A Vessel Crew Scheduling Problem: Formulations and Solution Methods

Thesis for the degree of Doctor of Philosophy

Alexander Leggate
Department of Management Science, University of Strathclyde

July 12, 2016

Abstract

Crew scheduling problems have been well studied in a variety of areas within transportation and logistics; however, the application of mathematical modelling techniques within a specifically maritime setting has received relatively little attention in the literature. This research focusses on the crew scheduling problem faced by a large, global company providing support services in the offshore oil industry.

Starting with an introduction to crew scheduling and the maritime industry, this thesis goes on to review literature in related areas before giving details of the broader business context and the specific crew scheduling problem in which we are interested. With the company currently using a manual method to update their crew schedules weekly on a rolling basis, often under time pressure, we argue that there is scope for a decision support tool to be introduced which will help the crew planners to find feasible, and potentially good quality, schedules.

Two main formulations are presented - a *Task-Based* model, which makes simplifying assumptions about crew contracts and working patterns, and a more realistic but more complex *Time-Windows* model. Both of these models can be likened to the crew recovery problems seen in other transportation scheduling literature, and can be solved with the objective of minimizing either the number of changes from the existing schedule or the cost of these changes.

While the Task-Based model proved relatively easy to solve, a number of solution methods had to be considered for the Time-Windows problem. Ultimately, a heuristic method was proposed to underpin the scheduling tool. This heuristic method finds an initial solution with a low number of changes, before performing a neighbourhood search which seeks to reduce the solution cost. Results however show there is still much room for improvement, and the thesis concludes with ideas for further extending this research.

Contents

Chapter 1

Introduction

Crew scheduling problems have been studied in a variety of contexts within the field of transportation and logistics. Numerous formulations have been proposed to solve the problem under various sets of rules and with a number of differing objectives. Many various solution methods have been proposed, ranging from exact solution methods to approximation algorithms and heuristics. However, all these cases display a number of similarities common to crew scheduling problems in any given setting.

These common characteristics of crew scheduling problems begin with the fact that there will be a set of employees (the crew), and a set of tasks which should be carried out. These tasks will generally have a defined time window in which they will be carried out, and most likely a location or route on which they will be performed. The aim of the crew scheduling problem is to allocate the crew to these tasks in the best possible way (as defined by the objective function), subject to various legal and contractual requirements, as well as physical constraints imposed by the geographical and temporal aspects of the tasks.

1.1 Importance of Crew Scheduling Problems

There are a number of reasons why crew scheduling problems are important to study. Firstly, for most of the transportation industry, crew costs are a large (if not the largest) single cost element in their operations. With increasing global competition, there is a constant motivation for companies to keep their costs as low as possible, and this can be achieved through effective planning of operations. It is often the case that even a small percentage saving in crew cost translates to a very significant amount in real terms.

There is also the matter of the difficulty in solving crew scheduling problems. Because of the number and complexity of the constraints that are involved, finding even a feasible schedule for the problem in question is often a non-trivial task. Any organisation currently scheduling their crew by hand will therefore be focussed almost exclusively on finding a feasible schedule, regardless of cost or any other requirement.

However, the complexity of the problem also means that even if the process is automated then the computational time required to find good solutions can be longer than desirable. This is especially true if a company requires schedules to be produced in real-time (or near to real-time). There is therefore often a trade off between the time taken to find solutions and the quality of these solutions. Consequently, there is still much interest from the research community in developing algorithms which can find better solutions in shorter time-frames, and so it is not surprising that a great deal of literature has been devoted to this. In particular, the airline crew scheduling problem appears to be the most well studied, although there have also been publications in the context of train and bus crew scheduling (reviews by Ernst et al. (2004a,b) and more recently Van den Bergh et al. (2013) discuss a variety of application areas).

Once the schedules have been drawn up, there is still the potential for them to require change at a later date. In particular, it is likely that one of any number of incidents could occur to disrupt a schedule as it is being executed, ranging from break-downs or signal failures, to traffic jams or road accidents, to weather-related disruptions. Known as the crew *recovery* problem (and sometimes referred to under the heading of "disruption management"), this is a natural extension of the crew scheduling problem, and is also subject to a great amount of discussion in the literature. It is often critical that solutions are produced in real-time, and so there is an interest in innovations that will make the problem easier to solve, for example techniques to reduce the size of the rescheduling problem as used by Rezanova and Ryan (2010).

1.2 Vessel Crew Scheduling

It is interesting to note that, with all the importance that is attached to crew scheduling, literature applying optimization techniques to crew scheduling for companies operating fleets of ships appears to be extremely limited. This is surprising, since crew costs in the maritime transport industry are no less significant than in other transportation areas. Stopford (2009, pp. 226-229) for example states that crew costs, from basic wages through to pension payments and travel costs, can account for up to half of a vessel's operating costs[1], and so it is not clear why this should be the case.

In their overview to maritime transportation, Christiansen et al. (2007) discuss possible reasons for the relative lack of papers concerning scheduling in general in a maritime context, and suggest the following:

[1]Operating costs are defined by Stopford (2009) as the day-to-day running costs such as crew costs, stores, regular maintenance, etc., but not including fuel costs.

- The fact that the shipping industry (in particular, cargo shipping) has a low visibility as compared to road, rail or air.
- The fact that maritime transportation problems are varied and so decision support systems require a higher degree of customization.
- The level of uncertainty in maritime operations, leading to a need for frequent re-planning.
- The tradition in the shipping industry makes it less likely to be receptive to new ideas.

While three of these points are consistent with our experiences in our case study, the argument concerning high levels of uncertainty has a converse - frequent replanning can be difficult or time-pressured, and could be greatly aided by some kind of decision support system. This is discussed more fully in section 4.2.

In more general terms, it is also possible to identify some of the ways in which the crew scheduling problem in the maritime context differs from other transportation settings. Firstly we must consider the kind of company with which we are dealing, as there are clearly many companies which fall under the description of *maritime transportation.* For example, in terms of scale a company may operate local ferry services between groups of islands, or at the other end of the range may operate on a global scale; meanwhile, in terms of size the company may be a small independent company with a handful of vessels, right up to a large multi-national business which owns a large fleet. For our purposes here, we consider the case of a large company operating numerous vessels on a global scale, as this is potentially the most interesting and most diverse from other settings in terms of crew scheduling.

There are also numerous modes of operations in which a maritime company can be engaged. Christiansen et al. (2013) discuss three broad areas for maritime transportation:

- *Liner* shipping, whereby ships run to a timetable on pre-determined routes, carrying either goods (*cargo liners*) or people (*passenger liners*) which require or desire transportation on that particular route.
- *Tramp* shipping, which generally involves cargo transportation on an *ad hoc* basis, travelling from port to port in response to relevant supplies and demands with a view to maximizing profit.
- *Industrial* shipping, which is similar to tramp shipping in that there are no pre-defined routes, but with the difference that the vessel owner is transporting their own cargo rather than transporting goods on behalf of third parties.

In addition to these, another category of vessel which cannot strictly be included within these modes of maritime transportation is that of the Offshore Supply (or Service) Vessel (OSV). The role of these vessels is to provide support services to offshore industries, such as oil and gas production as well the construction and maintenance of offshore wind farms, all of which are of significance in the UK at present. A brief introduction to OSVs is given by Barrett (2008), listing tasks such as construction support, remote operated vehicle (ROV) operations, surveying, diving support, and deep-water lifting and installation among those performed by this type of vessel. All of these are undertaken by our case study company.

Whatever the specific line of business of these global-scale companies, they will have a number of factors common to them which will impact on their crew scheduling. Not least of these is the large number of employees on a company's books. It may be the case that crew will have regular working patterns with regular recurring assignments; however this will not always be the case and complex crew scheduling problems may arise when these normal working patterns are disturbed.

One of the reasons why the problem may be more complex in these cases is the length of the planning horizon. A typical airline will schedule their crew on a monthly basis, while a bus company might be able to break down their problem to single days at a time. By contrast, a globally-operating shipping company will seek to draw up their schedules several months in advance and will plan several months at a time. The need for this stems partly from the length of the duty periods, which are in the order of weeks rather than hours in other cases, and partly from the global nature of operations where such matters as transportation and visas must be arranged well enough in advance for crew to be able to carry out their duties.

In turn, the arrangement of transportation is more complex here than in, for example, airline crew scheduling because of the relationship between crew, their home bases, and the location of the tasks. Generally, airline crew will have their tasks combined into groups, known as *pairings*, which start and finish at the hub airport where the crew are based. In the event that a pairing cannot be formed with two (or more) working tasks, a dummy task where an employee travels on a flight as a passenger rather than as crew can be included - this kind of assignment is known as a *deadhead* (see for example Barnhart et al. 1995). By contrast, normal practice in the global maritime sector is to employ crew from numerous countries around the world, each of whom will have their own designated home port from which the company will transport them to their assigned vessel. Deadheading in this case is not an option, since even in the unlikely event that one of the company's vessels is setting sail from the employee's home port the length of time taken to travel is significantly more than if a flight is taken. Therefore transportation costs are an important part of the cost of assigning a crew member to a task, and planning must be done sufficiently in advance that this transportation can be booked.

Other issues arise because of the rules, both legal and contractual, which are imposed on the problem. Some of these rules appear unique to vessel crew scheduling, while others are commonplace in scheduling, but may have a different bearing in this case. Firstly, there is the issue of maximum working period, which exists in all crew scheduling problems. In air, bus or rail transport, this would usually restrict the number or cumulative duration of tasks carried out within a working day, however in our vessel scenario it instead defines the maximum length of a single duty (in terms of days). Therefore, rather than an employee being allocated several consecutive tasks in this case, they are more commonly allocated single 'tasks', or blocks of offshore work, alternating with rest periods (i.e. periods during which they are not at sea).

Another consideration is the training and expertise of the employees, and whether they are suitable for certain tasks. This can depend on a number of factors, for example the type of vessel, the type of project it is carrying out, and the location of the vessel. It can also depend on the specific crew role, of which there can many on board the vessel. For example, an engineer may be qualified to work in the engine room of any ship in a company's fleet, whereas a deck foreman may require specialist skills depending on whether the ship is engaged in a diving project, a surveying project, a deep-sea pipe-laying project, etc. Comparing to other contexts, it appears that qualification constraints might be disregarded in some settings - Huisman and Wagelmans (2006) for example assume that all their bus drivers are identical. They do however arise in the airline context, as with a pilot's knowledge of a particular airport, as in Cappanera and Gallo (2004). It can be argued though that there are more of these constraints to consider in the vessel setting, especially when we also take into account another matter which could be counted as a qualification constraint - that of crew nationality. This has several consequences: ships registered under certain flags can only employ crew from certain countries, it may not be desirable for crew of certain nationalities to work together, and certain locations where the vessel operates may be nations that require work permits or visas for the crew working there.

A further element which complicates the vessel crew scheduling problem is the degree of uncertainty on the demand side. This is perhaps less applicable in the case of liner shipping, where routes and timings are clearly pre-defined, but in tramp shipping there is by definition an uncertainty in the future locations of the vessels, which could in turn make crew planning more difficult. In terms of our company in the OSV area of shipping, the uncertainty arises from the nature of the contracts with client oil companies. Vessels will be assigned to a particular project in a particular area, and these projects may require specific crew expertise. In some cases, the vessel will operate in that area for a substantial period of time. However, in some cases the vessel's role in a particular project will be relatively short term, as compared to the length of the planning horizon, and so when roles are being assigned on that vessel it is not known with certainty that this role will eventually be required. Similarly, new projects might be arranged at short notice, and so even after the initial scheduling has taken place, changes may need to be made. Consequently, as is discussed in greater detail in section 4.2, these companies will schedule crew on a rolling basis, initially scheduling assignments several months in advance, and updating the uncertain tasks as more information becomes available.

1.3 Outline of the Thesis

In light of the importance of the crew scheduling problem to our company, and the difficulties which are often experienced in finding suitable solutions, this thesis sets out to propose a framework for a decision support tool which could aid the planners in their decision making. The research process began by discussions with the client company in order to further understand the problem. Based on these discussions, two model formulations are proposed - a simplified “Task-Base” model, and a more realistic but more complex “Time-Windows” model. As full real datasets were not available from the company, datasets had to be generated at random within parameters provided by the company and based on some assumptions. Using these datasets, various solution methods could be applied to the two formulations, with the approaches being compared with respect to key metrics such as solution time, solution quality, and number of solutions found. Statistical techniques were later applied to determine whether the factors varied during data generation had a significant influence on the relative performances of the solution methods. The same statistical methods were also applied when testing the heuristic solution method designed, in order to determine the most suitable group of settings to use in this algorithm. Based on the results from examining these solution methods, possible structures for a decision support tool to aid the planers at the client company can be proposed.

In more detail, the thesis proceeds as follows. Firstly, in chapter 2 we examine relevant and related literature in more detail. This includes crew scheduling literature from other transportation settings such as airline or urban transit as well as scheduling in a more general context. We will also see that there is a body of literature concerning optimization problems in maritime settings, although there appears to be very little which can be termed “vessel crew scheduling”. The chapter concludes with a discussion of those papers found which can be categorised as considering a vessel crew scheduling problem. Following this, chapter 3 outlines the methods used in this research, including an overview of some key statistical concepts, before the detailed problem description is given in chapter 4. This sets out the crew scheduling problem as it appears at our case study company, a large company operating OSVs to provide services to the oil industry world-wide, including a discussion of the company’s current approach and proposals for how a this could be improved upon.

The next two chapters present formulations and solution methods for Vessel Crew

Scheduling Problem. Firstly in chapter 5, a simplified formulation of the problem described as a ‘Task-Based’ formulation is given. This formulation is shown not to be realistic enough to be used directly on our particular problem, but is argued to still be relevant and to provide a means of solving the Vessel Crew Scheduling Problem at a company where the simplifying assumptions do hold. Two solution approaches using a commercial solver are presented, one which returns the lowest cost solution found within a given time limit, and the second which minimizes the number of changes from the current schedule. This change minimization approach applies a cost limit which can be reduced iteratively, producing a Pareto-optimal set of solutions rather than a single solution. Computational results for both solution methods are given, along with a proposal for how these methods can be used in practice at a company to support the solution of their Crew Scheduling problem.

Chapter 6 meanwhile presents a more realistic, but more complex, formulation for the Vessel Crew Scheduling Problem, termed the ‘Time-Windows’ model. The methods used to solve the Task-Based problem were far less effective on this more difficult problem, and so other solution methods are proposed here, including using the Task-Based problem to approximate the problem and Heuristic methods. A Column Generation approach is also proposed although the effectiveness of this on the problem was not tested in this research. Of the methods that were tested, the Heuristic approach appears to be the most useful in providing multiple solutions of a reasonable quality in a short time, although it is noted that improvements could still be made to the algorithm. As with the Task-Based problem, proposals are given for how these methods can be used as part of a decision support tool in practice.

In both chapters 5 and 6, ideas are also proposed for future work, and these are drawn together as part of the conclusions given in chapter 7. In this final chapter, we also discuss how the work fits in to the broader context of other scheduling literature as well as the more specific implications for crew scheduling at our company and for others operating in the maritime industry.

Overall, the contributions of this thesis are as follows. Firstly, it applies an optimization approach to a crew scheduling problem which has not previously been discussed in the literature. Indeed, while some authors have suggested that crew scheduling in this context is not of interest, it is shown here that the company’s planners would in fact benefit from assistance in finding suitable schedules and that potentially the company could make considerable financial savings by employing such an approach. As discussed further in chapter 7, we can also see that the models proposed here could be useful in other similar settings.

Meanwhile, a more specific contribution is made by the development of a heuristic algorithm for solving the crew scheduling problem. Some observations and conclusions relating to the settings used here may also be relevant to solving similar problems by the same approach. In particular, it is concluded that the heuristic algorithm is more effective not when the search space is examined exhaustively at each iteration to find the best possible new solution, but rather operates better when a subset of the neighbourhood is considered with any improving solution accepted. In addition, it is concluded that the random perturbations ("kick") used to move the search away from local optima is more likely to do harm than good. This is likely because the kick is not well enough defined for the contour of the search space, where large numbers of local optima appear to exist, some with much greater solution values than others. This lesson can be generalised to other local search algorithms operating in similarly unpredictable search spaces.

Chapter 2

Literature Review

In this chapter we review literature from areas related to vessel crew scheduling, and discuss what kind of bearing these ideas and problems can have on our problem. An idea of the areas which may be of interest can be obtained from reviews of staff scheduling applications, formulations and models such as that given by Ernst et al. (2004b).

Amongst other things, this paper discusses the crew 'rostering' process, which the authors see as being divided into six modules (although not all models will consider all of these, while others may consider modules in combination or concurrently). The modules they discuss are:

1. Demand modelling;
2. Days off scheduling;
3. Shift scheduling;
4. Line-of-work construction;
5. Task assignment; and
6. Staff assignment.

Looking at demand modelling, which is a potential concern in our case, Ernst et al. (2004b) divide this into task-based demand, flexible demand, and shift-based demand. Flexible demand is the case where the "likelihood of future incidents are less well known" and so forecasting / queuing theory techniques are used to predict what staffing might be required. The other two forms of demand modelling come about when demand is well known in advance, or the shift requirements are already laid out, e.g. for nursing.

It is also worth noting some of the comments on constructing lines of work. This stage refers to the creation of rosters for each staff member over the planning horizon. It is noted that this can be either cyclic (i.e. giving regular day-by-day or week-by-week shift patterns), or acyclic (e.g. in a call centre where demand varies from day to day). Also commented on here are the constraints that are present - there are “hard” constraints, and “soft” constraints. The hard constraints must be satisfied, but the soft constraints can be violated, although this is not desirable. In these cases, the soft constraints may be included in the Objective Function, with an appropriate penalty cost coefficient.

The paper discusses several application areas, including transportation as well as others such as call centres and financial services. Transportation is highlighted as being a diverse area, but with two common features across all problems: the tasks have both temporal and spatial dimensions, and all tasks are dictated by some kind of timetable. This is interesting to note, as it highlights a potential difference between most transportation crew scheduling problems and some shipping problems, for example in the case of a ‘tramp’ shipping company (discussed in more detail in section 2.3) where there is by definition no timetable and the routes of the ships are not known in advance.

By contrast, in the call centre situation, exact details of tasks and demand are not known in advance. Instead, it is only known that a certain number of staff are required at any one time, and this information is obtained from forecast demand, and from queuing models and simulation. However, there is also a difference between this situation and a shipping company, in that all work is carried out at only one location - there is no ‘spatial dimension’ to the tasks.

Finally, financial services provide an interesting application area in the form of audit staff scheduling. Here, demand is not known well in advance, and staff have to travel to various locations to carry out the work. There is also a mix of skill requirements on audit jobs, much as there are on ships. According to Dodin et al. (1998), the lack of advance knowledge of demand is dealt with in auditor scheduling by having a relatively short planning horizon, i.e. planning is only carried out over such a horizon that demand is known.

A final point to note from the review by Ernst et al. (2004b) is regarding computer software which was available at time of writing. In particular they note that packages which exist tend to be either too industry- or organisation-specific to be transferred to another application area, or are too general to facilitate any automated roster generation. This would suggest that any existing crew scheduling software would not be directly applicable in our specific problem.

Having identified transportation settings as one of the key areas of interest, we begin the body of this review by looking in section 2.1 at crew scheduling in non-maritime transportation settings, before going on in section 2.2 to look at more general scheduling literature which is relevant to our work. Next, section 2.3 looks at how optimization techniques are applied in other maritime contexts, before section 2.4 discusses in particular the related (but distinct) Sailor Assignment Problem, which occurs predominantly in naval vessels. Finally, in section 2.5 we outline the few pieces of literature which discuss what can be described as a vessel crew scheduling problem.

2.1 Transportation settings

As mentioned, it would appear that only a small amount of literature exists which discusses crew scheduling in a maritime setting, and we discuss this later in the chapter. Firstly however, we note that there is a large body of publications concerning the scheduling of employees in other transportation settings. These predominantly concern crew scheduling for airlines, although bus and train crew are also considered. In this section, we review these two areas of the literature, and discuss the similarities and difference between the problems tackled in these pieces of research and the problem with which we are concerned. We also have a separate discussion of so-called 'schedule recovery problems', where the objective is not to create a new schedule but to correct one which is no longer feasible, for example because of a delayed flight.

2.1.1 Scheduling for Air Crew

The majority of transportation scheduling literature relates to airline crew, and it is suggested by Christiansen et al. (2007) that "in many aspects aircraft are similar to ships" (p.192), in terms of factors such as operational uncertainty, the international nature of their routes, and the size of capital investment involved. It must therefore be considered that scheduling of airline crew might have a bearing on how to approach vessel crew scheduling.

There are some aspects which are common to a large amount of crew scheduling literature in general, and airline crew scheduling literature in particular. For example, the majority of researchers appear to use real-world problems and real-world data to model their problems, with the collaboration of a company. In addition, a large number of models which are produced seem to rely on a network flow formulation of some kind. However, as can be seen in the remainder of this section, there are also some differences between the ways the problem is approached.

Firstly, there is the question of the way in which the problem is modelled. One key difference is that some models seek to schedule both crew and aircraft together, while others will only focus on a particular stage of the overall problem. Generally, it is this second option which is taken, with papers dealing specifically with the *crew pairing problem* (where flights are grouped into *pairings* that start and end at the same base) or the *crew rostering problem* (Kohl and Karisch 2004), or ignoring crew altogether and instead looking at the *aircraft assignment problem.* However Papadakos (2009) for example suggests several ways of integrating the problems so as to achieve a better overall solution. An obvious down side of an integrated approach is that it makes for a much larger model, and therefore a problem which is harder to solve. It would also make little sense to model in this way in a situation where aircraft (or vessel) locations are pre-determined and therefore unaffected by the requirements of the crewing problem. In these cases, it would probably be the crew rostering problem which is the most comparable to the vessel crewing problem.

Secondly, while it seems common to develop a model which uses a network flow representation, there are several different ways in which this can be applied. For example, Cappanera and Gallo (2004) define a multicommodity network flow problem, with a network to represent the tasks to be carried out, and the crew as commodities which must flow through the network. A contrasting use of networks is that employed by Gamache et al. (1999), who define a column generation method with the master-problem formulated as a simple IP to select the best rosters from a set of possibilities, while the sub-problem generates schedules for each employee which might be useful for improving the solution. It is the sub-problem in this model which makes use of the network formulation, with nodes representing time points, arcs representing tasks or rest periods, and arc costs being allocated based on the reduced costs taken from the master-problem. Promising schedules are therefore found by identifying the shortest path through the network.

Solution methods will also form a significant difference between different pieces of research. Gamache et al. (1999) for example use a path-building algorithm for their sub-problem, which makes use of 'Resource Extension Functions' (more detail on which can be found in Irnich 2008). The master problem is solved using a variant on branch and bound, where high fractional values are heuristically rounded up in order to reduce the search space. Computational experiments in this case have shown that solution time is reduced by a factor of at least 1,000 with this heuristic, while the solution obtained had a value in the worst case that was still within 0.6% of the optimal linear relaxation solution.

Partially, based on the approach by Gamache et al. (1999), Butchers et al. (2001) use a similar Resource Constrained Shortest Path (RCSP) method for roster generation for their crew scheduling problem at Air New Zealand. Meanwhile, Kohl and Karisch (2004) discuss a related column generation technique which applies a *k-shortest path generator* to the RCSP. Rather than considering the roster rules during path construction, this method works by finding the shortest path in terms of reduced cost and subsequently testing it for feasibility - if the roster is infeasible, it looks for the next shortest path and so on until a feasible roster is found.

An RCSP-type column generation approach is also used by Saddoune et al. (2011) for an integrated crew pairing and scheduling problem. A difference here from the above papers is the use of heuristics to created clusters of variables and constraints which can be aggregated in the master problem. It would appear that the use of heuristic methods are a more recent development in airline crew scheduling, with mention of heuristic techniques in a review by Gopalakrishnan and Johnson (2005), which focusses mainly on papers published before 2000, largely restricted to such matters as branching strategies applied within a branch-and-bound approach. More recently for example, Maenhout and Vanhoucke (2010) have developed a hybrid scatter search heuristic, which is a form of evolutionary algorithm, for solving the air crew rostering problem.

Others on the other hand have been less willing to compromise on optimality. Hoffman and Padberg (1993) for example argue that because of the large amounts of money that airlines spend on crew costs (they state that major US carriers may spend over \$1.3bn per year on crew, which would equate to around \$2.2bn in today's terms[1]), any percentage saving, however small, will in fact mean a substantial monetary saving. They propose a set partitioning problem formulation, which can be solved using a branch and cut method, which includes the generation of valid inequalities, to give a provably optimal solution. Clearly how researchers approach this speed-versus-optimality trade-off will depend very much on the needs of the client company, assuming this is the format that the research takes.

Another approach to the airline crew scheduling problem altogether is taken by Gamache et al. (2007). Here instead of being concerned with the cost of the schedule, rosters are allocated to crew based on their preferences. In order of seniority, each crew member in turn is allocated their most favoured schedule provided it will still allow a feasible allocation to the remainder of the employees. The feasibility test is formulated as a graph colouring problem - nodes represent the tasks yet to be assigned, and two nodes are connected if their associated tasks could not be performed by the same worker. If the minimum number of colours is greater than the number of employees still to be allocated a roster, then the last schedule allocated does not leave a feasible solution for the remaining workers. Clearly in the context of scheduling at an airline where such a preferential bidding system is in operation, such a method would be acceptable. However, it should be obvious that the overall satisfaction with the entire staff with the resulting schedule is likely to be far from optimal, and therefore it could be argued that such a method would have no place in a situation where costs, rather than preferences, are the priority. However, it should be noted that this method will find a feasible solution within minutes, compared to hours that an IP model might take to solve, and so a solution found using this method could potentially be used to aid a branch and bound approach by providing an upper bound on the solution value.

[1]The inflation rate of the US Dollar from 1993 to 2016 was $\approx 2.31\%$, meaning \$1.3bn in 1993 would be valued at approximately \$2.197bn in 2016. This was calculated using Calculator.net (2016).

2.1.2 Scheduling Literature for Bus and Train Crew

There is also a smaller amount of literature on crew scheduling in other settings. An example of a model used for rostering of bus crew is Haase et al. (2001). The approach proposed here is simultaneous scheduling of both vehicles and drivers, which makes sense since a bus requires a single driver and cannot travel without one. It is also argued by the authors that "crew costs dominate vehicle costs", and therefore it would make little sense to first schedule the movement of buses, then find a driver schedule which would fit with this - this would most likely lead to a sub-optimal solution over all. The planning horizon is one day, since for the most part buses will return to the depot at the end of each day. We can see that this does not hold much similarity with the possible problem of global-scale vessel crew scheduling, but might have some relevance to smaller scale, short distance services.

The problem is formulated as a set partitioning model with side constraints, with a network designed with nodes representing points in time and space and linked by arcs which represent travel or waiting periods. The solution methods suggested here are column generation - where possible sets of duties for the crew are created - and branch and bound. Perhaps the most interesting point about this particular model however is that it has been built from a much more theoretical basis than the majority of crew scheduling literature appears to have been. Many researchers will look to work with a company and build a model based on that company's specific problem, while Haase et al. (2001) have built a model around a general vehicle and crew scheduling problem (VCSP). They have gone on to test their model using data which has been randomly generated in such a way as to represent a plausible real-world problem. For this reason, it could be argued that their model is more robust since it has not been constructed for a specific company or scenario, and so it could be a more generalisable and theoretically useful model. Conversely, it has the disadvantage of not having been tested on an actual real-world instance of the problem.

Other existing work in the literature use a column generation approach similar to Gamache et al. (1999) discussed for airline crew scheduling above. Both Jütte et al. (2011) and Jütte and Thonemann (2012) use an RCSP formulation to generate rosters for their railway crew scheduling problem, and use the similar dynamic programming algorithm to find feasible reduced-cost rosters. A similar technique is also used to set equal to 1 certain variables in the master problem in order to achieve quicker convergence to an integer solution.

Heuristic approaches also feature in the scheduling literature for bus and rail transport. Elizondo et al. (2010) for example developed an evolutionary algorithm, which is used to construct partial schedules for crew in an underground rail network. Interestingly, in this paper the new approach is compared with a tabu search algorithm and with a 'greedy' heuristic, using a mixture of real and randomly generated instances. The results were mixed, with no method significantly outperforming the others. Meanwhile, Lourenço et al. (2001) use heuristics to solve a bus driver scheduling problem. Their approach is different to that of Haase et al. (2001) described above, with the problem being formulated as a multi-objective problem rather than simply cost-based. For their problem, Lourenço et al. (2001) have developed a metaheuristic approach, combining tabu search and genetic algorithms to create their solution method.

2.1.3 Crew Schedule Recovery Problems

The above discussion of scheduling literature has been concerned with constructing a schedule from scratch. However, as is discussed in detail in the problem description given in chapter 4, the nature of our crew scheduling problem is that schedules are constructed on a rolling basis, with assignments which are put in place perhaps later being updated or amended. This has some similarities with the 'recovery' problem, which arises in other transportation settings when disruption has occurred to the schedule which had been planned. Therefore we will here outline some of the literature in this area.

An overview of some of the most recent methods of disruption management in the airline industry is provided by Clausen et al. (2010). They observe that one of the key goals of the recovery problem in this setting is to solve the problem quickly, leading to techniques being developed to reduce the problem size. This can be done by reducing the time window for which tasks are re-planned, as opposed to re-planning the entire day's tasks; also, the number of crew being re-assigned can be restricted to only the affected crew plus a certain number of additional *candidate crew*.

Also observed in this overview is the different assumptions under which the airline crew recovery problem can be formulated. For example, some authors might accept that additional flights could be cancelled or delayed as part of the plan to bring crew operations back on schedule. This however does not seem to tie in very well with the nature of the vessel crew scheduling problem which we have investigated, since one of the key concerns for the company (even over and above cost) is fulfilling the requirements of the clients without crewing difficulties affecting operations. This seems more in line with those authors who assume that the flight schedule has been fixed before the airline crew recovery problem is solved.

There is also a difference in crew recovery models in the airline industry which is highlighted by Nissen and Haase (2006) - the difference between the models designed for European and for US airlines. The reason for the difference here is the way that crew are paid in the two regions, with European airlines generally paying their crew fixed salaries, and North American crew being paid according to the time they have worked. The authors here argue that this changes the focus of the crew recovery models in each case - the North American payment system leads to an emphasis on cost-minimization, which in turn means that there is close reliance on *pairings* for the problem in this region, since they are the main base units of costs. Conversely, it is suggested that with crew salaries fixed, the cost implications of changing the schedule are reduced, and the main emphasis turns instead to making the new schedule as close as possible to the previous, disrupted one. As a result, the pairings are less important, and the rescheduling can be done based on the individual duties to be carried out.

In terms of problem formulation, as with crew scheduling problems it appears that networks are often used to represent the models. Nissen and Haase (2006) propose a *duty-period-based* network model, while Wei et al. (1997) describe their model as having a multi-commodity network flow formulation. Similarly, Rezanova and Ryan (2010) describe a resource constrained shortest path problem to generate feasible new schedules for their crew.

What distinguishes the Rezanova and Ryan (2010) models from the others discussed here is that it is designed for the train driver recovery problem rather than for airlines. However, it is worth noting that it still exhibits some of the same properties as the airline models - as well as the network-based formulation that is present, the authors also make use of the two techniques discussed by Clausen et al. (2010), reducing the size of the problem by restricting the time over which duties are considered, and also the number of crew considered. Also, the priority in this case is to return to the original schedule as quickly as possible, similar to the European airline problem discussed by Nissen and Haase (2006).

An alternative approach to the railway crew recovery problem can be found in Potthoff et al. (2010). In this case, the problem size is reduced by first considering the ‘core’ problem - i.e. the subset of tasks or duties which *must* be carried out. Once an initial solution to the core problem has been constructed, their algorithm improves the solution by examining the possibility of adding in other duties from the ‘neighbourhood’. The neighbourhood in this case is defined to include the duties and tasks which are not covered by the current solution.

2.2 Scheduling Problems in General

Outside the area of transportation, there are various other settings where the scheduling of crew or other personnel takes place, and indeed other problems where resources or activities rather than employees must be scheduled. As well as the review by Ernst et al. (2004b) discussed in the introduction to this chapter, there have been several papers reviewing crew and personnel scheduling published in the past decade. One such recent review is by Van den Bergh et al. (2013), which discusses over 300 papers in the area of across many application areas. Their intention is to provide an up-to-date review of personnel scheduling literature published since the review by Ernst et al. (2004b), an annotated bibliography by Ernst et al. (2004a), and others. It is interesting to note here that there are no commercial maritime applications mentioned, but there are some 'military' applications discussed. These include shipboard manpower planning in the US Navy (Li and Womer 2009), which is discussed in detail in section 2.3, and crew scheduling in the Royal Australian Navy (Horn et al. 2007), which is discussed in detail in section 2.5 below. An interesting observation made by Van den Bergh et al. (2013) relates to how personnel scheduling has changed since first being introduced in the 1950s. They observe that "the relative importance of satisfying employee needs in staffing and scheduling decision has grown" (p.367) - certainly we know this to be the case at our company, and it has played an important part in how we have approached solving the problem.

Looking towards other specific areas, there are several examples in the literature of scheduling problems in the telecommunications industry, where tasks must be scheduled or assigned to engineers or technicians. Tsang and Voudouris (1997) for example develop two local search algorithms for scheduling engineers for British Telecom (BT), a 'fast' local search and a 'guided' local search. Tests carried out show both algorithms compare favourably, both in terms of solution quality and computational time, with genetic algorithm and simulated annealing methods for this problem. Nguyen and Wright (2014) meanwhile develop a variable neighbourhood search algorithm to solve the 'workload balancing problem' for a telecoms company. Here the problem is not necessarily to assign the tasks to specific employees, but to balance the workload across time periods based on anticipated demand. Imbalance is measured as the sum of squared deviations, a quadratic expression, which must be minimized. The algorithm developed here is shown to generate high quality solutions in a short computational time. The problem studied by Kovacs et al. (2012) is to find a routing and schedule for service technicians, with the possibility of non-covered tasks to be 'outsourced' in a comparable way to the ability to use 'agency crew' in our problem (see problem description in chapter 4). Their solution method is the 'adaptive large neighbourhood search', or ALNS, approach introduced by Ropke and Pisinger (2006), in which heuristic operators are chosen for each iteration according to how they have performed at previous iterations. This approach was shown to give around 10% lower cost for real world instances compared to manually generated solutions.

An alternative application area for scheduling is that of sport. This can for example concern fixture scheduling, such as that of an ice hockey league as studied by Nurmi et al. (2014), or scheduling the Brazilian football league fixtures as discussed by Ribeiro and Urrutia (2012). In both these cases, there are a number of 'soft' constraints which should be satisfied where possible, and 'hard' constraints which must be satisfied at all times. In both these papers, the authors take a multi-stage approach, breaking the problem down into subsections - in the case of Nurmi et al. (2014), this begins with creating 'minitournaments' hosted by specific teams, while Ribeiro and Urrutia (2012) begin by creating feasible home-away sequences. From here, a full fixture schedule can be built up in steps. A different problem which is discussed by Wright (2007a) is that of allocating umpires to a set of fixtures in the Devon Cricket League. A metaheuristic approach is developed to solve this problem, although this paper is interesting in that it notes that the formulation of the problem was much harder to achieve than the solution approach. The author notes that this is in contrast to most academic literature on scheduling, which "tend to assume that formulation is reasonably straightforward" (p.439). In the end, a formulation was arrived at where the majority of constraints were treated as 'preferences', with an associated term in the objective function which penalises violation of these preferences, in some cases in a non-linear manner.

The problem studied by Wright (2007a) is also the motivation of some detailed heuristic experiments in Wright (2007b). Here, an iterated local search (ILS) algorithm is applied to the umpire scheduling problem. The paper sets out the key elements of a generic ILS algorithm - an initial solution, a local improvement scheme, a perturbation or 'kick', solution acceptance criteria and stopping criteria - but notes that some of the details of the algorithm must be determined by experimentation. The author outlines a detailed experiment to see the effect of varying such parameters as the number of iterations, the nature of the kicks which are carried out, and the rules for accepting new solutions. The rules for accepting non-improving solutions include a probabilistic settings, a simulated annealing rule, and a threshold acceptance rule. By carrying out 100 runs with each combination of settings, the author is able to draw a number of conclusions about the best settings to use, although it is noted that this is only strictly applicable to the problem instance in question, and that other experiments should be carried out.

2.2.1 Scheduling under Uncertainty

An important subset of the literature on scheduling problems is that regarding scheduling under uncertainty. Discussed above were recovery problems in transportation areas, but it is also worth considering how uncertainty affects other settings. Similarities can be drawn between uncertainties in our problem and, for example, scheduling processes in manufacturing or production, and it therefore seems useful to give a brief overview of the approaches and terminology used here.

Lütjen and Karimi (2012) discuss the problem of controlling inventory and scheduling for wind farm installation, and also present a general overview of scheduling under uncertainty, which it is interesting to consider with comparison to our case study problem. According to the authors, the three approaches to 'dynamic' scheduling are:

- The *proactive* scheduling approach, which attempts to create robust schedules which can handle disruption, but requires information to be known about the uncertainties and their distributions. This at present is not available at our company.
- The *reactive* scheduling approach, which works without an initial scheduling and effectively makes decisions on a real-time basis. Since logistics planning is required, and crew require advance notice of their postings, this would not be workable at our company.
- The *predictive-reactive* scheduling approach is a hybrid of these, where an initial schedule is created but is revised either periodically or in direct response to disruptions (or a combination of both).

It is interesting that, even though this terminology originates in production scheduling, it is applicable to our problem. As discussed in detail in chapter 4, the company's current approach to scheduling can be described as a predictive-reactive one, with revisions made periodically on a weekly basis (and occasionally with *event-driven* changes made if the disruption occurs very soon before the execution date). It is worth noting also that the recovery problems discussed in section 2.1.3 also fall into the category of predictive-reactive scheduling, with existing schedules being revised in response to disruptions.

More generally, there are several recent review papers in this area which may be of interest. One by Li and Ierapetritou (2008) discusses some papers on reactive scheduling, as well as giving an overview of preventative / proactive scheduling techniques including stochastic optimization, robust optimization, fuzzy programming and sensitivity analysis. A large amount of literature is also reviewed by Vieira et al. (2003) with regard to predictive-reactive scheduling and rescheduling methods, in addition to some useful definitions of terminology used in this area. Another survey has also been produced by Ouelhadj and Petrovic (2009) and, while it seems to add little in terms of definitions or discussion over and above the paper by Vieira et al. (2003), it does provide some more recent references in the area.

2.3 The Maritime Industry

As noted earlier, literature regarding the scheduling of crew is very scarce in the area of maritime transportation. However, operational research techniques in general, and optimization in particular, in this sector are in fact quite well used. At least two journals have devoted special issues to maritime transportation - see introductions by Psaraftis (1999) and Christiansen and Fagerholt (2011) for details - although neither of these contained any discussions of crew scheduling problems.

A large amount of shipping literature seems to relate to routing and scheduling. Christiansen et al. (2013) for example present a review of literature in this area, discussing problems relating to liner, industrial and tramp shipping (definitions of these shipping types can be found in section 1.2 above). This was the fourth in a series which had been produced roughly every ten years since 1983, the next most recent being the review by Christiansen et al. (2004), building on earlier ones by Ronen (1993, 1983).

A more general overview to maritime transportation is given by Christiansen et al. (2007), which we observed in section 2.1.1 likened shipping to air transportation. The paper also has the following to say about crew scheduling:

> *"Crew scheduling for deep-sea vessels is not a major issue. Crew members spend months on the vessel and then get a long shore leave. For short-sea vessels the crew may change frequently, and crew scheduling may be an issue."* (pp.263-264)

It is interesting that, even though our company falls into the category of *deep-sea* shipping, that our experience shows crew scheduling to be an important issue. We feel that this is an important contribution of this research.

2.3.1 Offshore Supply Vessels

Another shipping type mentioned (but not particularly covered) by Christiansen et al. (2013) is offshore supply vessel, or OSV. Barrett (2008) gives a brief introduction to this kind of shipping, mentioning that the vessels cover tasks including providing construction support, remote operated vehicle (ROV) operations, surveying, diving support, and deep-water lifting or installation. All of these are carried out by our company, and this is clearly the category into which our company falls.

A search of the literature related to the OSV area of shipping also reveals a focus on routing or scheduling of the vessels, with no mention of the crew. Examples of this include Gribkovskaia et al. (2008), Halvorsen-Weare et al. (2012) and Fagerholt and Lindstad (2000). A narrower definition of the offshore supply vessel seems to be used by Aas et al. (2009), which focusses only on the actual supply of oil platforms rather than the supplementary services mentioned above.

2.3.2 Other Shipping-related Literature

There are numerous other problems which have been studied connected to the shipping industry. Fagerholt et al. (2010) for example outlined a simulation- and optimization-based methodology for a strategic planning problem. This, it was proposed, could be used as a decision support tool for routing and scheduling, contract analysis, and decisions around the fleet size and mix problem. Other papers discuss offshore wind farms, with Scholz-Reiter et al. (2010) proposing a MILP model for planning the installation of the wind farms, taking into account the uncertain nature of the weather conditions, and Lütjen and Karimi (2012) on the presenting a simulation-based model for controlling inventory and scheduling wind farm installation.

Other papers discuss scheduling of on-shore operations linked to shipping. Legato and Monaco (2004) for example discuss the scheduling of employees at a marine container terminal. Interestingly, their problem like ours involves a large degree of uncertainty, in this case primarily arising from the variable arrival times of vessels at the terminal. However, unlike our problem crew only need to be present when a vessel is requiring loading or unloading, and hence there are no "well defined 'tasks' to perform... with fixed start and finish times" (p.770). In this case, the problem can be decomposed into a long-term planning side, where working days are determined (since arrival days of vessels can be known with accuracy), and a short-term planning stage where shift patterns are later determined (since exact arrival times are not certain).

One paper which relates a little more closely to vessel crew scheduling is by Li and Womer (2009). This paper deals with the *shipboard manpower scheduling problem*, with the objective being to allocate crew duties during the course of a voyage, and to use a minimal number of crew. A MILP formulation is presented, which can be solved using a multi-phase decomposition method with the emphasis on finding a feasible solution. Parallels can be drawn between this problem and the vessel crew scheduling problem. For example, in both cases crew members can be *multi-skilled* (i.e. able to fulfil a number of different roles on board a ship), but for any single journey or duty period they could be allocated a single role or task. Also, in our problem we will have a number of roles and time periods to which a crew member might be assigned, and there will be temporal restrictions on these, similar to those present in Li and Womer (2009)'s model. However, there will also be some differences, for example no consideration is given by Li and Womer (2009) to the issue of transportation. In our problem, a significant consideration is the cost of, and indeed time required to arrange, the transportation of a crew member to and from the vessel to which they are assigned.

2.4 Sailor Assignment Problem

Another problem which is related, but distinct, from the crew scheduling problem is the sailor assignment problem. This problem has been present in the literature for some time. Liang and Thompson (1987) discuss the first implementable computer model for the problem, but also note that some unsuccessful attempts at this were made in the 1960s. More recent descriptions of the sailor assignment problem include Blanco and

Hillery (1994), Holder (2005) and Garrett et al. (2007). From these papers, it is possible to identify some similarities and also some key differences as compared to our crew scheduling problem.

One of the similarities between the two problems is the time scale over which planning is carried out, with Holder (2005) stating that the sailors must be assigned their new job six to nine months in advance, as compared to the ideal of three to six months at our company. The key difference however is that each sailor goes through a *rotation* roughly every three years, and will carry out their new assignment until the time of their next rotation.

Linked to this, it is worth noting that it is the sailor who contacts the *detailer* to look for their next assignment, rather than the other way round. At this time, the detailer will create a list of jobs from the current *requisition list* (i.e. the jobs that are due to be assigned in six to nine months' time) to offer to the sailor. There is a similarity here in that, while our company does not offer a list of tasks to the employee, there is a degree of discussion between planners and crew as to which tasks they are to be assigned. This aspect of the process gives rise to a requirement that the optimisation must be done quickly, as with at our company, since the list must be created while the sailor and detailer are in conversation.

An interesting aspect is the role that time factors play. In the Navy's assignment problem as described by Holder (2005), the requisition list is updated with new jobs every two weeks, while once a set of jobs has been offered to one sailor they cannot be offered to a second until a choice has been made or a deadline reached. The list of tasks, as with our problem, is therefore subject to change. Interestingly, Holder (2005) adopts a similar approach to us with regard to this, in that they assume that the information is fixed at the time of the model being run. However, a crucial difference is that no mention is made of tasks being changed once they are assigned - so far as it appears from the literature, once an assignment is made it is fixed.

In terms of the formulation of the problem, Blanco and Hillery (1994) mention "complex eligibility rules, multiple objectives and nonstandard constraints", but unfortunately do not present these mathematically. Garrett et al. (2007) meanwhile state that there are "some ninety goals and constraints", although the main emphasis seems to be on the construction of a *fitness function* which should be maximised in order to find the best solution. The actual constraints of the mathematical formulation given are very simple - no sailor is to be assigned more than one job, and to job is to be assigned to more than one sailor. Similarly, Holder (2005) defines a cost function aggregating training suitability, location, navy priority and the sailor's geographical preference, subject to the same assignment constraints.

In terms of solution methods, Holder (2005) proposes using a *path-following-interior-*

point algorithm to find all the jobs which can be assigned to the current sailor in an optimal (or near optimal, if a longer list is required) solution to this assignment problem. Even though the motives for requiring multiple optimal solutions are a little different from our problem, it is interesting to note that they are not necessarily concerned with a single solution to the mathematical formulation.

2.5 Vessel Crew Scheduling

As well as the numerous related papers discussed above, there is also a small group of papers in the current literature which can be categorised as discussing a vessel crew scheduling problem. All of these have a degree of similarity with our problem, but also have specific differences in the problem setting or the approach identified as the best for modelling or solving the problem. These papers are discussed, in turn, here.

2.5.1 Harbour Pilot Assignment Problem

The paper by Wermus and Pope (1994) is cited by Christiansen et al. (2007) as a "special type of [short-sea] crew scheduling problem", and discusses a simple heuristic for scheduling pilots in a harbour. While this is near to our problem in terms of setting, the problem definition is in fact quite different. The problem size for example is only seven or eight employees, compared to the hundreds which our company may have to deal with at one time. Also cost is not a concern in this case, with the objective being to allocate the employees to working and standby days in a "fair and equitable" manner. Unfortunately, no mathematical description of the problem is presented in the paper.

2.5.2 Navy Crew Scheduling Problem

As mentioned above (section 2.2), Horn et al. (2007) discuss a scheduling problem for the Royal Australian Navy. The key difference between their problem and ours is that theirs is an integrated vehicle and crew scheduling problem. However, it differs from other integrated problems in that there is no fixed timetable for the vehicles to perform, only a set of vessel assignments which must also be scheduled. It turns out that the crew side of the problem is not as complex as in our case, as crews operate as a group with no interchange between them - crew can therefore be assigned as a group to a vessel rather than having to be considered individually. The problem however was still found to be too complex to be solved using an ILP model, and so a simulated annealing procedure was developed.

2.5.3 Ferry Crew Scheduling Problem

Another paper which relates very closely to our problem is by Ammar et al. (2013), which discusses to the scheduling of crew on board ferries operating between Sfax and the Kerkennah islands in Tunisia. As with Horn et al. (2007) above, it appears that crew are allocated in teams rather than as individuals. There is also a difference to our problem with respect to timescales - being a short-distance ferry service, the crew are expected to work for periods in the order of hours, with restrictions on the number of hours between shifts and the number of days on which they are permitted to work. The constraints imposed are therefore similar to those present in airline or urban mass transit systems rather than the long term nature of our problem. There is therefore not the issue of uncertainty associated with long-term planning that we face in our problem.

2.5.4 Cruise Crew Scheduling Problem

Perhaps the most similar to our problem in terms of problem description is that studied by Giachetti et al. (2013), who propose a “decision support system for crew scheduling in the cruise industry”. The company under focus in this paper operates cruises in various locations around the world, ranging from three nights to a month or more in length. In terms of the global scale of the company, this is similar to our problem, but the key difference is that as a cruise line the vessel schedules are drawn up well in advance and will thereafter be unchanged. The scheduling problem is therefore not subject to the late-notice changes to the vessel schedule (or requirements) that we face.

In terms of crew, there are again some similarities. For example, the crew are hired internationally in both cases, and in both cases the company is subject to transportation costs of getting the crew from their home port to their assignment (referred to by Giachetti et al. (2013) as *movement cost*). There is however an additional complication for our company regarding the crew, relating to their experience and their costs. The Giachetti et al. (2013) problem is defined such that for each job category there is a set of crew who are qualified to work in that role, all of whom have the same set of skills and experience, and all of whom cost the same to employ. This is in contrast to our problem where in certain crew roles there will be different required skill sets and different levels of experience and where, depending on crew nationality and contract type, employees who are otherwise equivalent in terms of ability will cost different amounts to utilise.

In terms of goals of the problem, the need to fill all assignments is key in both cases. However, the means of doing this is different - our company aims to allocate an employee to each role in each time period, while the cruise company anticipate potential no-shows or other absences and *overbook* the number of crew on board in order that a *PAR level* of service can be maintained. Other requirements of the cruise company’s process is that a mix of nationalities and language skills are present on board each vessel. While this is not in itself present in our problem, these requirements could be thought of as analogous with the experience, or minimum aggregate skill level, constraints.

The proposed solution approach by Giachetti et al. (2013) involves a two-stage process. The first stage is a *demand planning module*, which takes past data and uses it to predict no-shows and thereby estimate the overbooking requirement for each vessel. This is in contrast to our problem where vessel requirements are considered to be known (if changeable), and so a demand model is not used in our case. The second stage is the *crew scheduling optimization module*, which is formulated as a goal programming problem, although it is not discernibly different from a mixed integer linear programme. In order to keep the problem to manageable size, Giachetti et al. (2013) argue that different crew categories are independent and therefore can be solved separately, although it is not clear how this can be completely true since the nationality mix constraint appears to refer to *all* crew on board rather than just crew in a given role. The principle however of decomposing the problem by crew role is similar to the current approach by our company, where different *planners* are responsible for different crew groups.

2.6 Summary of Literature

To summarise the literature discussed in this chapter, we have seen that there are a variety of application areas which have a relationship to vessel crew scheduling. These include scheduling problems in other contexts, particularly other transportation settings (such as airline or rail) of crew scheduling problems, and also other maritime scheduling and optimization problems. These have been solved by a number of approaches, including column generation, branch-and-bound, branch-and-price, and numerous heuristic methods; often, more than one of these approaches was combined to form the proposed solution method.

In section 2.5, four types of problem were discussed which exist in the literature which could be classified as ‘Vessel Crew Scheduling’. It was noted that while each of these had similarities with the problem we have studied, each also has a number of differences which set our problem apart. We argue that the crew scheduling problem faced by our case study company is sufficiently distinct that it warrants separate study to understand, formulate and solve in a manner which would be acceptable to the company.

Chapter 3

Research Methods

Before discussing the specific details of the problem (Chapter 4), this chapter briefly discusses the methods which are applied in this research. This begins in section 3.1 with an outline of the general research approaches used, namely mathematical modelling and case study, including the potential drawbacks of these. Then, in section 3.2 we give an overview of some of the statistical methods used to analyse some of our results in the later chapters.

3.1 Research Approaches

The main approaches used in this research are modelling, specifically mathematical modelling, and a case study. The first thing to note about these is that they are two very different, contrasting approaches. In Operational Research, tools and methods are often split into two groups - 'hard' OR and 'soft' OR. If we use these terms, it is easy to identify that modelling, especially mathematical modelling falls into the 'hard' category, while case studies are more appropriate for dealing with the 'soft' side of a research project. In this way, a combined modelling and case study approach can be seen as combining two contrasting and complementary methods. To show this, we will first discuss each method separately, before examining how they work in conjunction.

If we consider modelling, it is quite easy to argue that this is very much suited to the study of the vessel crew scheduling problem, or for that matter any scheduling problem. Indeed, it can be seen in Chapter 2 that virtually every scheduling problem discussed in the literature is solved using some kind of mathematical modelling approach. Thinking of the reasons for this, we can see that in scheduling problems the number of different possible combinations of shift assignments increases exponentially with every extra staff member or shift that is included, and so we must have some means of methodically choosing a good (or preferably, the best) allocation. There is also the matter of not just having to find a feasible allocation of duties, but requiring that some property, usually cost, is minimized.

It follows that if we can express all the relevant factors in the problem in mathematical terms, we can use mathematical modelling to describe the problem and simplify it into a form which can be understood and solved systematically.

In contrast, by using a case study to investigate crew scheduling within a company we can gain a better understanding of the softer issues surrounding the problem. In management science, the technique of problem structuring is often seen as an important stage in any project. Pidd (2003) for example states

> *"soft approaches begin with the assumption that problem definition is not straightforward, but is itself problematic"*

and this assumption is often found to be correct. Consequently, it is important to have this stage in a project, where the client's problem can be discussed, analysed and clarified. Particularly, this is important in the case of Vessel Crew Scheduling when it can be seen that there is very little current literature which is similar in definition to our problem. Only once we know the true requirements of the problem can we proceed to trying to solve it. Therefore it stands to reason that, before a model can be built, it would be required to find out about the crew scheduling problem from a company who currently has experience of this problem. In general, a case study would involve going to this company and finding out about their crew scheduling problem - what their objective for the problem is, and what constraints they face, for example rules and regulations about shift patterns, payment and contract structures within the company, etc. This approach therefore ensures that the problem being considered is one with practical relevance - a 'real-world' rather than purely theoretical problem.

In this way, there is a complementarity between the mathematical modelling approach and the case study approach. This can allow the creation not only of a theoretically useful model, but also a practically useful one. This leads to the conclusion that one of the final goals of this research should be to generate a model which can be applied in practice at the case study company - this is discussed in more detail in section 4.2.3.

3.1.1 Potential Drawbacks

Outlined above are some of the arguments in favour of adopting the mathematical modelling and case study research approaches. However, there are also some problems which are inherent to each which must be acknowledged and, if possible, addressed.

Taking firstly modelling, it is well known that the purpose of a model is to simplify a complex situation so that it can be better understood and analysed. The simplification idea makes modelling very useful, but it should be considered that there is also a considerable risk of problems with this process. With every simplifying assumption which is made during the modelling process, there is a step away from the real-life situation. There is therefore a balance to be attained - too few simplifying assumptions and we will have an overly-complicated model; too many and we risk taking the model too far away from the real-life situation such that it is no longer useful. This risk is particularly real for mathematical or optimization models. As Pidd (2003) observes:

> *"the method guarantees that an optimum solution to a correctly formulated model will be found, if it exists. ... [T]his does not guarantee that the optimum solution to the model will be the best solution in practice. This can only be the case if the model is a perfect fit to the system being modelled, which is unlikely."*

The need to find this balance in this particular case will become more evident in the chapters 5 and 6, where first an easy to solve but over-simplified Vessel Crew Scheduling model is presented, followed by a more complex but more realistic (and practically useful) formulation. As will be seen, the over-simplistic model can still give useful insights into how to tackle the more complex problem. Clearly, the problem of achieving the balance between simplicity and realism can be partially addressed through the case-study part of the process. By talking to a client with actual experience of the problem, we can better understand which factors are important and which are difficult to account for, and based on this can make an informed decision about what factors are most important to be included in the model.

The case study approach also has its potential drawbacks, not least that it firstly requires the buy-in of a client company. To this end, it can be necessary to visit a company (or possibly several companies) and try to 'sell' this project to them, for example by explaining the potential benefits that a new crew scheduling optimization model could yield. There is also a potential problem of the case study approach with respect to the research value of the output from the project. While it is certainly necessary to carry out the research as a case study in order to show the model works in practice, there is the associated implication that the model might *only* be applicable in the case study situation, and not elsewhere. To try to address this problem, we will attempt at the modelling stage (including formulating the problem and developing solution methods) to create a model that, while taking into account the factors which are of importance to the client company, leaves some room for flexibility which could allow the model to be more generalisable.

3.2 Statistical Overview

When evaluating some of the solution methods discussed in chapters 5 and 6, it will be necessary to use some statistical analysis to carry out some comparisons. Here, we will briefly overview the key concepts and methods used. This comprises discussion of what is meant by hypothesis testing, Type I and Type II errors and the power of a test, along with an outline of the tests used - the F-test for Analysis of Variance, the Kruskal-Wallis test and Pearson's correlation coefficient.

3.2.1 Hypothesis Testing

Hypothesis testing is a technique used in statistics to attempt to prove that a theory or belief is correct. To begin with we must have two hypotheses - a *null* hypothesis and an *alternative* hypothesis. The null hypothesis (H_0) is usually the opposite of what we hope to prove - for example, if we are testing to see if there is a difference between two solution methods, the null hypothesis would be that there is no difference. The alternative hypothesis (H_A or H_1) is then the theory or belief which we hope to prove, for example that there is a difference between the two methods. The null hypothesis is assumed to be true unless sufficient evidence can be found to disprove it - it is this evidence we are trying to find by carrying out the hypothesis test.

Evidence is calculated in the form of a *test statistic.* The exact definition of the test statistic varies according to the specific test being used, but it is a single value which is calculated from the data. If we assume that the null hypothesis is true, then the value of a test statistic calculated from a random sample of the data is expected to follow a known distribution - which distribution will again depend on the statistical test being used. From this, we can calculate the probability p of observing a test statistic of this value x or greater, assuming the null hypothesis is true, as

$$P\left(X \geq x\right) = p$$

This probability p is known as the *p-value.* If the p-value is less than a specified probability, known as the significance level α and usually defined to be 5% (or 0.05), this means it is highly unlikely to have obtained a test statistic at least as extreme as this if the null hypothesis is correct. We therefore have sufficient evidence to reject the null hypothesis and accept the alternative hypothesis instead. If the p-value is not less than α, we cannot claim to have disproved the null hypothesis.

An equivalent way of looking at this is by considering a cut-off level X^* for which

$$P\left(X \geq X^*\right) = \alpha$$

In this case, if the test statistic x takes a value $x > X^*$ then we also have sufficient evidence to reject the null hypothesis and accept the alternative hypothesis instead.

There are various different types of statistical tests which can be carried out, depending on the kind of comparison which is being made and what assumptions can be made about the population the data is drawn from. In particular, there are two kinds of tests

- parametric and non-parametric. Parametric tests depend on the assumption that the distribution underlying the data comes from a Normal distribution, and will often require the calculation of a variance to obtain the test statistic. Where this assumptions does not hold, a non-parametric test which does not make assumptions about the distribution of the underlying population can often be used instead. However, where possible it is preferable to use a parametric test - as Pallant (2011) notes:

> *"Despite being less 'fussy', non-parametric statistics do have their disadvantages. They tend to be less sensitive than their more powerful parametric cousins, and may therefore fail to detect differences between groups than actually exist. If you have the 'right' sort of data, it is always better to use a parametric technique if you can."* (p.213)

Note that power of a test is discussed in section 3.2.2 below.

3.2.2 Type I and Type II Errors

The significance level, α, described above describes the probability that we observe a test statistic value at least as extreme as that calculated assuming the null hypothesis is correct. In other words, it is the probability that we incorrectly reject the null hypothesis. This kind of error is known as Type I error.

Conversely, a Type II error occurs when the test fails to reject the null hypothesis when it is in fact false. This will happen with some probability β, and depends on a number of factors including the sample size used, the size of the effect being tested for, and the defined significance level α (Pallant 2011). For a larger sample size or a larger effect which is being identified, the probability of not identifying a difference if one exists is reduced, while as might be expected the values of α and β have an inverse relationship - decreasing the significance level will make it less likely that the null hypothesis will be incorrectly rejected, but will increase the probability of a Type II error.

The complement of the probability of a Type II error, $(1 - \beta)$, or the probability of rejecting the null hypothesis if it is indeed false, is known as the *power* of a test. As described above, parametric tests depend on more assumptions, but for the same sample size and effect size and the same significance level they will achieve a higher power (or lower probability of Type II error) than their non-parametric equivalents.

3.2.3 Tests Used

There are two main kinds of tests used to analyse the results presented in chapters 5 and 6 of this thesis - tests for differences of output measures between different groups of inputs, and tests for correlation. For the tests of differences between groups, both the parametric and non-parametric tests are used, depending on whether or not the data appears to be drawn from a distribution that is approximately Normal.

3.2.3.1 Analysis of Variance

The One-Way Analysis of Variance (ANOVA) is a parametric method, and can be used when we are analysing a single continuous output variable, with cases which can be divided into three or more categories (Pallant 2011). We can then use the ANOVA method to examine whether there is a difference in this output variable between the different categories, with a null hypothesis that there is no difference between the groups.

The test statistic calculated is the F-statistic, which is a ratio of the total 'between-group variance' divided by the 'within-group variance'. In the F-test, the test statistic is compared with the F-distribution with $(K-1),(N-K)$ degrees of freedom, where K is the number of groups and N the total sample size, to obtain a p-value. A high value of the ratio, indicating a large amount of between-group variance compared to that within the groups, will return a small p-value which, if less than α will indicate that there is a significant difference between the groups with respect to the output variable.

3.2.3.2 Kruskal-Wallis Test

The Kruskal-Wallis test is the non-parametric equivalent of an Analysis of Variance (Pallant 2011), and can be used to determine the same information about the effect of different groups on an output in the case where assumptions about an underlying Normal distribution do not hold.

Rather than using the variance, the Kruskal-Wallis test statistic is calculated based on ranking the cases according to the values of the output variable. The calculation is analogous however - while variance is calculated as the sum of the squared deviations of each observation from the mean, here we calculate the sum of the squared deviations of each observations *rank* from the mean *rank*. The test statistic is then the ratio of the total between-group variance of rank divided by the within-group variance of rank. This test statistic is compared to a χ^2 distribution with $K-1$ degrees of freedom, where K is the number of groups, to obtain a p-value. As with the F-test above, a high value of this ratio indicates a large amount of between-group variance compared to that within the groups and will return a small p-value. As before, if this is less than α will indicate that there is a significant difference between the groups with respect to the output variable.

It can be seen that the actual values of the dependant variable are not considered here, except insofar as they determine the rankings. Consequently, information is being discounted in this calculation as compared to the ANOVA calculation discussed above, and so it can be seen that this test may be more likely to fail to identify a difference even if one exists (i.e. commit a Type II error). Therefore, as discussed above, we should always use the more powerful parametric form of a test, in this case the F-test, where possible.

3.2.3.3 Pearson's Correlation

Correlation analysis can be carried out to determine whether there is a relationship between variables. Specifically, here we will examine if there is a relationship between the values of two continuous variables, for which we use Pearson's "product-moment" correlation (Pallant 2011). This returns a value between -1 and 1, with values close to -1 indicating a strong inverse relationship between the variables, and values close to $+1$ indicating a strong positive relationship. Values close to zero suggest there is little or no relationship between the values of the variables. A p-value can be calculated to determine whether the correlation coefficient found is sufficient indication of a significant difference.

The correlation coefficient is calculated as the ratio of the covariance of the two variables (i.e. the product of the deviations from the variable means for each case, averaged across all data points) divided by the sample variance of each of the two variables of interest. In order to test whether or not this correlation is significant, we must transform the correlation coefficient, which can be referred to as r, into a test statistic t. This is calculated as

$$t = r\sqrt{\frac{n-2}{1-r^2}}$$

where n is the sample size, or number of data pairs in the sample. This test statistic is compared with a Student's t-distribution with $n-2$ degrees of freedom to obtain the p-value, and if this is less than α this will indicate we should accept the alternative hypothesis that the two variables are correlated. Note that the test statistic value increases as the square root of the sample size, meaning that for larger samples a larger test statistic will be obtained, and therefore suggesting (as might be expected) that we are more likely to be certain that a correlation is significant when n is larger.

Chapter 4

The Vessel Crew Scheduling Problem

This chapter sets out the formal description of the Vessel Crew Scheduling Problem as is faced by the client company. However, before describing the specific details of the problem we will set the problem in context by describing the wider business process within which it fits.

4.1 Business Process

As indicated in section 2.3.1, the nature of the business is to provide services (such as construction support, remote operated vehicle (ROV) operations, surveying, diving support, and deep-water lifting or installation) to companies in the offshore oil industry. Their operations therefore are dictated by the needs of their client companies, and specified by contracts agreed with clients on a project-by-project basis. Projects may be of various lengths, from a matter of weeks to several years, and would normally be agreed well in advance of the operational dates; however in some cases projects may be requested, or changes to existing projects be negotiated, at relatively short notice.

Within the company, there are several groups which deal with each of the various aspects as the company's operations. These include Project Managers, Vessel Managers, Planners and the Crewing group in terms of onshore staff, while Offshore Managers on board vessels and individual crew members are also involved, as well as the client companies and an external travel agent. The relationships between these within the process are shown in Figure 4.1, which uses the principles of Business Process Modelling and Notation (BPMN; more information can be found in Weske (2007), for example) 2.0 to show the procedure as a business process map.

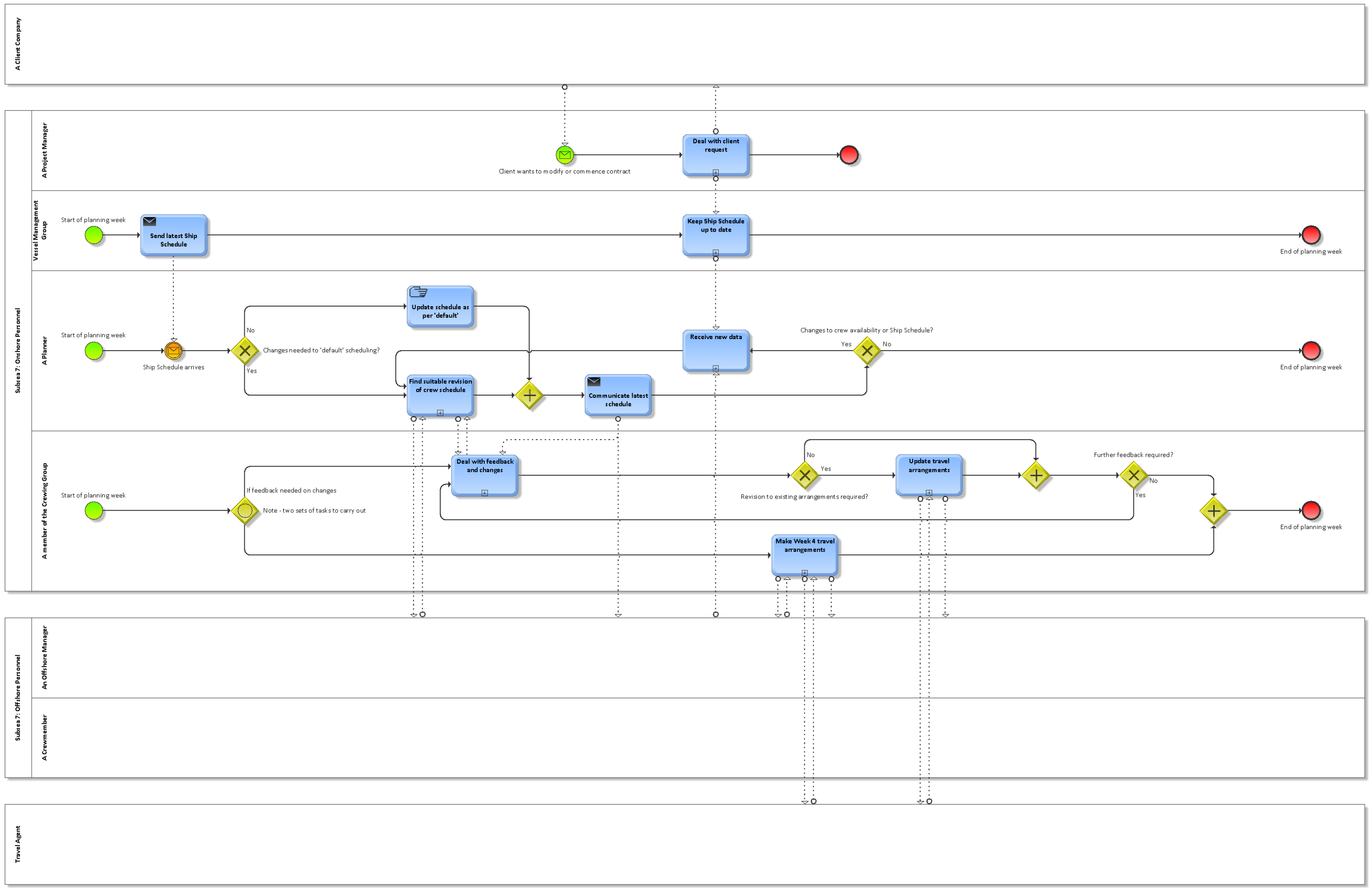

A Client Company
Subsea 7: Onshore Personnel
A Project Manager
Client wants to modify or commence contract
Deal with client request
Vessel Management Group
Start of planning week
Send latest Ship Schedule
Keep Ship Schedule up to date
End of planning week
A Planner
Start of planning week
Ship Schedule arrives
Changes needed to 'default' scheduling?
No
Yes
Update schedule as per 'default'
Find suitable revision of crew schedule
Communicate latest schedule
Receive new data
Changes to crew availability or Ship Schedule?
Yes
No
End of planning week
A member of the Crewing Group
Start of planning week
If feedback needed on changes
Note - two sets of tasks to carry out
Deal with feedback and changes
Revision to existing arrangements required?
No
Yes
Update travel arrangements
Make Week 4 travel arrangements
Further feedback required?
No
Yes
End of planning week
Subsea 7: Offshore Personnel
An Offshore Manager
A Crewmember
Travel Agent

Figure 4.1: Business Process Map showing crew scheduling process in context

This diagram shows the process at a high level - a more detailed diagram, with all sub-processes expanded to show their constituent tasks, can be seen in Figures A.1, A.2 and A.3 in Appendix A.

As the diagram shows, the Project Managers are responsible for dealing directly with the client companies, and determining what is required in order for the company to deliver the project. This interaction is shown in more detail in Figure 4.2 below, including an expansion of the sub-process of dealing with the client request.

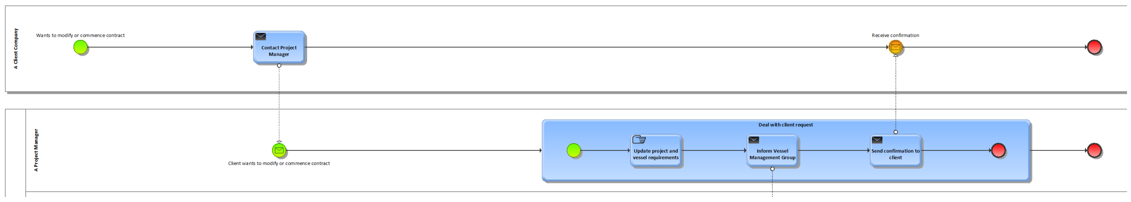

A Client Company
Wants to modify or commence contract
Contact Project Manager
Receive confirmation
A Project Manager
Client wants to modify or commence contract
Deal with client request
Update project and vessel requirements
Inform Vessel Management Group
Send confirmation to client

Figure 4.2: Zoom in on detail of interaction between a client company and a Project Manager

The Project Managers pass the new details to the Vessel Managers, who are responsible for the assignment of the vessels to specific projects and also determining when vessels should be brought into dock for maintenance. Figure 4.3 below shows the process followed by the Vessel Managers to produce a weekly 'Ship Schedule' and to keep this up to date.

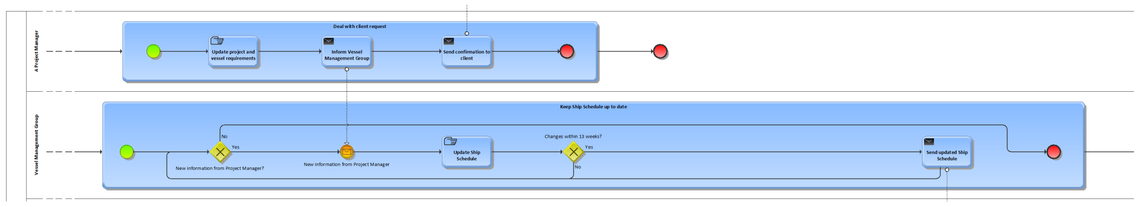

A Project Manager
Deal with client request
Update project and vessel requirements
Inform Vessel Management Group
Send confirmation to client
Vessel Management Group
Keep Ship Schedule up to date
No
Yes
New information from Project Manager?
New information from Project Manager
Update Ship Schedule
Changes within 13 weeks?
Yes
No
Send updated Ship Schedule

Figure 4.3: Zoom in on detail of interaction between Project Manager and Vessel Managers

This Ship Schedule is passed at the start of each planning week to the Planners, whose responsibility it is to allocate crew to the vessels according to the vessels' requirements and to keep the crew schedule up to date for at least the coming 13 weeks (i.e. 3 months). If any changes to the Ship Schedule fall within this 13 week planning period then the Vessel Managers should send an additional update to the Planners so that the necessary crew schedule changes can be made as soon as possible.

As described in more detail below (section 4.2), the Planners may be able to update the crew schedules according to a 'default' plan which will see employees allocated their regular assignments. However, in some cases it may be necessary to deviate from this default plan, and when this happens the Planners must confirm the alterations with the crew concerned, the Offshore Managers, and in cases where transport arrangements are affected the Crewing group. This part of the process is shown in greater detail in Figure 4.4 below. In addition to updating the schedule at the start of the planning week, it is also possible that new information relating to vessel requirements (from the Vessel Managers) or crew availability during the planning period will arrive during the week and this will necessitate further changes to the schedule. This part of the process is shown in greater detail in Figure 4.5.

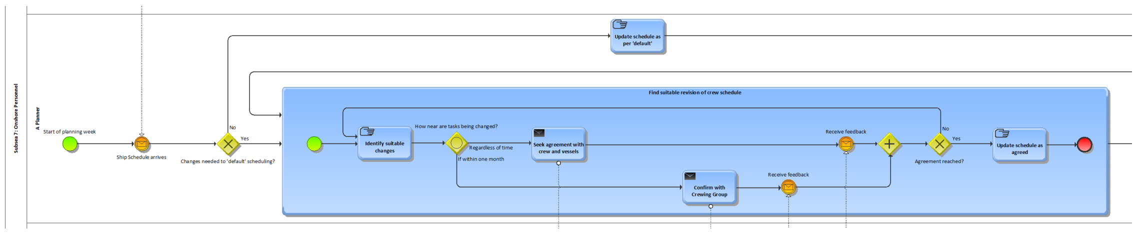

Subsea 7: Onshore Personnel
A Planner
Start of planning week
Ship Schedule arrives
No
Yes
Changes needed to 'default' scheduling?
Update schedule as per 'default'
Find suitable revision of crew schedule
Identify suitable changes
How near are tasks being changed?
Regardless of time
If within one month
Seek agreement with crew and vessels
Confirm with Crewing Group
Receive feedback
Receive feedback
No
Yes
Agreement reached?
Update schedule as agreed

Figure 4.4: Zoom in on detail of process of updating or revising crew schedule

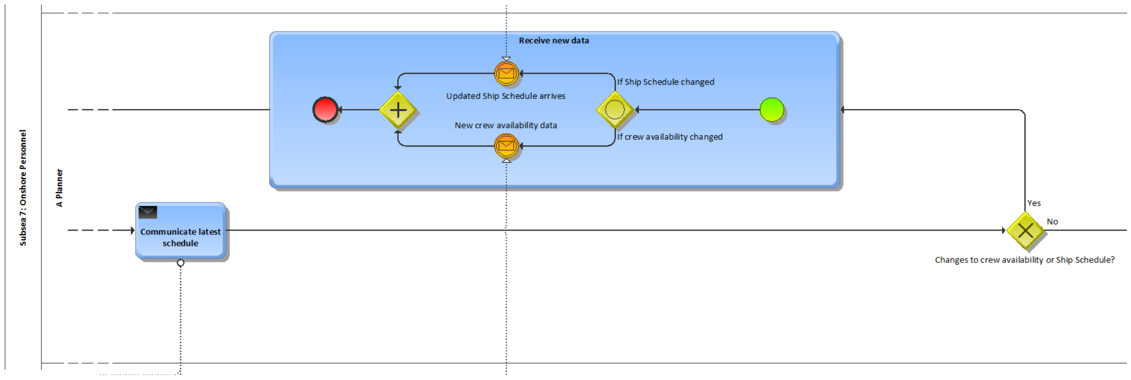

Subsea 7: Onshore Personnel
A Planner
Receive new data
Updated Ship Schedule arrives
New crew availability data
If Ship Schedule changed
If crew availability changed
Communicate latest schedule
Yes
No
Changes to crew availability or Ship Schedule?

Figure 4.5: Zoom in on detail of action taken when Planner receives new information

The Crewing group, meanwhile, are responsible for making travel arrangements for the crew. Each employee has a designated ‘home port’, to which it is their responsibility to travel; however, it is the company’s responsibility to arrange their transfer to their assigned vessel, or in some cases to another port which is the closest to the vessel. Bookings will be made four weeks ahead of the crew change date, although Crewing may also be affected by new information about employee availability or vessel requirements. If changes are needed within four weeks, Crewing must be consulted to take into account new travel arrangements - this interaction is show in more detail in Figure 4.6. Subsequently the Travel Agent must again be contacted to change the bookings once the schedule changes are confirmed - this sub-process is shown in expanded form in Figure 4.7.

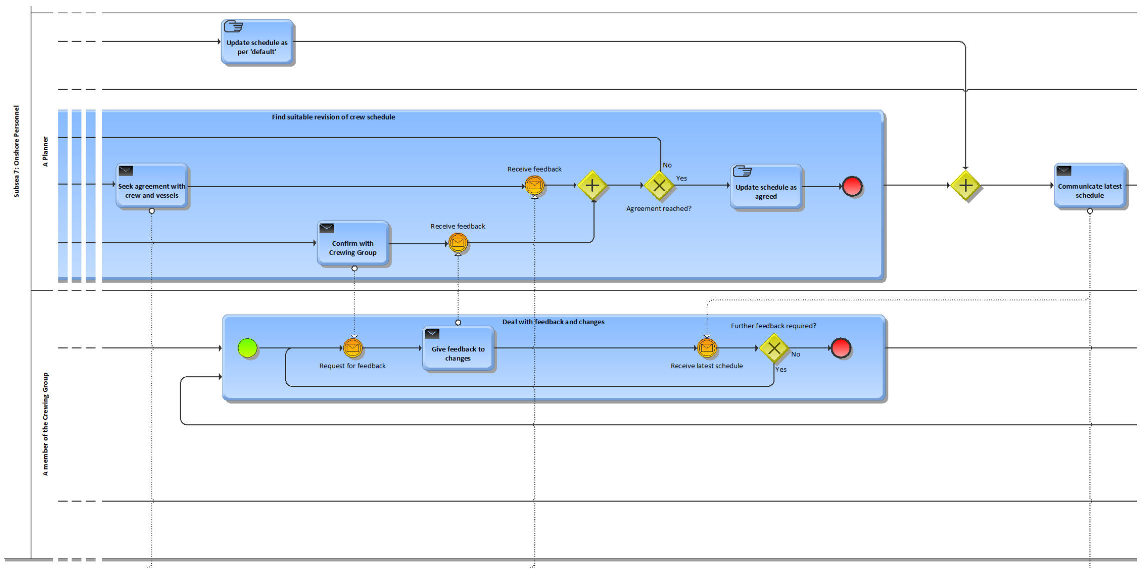
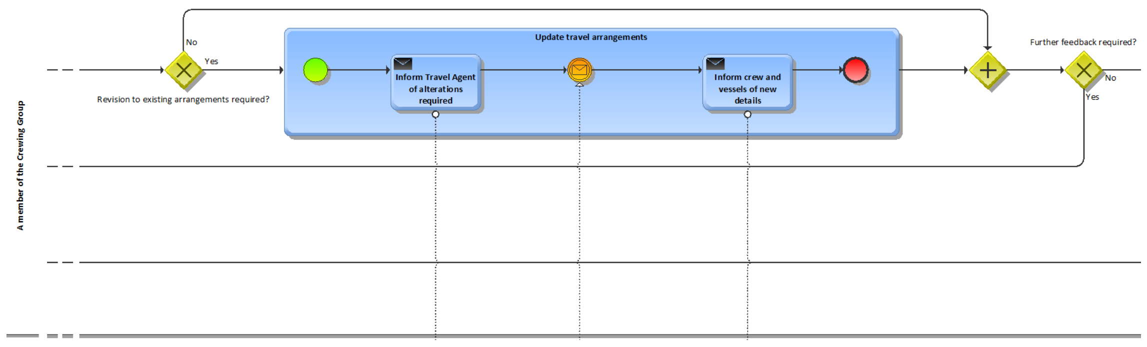

Subsea 7: Onshore Personnel
A Planner
A member of the Crewing Group
Update schedule as per 'default'
Find suitable revision of crew schedule
Seek agreement with crew and vessels
Receive feedback
No
Yes
Agreement reached?
Update schedule as agreed
Confirm with Crewing Group
Receive feedback
Communicate latest schedule
Deal with feedback and changes
Request for feedback
Give feedback to changes
Further feedback required?
Receive latest schedule
No
Yes

Figure 4.6: Zoom in on detail of interaction between Planners and Crewing when changes are required

In addition to dealing with changes to existing travel arrangements, over the course of a week the Crewing group must book the new travel arrangements which now fall within their four-week planning period. This sub-process is shown in expanded detail in Figure 4.8. The schedules produced by the Planners specify the week-by-week assignments; however, the specific day and time of the crew change is flexible and dependent on the itinerary of the vessel that week. The Crewing employees must therefore contact the Offshore Managers to arrange the specific crew change dates, before dealing with the external Travel Agent in order to book the required transportation.

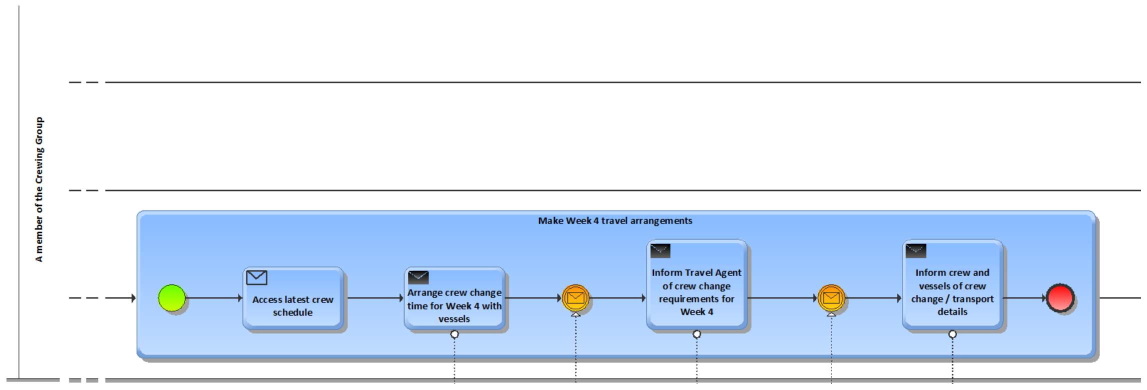
In terms of the offshore employees, the Offshore Manager is responsible for the administrative tasks on board their respective vessels. As described above, with regard to the scheduling process this involves consultation with Planners about changing the regular schedules and confirming crew change dates with the Crewing personnel. In addition, when transport arrangements are made or altered, the Offshore Managers will be informed of the new arrangements. Details of this are not shown here in diagram form, but are contained in the full process maps given in Appendix A.

The crew members meanwhile have several roles in the scheduling process (again not detailed here - please see Appendix A). Perhaps the most important role, and the one which must be instigated by the employee rather than the onshore staff, is to inform the Planners if their availability changes (for example through illness, or if they wish to register a vacation period). Another key role is to provide the planners with feedback when they



A member of the Crewing Group
No
Yes
Revision to existing arrangements required?
Update travel arrangements
Inform Travel Agent of alterations required
Inform crew and vessels of new details
Further feedback required?
No
Yes

Figure 4.7: Zoom in on detail of sub-process of updating travel arrangements



A member of the Crewing Group
Make Week 4 travel arrangements
Access latest crew schedule
Arrange crew change time for Week 4 with vessels
Inform Travel Agent of crew change requirements for Week 4
Inform crew and vessels of crew change / transport details

Figure 4.8: Zoom in on detail of sub-process of making new travel arrangements

are asked to deviate from the default schedule. An employee's willingness to change their schedule may depend on the kind of change required, how near in the planning horizon the affected week falls, and the general inclination of the employee to agree to alterations. The remaining elements of the crew members' involvement in the scheduling process is simply that of receiving new information, either relating to updates to their schedule of assignments, or relating to newly made or updated transport arrangements.

The final actor to consider in the scheduling process is the external travel agent, whose role is simply to react to information sent to them by the Crewing group. Mainly, this comprises details of the employees who require transport to be booked, along with the required arrival dates and times in order for those employees to board their vessels; however, it may also involve requests to change existing bookings, including amending arrival times, changing the employee who is booked on a given route, or perhaps cancelling a booking completely. The travel agent will make the necessary arrangements, and communicate the details (in particular, the required rendezvous points for offshore crew) back to the Crewing team. As with the offshore employees, a detailed diagram is not given here, but is contained in the full process maps in Appendix A.

4.2 Problem Description

Having discussed the broader context of the scheduling process in section 4.1 above, we can now look at the specific description of the crew scheduling problem. We now focus in on the tasks in the process carried out by the Planners who, although having input from various other actors in the broader process, have the final responsibility for the crew schedules.

As described above, a key part of the Planners' responsibilities is the sub-process of finding suitable revisions to the crew schedule, as shown in Figure 4.4. It should be noted that this process is iterative, and in some cases it may take several attempts to find a revision that is acceptable to all parties. It can also be seen from the main process map (Figure 4.1) that the sub-process as a whole may have to be carried out several times during the planning week, depending on how many changes arise during that time. Because some of these changes may relate to tasks to be carried out very soon, e.g. the next week, it may also be the case that the determining of a suitable new schedule must be done very quickly. It is for these reasons that this sub-process is a key part of the crew scheduling process, and it is desirable that it should operate as efficiently and effectively as possible.

This section goes on to discuss in detail the main concerns for the Planners during this sub-process and how it is currently approached. This allows us to identify potential for improvements to the process.

4.2.1 Detail of the Problem

Clearly, the fundamental aim of any crew scheduling problem is to ensure that all of the required tasks are assigned to one of the available employees. The 'tasks' in the case of this problem are defined according to the crew roles which must be filled on board each of the company's vessels. The different crew roles are numerous, but can be divided into two broad categories. The first, termed *marine crew*, includes roles such as ship's captain, other bridge crew, and engineering crew. These roles must be covered at all times regardless of the nature of a vessel's current assignment, and must even be covered when a vessel is unassigned or undergoing maintenance. The requirements for the second category, termed *project crew*, will vary depending on a vessel's assignment, e.g. diving crew will only be required when a vessel is engaged in a diving project. This category also includes crew which work on deck (riggers, deck foremen, etc.) who will be required for most projects but will *not* be required when a vessel is unassigned or undergoing maintenance.

The set of available crew is made up of employees of many nationalities, with the result that legal and contractual restrictions on crew can vary from employee to employee. In general, there are two kinds of crew, *regular* crew and *agency* crew, with regular crew greatly outnumbering the agency crew employed in the majority of crew roles. Regular crew are permanent employees of the company, and while contractual conditions will vary these can again be split into two broad categories:

- Some of the regular crew are employed on *fixed contracts*, meaning they are paid a salary and expected to work a certain number of days at sea per year. If they work fewer than these *guaranteed* days, they will still be paid for them (and so the company is wasting money), while if they work more than their guaranteed days over the year they will be entitled to additional pay, usually pro-rata according to their salary and with a possible overtime multiplier.

- The crew not on fixed contracts are referred to as *day rate* crew, since they are paid per day that they work at sea. Their rate of pay will usually be higher that the fixed contract crew, up to around 1.5 times the amount.

Agency crew on the other hand are not permanent employees of the company, but instead are found by external agencies to work on a temporary contract. Because they are available at short notice, agency crew are generally more expensive (up to twice the pay rate) per day than the regular crew, although this is in part balanced out by the agency crew often being local which leads to a slight saving on travel costs (as discussed below). In general, the company will make use of regular employees where possible, but agency crew will be utilised in a situation where there no regular crew are available, or are only available at a very high cost.

Crew availability to carry out a task depends on a number of factors. Not least of these are other commitments to which the employee might be subject, for example pre-arranged holiday periods or training courses which they must attend. In addition, it is possible that an employee might be unavailable due to illness or similar unforeseen circumstances. Also affecting an employee's ability to perform certain tasks is their training and experience. This relates primarily to the project-specific crew - for example, if the vessel is on a pipe-laying assignment, it will be desirable that at least one of the two deck foremen will have a suitable level of previous experience of pipe-laying projects. However, it can also apply to other crew - captains, for example, might be required to have previous experience of commanding a vessel in a certain region or in certain conditions.

Availability of an employee for a certain task might also depend on their nationality. For example, ships registered under certain flags are only permitted to employ crew from certain countries, while some crew may require visas or work permits (which take time to arrange, and at a cost) in order to work on vessels operating in certain regions. In addition, it may not be considered desirable for crew of certain nationalities to work together if it can be avoided.

Another set of restrictions on crew availability are the legal and contractual constraints placed on working and resting periods. Each employee will have a maximum number of weeks they can be assigned consecutively to work, and a minimum number of weeks they must have free in between offshore assignments. We note here that a working 'week' for the crew is a full seven day week. The exact restrictions on individual employees will vary depending on their nationality and the specific details of their contract, for example European Union nationals will have their contracts governed by legislation such as the European Union Working Time Directive (see European Union 2003)). In general, these requirements form the basis for a framework of regular working patterns which are used at the company.

Staying with the example of crew from EU nations, the majority of these employees will generally be assigned four weeks on board ship followed by a four week rest period (termed *four weeks on, four weeks off* or *4-on-4-off*), or sometimes *5-on-5-off* in the case of a vessel operating in a more remote region. On the other hand, crew of some nationalities are restricted to *2-on-4-off* working patterns, while others may regularly be assigned a *10-on-5-off* pattern. It should be noted that crew changes take place each week on each vessel, meaning that change-overs are staggered both with respect to the crew on board each vessel, and all the crew of a given grade. In other words, while there are no restrictions *per se* on the pattern of change overs, in reality only a subset of the crew on a vessel will change each week, and only a subset of each crew of a given grade across the entire fleet will change each week.

Because of the length of the employees' duty periods, and the need to arrange transportation for the crew in advance, the company has a much longer planning horizon than would be expected in other transportation settings. The company aims to plan at least 13 weeks ahead, and will arrange the logistics for the crew changes up to four weeks before the date. Note that the exact day of the crew change is only confirmed when the logistics are arranged - during the planning process before this, it is assumed that all crew changes happen on the same day every week.

The length of the duty periods and corresponding (comparatively) long planning horizon lead to an element of uncertainty in the planning process. This is in part as a result of changes in crew availability, with for example an employee falling ill at relatively short notice, and partly because of the flexible nature of the company's contracts with their clients which can lead to changes in the vessels' assignments, and therefore alterations to their requirements for crew over time. It is worth noting here that factors such as weather or traffic congestion which can cause significant disruption in air, bus or train crew scheduling are actually less of a concern here. Barring highly exceptional large-scale travel disruption, the delay will be in the order of hours for an employee to reach their vessel. While potentially inconvenient in the short term, this is a small proportion of the duty period lengths and therefore will not cause knock-on effects to the rest of the schedule.

As a result, the scheduling process is not simply carried out for a block of thirteen weeks at a time; instead, the current schedule for the coming thirteen weeks is reviewed on a weekly basis, with any necessary changes being made along with new tasks at the far end of the planning horizon being assigned. This job, carried out by the Planners is simplified by the existence of regular assignments, whereby when all else is equal crew will generally have a vessel to which they are always assigned, and will share their role with another employee working four-on-four-off (or five-on-five-off) in turn - this is termed as working *back-to-back*.

The problem becomes more difficult when, as described earlier, deviations from these regular back-to-backs are required. Because of the pressures involved with this stage of the process (particularly finding feasible solutions, seeking agreement between all parties and the possibility of requiring several iterations to achieve this), it is here that the Planners, and indeed the company as a whole, would most benefit from a tool to support the decision making. Firstly, we discuss below how the Planners currently approach this problem.

4.2.2 Current Approach to the Problem

Currently there is no optimization tool used to aid the decision making process. A visualisation software is used to show the current assignments of each individual crew member, and using an alternative view the crew members (if any) currently assigned to each role for each week that role is required. The Planners can also use this software to visualise the effects of changing the current assignments, whether by changing the duration for which an employee works in a certain role, or by removing and allocating employees from and to entire blocks (i.e. a set of consecutive weeks) of work. They will also need to make decisions about when it is necessary to cover a role which an agency employee rather than one of their regular crew.

In some instances, the Planners will be under time pressure when trying to identify solutions. However even when time is not a particular issue, it can be difficult for the Planners to find solutions, given the contractual restrictions placed on assignments and the potential issues surrounding unavailability or unwillingness of some employees to change their assignments. Because of these difficulties, and the lack of an optimization tool to aid decision making, the Planners' primary concern in the current process is to obtain feasible solutions, with even this objective often proving problematic.

4.2.3 Proposed Intervention

The Planners' task could be greatly aided by the addition of an optimization tool, designed to quickly and efficiently find feasible schedules for the current situation. Because of the need for iteration and dialogue, such a tool would ideally allow the problem to be solved multiple times, with small adjustments made to the constraints at each step to account as necessary for any new information from communication with the employees.

Clearly, using an optimization tool an idea of quality of solution can also be brought into consideration. This can best be judged on cost, which is of course of importance to the company even if the planners cannot always take it into account. Costs of an employee carrying out a task comprise their wages for that period in the case of day-rate and agency crew, and the cost of transporting the employee from their home port to the vessel to which they are assigned. Costs of fixed contract crew wages must be calculated differently - we must take into account their working weeks over the year, and consider either the extra payment due when the guaranteed days are exceeded, or the money wasted when they have not been fully utilised.

We can therefore say that the goal of our optimization tool will be to propose feasible, low-cost solutions for the crew scheduling problem. In order to do this, the problem had to first be formulated based on the information given by the company, before investigation could be carried out into how best to solve the problem in an efficient manner. The subsequent chapters go on to discuss the different proposed formulations, and the details and results of the investigations into these.

Chapter 5

A Task-Based Formulation

The first formulation constructed for the Vessel Crew Scheduling Problem made use of some assumptions designed to simplify the problem and make it easier to model. This *Task-Based* formulation assumed that the regular assignment pattern, as described in section 4.2.1, is pre-determined and fixed for each vessel. Specifically, we assume that the pattern follows that described for the majority of EU nationals, namely *4-on-4-off*, and *5-on-5-off* for vessels operating in a more distant region such as the South Atlantic. As a consequence, we must also assume that all employees' contracts require a minimum rest period length of at most four weeks and permit a maximum working length of at least five weeks.

Incorporating these assumptions leads to the formulations set out in this chapter. Firstly, a formulation for a 'basic' scheduling problem is presented, which assumes that there is no partial schedule at the start of the planning process. This assumption can then be removed, allowing the Basic model to be modified into a 'Recovery-type' problem which more accurately reflects the process in place at the company. We will also present an adapted formulation which seeks to minimize the number of changes in the solution rather than a cost function, before discussing the experience of computational work with these models.

5.1 Basic Problem

As indicated at the beginning of this chapter, we firstly outline a version of the scheduling problem which assumes no prior schedule (or partial schedule) exists at the start of the planning process, meaning that all schedules are constructed from scratch. This assumption allows us to set out the basics of our model in a simpler manner, before later extending to take into account the recovery problem (see section 5.2). This simplified model may also be useful in that it could be applicable in another vessel crew scheduling setting even if not for the setting considered in this case.

5.1.1 Definitions

Before giving the formulation, we will define all the notation used in the model.

Sets

We begin by defining the set of employees and the set of tasks to which they must be allocated within the given time period:

E_R is the set of all regular employees, with $|E_R| = m$.

E $= E_R \cup \{m+1\}$ is the set of all employees, where $m+1$ is the index used to denote agency employees.

G $\subseteq E_R$ is the set of fixed contract employees.

J is the set of tasks which are to be carried out, with $|J| = n_W$.

N is a set of dummy *rest* tasks, with $|N| = n_R$. These will be used to ensure employees' minimum rest periods are respected.

$J \cup N$ is the combined set of working and rest tasks, with $|J \cup N| = n_W + n_R = n$.

Main decision variables

We can now define the main decision variables as follows:

$$x_{ij} = \begin{cases} 1 & \text{if employee } i \text{ is allocated to task } j \in J \cup N \\ 0 & \text{otherwise} \end{cases}$$

As discussed previously in section 4.2.1, an employee may not be qualified to perform a given role, or may be unavailable in a given week (meaning they are ineligible for all tasks which overlap that week). In order to reduce the problem size, a simple pre-processing stage can be introduced which will define the decision variables dynamically. In the case of working tasks, this will mean that x_{ij} will only exist where employee i is eligible to carry out task $j \in J$. Similarly, given that each employee will be entitled to a minimum rest period duration, we will say that x_{ij} is not defined for any rest task $j \in N$ which is shorter in duration than employee i's minimum rest period entitlement.

Data

There are numerous pieces of data which are required by the model. These are as follows:

s_j is the start time of task j.

d_j is the duration of task j.

$C_\gamma \subseteq J \cup N$ are sets of tasks which overlap in time, with $\gamma \in \Gamma = \{1, 2, \ldots, \gamma_{max}\}$. The value of γ_{max} and the contents of the sets C_γ can be determined using an algorithm given in section 5.1.2 below.

K is the set of projects which require specialist knowledge, training or experience.

$P_k \subseteq J$ is the set of tasks which comprise project $k \in K$.

e_{ij} is the experience score of employee i with respect to task $j \in \bigcup_{k \in K} P_k$.

ϵ_k is the minimum total experience required across the tasks in project $k \in K$.

$b \in B$ is the chronologically ordered index of all tasks in $J \cup N$, such that $s_{b-1} \leq s_b$ for all $b \geq 2$.

c_{ij} is the cost to the company of assigning employee i to task j. Note that as well as including financial costs (arranging visas, transportation, wages), the user may also wish to include a non-financial *penalty* cost to account for, for example, when an employee is asked to carry out a less desirable assignment. The exact value of this penalty could be varied depending on the level of undesirability of the assignment.

ω_i is the maximum permitted number of consecutive working days for employee i.

ρ_i is the minimum duration of a rest period that employee i must be allowed between working periods.

w_b is the *work resource* value of task b, such that

$$w_b = \begin{cases} d_b & \text{for } b \in J \\ -M & \text{for } b \in N \end{cases}$$

where M is a large number which must satisfy $M \geq \max\limits_{\forall i \in E} \omega_i$.

r_b is the *rest resource* value of task b, such that

$$r_b = \begin{cases} 1 & \text{for } b \in J \\ -1 & \text{for } b \in N \end{cases}$$

W_{i0} is the number of consecutive working periods to which employee i has been assigned immediately prior to the start of the planning period.

R_{i0} is an indication of whether employee i requires rest at the start of the planning period, such that

$$R_{i0} = \begin{cases} 1 & \text{if employee } i \text{ requires rest} \\ 0 & \text{otherwise} \end{cases}$$

g_i is the number of days which are guaranteed to employee $i \in G$.

$\bar{W}_i$ is the number of days employee $i \in G$ is expected to work during the year *outwith* the current planning period.

μ_i is the effective rate of pay per day for employee $i \in G$, which can be considered an *under-time* rate in the event that the employee works less than g_i in the year.

ϕ_i is the daily rate at which overtime is paid to employee $i \in G$, in the event that the employee works more than g_i in the year.

Supplementary decision variables

Finally, there are several supplementary decision variables which are required to ensure feasibility and to correctly calculate the cost of the schedule:

W_{ib} is the accumulated *work resource* value for employee i once all tasks up to and including the task indexed b have been considered.

R_{ib} is the corresponding accumulated *rest resource* value.

u_i is the number of days short of their guaranteed amount that employee $i \in G$ is expected to work during the year.

o_i is the number of days over their guaranteed amount that employee $i \in G$ is expected to work during the year.

5.1.2 Calculating Overlapping Task sets

As described in section 5.1.1 above, this formulation makes use of sets C_γ to define tasks which overlap in time. This is in order to allow us to create constraints which will forbid any employee from being assigned two tasks which take place at the same time. One possible way to do this would be to create a constraint for every pair of tasks (for every employee), which could be constructed as follows:

$$M(x_{if} + x_{ig} - 2) \leq s_g - (s_f + d_f) \quad \forall i, \quad \forall f, g \in J \quad s.t. \quad s_g \geq s_f \tag{5.1}$$

where M is some large number such that $M \geq \left(\max_{\forall j \in J \cup N}(s_j + d_j)\right) - \left(\min_{\forall j \in J \cup N} s_j\right)$. This however would create $\frac{1}{2}n(n+1)$ constraints for every employee $i \in E_R$, i.e. a number of constraints of order $O\left(n^2\right)$. Since in reality a number of tasks will overlap for a given time period, the number of constraints required can be greatly reduced by considering multiple tasks in each constraint.

In order to do this, we first define an ordered list of time points $t_p, p = 0, 1, 2, \ldots, p_{max}$ such that each task start time s_j and task end time $s_j + d_j$ equate to a t_p for all $j \in J \cup N$, and $t_p > t_{p-1}\ \forall p$. Note that consequently, $t_0 = \min j \in J \cup N s_j$, i.e. the earliest start time of all tasks in $J \cup N$; and that $t_{p_{max}} = \max j \in J \cup N\,(s_j + d_j)$, i.e. the latest end time of all tasks. By examining the tasks which start and end at each of these time points, the sets C_γ can be constructed and the value of γ_{max}, where $0 \leq \gamma_{max} \leq \frac{1}{2}n(n+1)$, can be found. This is described in Algorithm 5.1.

Algorithm 5.1 Algorithm defining overlapping task sets C_γ

```
calculate all time points $t_p$ for the set of tasks $J \cup N$
set counters $\alpha = 0$ and $\beta = 0$
set property constraint required to be false, and let the calculation set $C^c = \emptyset$
while $\alpha \leq p_{max}$ do
  if $t_\alpha = s_j + d_j$ for some $j \in J \cup N$ then
    if constraint required is true then
      set $\beta = \beta + 1$
      set $C_\beta = C^c$
      reset constraint required to false
    end if
    for all $j \in J \cup N$ such that $t_\alpha = s_j + d_j$, remove $j$ from set $C^c$
  end if
  if $t_\alpha = s_j$ for some $j \in J \cup N$ then
    set constraint required to be true
    for all $j \in J \cup N$ such that $t_\alpha = s_j$, add $j$ to the set $C^c$
  end if
  if $\alpha = p_{max}$ then
    set $\gamma_{max} = \beta$
    set $\Gamma = \{1, 2, \ldots, \gamma_{max}\}$
  end if
  set $\alpha = \alpha + 1$
end while
```

5.1.3 Formulation

Having defined all the quantities required, we can now state this basic *task-based* formulation of the vessel crew scheduling problem as follows:

$$\min \sum_{\forall i,j} c_{ij}x_{ij} + \sum_{i \in G} (\mu_i u_i + \phi_i o_i) \tag{5.2}$$

subject to:

$$\sum_{i=1}^{m+1} x_{ij} = 1 \qquad \forall j \in J \tag{5.3}$$

$$\sum_{j \in C_\gamma} x_{ij} \leq 1 \qquad \forall i \in E_R, \gamma \in \Gamma \tag{5.4}$$

$$\sum_{i=1}^{m+1} \sum_{\forall j \in P_k} e_{ij}x_{ij} \geq \epsilon_k \qquad \forall k \in K \tag{5.5}$$

$$W_{i,b-1} + x_{ib}w_b \leq W_{ib} \qquad \forall b \in B, i \in E_R \tag{5.6}$$

$$0 \leq W_{ib} \leq \omega_i \qquad \forall b \in B, i \in E_R \tag{5.7}$$

$$R_{i,b-1} + x_{ib}r_b \leq R_{ib} \qquad \forall b \in B, i \in E_R \tag{5.8}$$

$$0 \leq R_{ib} \leq 1 \qquad \forall b \in B, i \in E_R \tag{5.9}$$

$$u_i \geq g_i - \left(\bar{W}_i + \sum_{j \in J} d_j x_{ij} \right) \qquad \forall i \in G \tag{5.10}$$

$$o_i \geq \left(\bar{W}_i + \sum_{j \in J} d_j x_{ij} \right) - g_i \qquad \forall i \in G \tag{5.11}$$

$$x_{ij} \in \{0,1\} \quad \forall i,j \text{ s.t. } x_{ij} \text{ is defined} \tag{5.12}$$

$$u_i, o_i \geq 0 \qquad \forall i \in G \tag{5.13}$$

The objective here, given by equation (5.2), is to minimise the total cost of the schedule. This takes into account both the direct cost of assigning employees to each task, and also the costs incurred relating to the guaranteed days specified for the fixed contract crew. Constraint set (5.3) ensures that each task is covered, while set (5.4) ensures that no employee can be assigned more than a single task from each overlapping task set C_γ, as described in section 5.1.2 above. Constraint set (5.5) meanwhile requires that the experience constraints for certain specified projects are met. The maximum work period lengths are covered by inequalities (5.6) and (5.7), with (5.8) and (5.9) similarly covering the minimum rest period duration between tasks for each employee. Finally, (5.10) and (5.11) allow for the calculation of the amount of under- or over-time that each fixed contract employee will be expected to work during the year.

5.2 Recovery-type Problem

The formulation presented in the section 5.1 above could be useful in a situation where planning is carried out in advance and the schedule can simply be implemented when the time comes, with no need for revisions. However, as described, the nature of our particular problem is that changes must frequently be made to the existing schedule in light of new information about crew availabilities or vessel requirements. As a result, an approach more akin to the recovery problems seen in other scheduling settings is more appropriate. The formulation described above can be modified to take this into account.

5.2.1 Additional Definitions

In order to modify the problem, some additional definitions must be made. This involves the existing schedule which is used as input data, new decision variables which are required, and modifications to the cost coefficients and objective function.

The existing schedule

In section 5.1 we assumed that schedules were being constructed from scratch; however, we know that in practice at the start of a given planning week we have a schedule which was determined during the previous planning week. This schedule can be considered as a binary $(m+1) \times n_W$ matrix $\mathbf{X}^*$ of assignments, which will become input data for the new planning step. The elements of the matrix $\mathbf{X}^*$ will take the following values:

$$x^*_{ij} = \begin{cases} 1 & \text{if employee } i \text{ is allocated to task } j \\ 0 & \text{otherwise} \end{cases}$$

for all employees $i \in E$ and working tasks $j \in J$. It should be noted that $\mathbf{X}^*$ will be a sparse matrix.

Decision variables

As described in Chapter 4, new information may be available to the Planners at the start of the new planning week which means some assignments in $\mathbf{X}^*$ are no longer feasible. In addition, any tasks which start in the final week of the planning horizon will not have been considered in the previous planning week, and so these will be currently unassigned. The problem is now therefore to find suitable modifications to $\mathbf{X}^*$ such that all tasks are once again covered, and all constraints are once again complied with. Our decision variables for the Recovery-type problem can therefore be defined as follows:

$$y_{ij} = \begin{cases} 1 & \text{if there is a change to employee } i\text{'s schedule with respect to task } j \\ 0 & \text{otherwise} \end{cases}$$

These new y_{ij} decision variables can be combined with the x^*_{ij} terms (i.e. individual elements of $\mathbf{X}^*$) to describe the resulting new schedule. For notational simplicity, we can rewrite these combined terms by defining a new quantity z_{ij} which takes the following values:

$$z_{ij} = \begin{cases} x^*_{ij} - y_{ij} & \text{if } x^*_{ij} = 1 \\ x^*_{ij} + y_{ij} & \text{if } x^*_{ij} = 0 \end{cases}$$

Note that alternatively z_{ij} could be represented as

$$z_{ij} = x^*_{ij} + (1 - 2x^*_{ij})y_{ij}$$

Cost coefficients

As the focus of our problem is now on making changes to an existing schedule, the cost of assigning an employee to a given task is now less important. Instead, we must be concerned with the cost of changing an employee's assignment to a task, which will take into account several elements, such as the amounts incurred or saved with respect to wage and transportation costs.

We should note however that these savings will not be an exact reflection of the initial costs, particularly with respect to transportation. As discussed earlier, four weeks in advance of implementation, the transportation arrangements will be finalised and, while efforts are made to make tickets transferable, this means that some money will not be recovered if an employee is removed from a flight. In addition to these tangible costs, it may be advisable to introduce a penalty element to these cost values to account for the preference of the company to avoid unnecessary changes. This will particularly be the case with respect to short-notice changes (i.e. within four weeks of the current date).

It is these factors that necessitate our new definition of solution quality, as clearly the best solution in terms of pure costs will not necessarily be the cheapest to move to from the starting solution $\mathbf{X}^*$. We therefore define our new cost coefficients, as well as an equivalent cost-of-change piece of data for fixed-contract crew, as follows:

c'_{ij} is the cost (or saving, if negative) of changing the assignment of employee i with respect to task j. Note that, as with the assignment cost c_{ij} for the basic problem, this cost may include a penalty cost over and above the financial costs of making the change. This penalty could, amongst other things, account for the disruption caused by asking an employee to change their assignment, especially at short notice.

Ω_i is the expected additional overtime or undertime costs for employee $i \in G$ according to the previous version of the schedule.

We note that generally there will be a positive value (i.e. cost) to c'_{ij} where employee i is not assigned to task j in solution $\mathbf{X}^*$, while if $x^*_{ij} = 1$ then c'_{ij} will most likely be negative (i.e. a potential saving). Consequently it is possible that the combined effects of two changes will result in a net cost very close to zero, for example re-assigning employee i from task j to task j' when tasks j and j' are both on board the same vessel or on different vessels in the same region.

5.2.2 Formulation

The formulation of the Recovery-type problem is similar to that of the Basic problem given in section 5.1.3, but incorporating the changes outlined above. Where the Basic problem used the x_{ij} variables in the constraints (5.3 - 5.11) these are replaced by the new z_{ij} variables, while the objective function is updated to reflect that we are now concerned with the cost of *changing* assignments. The full formulation is as follows:

$$\min \sum_{\forall i,j} c'_{ij} y_{ij} + \sum_{i \in G} (\mu_i u_i + \phi_i o_i - \Omega_i) \tag{5.14}$$

subject to:

$$\sum_{i=1}^{m+1} z_{ij} = 1 \qquad \forall j \in J \tag{5.15}$$

$$\sum_{j \in C_\gamma} z_{ij} \leq 1 \qquad \forall i \in E_R, \gamma \in \Gamma \tag{5.16}$$

$$\sum_{i=1}^{m+1} \sum_{\forall j \in P_k} e_{ij} z_{ij} \geq \epsilon_k \qquad \forall k \in K \tag{5.17}$$

$$W_{i,b-1} + z_{ib} w_b \leq W_{ib} \qquad \forall b \in B, i \in E_R \tag{5.18}$$

$$0 \leq W_{ib} \leq \omega_i \qquad \forall b \in B, i \in E_R \tag{5.19}$$

$$R_{i,b-1} + z_{ib} r_b \leq R_{ib} \qquad \forall b \in B, i \in E_R \tag{5.20}$$

$$0 \leq R_{ib} \leq 1 \qquad \forall b \in B, i \in E_R \tag{5.21}$$

$$u_i \geq g_i - \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) \qquad \forall i \in G \tag{5.22}$$

$$o_i \geq \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) - g_i \qquad \forall i \in G \tag{5.23}$$

$$z_{ij} = x^*_{ij} + (1 - 2x^*_{ij}) y_{ij} \quad \forall i,j \text{ s.t. } y_{ij} \text{ is defined} \tag{5.24}$$

$$y_{ij} \in \{0,1\} \quad \forall i,j \text{ s.t. } y_{ij} \text{ is defined} \tag{5.25}$$
$$u_i, o_i \geq 0 \qquad \forall i \in G \tag{5.26}$$

Notice that equations (5.24) have been added to define the link between the old schedule, the changes, and the new schedule (i.e. quantity x^*_{ij} and variables y_{ij} and z_{ij} respectively); while (5.12), which previously gave the binary definition of x_{ij}, has been replaced with (5.25), the equivalent for the new y_{ij} variables. In the same way as our basic scheduling formulation, we consider y_{ij} and z_{ij} to be defined dynamically in a pre-processing stage, meaning that they only exist where employee i is eligible to carry out task $j \in J$ and where the length of a rest task $j \in N$ is at least as long as the minimum rest period entitlement for employee i. Also, as discussed, the objective function (5.14) has been modified and now states that we seek to minimize the total cost of the changes.

5.3 Minimizing the Number of Changes

An alternative approach to that outlined above would be to seek to minimise the *number*, rather than the cost, of the changes. This might be particularly relevant if a planner were seeking a solution which would be quick or easy to implement and which would require minimal discussions with employees.

This could be achieved simply by setting $c'_{ij} = 1$ for all $i \in E$ and $j \in J$, and $\mu_i = \phi_i = \Omega_i = 0$ for all $i \in G$ in the formulation presented in section 5.2 above. However in order to be of more use in practical terms, we argue that it is desirable to continue to use the true values of these cost coefficients. This would allow the user, if desired, to set an upper limit Λ on the cost of making these changes. We note however that because not all changes have the same cost, and some may even incur a saving, there is no guarantee that a solution with a smaller number of changes will also have a lower cost.

5.3.1 Additional Definitions

The changes which must be made include modifying the objective function, setting a cost limit, and introducing new variables to ensure costs are calculated correctly.

Objective function

Firstly, the objective function must be updated to consider only the number of changes and not their cost. The new objective is therefore given as:

$$\min \sum_{\forall i,j} y_{ij} \tag{5.27}$$

This is, as discussed above, the equivalent of setting $c'_{ij} = 1$ for all $i \in E$ and $j \in J$, and $\mu_i = \phi_i = \Omega_i = 0$ for all $i \in G$, in the cost-minimization objective (5.14).

Cost limit

We can then modify the cost-minimizing objective to give a constraint which places an upper limit Λ on the cost:

$$\sum_{\forall i,j} c'_{ij} y_{ij} + \sum_{i \in G} (\mu_i u_i + \phi_i o_i - \Omega_i) \leq \Lambda \tag{5.28}$$

We note here that in some cases a sensible upper limit may not be known, or may not be desired when minimizing the number of changes is the only concern. In this case we can set

$$\Lambda = \sum_{\forall i,j} |c'_{ij}| + \sum_{i \in G} (366\,(|\mu_i| + |\phi_i|) + |\Omega_i|) \tag{5.29}$$

as this is the highest possible value of the cost function, given that under- and over-time are calculated as number of days over a calendar year.

Additional variables

The need for additional variables arises from the need to ensure that undertime and overtime variables u_i and o_i for $i \in G$ take the correct values. As a reminder, u_i should be *exactly equal* to $g_i - \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij}\right)$ if this expression is positive, and should be zero otherwise. Similarly, $o_i = \max\left\{0, \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij}\right) - g_i\right\}$.

In the cost-minimizing formulation, inequalities (5.22) and (5.23) were sufficient to ensure this since u_i and o_i appear in the objective function (5.14) with positive coefficients (note that a situation is not conceivable in practice where μ_i or ϕ_i would be non-positive). However, the new minimum-change objective function (5.27) exposes the lack of upper bound placed on the undertime and overtime variables. We therefore propose the introduction of two additional variables for all employees $i \in G$:

$$\psi_i^u = \begin{cases} 1 & \text{if the undertime value for employee } i \text{ is non-negative} \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_i^o = \begin{cases} 1 & \text{if the overtime value for employee } i \text{ is non-negative} \\ 0 & \text{otherwise} \end{cases}$$

Using these, we can supplement inequality (5.22) with the following two expressions to ensure proper definition of u_i:

$$u_i \leq g_i - \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) + M \psi_i^u \quad \forall i \in G \tag{5.30}$$

$$u_i \leq M \left(1 - \psi_i^u\right) \quad \forall i \in G \tag{5.31}$$

and similarly use the following two expressions to supplement inequality (5.23):

$$o_i \geq \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) - g_i + M \psi_i^o \quad \forall i \in G \tag{5.32}$$

$$o_i \leq M \left(1 - \psi_i^o\right) \quad \forall i \in G \tag{5.33}$$

Note that in the above expressions, M is a suitably large number which is greater than or equal to the maximum possible value for the number of days of under- or over-time. Since these are calculated on an annual basis, we can set $M = 366$.

An alternative representation of these constraints can be derived by noting that by definition at most one of u_i and o_i can take a positive value for a given employee i. We can therefore relate ψ_i^u and ψ_i^o by saying that $\psi_i^u + \psi_i^o \leq 1$ for all employees $i \in G$. This allows us to combine the two into a single variable ψ_i which, for a given employee $i \in G$, will take the following values:

$$\psi_i = \begin{cases} 1 & \text{if the overtime value for employee } i \text{ is non-negative} \\ 0 & \text{if the undertime value for employee } i \text{ is non-negative} \\ 0 \text{ or } 1 & \text{if } o_i = u_i = 0 \end{cases}$$

Using this alternative definition, we can substitute ψ_i for ψ_i^o in equations (5.32) and (5.33) to give the following for the overtime values:

$$o_i \leq \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) - g_i + M \psi_i \quad \forall i \in G \tag{5.34}$$

$$o_i \leq M \left(1 - \psi_i\right) \quad \forall i \in G \tag{5.35}$$

and replace ψ_i^u with the expression $(1 - \psi_i)$ in equations (5.32) and (5.33) to obtain the following to calculate the undertime values:

$$u_i \leq g_i - \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) + M \left(1 - \psi_i\right) \quad \forall i \in G \tag{5.36}$$

$$u_i \leq M \psi_i \quad \forall i \in G \tag{5.37}$$

5.3.2 Formulation

We can now update the formulation given in section 5.2 with the new information discussed in section 5.3.1 above. This gives us the following formulation for the task-based problem with an objective of minimizing the number of changes:

$$\min \sum_{\forall i,j} y_{ij} \tag{5.38}$$

subject to:

$$\sum_{\forall i,j} c'_{ij} y_{ij} + \sum_{i \in G} (\mu_i u_i + \phi_i o_i - \Omega_i) \le \Lambda \tag{5.39}$$

$$\sum_{i=1}^{m+1} z_{ij} = 1 \qquad \forall j \in J \tag{5.40}$$

$$\sum_{j \in C_\gamma} z_{ij} \le 1 \qquad \forall i \in E_R, \gamma \in \Gamma \tag{5.41}$$

$$\sum_{i=1}^{m+1} \sum_{\forall j \in P_k} e_{ij} z_{ij} \ge \epsilon_k \qquad \forall k \in K \tag{5.42}$$

$$W_{i,b-1} + z_{ib} w_b \le W_{ib} \qquad \forall b \in B, i \in E_R \tag{5.43}$$

$$0 \le W_{ib} \le \omega_i \qquad \forall b \in B, i \in E_R \tag{5.44}$$

$$R_{i,b-1} + z_{ib} r_b \le R_{ib} \qquad \forall b \in B, i \in E_R \tag{5.45}$$

$$0 \le R_{ib} \le 1 \qquad \forall b \in B, i \in E_R \tag{5.46}$$

$$u_i \ge g_i - \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) \qquad \forall i \in G \tag{5.47}$$

$$o_i \ge \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) - g_i \qquad \forall i \in G \tag{5.48}$$

$$u_i \le g_i - \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) + M(1 - \psi_i) \qquad \forall i \in G \tag{5.49}$$

$$o_i \le \left(\bar{W}_i + \sum_{j \in J} d_j z_{ij} \right) - g_i + M\psi_i \qquad \forall i \in G \tag{5.50}$$

$$u_i \le M\psi_i \qquad \forall i \in G \tag{5.51}$$

$$o_i \le M(1 - \psi_i) \qquad \forall i \in G \tag{5.52}$$

$$z_{ij} = x^*_{ij} + (1 - 2x^*_{ij}) y_{ij} \quad \forall i, j \text{ s.t. } y_{ij} \text{ is defined} \tag{5.53}$$

$$y_{ij} \in \{0, 1\} \quad \forall i, j \text{ s.t. } y_{ij} \text{ is defined} \tag{5.54}$$

$$u_i, o_i \geq 0 \qquad \forall i \in G \tag{5.55}$$
$$\psi_i \in \{0, 1\} \qquad \forall i \in G \tag{5.56}$$

The majority of the constraints are unchanged from the cost-minimizing formulation, with inequalities (5.40 - 5.48) and constraints (5.53 - 5.55) being identical to expressions (5.15 - 5.26) in section 5.2.2. As discussed however, the objective function (5.38) is now different, with the objective being to minimize changes, and the previous objective function expression has been incorporated into inequality (5.39) to allow a cost limit to be placed on the solution. Also added to the problem are constraint sets (5.49 - 5.52), which allow undertime and overtime values to be correctly evaluated; along with expression (5.56) which gives the correct definition of new variable ψ_i.

5.4 Generating Data

Having formulated the cost-minimizing and change-minimizing problems as set out in the preceding sections (5.2 and 5.3 respectively), the next desired step in the process was to investigate potential solution methods for these problems. Ideally, this would be done using real data as supplied by the company; however this was unfortunately not practical. Some of the data relating to costs was thought to be too sensitive to release, but primarily the issues arose from the data storage at the company, with different departments maintaining different elements of the data and therefore making data collation difficult. In addition, some of the required data items were not recorded by the company at all and would be difficult to calculate or quantify.

Instead, key parameters were identified which would describe the make-up of a typical data set and which the company was able to supply. These included:

- Outline crew data including role, nationality and contract type code.
- Number of vessels, and the numbers of each role normally required on board.
- An estimate of the absence rate due to sickness.
- A broad estimate of the daily cost of fixed contract, day rate and agency crew.

Using this information, and with some randomization introduced, it was possible to generate datasets which while not being strictly 'real' would be realistic. This section goes on to outline the procedure for doing this, as well as describing how uncertain elements were accounted for.

5.4.1 Procedure for Data Generation

We now give a step-by-step outline of the process for generating the realistic datasets (note that full details of the code implemented in FICO Xpress can be found in the appendix, section E.1.1). Since full real datasets were not available, there was a degree of uncertainty about how some of the data elements should be generated, and this lead to assumptions having to be made. For example, while an average daily absence rate was available, patterns of crew absence was not known, meaning assumptions had to be made about the length and frequency of absence. Assumptions also had to be made in order to simplify the situation in line with the Task-Based formulation. Section 5.4.2 below gives a discussion of attempts to mitigate the possible effect of poor assumptions by generating datasets using varying assumptions about some of the factors. In the step-by-step procedure below, all assumptions are highlighted in bold within the relevant steps. Note that, as described earlier, each crew group is distinct and currently scheduled independently from the other groups - consequently, we consider the crew groups to be separate problems, and so the data need only be generated for a single crew group at a time.

Step 1 - Set task lengths:

- Using the real crew data, count the number assigned to each contract type and identify the dominant contract type.
- **Assume all crew in group are contracted to this dominant type.**
- Regular on/off patters will be decided accordingly:
  - If predominantly on Norwegian contracts then 2 weeks on, 4 weeks off;
  - If Singapore then 10 weeks on, 5 weeks off;
  - If other, then 4-on-4-off or 5-on-5-off depending on vessel location.

Step 2 - Calculate total number of roles required:

- This is just the sum of the number of the role of interest required on board each vessel.

Step 3 - Generate *starting points* for each vessel, and determine number of tasks:

- From **Step 1** above, will know the standard duration of task on board each vessel.
- **Assume an even spread of crew change dates.**

- **If there is more than one of a role required on board a vessel, assume that their change-over times are staggered.**
- At random, assign a length of time each role has remaining on its initial task at the start of the planning horizon.
- From this, we know the remaining length of the initial task for each vessel, and the regular lengths of tasks thereafter.
- We can therefore calculate the number of tasks each role must be divided into, and therefore the total number of tasks overall.

Step 4 - Assign crew to a regular vessel:

- **Assume all crew have a regular vessel to which they will normally be assigned.**
- For each role, assign two (or, depending on crew-vessel ratio, three) crew to be a regular in that role.

Step 5 - Assign task start times, and determine the standard assignments of crew:

- For the crew assigned as regulars to each role, define an ordering indicating who will work first and who will work second (and possibly third) in that role.
- The employee defined as being first will be assigned the initial task in that role.
- Can work along the timeline, defining the next task to begin at the appropriate time and assigning the next regular employee in the sequence to that task.
- **Assume that these regular patterns have also been observed immediately prior to the planning horizon starting, which therefore allows a calculation of the *work resource* and *rest resource* values at time zero.**
- **Assume that assignments for the final week of the planning horizon have not yet been determined - if a task starts at this point, then it is assumed to be unassigned.**
- **Assume that if there is not enough crew to cover all roles with two regular employees, then agency employees will be used to fill any gaps.**

Step 6 - Generate employee availabilities:

- From data given by the company, an employee is unavailable (due to sickness) on a given day with probability 0.008.
- **Assume that an employee's availability on a given day is dependent on their availability the previous day.**
  - **We can say that if an employee is *unavailable* on a given day, then they are unavailable the following day with probability p.**
  - **Similarly, if they are *available* on a given day then they are unavailable the following day with probability q.**
  - **In order that their overall probability of being unavailable remains at 0.008, we require the following relationship between p and q:**

$$0.008 \times p + 0.992 \times q = 0.008 \tag{5.57}$$

    i.e. $p = 1 - (124 \times q)$, **or** $q = (1 - p)/124$.

- **Assume that unavailability is only known with this certainty for the first four weeks of the planning horizon - the probability of being unavailable can be reduced over the remainder of the planning period.**
  - **This can be done on a straight-line basis, such that for a given day index $d > 28$, the reduction factor $r(d)$ by which the availability probability would be multiplied is calculated as:**

$$r(d) = \frac{(d - 28)}{(n - 28)} \tag{5.58}$$

    where n is the number of days in the planning horizon. The result would be that an employee is unavailable on a given day d with probability $p \times r(d)$ if they were unavailable the previous day; and with probability $q \times r(d)$ if they were available the previous day. The overall effect on the probability of unavailability over the planning period can be illustrated graphically, as shown in Figure 5.1. *Note that this assumption has not been validated, therefore when generating datasets this property can be switched off so that a comparison can be made.*

- Generate at random each employee's availability for day zero (immediately before the planning period begins), given that they are unavailable with probability 0.008.

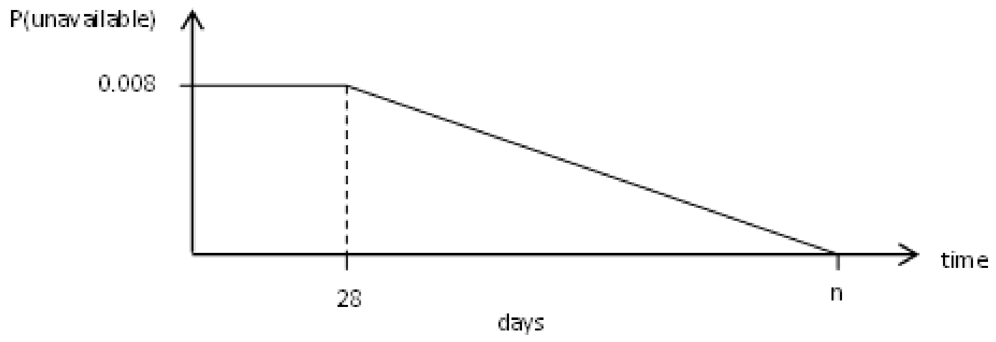


Figure 5.1: Graph showing effect of reduction factor over the planning period.

- Using appropriate values of p and q, randomly determine whether each employee is available on each of the subsequent days up to the end of the planning horizon. *Note that the values of p and q should be varied across the datasets as the true values are not known.*
- Based on the day-by-day availabilities we can calculate each employee's availability for each task, since an employee is available for a task if and only if they are available over all the days that the task covers.

Step 7 - Generate costs of changes for regular crew:

- This can be broken down into two main cost types - transport (i.e. the employee boarding or leaving the vessel), and the actual cost of the employee working the given task.
- With respect to transportation:
  - **Assume that this part of the cost will depend on where the crew are normally based and where the vessel is usually operating:**
    - **Crew are European, North American, Asian, Australasian or *other*.**
    - **Vessels operate in Europe, Africa, USA, Brazil, Asia-Pacific, or *other*.**
  - **Assume that if either the boarding or the departing are due to take place after the first four weeks of the horizon then cost of making the change is less severe - the estimated cost would therefore be halved.**
  - **Assume that because of the cost associated with arranging for the change to be made, any saving made by cancelling a flight will not**

completely cover the cost of an otherwise identical employee being booked in to take the same flight instead.

- With respect to the working costs:
  - **Assume that if employee is on a fixed contract then there is no additional cost directly associated with them working a particular task. Otherwise, the cost of making the changes depends on the crew nationality - Norwegian crew will have a higher day-rate cost.**

- By adding these costs, this allows us to calculate the total cost of changing the assignment. However, may also wish to consider an additional penalty cost associated with making a change, which can be defined as some factor K. If c_{ij}^F is the financial cost of changing the assignment of employee i to task j, then the penalised cost c'_{ij} of this change can be calculated as:

$$c'_{ij} = \begin{cases} K \times c_{ij}^F & \text{if } c_{ij}^F > 0 \\ c_{ij}^F / K & \text{if } c_{ij}^F < 0 \end{cases} \tag{5.59}$$

  Note that it is not clear what a sensible penalty factor should be in this case, therefore we should generate numerous datasets with varying degrees of disruption factor. This will include $K = 1$, which gives $c'_{ij} = c_{ij}^F$ (i.e. only the financial cost is applied), as well as other values of $K > 1$.

  - **Assume that there may be a greater penalty for disrupting assignments within the first four weeks of the planning period, so will define to different K values to ensure this. These will be represented by K_N for the near-term disruption factor, and K_L for the long-term (i.e. after the first four weeks) disruption factor, and we will have that $K_N \geq K_L$.**

Step 8 - Generate costs of changes for agency crew:

- This can be done in a similar way to the regular crew.
- **Assume that all agency crew will be sourced (relatively) locally and so the transportation costs will be low.**
- **Assume that there is no penalty factor associated with disrupting assignments, as agency crew by definition should be available at short notice - therefore the penalty costs mentioned above do not apply.**

- However, may have a penalty arising from the undesirability of employing agency crew, and this will be used in the same way as a multiplying or dividing factor as appropriate. If we call this agency penalty K_{AG}, and have a financial cost of $c^F_{m+1,j}$of changing agency assignment to task j, then the penalised cost $c'_{m+1,j}$ used in the formulations above can be calculated as:

$$c'_{m+1,j} = \begin{cases} K \times c^F_{ij} & \text{if } c^F_{m+1,j} > 0 \\ c^F_{m+1,j}/K & \text{if } c^F_{m+1,j} < 0 \end{cases} \tag{5.60}$$

  Note again that it is not clear what a sensible penalty factor should be, and therefore we should generate numerous datasets with the penalty factor taking values $K_{AG} \geq 1$.

Step 9 - Generate terms of the fixed contracts:

- Under- and over-rates will depend of crew nationality / contract type, as certain Norwegian contract types are much more expensive.
- **Assume that all fixed-contract employees are contracted to work 26 weeks in the year.**
- **Assume the initial solution (before any cancellations had to be made due to absence / illness) had all fixed-contract employees fully utilised - i.e. all were set to work 26 weeks in the current contractual year.**
- Can therefore calculate the expected working time outwith the planning horizon.
- From this, can calculate the amount of working time to which crew are currently assigned, taking into account that they cannot work tasks for which they are now unavailable.
- Can therefore calculate how many weeks under- or over-time pay to which they employee would be entitled under the current solution, taking into account unavailability.

Step 10 - Finally, can write the required information into a data file:

- File heading, and an indication of the values of the parameters used (as discussed above).
- Basic information, such as the number of days in the planning horizon and the number tasks.
- Number of *Projects*:

  - **We will assume that for the majority of crew groups this will not be relevant, and so can be given as zero.** *Note that this effectively removes constraint (5.17) from the cost-minimization formulation and (5.42) from the change-minimization formulation, as it means the project set $K = \emptyset$.*

- Employee labels, and which of them are on fixed contracts (subset G in the above formulation).

- Start time and durations of the tasks.

- The minimum rest and maximum working periods of the employees.

- The eligibility matrix, showing whether or not an employee is available to carry out a task.

  - **Assume that if an employee is assigned to a task starting at time zero then their eligibility $= 1$, regardless of what has been calculated above.**

  - **Assume that agency crew will be available and eligible to carry out all tasks.**

- Work and rest *resource* values at time zero (quantities W_{i0} and R_{i0} respectively in the above formulation).

- The initial solution, allowing for availability data. This can be calculated by multiplying together the two binary values indicating the initial assignments and the availabilities. Clearly for agency crew, this will just be a case of printing the initial assignment value.

- Give the change costs (the c_{ij} values in the above formulation), as well as the contractual information for the fixed-contract crew (i.e. under-rate, over-rate, current excess, number of guaranteed days, expected working time - μ_i, ϕ_i, Ω_i, g_i and $\bar{W}_i$ respectively above).

- Project data:

  - **As above, assume that this is not relevant and so leave as blank.**

  - *Note that this will be relevant to a small number of crew groups, although details of this may be considered at a later date.*

5.4.2 Dealing with Assumptions and Uncertainty

As noted in the above description, it was not always possible to validate the assumptions, even if they appear intuitively sensible. Similarly, it was not certain what values should be set for some of the parameters, such as the disruption penalties. Steps therefore had to be taken to mitigate the potential effects of these choices. The best way of doing this was felt to be to generate numerous data sets with different values of these uncertain parameters. Results could then later be compared for these different groups to determine if the settings made a significant difference to solution time, solution quality, etc.

Overall, four factors were identified which should be varied across datasets - these are described below:

Availability probabilities

The probability that an employee is available on a given day was assumed to be dependant on their availability the previous day, while the probability of an employee being unavailable on a given day was estimated to be 0.008. Using p and q as defined in **Step 6** of the above procedure (section 5.4.1), and their relationship given in equation (5.57), it can be seen that in order to maintain the overall probability of unavailability as 0.008 it follows that as p increases q must decrease, and vice-versa. The true values of p and q are not known, but it can be seen that a higher value of p would give rise to fewer longer-term absences, while a lower value of p (and therefore higher value of q) would give rise to more but generally shorter-term absences. To deal with this uncertainty of the values, it was decided to use three different values of p and q when generating the datasets. These were:

1. $p = 0.8$ and $q = 0.0016$;

2. $p = 0.5$ and $q = 0.004$; and

3. $p = 0.2$ and $q = 0.0065$.

Probability reduction factor

The use of the probability *reduction factor* $r\,(d)$, described in equation (5.58), is based on the idea that while the overall absence rate might be 0.008 it is less likely that an employee's absence will be known two or three months in advance. This ties in with one of the initial premises of the problem - that a schedule is drawn up, but that this schedule later becomes infeasible because of changes to employee availability (and other factors). While this may be a plausible argument in favour of the reduction factor, it has unfortunately not been possible to validate its use, and therefore it was decided to generate data sets with the following settings:

1. Reduce probabilities p and q by multiplying by reduction factor $r(d)$; and
2. Do not use reduction factor $r(d)$, and generate data using the unreduced values of p and q.

Disruption penalty

The disruption penalty K defined in equation (5.59) was proposed to take into account a penalty cost over and above the monetary cost of making changes to the schedule. It is known that making changes is not desirable, but it is uncertain how undesirable the changes are. Also, it is known that making a change within four weeks of the task taking place is less desirable again, since this also entails disruption to travel arrangements. Therefore the factor K is split into two factors - K_N for activities in the near-term (i.e. within four weeks), and K_L for the activities taking place in the longer term.

Four different levels were defined for each penalty term, with $K_N, K_L \in \{1, 2, 5, 10\}$. The lowest level, $K_N = K_L = 1$, means only the financial costs of the changes are taken into account; the highest level, $K_N = K_L = 10$, means positive costs are increased to ten times their actual cost, and negative costs reduced to one tenth. Given that it does not make sense for the long-term penalty to be greater than the near-term penalty, this gave ten combinations of values for K_N and K_L which would be used when generating datasets:

1. $K_N = 1$, $K_L = 1$;
2. $K_N = 2$, $K_L \in \{1, 2\}$;
3. $K_N = 5$, $K_L \in \{1, 2, 5\}$; and
4. $K_N = 10$, $K_L \in \{1, 2, 5, 10\}$.

Agency penalty

The agency penalty factor was proposed to take into account the fact that the use of agency crew is also undesirable to the company. Like the disruption penalty, it is uncertain to what extent this is true over and above the monetary cost of using agency crew. Therefore, as with the disruption penalty, the values of $K_{AG} \in \{1, 2, 5, 10\}$ were used for the datasets in order to judge the potential impact of adjusting this factor.

5.4.3 Datasets Generated

In order to fully take into account the different levels of the factors discussed in section 5.4.2 above, it was decided to generate a dataset using each of the possible combinations of the above parameters. This gave 240 combinations in total.

It also had to be decided which crew group(s) data should be generated for, as each crew group could be considered a separate problem. The *Captains* crew group was selected as being a good example to test the formulation on. While each vessel may only require one captain at any time, it is a role that is required by every vessel. The group therefore has to fulfil roles on ships in various locations - giving a wider range of task lengths and transportation costs - and still requires a considerable number of employees in the pool to keep the roles covered.

The following section (5.5) discusses computational results with the 240 Captains datasets generated.

5.5 Computational Results

Using the datasets generated as described in 5.4 above, the potential of the cost-minimization (section 5.2) and change-minimization (section 5.3) models was tested. All test runs were carried out on a Dell Optiplex 790 computer running Windows 7 32-bit, with an Intel Core i5 3.10 GHz processor and 4GB RAM. The solution methods were varied as appropriate for each case, although the overall principles for solving both formulations are the same. This is described in section 5.5.1 below, before the results themselves are presented in section 5.5.2.

5.5.1 Test Conditions

As discussed in chapter 4, the main issue with the current procedure at the company is that it can be difficult to find good (or at times even feasible) solutions within the short time available, and this can be exacerbated by the need to confirm the changes with the crew and possibly find an alternative if the changes cannot be agreed. We therefore say that for a mathematical model to be of practical use, the company would require it to solve in a relatively short space of time. Consequently, rather than investigating how long the optimal solution would take to find, the aim of the computational tests was to discover the number and quality of solutions which could be found within a reasonable time limit. As indicated, the approach taken was different for the two models, and these are discussed separately below.

5.5.1.1 Testing the Cost-minimization model

For the cost-minimizing formulation (as described in section 5.2), the aim was to obtain a single high quality solution within a set time limit. As discussed, in practice it may subsequently be required to run the model multiple times during one solution process. In particular, this might be necessary if the first solution given proposes a solution which violates one of the ‘implicit’ constraints such as two employees who do not get on being asked to work together. If this occurs, an additional constraint could be added forbidding this, and the model resolved.

For this model, it was decided to use the FICO Xpress software, making use of its inbuilt solution methods. These consist of preliminary heuristics and cutting-plane phase, followed by a branch-and-cut algorithm. Preliminary tests indicated that the default cutting plane strategy of Xpress was insufficient for this model, and so the number of rounds of Cover and Gomory cuts was increased from the default setting. In terms of a time limit, it was taken into account that the planner may wish to run the model multiple times during a solution cycle, and that they would therefore be unwilling to wait a large amount of time to obtain a solution. With this in mind, two time limit settings were chosen, a two-minute limit and a ten-minute limit, which were considered to be at the shorter and longer end respectively of what would be a reasonable solution time. Settings were also altered with respect to the acceptable optimality gap in the solution process. As stated, it is important to obtain high quality solutions, but it is not necessary to obtain the exact optimum solution. This is particularly true when we consider that the solution first proposed may not be acceptable in practice. For this problem, the settings were changed such that the programme would terminate when a solution within 5% of the optimal solution was found.

Bearing these considerations in mind, and based on some preliminary testing, two groups of settings were chosen to test the solving of the Cost-minimization model:

1. Two-minute time limit; maximum rounds of Cover cuts = 30; maximum rounds of Gomory cuts = 10; acceptable optimality gap = 5%.

2. Ten-minute time limit; maximum rounds of Cover cuts = 50; maximum rounds of Gomory cuts = 10; acceptable optimality gap = 5%.

Other than altering these settings, Xpress was free to use its own methods to solve each problem. As stated, this consists of some initial heuristics along with the cutting algorithm, followed by a branch-and-cut search. The nature of branch-and-cut is that there are two quantities to monitor - the value of the best solution found, which (for a minimization problem) will approach the optimal value from above as the algorithm progresses; and the value of the best bound, which will approach the optimal value from below. Only when these two values are equal can optimality be proved. Since a time limit is being imposed on the algorithm, there may not be time to find a provably optimal solution for every instance. However, in some cases it is possible that the objective has reached its optimal value and it is only the bound which has not been improved sufficiently to prove this. In order to take this possibility into account, a third run was made for any instances for which the optimal solution had not otherwise been found. This used the following settings:

3. Six-hour time limit; maximum rounds of Cover cuts = 1000; maximum rounds of Gomory cuts = 1000; acceptable optimality gap = 0.0001% (the default value).

Results from this extended run allow us to comment in a more informed way on the quality of the solutions found in the main test runs where the time limit has been reached before optimality has been proven.

The full code implemented in FICO Xpress for these test runs can be found in the appendix in section E.1.2.

5.5.1.2 Testing the Change-minimization model

The change-minimization model was also solved using the FICO Xpress software, but a different approach was taken to do this. Preliminary tests in this case indicated that when the cost limit Λ was set to a relatively high value, optimal solutions could be found in a matter of seconds. However, the cost of the minimal-change solutions found under these settings was found to be very high compared to the best known solution value for the instance. This gave rise to the idea for an algorithm which would iteratively lower the maximum cost of the solution while seeking to minimize the number of changes at each iteration.

When setting a time limit for the algorithm, preliminary testing found that numerous iterations could be carried out within a two minute time limit. In addition, it should be remembered that that change-minimization would be intended for use when there was particular time pressure on the Planner. Because of this, no tests were carried out using a ten-minute time limit as had been the case for the cost-minimization problem - the two-minute time limit only was used to test the algorithm for the change-minimization problem. An outline of the procedure is given in Algorithm 5.2, while full detail of the code implemented in FICO Xpress can be found in the appendix in section E.1.3.

In this algorithm, the initial value of $\pi = 0.1$ has been chosen in an attempt to strike a balance between bringing the solution cost down quickly, and making the problem too difficult to solve (or infeasible) before a reasonable range of solutions has been generated. The lower setting of $\pi = 0.05$ means that in the event that the algorithm terminates before the time limit τ_T is reached then, assuming the problem is feasible, we know we have a solution with a cost within 5% of the optimal value. This mirrors the settings described for the cost-minimization model above.

The algorithm differs from the cost-minimization approach, however, with respect to the number of solutions which are generated. Here, a solution is produced at the majority of iterations, and generally from one solution to the next the cost is decreasing while the number of changes is non-decreasing. This produces a Pareto-optimal set of solutions with respect to the two objectives of minimizing number and cost of changes. If used in practice,

Algorithm 5.2 Algorithm for solving the Change-minimization problem

```
set $terminate$ = false, and set $\Lambda$ at maximum value (see equation (5.29))
solve the problem with cost limit $\Lambda$ and no time limit
if no feasible solution is found then
  set $terminate$ = true
else
  set solution cost $\kappa = \sum_{\forall i,j} c'_{ij} y_{ij} + \sum_{i \in G} (\mu_i u_i + \phi_i o_i - \Omega_i)$
  {Note that this is simply the value of the left hand side of constraint (5.39) in the
  change-minimization formulation.}
end if
set initial proportion $\pi = 0.1$
set iteration time limit $\tau_I = 30$ seconds, and overall time limit $\tau_T = 120$ seconds
while $terminate$ = false do
  calculate new cost limit $\Lambda = \begin{cases} -10 & \text{if } \kappa = 0 \text{ and } \pi = 0.1 \\ -1 & \text{if } \kappa = 0 \text{ and } \pi = 0.05 \\ \kappa - \lfloor \pi\kappa \rfloor & \text{otherwise} \end{cases}$
  solve problem with cost limit $\Lambda$ and time limit $\tau_I$
  calculate total running time $\tau_R$
  if $\tau_R \geq \tau_T$ then
    set $terminate$ = true
  else if $\tau_R \geq \tau_T - \tau_I$ then
    set $\tau_I = \lfloor \tau_T - \tau_R \rfloor + 1$
  else
    $\tau_I$ remains unchanged
  end if
  if integer solution found at this iteration then
    calculate solution cost $\kappa$
  else if $\pi = 0.05$ then
    set $terminate$ = true
  else
    set $\pi = 0.05$
  end if
end while
```

this would allow Planners to choose a solution which best satisfies their preferences in the trade off. In addition, the Planners would be able to look out for violations of the hard-to-model constraints, and may be able to select what they believe to be a truly feasible solution at their first attempt, instead of having to add an extra constraint and resolve as with the cost-minimization problem. The provision of multiple solutions and the practical implications of this are discussed in more detail in sections 5.5.2 and 5.6 below.

5.5.2 Result Details

We now present the results of the tests, conducted as described in section 5.5.1 above. Note that in order to control for the random elements which are present in the FICO Xpress solution methods, the test runs were repeated and compared. In the cost-minimization runs, all solution values were identical although there was a small variation in the values of the bounds found. This had a small impact in the optimality gaps; however no instance showed more than a 0.002 of a percentage point difference between the first and second runs. There was a small change in the number of iterations achieved (±1) for four of the instances when the change-minimization tests were re-run, but only in one case did this allow an additional solution to be found. Overall, these difference were not considered significant, and therefore for clarity all the results discussed in this section refer to a single test rather than the combined results.

We begin the discussion of the results with the cost-minimization approach, before moving on to discuss the change-minimization formulation. Finally, we investigate the bearing that the parameter values which were varied during instance generation (as discussed in section 5.4.2) may have had on the results.

5.5.2.1 Cost-minimization results

The results of the tests on the Cost-minimization approach can be broken down in three different ways. Firstly, Table 5.1 breaks down the results by apparent solution quality, i.e. the quality of solution that FICO Xpress was able to prove in the time available during the run itself. This analysis reflects only the gap from the solution value to the best bound found during the time limit, and does not take into account the best known solutions or bounds for an instance which have been found in other test runs. The table indicates for both the two-minute and ten-minute test runs which instances could not be solved to within 5% of the optimal solution (the gap set as acceptable, as described in section 5.5.1 above), which could be solved to within 5%, and which were solved to proven optimality. Note that the cases of those solved to optimality will have occurred by chance - since the programme is set to terminate at the first solution found within 5% of the optimal, it will be by luck rather than design if this first within-5% solution happens to

Table 5.1: Results of trial runs on Captains data, summarised by apparent solution quality (i.e. gap to bound)

	10-minute settings			
2-minute settings	Proven optimality	Proven 5% gap	Not within 5% gap	TOTAL
Proven optimality	98	-	-	98
Proven 5% gap	22	105	-	127
Not within 5% gap	-	3	12	15
TOTAL	120	108	12	240

be an optimum. The table shows that 98 instances ($\approx$ 40.8% of the total) were solved to proven optimality in the two-minute run, and 120 instances (50% of the total) were solved to proven optimality in the ten-minute settings. Overall, only 15 instances (6.25%) could not be provably solved to within 5% of the optimal solution within two minutes, with the ten-minute settings solving 3 of these (1.25%) to within the 5% gap. For the 12 instances which could not be solved to within a 5% gap by either setting, no pattern could be found connecting these or distinguishing them from the other test instances.

It is of course possible that a good quality (or even optimal) solution has been found, but that the time has been insufficient to improve the lower bound enough to prove this. Therefore the second breakdown of results shown in Table 5.2 presents the results according to the best known solution quality, using information on the best known bound gained from *all* test runs (including the third group of settings where necessary) to judge the quality of the solution regardless of what could be proved during the run itself. The table shows that for 104 ($\approx$ 43.3%) of the instances the optimal solution was found by both the two- and ten-minute settings, while a further 21 instances (8.75%) had an optimal solution found in one (but not both) of the two runs. We note that, as can be seen, for 2 instances ($\approx$ 0.83%) an optimal solution was found with the two-minute settings but not for the ten-minute settings. While this may appear counter-intuitive, this can be put down to the random element within the solution algorithms, meaning that in a case where neither the two-minute nor ten-minute programmes were able to terminate at a near-optimal solution (as was the case here) it will not always be the case that the ten-minute settings will obtain the best solution. Meanwhile, it can also be seen that only 6 instances (2.5%) could not be solved to within 5% of the best known bound by the two-minute settings, and only 5

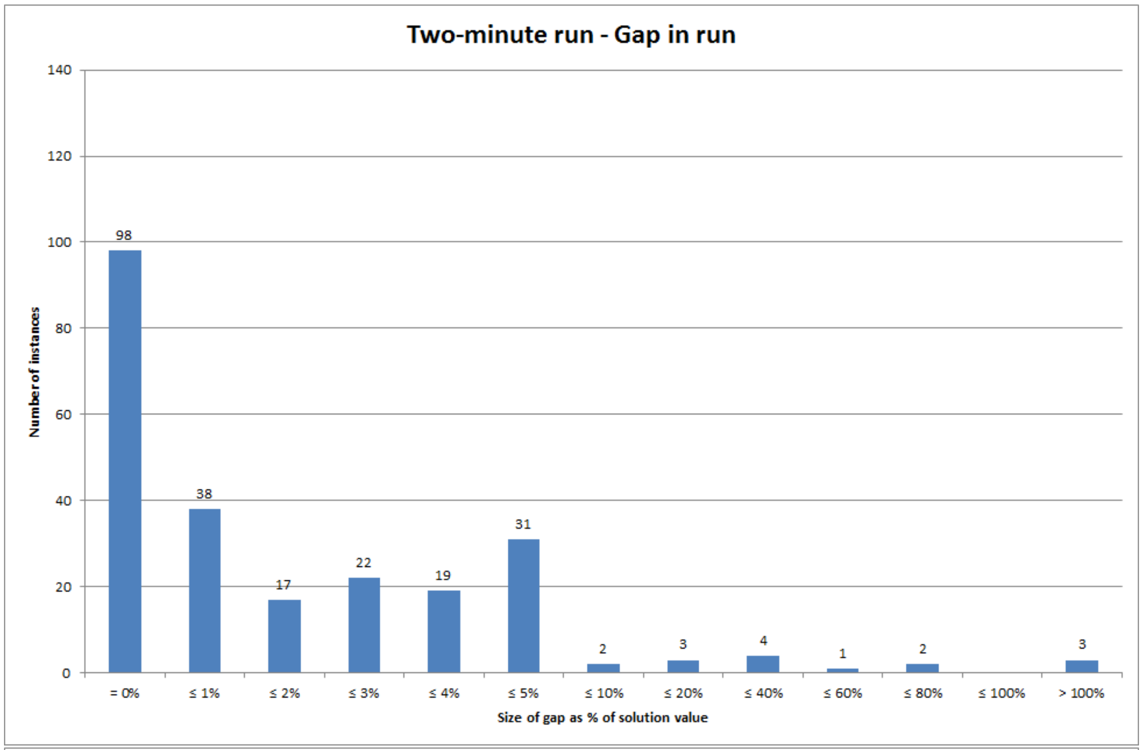
Table 5.2: Results of trial runs on Captains data, summarised by best known quality of the solution found

	10-minute settings			
2-minute settings	Optimal found	Within 5% of optimal	Not within 5% of best known bound	TOTAL
Optimal found	104	2	-	106
Within 5% of optimal	19	109	-	128
Not within 5% of best known bound	-	1	5	6
TOTAL	123	112	5	240

($\approx$ 2.08%) were not within 5% using the ten-minute settings.

As anticipated, there is a discrepancy between Table 5.1 and Table 5.2, indicating that some solutions found are in fact of better quality that could be proven in the time available in the run itself. This discrepancy can best be highlighted by Table 5.3, which combines the information shown in the previous tables. Doing this allows us to categorise the solution quality for each instance, under both the two-minute and ten-minute settings, with respect to both apparent solution quality (i.e. what could be proved during the run in question) and the best known bound for the instance. It can be seen that on the two-minute test run, of the 15 instances which were not provably solved to within 5% of the optimal solution, 9 (60% of the 15) of them did in fact produce a solution which was within this level of quality. The percentage is similar for the ten-minute settings, with 7 instances solving to within 5% of optimal out of the 12 for which this size of gap could not be proved ($\approx$ 58.3%). This is noteworthy because, while of course a proven level of solution quality is desirable, it is more important for the company that the solution method consistently finds good quality solutions.

The results for the cost-minimization approach can also be summarised in histograms, showing the percentage gaps for each instance and each group of settings with respect to both apparent and best known solution quality. These can be shown in Figure 5.2 for the two-minute settings, and Figure 5.3 for the ten-minute settings. These histograms show a different scale compared to the tables given earlier, with the count of instances which solved to within 5% of the respective bound now grouped into single percentage blocks. In all four charts, it can be seen that the most common size of gap for those instances which



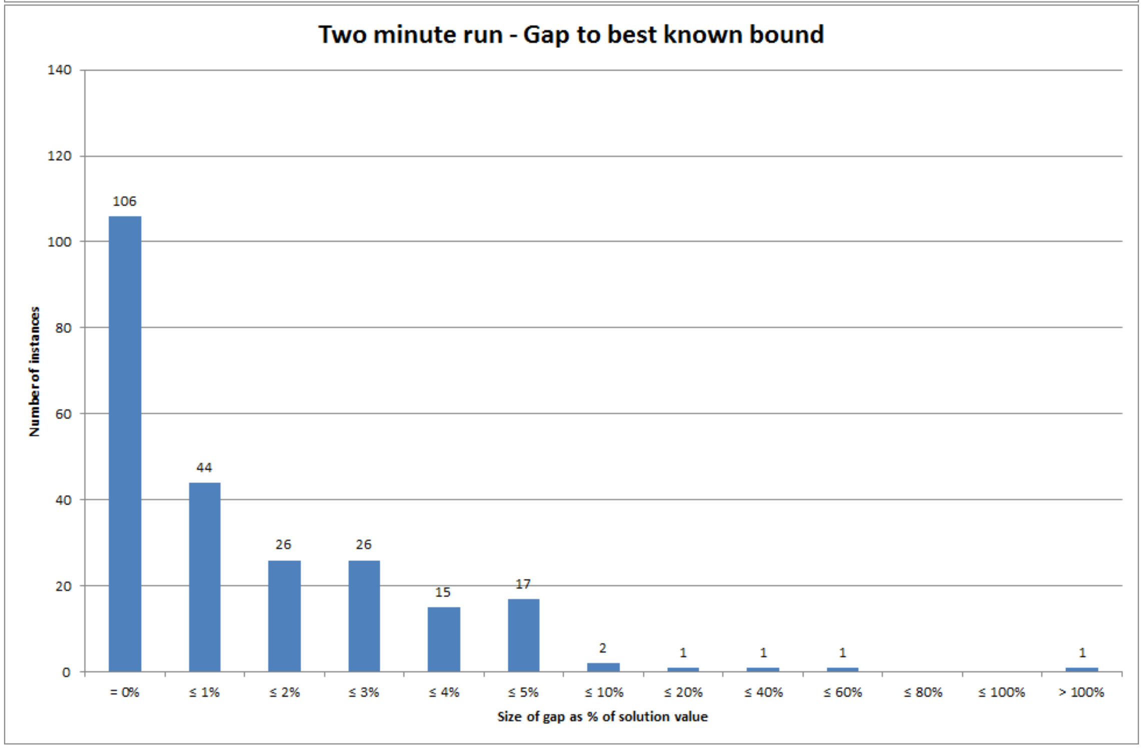
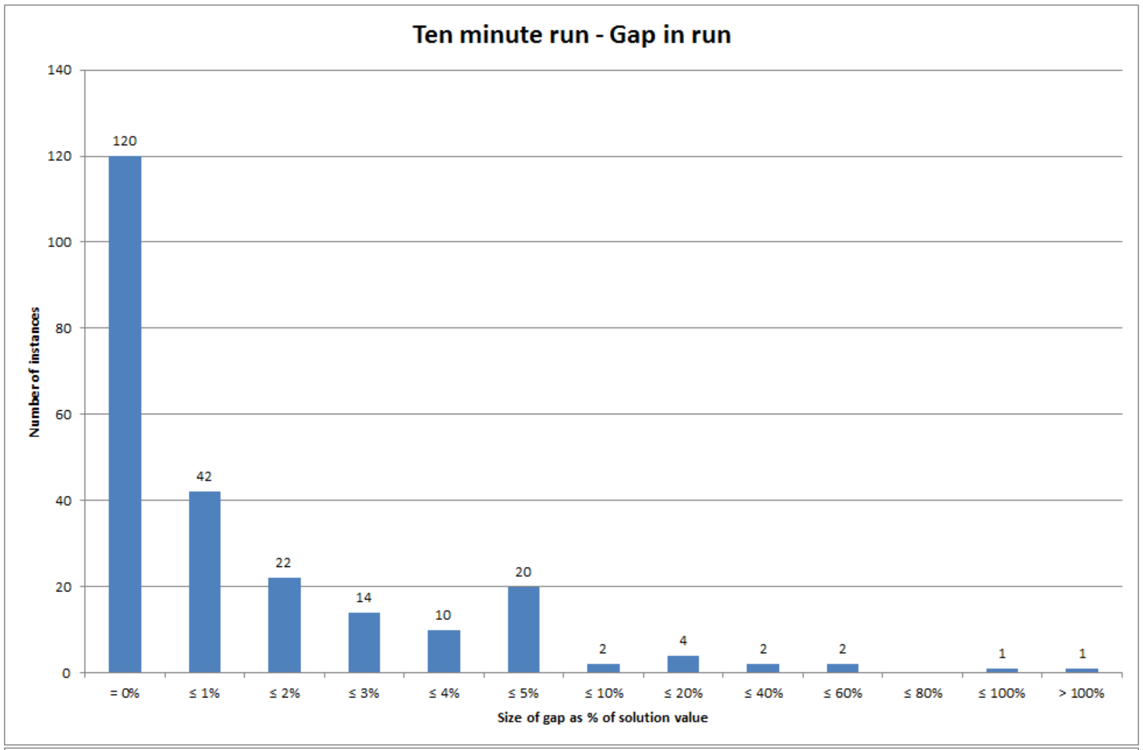


Figure 5.2: Gaps found in two-minute cost minimization runs, as compared to both the bound found during run (top) and the best known bound (bottom).



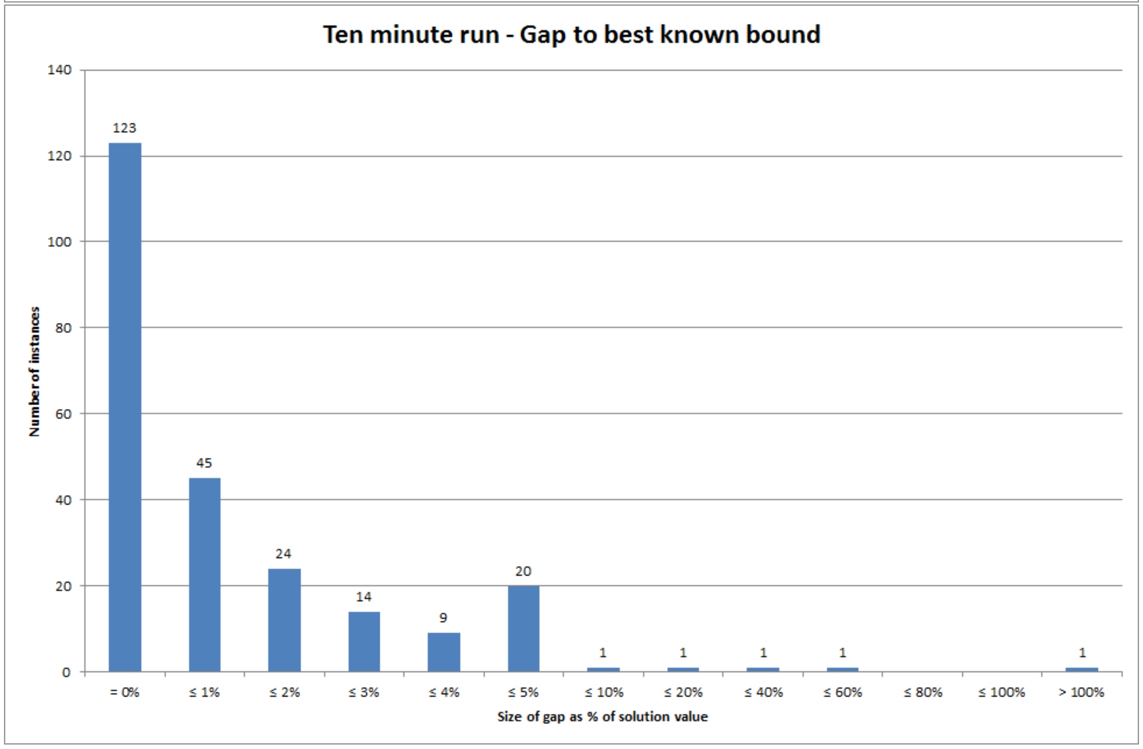


Figure 5.3: Gaps found in ten-minute cost minimization runs, as compared to both the bound found during run (top) and the best known bound (bottom).

Table 5.3: Results of trial runs on Captains data, showing both apparent and best known solution quality

		10-minute settings						
		Optimal solution found			Soln found within 5% of optimal			
2-minute settings		Proved	Proved within 5%	Not proved within 5%	Proved	Not proved	Not within 5% of best known bound	TOTAL
Optimal solution found	Proved	98	-	-	-	-	-	98
	Proved within 5%	5	-	-	-	-	-	5
	Not proved within 5%	-	-	1	-	2	-	3
Soln found within 5% of optimal	Proved	17	-	-	105	-	-	122
	Not proved	-	-	2	2	2	-	6
	Solution not within 5% of best known bound	-	-	-	1	-	5	6
	TOTAL	120	0	3	108	4	5	240

solved to within 5% of optimal, but not to optimality, is to be within 1% of the respective bound.

The graphs also give additional information about the size of the gaps for those instances which could not be solved to within 5% of the respective bound. It can be seen that for the two-minute settings (Figure 5.2), the majority of those which could not be provably solved to within 5% during the run itself (9 out of the 15 instances) are concentrated at the lower end of the graph, with gaps less than or equal to 40%. When it comes to the gap with respect to the best known bound, we again see that the majority of instances which were not solved to within 5% of the best bound (5 out of the 6 instances) are concentrated at the lower end, with two solving to within 10% of the best bound, and three more being within 60% (in fact, the results show that these three are all within 45% of the best known bound). The major outlier here is the single instance which was not solved to within a 100% gap from the best known bound.

A similar pattern can be observed in Figure 5.3 for the ten-minute settings. We see 8 of the 12 instances which were not provably solved to within 5% of the bound found in the run were solved to within 40%, while 4 of the 5 instances not solved to within 5% of the best known bound were solved to within 60% (and indeed, as above, to within 45%) of the best bound. Again, there is the single outlier which even for the ten-minute settings was not solved to within a 100% gap of the best known bound.

Further investigation into this outlying instance revealed it to be one which has a solution value and best bound unusually close to zero (as compared to the other test problems). The ten-minute settings produced a solution with a cost of 20.5 units, while the average solution cost across all instances was 12306.5 units; and the best known bound for this instance is -50.515 units, while the average best bound across all instances was 12206.1 units. The gap therefore for this instance was 71.015 units, which is in fact smaller than the average gap to best known bound across all instances of ≈ 100.43 (which includes the 123 instances with a gap of exactly zero). However, because it is being divided by a number much closer to zero than on average for the other instances, the size of the gap in percentage terms is greatly magnified. We therefore feel that this particular instance of what appears to be a very large optimality gap is not in fact a cause for concern.

5.5.2.2 Change-minimization results

We now go on to present the results for the change-minimization model. Since a different approach is taken to solving this problem, the results should therefore also be presented in a different manner. There are now two factors to consider - the cost of the solution found, and the number of changes it entails. Firstly we recall from section 5.5.1 that, unlike the cost-minimization approach, multiple solutions can be found in a single run of the solution algorithm. This means the planner can be presented with several alternative solutions which form a *Pareto-optimal* set between minimizing cost and number of changes. An example of this for one of the datasets can be seen in Figure 5.4 below. In terms of how this relates to the progression of the algorithm, it is the solution plotted furthest towards the lower-right corner which would have been found first. The imposition of the decreasing cost limits produces solutions progressively to the left of the previous solution. At first, several solutions are found with the same (minimal) number of changes but increasingly lower costs, until eventually the number of changes increases in order to find a solution within the cost limit. The pattern in this example is typical of that found for other instances using this algorithm, although we should not always expect the number of solutions found to be as high as this.

The number of solutions found by the algorithm for each instance within the time can be summarised in Figure 5.5 below. This histogram shows that the 34 solutions found for instance R133 was in fact the highest for all instances. Overall, the distribution of the number of solutions found follows a slightly skewed normal distribution, with a longer tail to the right of the graph (the mean value of 13.3 changes being a little higher than the

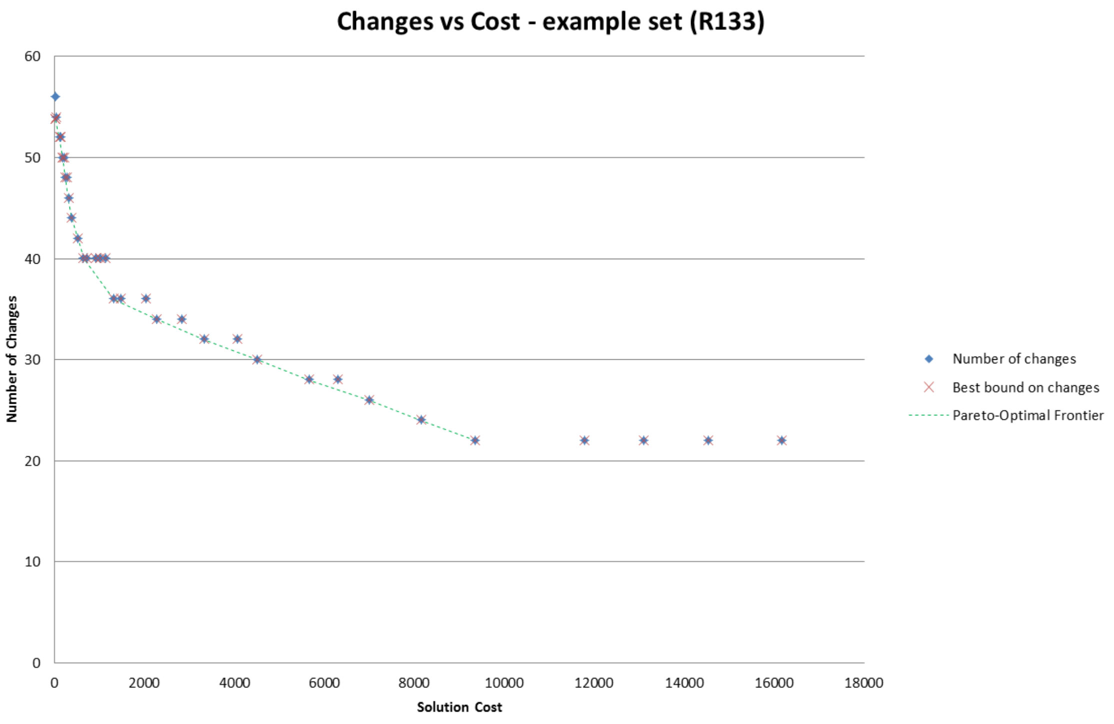


Figure 5.4: Graph showing cost and number of changes for solutions to test set R133.

median of 12 and the mode of 11). All instances found at least 5 solutions within the two minute time limit, demonstrating that in every one of these cases a Planner would have been presented with at least some degree of choice.

Looking next at the quality of solutions produced, we will now consider the effectiveness of the procedure set out in Algorithm 5.2 in producing low cost solutions. This is illustrated in Figure 5.6, which shows the gap to the best known bound for the lowest cost solution found for each instance (note that there is no bound generated during the run in this case, except in a rare case where the reduction of the cost limit results in an infeasible problem). Comparing this graph to the histograms in the lower halves of Figures 5.2 and 5.3, and to Table 5.2, indicates that in terms of the lowest cost solution found for each instance this cost-reducing change-minimization algorithm is reasonably competitive. It can be seen that only 21 instances (8.75% of the total) resulted in the optimal solution being found, compared to $\approx$ 44.17% and 51.25% of instances for the two-minute and ten-minute cost-minimization respectively. However, 50 instances ($\approx$ 20.83% of the total) were solved to within a 1% gap to the best known bound and 57 instances (23.75%) were solved to within 2%, an improvement on the to $\approx$ 18.33% and $\approx$ 10.83% respectively for the two-minute, and 18.75% and 10% respectively for the ten-minute settings. Considering the first six

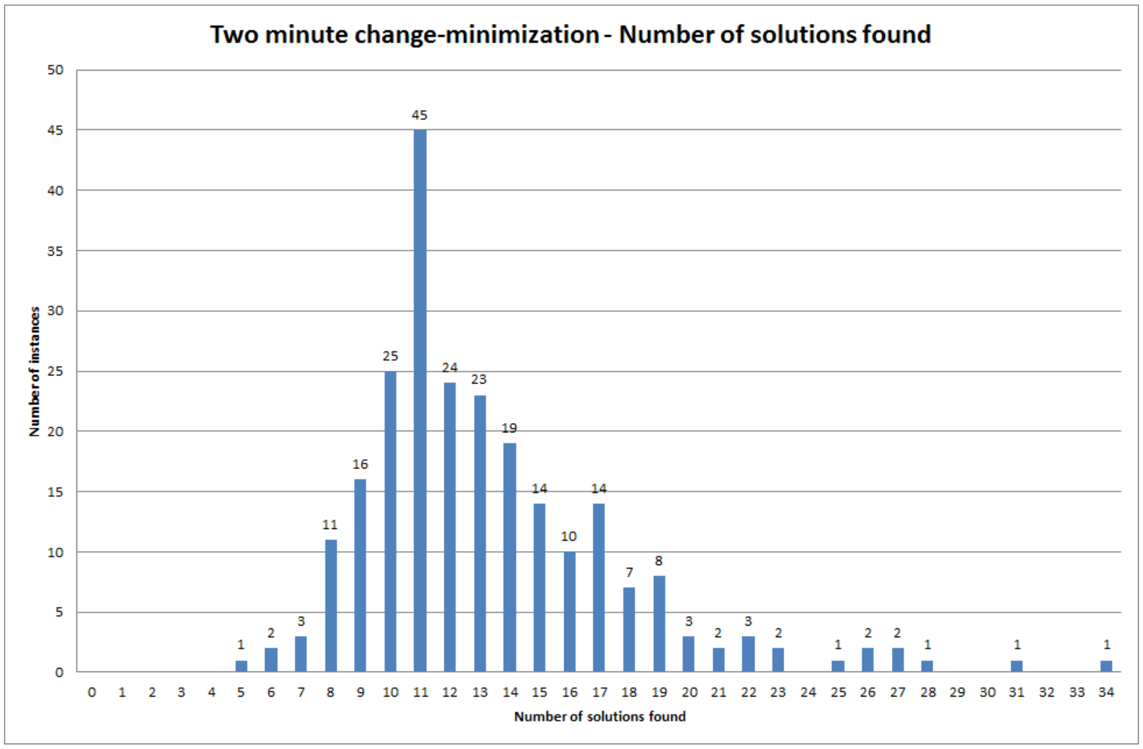


Figure 5.5: Number of solutions found by each instance using algorithm for change-minimization approach.

columns of the graph in combination, we see that 198 instances (82.5% of the total) were solved to within 5% of the best known bound. This is fewer than for the cost-minimization approach, but is quite close to the 97.5% and ≈ 97.92% observed respectively for the two-minute and ten-minute settings. As with the cost-minimization results, a number of the instances in the tail of the graph are focussed at the lower end, with 27 of the 42 instances which were not solved to within 5% still resulting in a solution of cost within 40% of the best known bound (and all these are in fact within 35%, as before).

However, there are considerably more instances for which the gap to the best known bound is greater than 100%, with 11 instances (≈ 4.58% of the total) falling into this category for the change-minimization, as compared to just a single instance for each of the cost-minimization tests. Further investigation into this revealed that all 11 of these instances shared a common factor of the best known bound being negative. This has the effect of inflating the percentage measure of the gap as a positive solution cost decreases towards zero. It could in fact be seen that numerically the size of the gap was not noticeably larger for these instances than the average. The most likely reason for the increased number of instances falling into this category is that the change-minimization algorithm does not

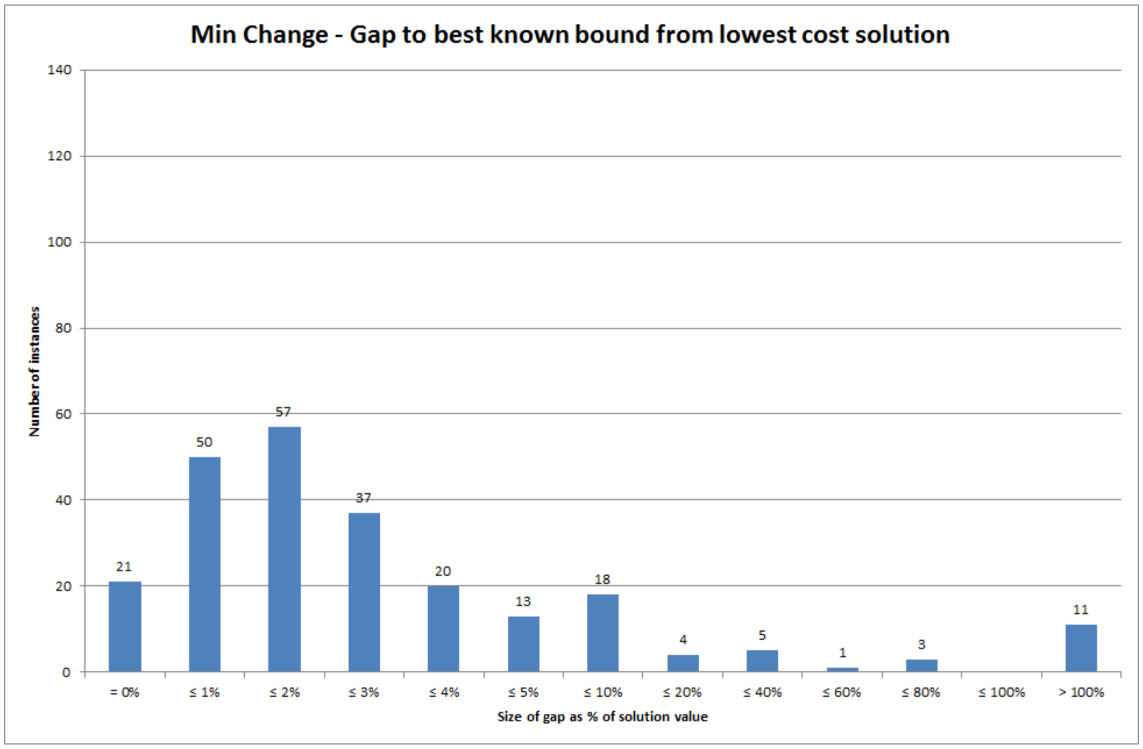


Figure 5.6: Gap to best bound for lowest-cost solutions found in two minute Change-minimization run

provide sufficient opportunity for the solution cost to reach a negative value. In real terms, the size of decrease of the cost limit becomes smaller as the solution cost approaches zero. In contrast, the cost-minimization approach makes no distinction between positive and negative costs, and will have no barrier to reaching negative cost values. This is something which could be investigated further in the future, with a view to improving the ability of the change-minimization algorithm to deal with small positive cost values which could become negative.

An alternative comparison which may be of interest is to examine the difference between the lowest cost solution found by the change-minimization procedure and the solution found in the two-minute cost-minimization test. We can consider the amount by which the best change-minimization solution is *greater* than the two-minute cost-minimization solution, and represent this as a percentage of the cost-minimization solution, giving a gap value which has a similar form to that of the optimality gap. The outcome of these calculations can be summarised in Figure 5.7 below. The graph shown is encouraging, as it shows that for 41 instances ($\approx$ 17.08% of the total) the change-minimization approach finds a solution with a value *less* that that of the cost-minimization solution, while for another 23 instances ($\approx$ 9.58% of the total) the solution value found was identical. For a further

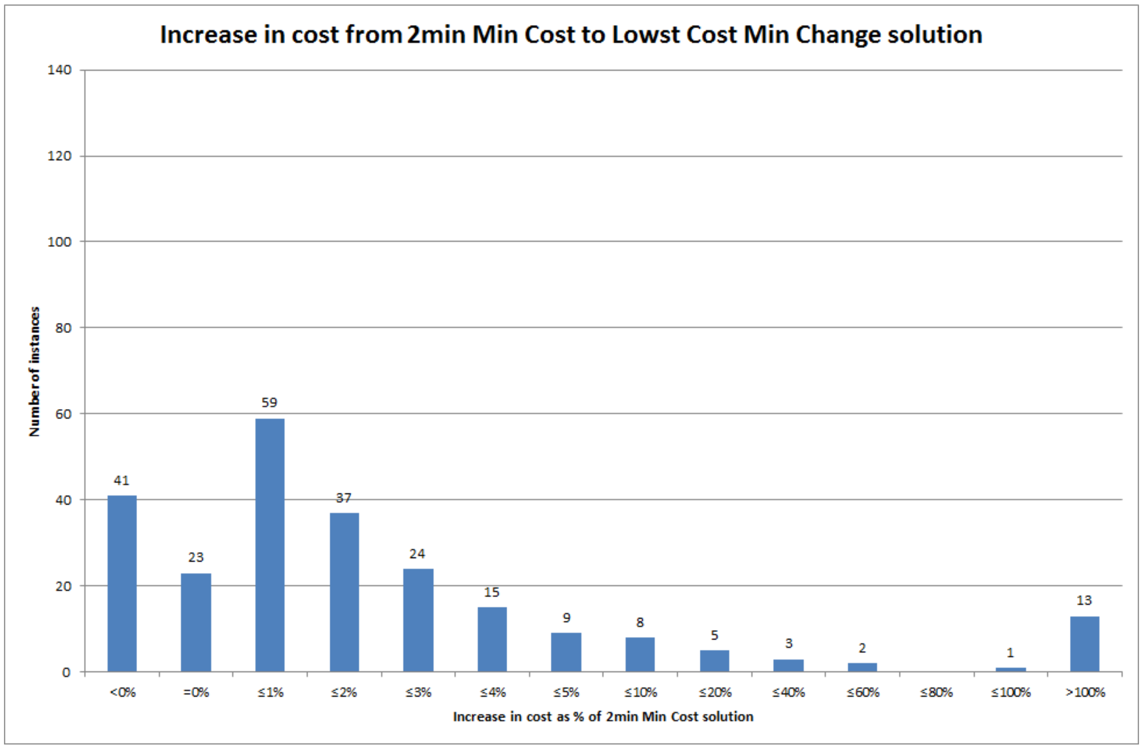


Figure 5.7: Increase in cost from two minute Cost-minimization solution to best Change-minimization solution

59 instances (≈ 24.58 of the total), the best change-minimization solution was within 1% of the two-minute cost-minimization solution; and a combined 144 instances (60% of the total) had a lowest cost change-minimization within 5% of (but not equal to) two-minute cost-minimization solution. As with similar graphs earlier, it can also be seen that 16 of the 32 remaining instances had a change-minimization solution within 40% (and also within 35%) of the cost-minimization solution, although 13 instances (≈ 5.42% of the total) were not within 100% of the two-minute cost-minimization solution. Overall, the results shown in this figure suggest that the ability of the change-minimization procedure to reduce cost is quite close to that of the cost-minimization approach for a large majority of instances.

It should be noted that the results presented in Figure 5.7 refer to the *best* (i.e. lowest-cost, and final) solution found by the change-minimization method. Solutions found earlier in the process will, by their nature, have a much greater gap to the cost-minimization solution. The first solution found by the process, i.e. the minimal-change solution with no cost limit, was found to have a median gap of ≈ 398.4% to the two-minute cost-minimization solution, equating to a cost of almost five times the magnitude. We also recall, as exemplified in Figure 5.4, that other minimal-change solutions may be found which have a lower cost. Therefore we also looked at the median gap from the cheapest minimal-change solution found to the two-minute cost-minimization solution, and this was found to be $\approx 223.0\%$, indicating a cost of over three times that of the cost-minimization solution.

We next look at the results with respect to the number of changes in the solutions found by the algorithm. We begin by discussing the results from the initial solving of the change-minimization problem, i.e. the minimal number of changes for each instance. It should be noted that for each of the 240 instances, FICO Xpress was able to solve the non-cost-constrained problem to optimality in a fraction of a second (between 0.12 and 0.30 seconds for each instance). We therefore do not present the results with reference to optimality gap, but instead show the actual numbers of changes in the optimal solutions - this can be seen in Figure 5.8 below. Since all 240 datasets were based on the same basic

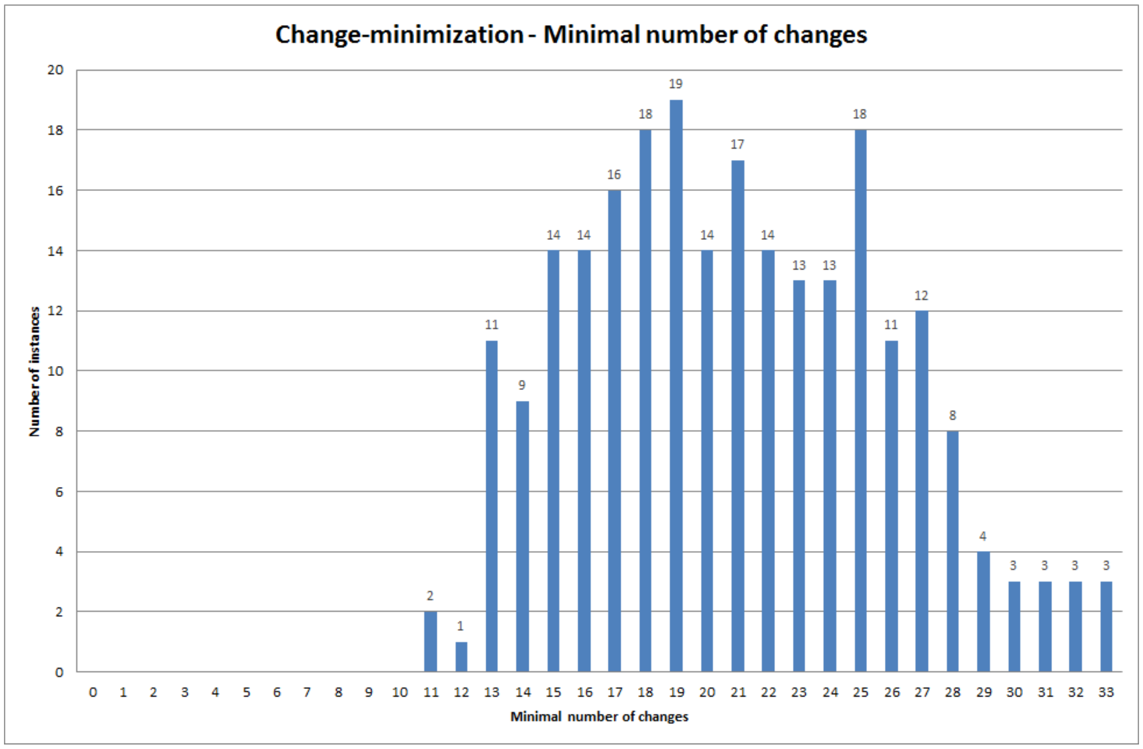


Figure 5.8: Number of changes in the minimal-change solution for each instance.

parameters, including number of roles, number of employees and the overall absence rate, it can be expected that to a certain extent the minimal number of changes required to make the schedule feasible should be similar. This can be seen to be the case, with all instances requiring between 11 and 33 changes in the minimal-change solution. The mean number of changes required was 20.95, with the median number equal to 21 and lower and upper quartiles of 17 and 25 respectively. This indicates a relatively even spread of the results across the range, although as the graph shows there is a degree of tapering in the tails.

These results can be compared to the number of changes required to implement the solutions found by the two-minute cost-minimization settings. In keeping with the results shown in Figure 5.7 above, we subtract the number of changes in the cost-minimization solution from the number in the minimal-change solution to give an 'increase' in the number of changes that the change-minimization approach produces (although note that clearly this cannot take a positive value). This can then be represented as a percentage of the number of changes in the cost-minimization solution, and the results are summarised in Figure 5.9 below. Given that the non-cost-constrained change-minimization problems were

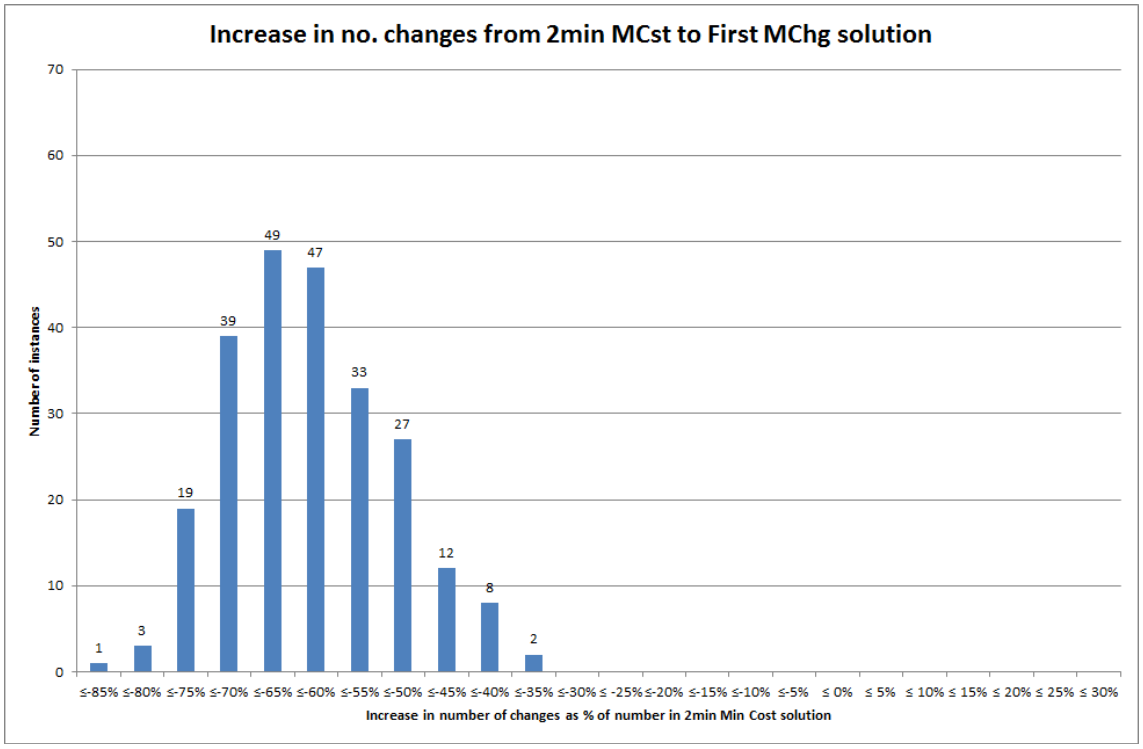


Figure 5.9: Percentage 'increase' in number of changes to minimal-change solution from two-minute cost-minimization solution.

all solved to optimality, it is clearly not possible that any of the cost-minimization solutions would have fewer changes (and therefore give a positive increase to the minimal-change solution). However, it can be seen from the graph that all 240 instances display a sizeable decrease in the number of changes when change-minimization is used, with a decrease of at least 35% when compared to the two-minute cost-minimization solution. Overall, the decreases produce a histogram which follows a roughly Normal curve, with values between a 36% and 89% decrease when the minimal-change solution can be found.

As well as examining the minimal number of changes for each instance, we can also consider the number of changes required by the final solutions (i.e. the lowest cost) found by the change-minimization algorithm. Similar to Figure 5.8 earlier, we can firstly show a summary of the number of changes in the lowest cost solutions for each instances - this is presented in Figure 5.10. As can be seen, there is a much wider range of solution values in

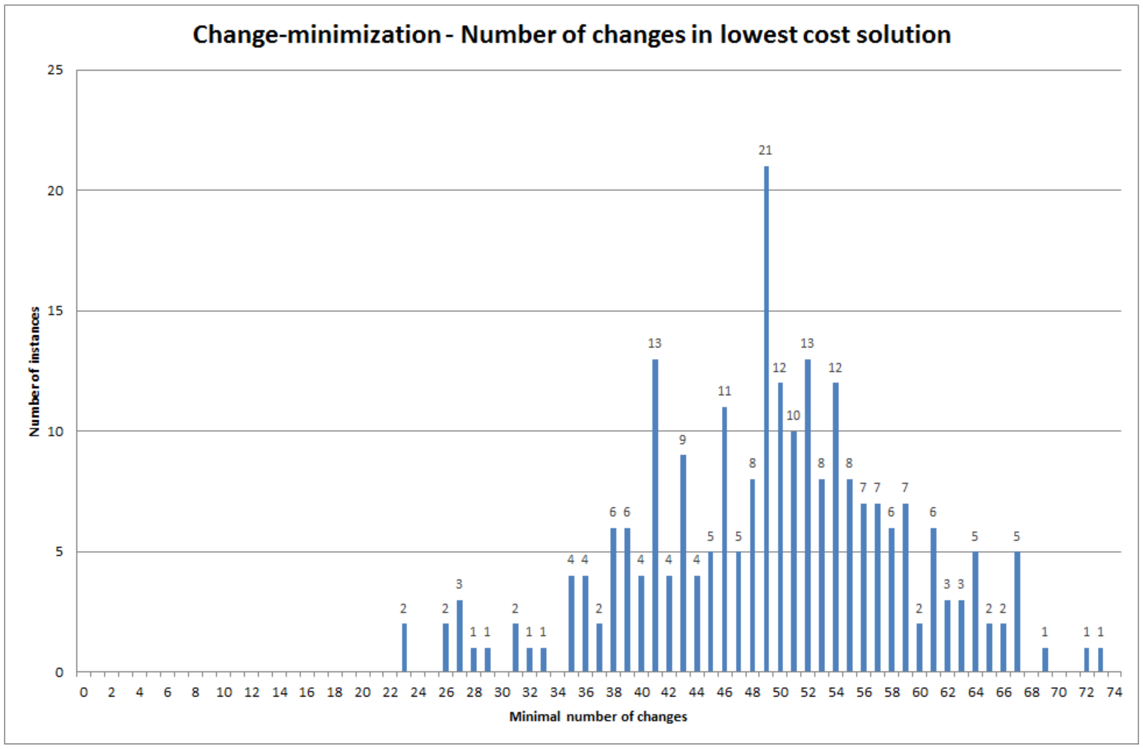


Figure 5.10: Number of changes in the lowest cost change-minimization solution for each instance.

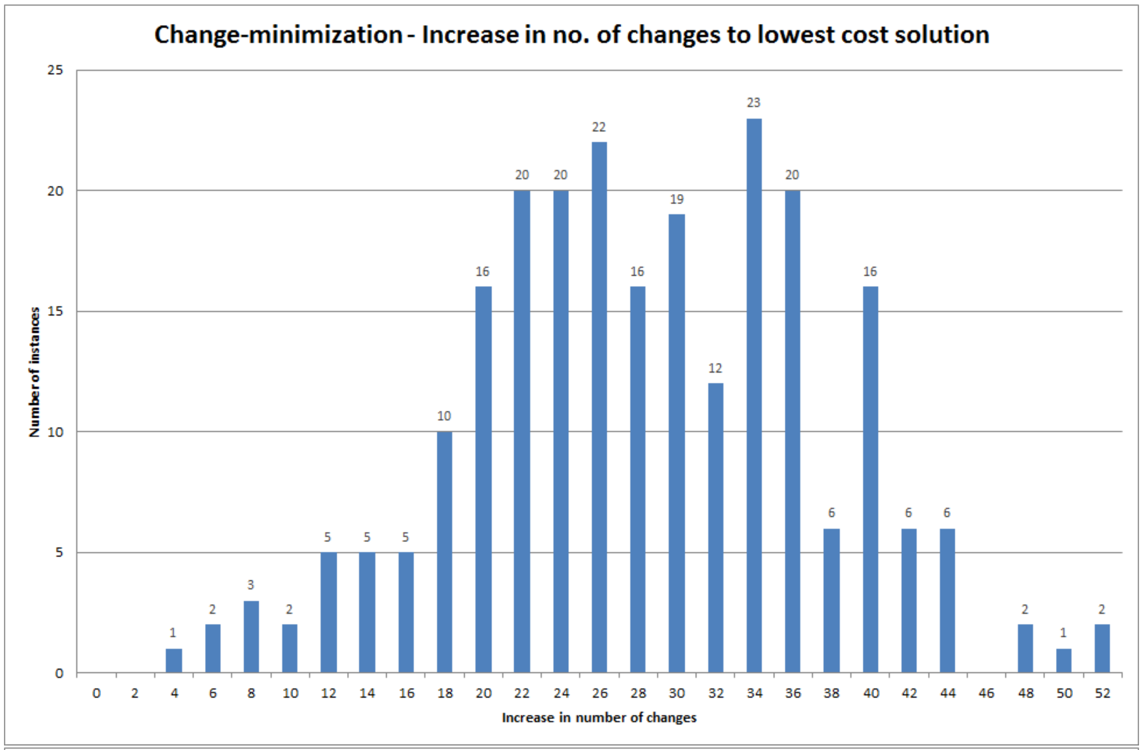
this case, from a minimum of 23 to a maximum of 73, and a more obvious bell shape to the curve as compared to the minimal-change results. This could be related to a number of factors such as the minimal number of changes for the instance, or the amount of restriction placed by the cost limit which as it becomes more restrictive will be more likely to produce a higher number of changes. The different cost patterns present in the instances, due to different parameter settings or the random elements, could also have a bearing on pattern observed here.

Overall, Figure 5.10 is probably less informative than its equivalent for the minimal-change solutions. Perhaps more useful would be to compare the number of changes in the minimal-change solution to the number of changes seen here for the lowest cost change-minimization solutions. We therefore calculate the increase in changes for each instance from the minimal-change to the lowest cost solution, and present two summaries of this in

Figure 5.11 - as a purely numerical value in the upper chart, and as a percentage of the minimal-change solution in the lower chart. Perhaps the most interesting point to notice about the upper chart is that all values are even numbers. While this at first may be surprising, it is actually a natural consequence of the structure of the problem. Consider that once a feasible solution has been found, it will contain a certain number of changes which may be an odd or even number. However if you wish to alter this solution, if you make an additional change by say removing employee i from task j, then you must make a corresponding second change to ensure task j is still covered, say by employee i'. If i' must be removed from another task in order to do this, that other task must also be covered, and so on. Looking at the pattern produced by the graph, it could be viewed as representing a bi-modal distribution with peaks at 26 and 34 changes (the mean is an increase of 28.33 changes). The range of the results is quite large, with instances showing an increase of between 4 and 52 changes from the minimal-change solution.

When these numbers are viewed as a percentage of the minimal-change solution (the lower chart in Figure 5.11), we again find a distribution which appears bi-modal, with peaks in the range between a 80% and 90% increase and between a 150% and 160% increase; however, there is a much longer tail to the right here. Overall, the mean increase in the number of changes is $\approx$ 145.92% of the minimal-change solution, i.e. around two-and-a-half times the number of changes. Aggregating some of these percentages, 68 instances ($\approx$ 28.33% of the total) have an increase of up to 100%, i.e. up to double, while a majority of instances (124, $\approx$ 51.67% of the total) have an increase of more than double but no more than treble the minimal-change solution. Only 48 instances (20% of the total) show an increase in the lowest cost change-minimization solution of more than 200% of the minimal-change solution.

Finally, we can compare the number of changes in these lowest cost change-minimization solutions to those in the two-minute cost-minimization solution. In a similar way to Figure 5.9 previously, we again subtract the number of changes in the cost-minimization solution from the number in the lowest cost change-minimization solution to give an increase in the number of changes (noting that unlike previously, this value may be positive). This can once again be represented as a percentage of the number of changes in the cost-minimization solution, with the results summarised in Figure 5.12 below. This graph shows that 4 instances ($\approx$ 1.7% of total) have an increased number of changes in the lowest cost change-minimization solution as compared to the two-minute cost-minimization approach. In addition, the modal column includes 27 instances ($\approx$ 11.3% of the total) where the number of changes were the same, i.e. an increase of zero, along with 31 instances ($\approx$ 12.9% of the total) which had an 'increase' of less than zero but greater than -5%. Clearly the majority of instances show a decreased number of changes even in the lowest cost change-minimization solution as compared to the cost-minimization approach. Although



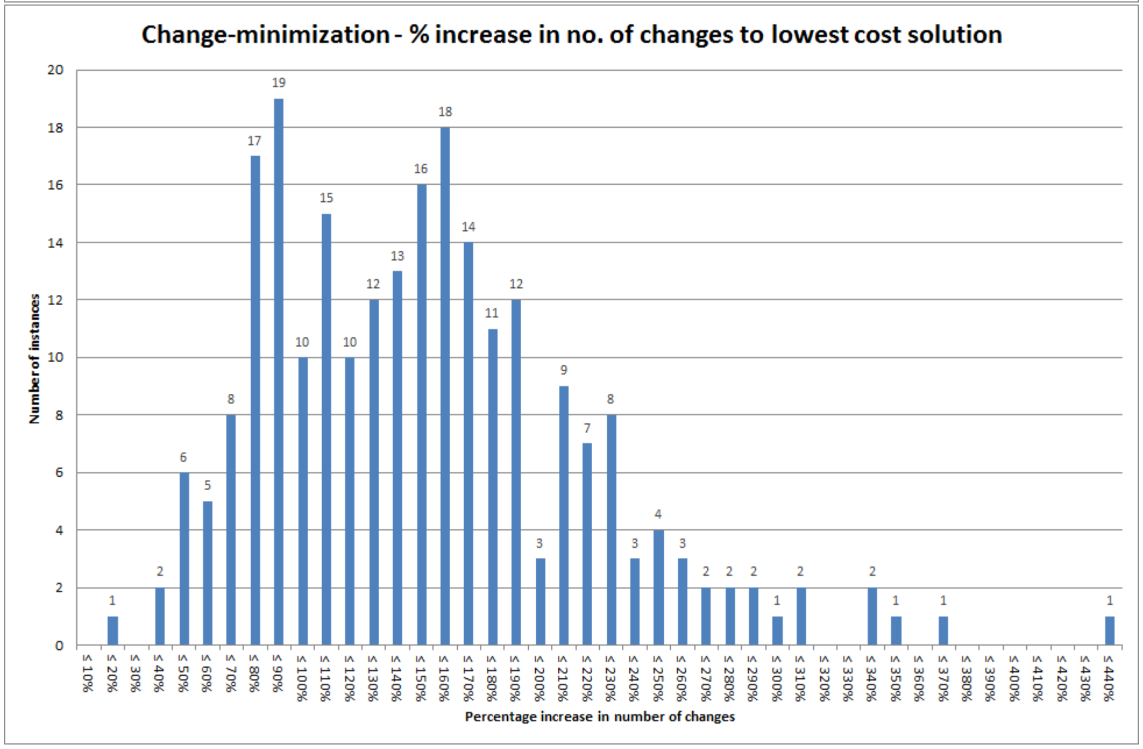


Figure 5.11: Increase in number of changes from minimal-change solution to lowest cost change-minimization solution, shown as numerical value (top); and as a percentage of the minimal-change solution (bottom).

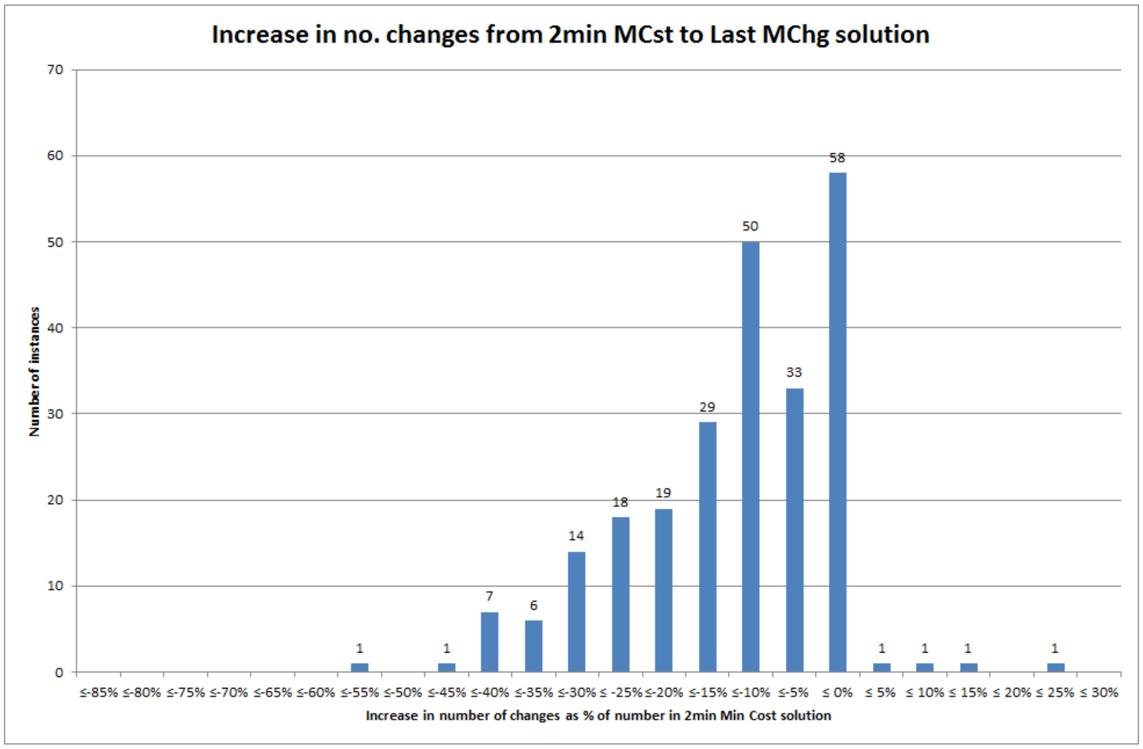


Figure 5.12: Comparison of number of changes in cheapest cost-min. solution against cheapest solution found by change-min. method

these decreases seem more likely to be a smaller percentage decrease, at the extreme the largest decrease was found to be ≈ 59.6% of the cost-minimization solution, meaning in that instance the final change-minimization solution still had less than half the number of changes.

Summarising the implications of these results on the change-minimization approach, a conclusion we can draw is that in general it should be preferable for the planners to make use of this approach rather than cost-minimization to solve their crew scheduling problem. As demonstrated, the cost of the solutions may at times not be as cheap; however, it does have the advantages of finding solutions which will entail less disruption, and therefore will be easier to implement, and will also more easily allow the proposal of multiple solutions without a need to re-solve manually. Further details of how we envisage this working in practice are discussed in section 5.6. Firstly however we must consider the potential impact of the parameter values used in the generation of the test instances.

5.5.2.3 Effect of parameter values on results

As discussed in section 5.4.2, several parameters required for data generation were uncertain and were therefore varied across a range when the datasets were being created. Having done this, it was of course necessary to investigate whether changes to these parameter values had any effect on the results, and consequently determine whether the company may observe a different pattern of results in practice depending on the 'true' values of these parameters.

Of particular interest was how these different parameter values would affect the performance measures which are of primary concern to the company - the cost of solutions, and the number of changes to the existing schedule that the solutions entail. Based on this, several results from both the cost-minimization and change-minimization approaches, as well as some comparisons between the two, were chosen to be examined. These were as follows:

1. From the cost-minimization results:
    - Running time, for both the two-minute and ten-minute time limits.
    - Number of changes in the solution for the two-minute settings only.
    - The gaps proven in the test runs, for both the two-minute and ten-minute settings.
    - An adjusted proven gap, with all values less than 5% taken to be equal to 5%, for both the two-minute and ten-minute settings. This takes into account the acceptable optimality gap which was set to 5%, and therefore we can consider any solution within 5% of optimal to be as good as any other.
    - The gap to the best known bound for each instance, for both the two-minute and ten-minute settings.
    - An adjusted gap to the best known bound, with all values less than 5% taken to be equal to 5% as above, for both the ten-minute and two-minute settings.
2. For the change-minimization results:
    - Running time.
    - Number of iterations.
    - Number of changes in the minimal-change solution and in the lowest cost solution.
    - Percentage gap to the best known bound for the costs of the following:
        - The first (i.e. non-cost-constrained) solution;

    - The cheapest minimal-change solution;
    - The lowest cost solution.
  - An adjusted percentage gap for the lowest cost solution, with all values less than 5% taken to be equal to 5%.

3. For comparing the change-minimization results with the two-minute cost-minimization results:

  - The percentage increase in the number of changes for both the minimal-change and lowest cost change-minimization solutions, with the two-minute cost-minimization solution as the base of comparison.
  - The percentage increase in cost for the non-cost-constrained, the cheapest minimal-change and the lowest cost change-minimization solutions, with the two-minute cost-minimization solution as the base of comparison.

This gave 24 sets of results to be examined, although 5 of these related to the 'adjusted' cost gaps.

Also to be decided was the kind of test to be used, and the Analysis of Variance approach was identified as the preferred option. As detailed in section 3.2, an Analysis of Variance tests the hypothesis that the data relating to different levels of a parameter in fact come from different distributions, with the null hypothesis being that the distributions are the same. The test used for this, the F-test, is the most powerful test for identifying differences between groups; however it is only applicable where a number of assumptions hold. One of the key assumptions is that the distribution underlying the data is a Normal distribution, and this must hold for each of the parameter levels separately. This could be judged from a visual inspection of histograms for each of the results being examined. The 19 histograms used for this are shown in Appendix B.1.

In some cases it was obvious from this histograms that the assumption of normally distributed data did not hold. For these results, the non-parametric equivalent of the ANOVA, the Kruskal-Wallis test, had to be used instead. This test is less powerful than the F-test, meaning that there is a greater probability that it will fail to reject the null hypothesis in a case where the distributions are in fact different (power is discussed in more detail in section 3.2.2). However, since the F-test is not valid here, it is the best test available.

The tests were carried out to determine whether seven parameter values had an influence on these outputs. These parameters, as described in section 5.4.2, are:

- Availability probabilities p and q;
- Whether or not the time reduction $r(d)$ is used;

- The near- and long-term disruption factors, K_N and K_L respectively;
- An aggregated disruption factor $\hat{K}$, calculated as a weighted average of the near- and long-term factors such that

  $$\hat{K} = \frac{(4 \times K_N) + (9 \times K_L)}{13}$$

  and
- The agency penalty factor K_{AG}.

The p-values for the statistical tests (see section 3.2.1 for more information about p-values and statistical tests) are reported in Table 5.4 for the parameters relating to availability and Table 5.5 for the parameters relating to cost, with an indication given in each table of which test was used. Note however that since the values of p and q are related by equation (5.57), the influence of the value of q on the results is identical to that of p. For this reason, results are only reported with respect to probability p. All values which are significant at the 5% level, i.e. for which there is evidence to reject the null hypothesis and say that there is a difference between the distributions for different levels of the parameter, are marked with an asterisk.

It is also possible for those outputs which appear to come from an approximately normal distribution to examine correlation. Specifically, for the six numerical parameters (i.e. excluding the boolean parameter relating to the reduction factor $r(d)$) it is possible to calculate a Pearson correlation coefficient - more detail on this can be found in section 3.2.3.3. A p-value can also be determined for this which will indicate if any of these coefficients represent a significant correlation. For those which do, we will have information additional to that provided by the Analysis of Variance which will show *how* the different parameter values influence the outputs of interest. The Pearson correlation coefficients (PCC) and their associated p-values are shown in Table 5.6. Note that as before, the results for the parameters p and q are essentially identical, save that the correlation coefficients are inverted (since p increases as q decreases and vice versa), and therefore only the results for probability p are shown here. Again, all p-values which are significant at the 5% level are marked with an asterisk.

Using the Analysis of Variance and the correlation results given here, we can make some observations about how the true values of these parameters may influence the running of the models in practice. We can look firstly at the factors which relate to crew availability, namely the absence probabilities p and q and the time reduction factor $r(d)$. As might be expected, we can see from Table 5.4 that these appear to have a significant influence on the number of changes required both for the two-minute cost-minimization approach and for the minimal-change and lowest cost solutions for the change-minimization algorithm.

Table 5.4: Results of ANOVA and Kruskal-Wallis tests for influence of parameters used in instance generation - Part 1

Outputs of interest			Test	Parameter	
				p	Use r (d)?
Cost-minimization settings	Running time	2mins	K-W	0.217	0.351
		10mins	K-W	0.283	0.137
	No. of changes	2mins (only)	F	0.004*	< 0.001*
	Proven % gap	2mins	K-W	0.701	0.430
		10mins	K-W	0.115	0.383
	Adjusted proven % gap	2mins	K-W	0.538	0.825
		10mins	K-W	0.458	0.576
	% gap to best known bound	2mins	K-W	0.341	0.231
		10mins	K-W	0.158	0.152
	Adjusted best known % gap	2mins	K-W	0.592	0.975
		10mins	K-W	0.434	0.636
Change-minimization settings	Running time		K-W	0.101	0.528
	Iterations		F	0.001*	< 0.001*
	Number of changes	Minimal-change	F	< 0.001*	0.002*
		Lowest cost sol.	F	0.002*	0.002*
	% gap to best known cost bound	Non-cost-constrained	F	0.280	0.393
		Chpst min-chng sol.	F	0.212	0.198
		Lowest cost sol.	K-W	0.187	0.481
		Adj. lowest cost	K-W	0.446	0.846
Comparison with min cost (2min setting)	% increase in number of changes	Minimal-change	F	< 0.001*	0.002*
		Lowest cost sol.	K-W	0.576	0.830
	% increase in cost	Non-cost-constrained	K-W	0.439	0.074
		Chpst min-chng sol.	K-W	0.013*	0.122
		Lowest cost sol.	K-W	0.413	0.676

Table 5.5: Results of ANOVA and Kruskal-Wallis tests for influence of parameters used in instance generation - Part 2

Outputs of interest			Test	Parameter			
				K_N	K_L	$\hat{K}$	K_{AG}
Cost-minimization settings	Running time	2mins	K-W	0.003*	< 0.001*	< 0.001*	0.008*
		10mins	K-W	0.006*	< 0.001*	< 0.001*	0.201
	No. of changes	2mins (only)	F	< 0.001*	< 0.001*	< 0.001*	0.960
	Proven % gap	2mins	K-W	0.024*	< 0.001*	< 0.001*	0.015*
		10mins	K-W	0.168	< 0.001*	0.010*	0.560
	Adjusted proven % gap	2mins	K-W	0.432	0.011*	0.104	0.056
		10mins	K-W	0.063	0.074	0.213	0.208
	% gap to best known bound	2mins	K-W	0.169	< 0.001*	0.001*	0.106
		10mins	K-W	0.227	0.001*	0.033*	0.767
	Adjusted best known % gap	2mins	K-W	0.158	0.027*	0.277	0.101
		10mins	K-W	0.049*	0.054	0.151	0.032*
Change-minimization settings	Running time		K-W	0.045*	0.001*	0.023*	< 0.001*
	Iterations		F	0.003*	0.125	0.037*	< 0.001*
	Number of changes	Minimal-change	F	0.515	0.808	0.941	0.780
		Lowest cost sol.	F	0.003*	0.007*	0.010*	0.561
	% gap to best known cost bound	Non-cost-constrained	F	0.001*	0.012*	0.005*	< 0.001*
		Chpst min-chng sol.	F	0.002*	0.029*	0.011*	< 0.001*
		Lowest cost sol.	K-W	0.010*	< 0.001*	< 0.001*	< 0.001*
		Adj. lowest cost	K-W	0.308	0.001*	0.018*	< 0.001*
Comparison with min cost (2min setting)	% increase in number of changes	Minimal-change	F	0.008*	< 0.001*	< 0.001*	0.868
		Lowest cost sol.	K-W	< 0.001*	< 0.001*	< 0.001*	0.413
	% increase in cost	Non-cost-constrained	K-W	0.002*	0.093	0.024*	< 0.001*
		Chpst min-chng sol.	K-W	0.008*	0.064	0.032*	< 0.001*
		Lowest cost sol.	K-W	0.181	0.014*	0.168	< 0.001*

Table 5.6: Correlation analysis for influence of parameters used in instance generation

Outputs of interest				Parameter				
				p	K_N	K_L	$\hat{K}$	K_{AG}
Cost-min	No. of changes	2mins (only)	PCC	-0.211	-0.397	-0.436	-0.484	-0.023
			p-value	0.001*	< 0.001*	< 0.001*	< 0.001*	0.723
Change-min	Iterations		PCC	-0.236	-0.222	-0.153	-0.204	-0.413
			p-value	< 0.001*	0.001*	0.018*	0.002*	< 0.001*
	No. of changes	Minimal-change	PCC	-0.580	-0.072	-0.036	-0.057	-0.005
			p-value	< 0.001*	0.263	0.574	0.382	0.938
		Lowest cost sol.	PCC	-0.227	-0.239	-0.175	-0.227	-0.075
			p-value	< 0.001*	< 0.001*	0.007*	< 0.001*	0.246
	% gap to best known cost bound	Non-cost-constrained	PCC	-0.096	-0.260	-0.204	-0.257	-0.443
			p-value	0.137	< 0.001*	0.001*	< 0.001*	< 0.001*
		Chpst min-chng sol.	PCC	0.111	-0.243	-0.179	-0.231	-0.411
			p-value	0.086	< 0.001*	0.006*	< 0.001*	< 0.001*
Comparison	% incr in no. of changes	Minimal-change	PCC	-0.376	0.215	0.300	0.309	0.011
			p-value	< 0.001*	0.001*	< 0.001*	< 0.001*	0.862

Specifically, from Table 5.6 we see that there is a significant negative correlation between all three of these change measures and the availability probability p, meaning that for higher values of p (and therefore lower values of q, i.e. longer but fewer absences) we expect to see a decrease in the number of changes required in the respective solutions. Intuitively this is sensible, as is the apparent influence of the reduction factor $r(d)$, with investigation showing that on average more changes are required for instances where the reduction factor is not used (i.e. the probabilities remain constant throughout) than when it is. These averages are as follows:

- For (two-minute) cost-minimization: average of ≈ 55.76 changes for instances using $r(d)$; average of ≈ 60.18 changes for the instances not using $r(d)$.
- For the minimal-change solution: average of ≈ 19.16 changes for instances using $r(d)$; average of ≈ 22.74 changes for the instances not using $r(d)$.
- For lowest cost change-minimization: average of ≈ 47.43 changes for instances using $r(d)$; average of ≈ 51.14 changes for the instances not using $r(d)$.

There is also a significant negative correlation between the value of p and the percentage difference in the number of changes between the two-minute cost-minimization solution and the minimal-change solution, suggesting that for higher values of p (i.e. longer but fewer absences) we would expect a greater decrease in number of changes, and therefore a greater advantage to using the cost-minimization approach if this were the case in reality. As shown in Table 5.4, there is also a significant influence by factor $r(d)$ on this measure, with investigation of the means showing an average of a $\approx 64.91\%$ decrease when $r(d)$ is used, and an average of a $\approx 61.21\%$ decrease when it is not. Note that there is no significant influence or correlation for either of these factors with respect to the other difference in changes measure, between the two-minute cost-minimization solution and the lowest cost change-minimization solution.

The parameters p, q and $r(d)$ also display a significant influence on the number of iterations carried out by the change-minimization algorithm. Table 5.6 shows there is a negative correlation with the value of p, indicating that for higher values of p (i.e. fewer, longer absences) we would expect fewer iterations to be carried out within the time limit (and potentially fewer solutions found). With regard to the reduction factor, examining the means shows that ≈ 13.23 iterations are carried out on average for instances where $r(d)$ has been used, while ≈ 15.29 are carried out on average for instances where it has not.

Meanwhile, the disruption and penalty factors K_N, K_L, $\hat{K}$ and K_{AG}, i.e. the factors relating to costs, can be seen in both Table 5.5 and Table 5.6 to have a considerably wider influence. This is especially true of the three parameters relating to the disruption penalty for the regular crew - near-term disruption parameter K_N, long-term disruption parameter K_L, and the weighted average of these $\hat{K}$. It can be seen in Table 5.6 that these disruption penalties have no correlation with the number of changes in the minimal-change solution (which is to be expected, as cost has no bearing at that stage of the solution algorithm), but that there is significant correlation with the number of changes required in the two-minute cost-minimization and lowest cost change-minimization solutions, as well as with the comparison between the cost-minimization and minimal-change solutions. For both the cost-minimization and change-minimization final solutions, the correlations are negative which indicates that the number of changes will be lower for higher values of the disruption factor (whether K_N, K_L or $\hat{K}$). Consistent with this is the positive correlation which exists between the disruption factor values and the percentage difference in number of changes, which indicates that there is a smaller decrease in the number of changes from the cost-minimization result to the minimal-change result (i.e. the cost-minimization number of changes is nearer to the the minimal number of changes) when the disruption factor values are higher.

As well as giving significant correlations, these outputs also have significant F-test p-values with respect to the disruption factors, as given in Table 5.5. This table also shows that the factors K_N, K_L and $\hat{K}$ all have a significant influence on the comparison between the changes in the two-minute cost-minimization solution and the lowest-cost change-minimization solution. This could not be analysed for correlation because the normality assumption did not hold, but examination of the histograms (see appendix B.2) showed that higher values of the disruption factor tended to give instances which had a smaller percentage gap between the number of changes. Note here that the agency penalty factor K_{AG} appears to have no significant influence or correlation with any of the change-related outputs.

The table of correlations (Table 5.6) also shows a significant relationship between all four penalty factors and the number of iterations performed by the change-minimization algorithm. Curiously, the result of the F-test to determine whether the long-term disruption factor K_L has an influence on the number of iterations is *not* significant (see Table 5.5) even though the correlation coefficient is. All the penalty parameters have a negative correlation, indicating that for higher values of the penalty factor there will be fewer iterations (and consequently fewer solutions) achieved within the time limit.

With regard to costs, we can see that all four of the penalty parameters have a significant correlation with the percentage gap to the best cost bound for both the non-cost-constrained and the cheapest minimal-change change-minimization solutions (see Table 5.6). That the influence of these parameter levels also produces a significant result from the Kruskall-Wallis test for the lowest cost change-minimization using both the true values and the adjusted values, with the exception of near-term penalty value K_N which does not appear to have a significant impact on the adjusted percentage gap. The correlations shown for the gaps for both the non-cost-constrained and the cheapest minimal-change solutions are all negative, indicating that for higher values of the penalty factors we would expect a smaller gap to the best known bound for the problem, i.e. it will be solved to a value closer to the optimum. No correlation test was available for the gap to the lowest cost change-minimization solution, but examination of the histograms (see appendix B.2) showed a similar general pattern for the disruption factors K_N, K_L and $\hat{K}$ with higher values being morel likely to result in optimal solutions and lower values more likely to result in solutions with larger gaps to the best known bound. The pattern for the agency penalty K_{AG} (see Figure B.23 in Appendix B.2) was a little different, with instances with lower values seemingly more likely to solve to optimal rather if they were solved to within a 5% gap, while instances with higher values were more likely to solve to within a 5% gap overall but less likely to result in an optimal solution.

There were no further cost-related outputs tested for correlation with these factors; however, as can be seen in Table 5.5 there are a number of other significant interactions between the penalty values and cost-related outputs according to the results of the Kruskall-Wallis tests. Looking first at the gap to the best bound for the cost-minimization approach, we see that each of the four penalty factors have a significant effect on either the adjusted or unadjusted gaps for the ten-minute settings; while only the long-term penalty K_L and combined penalty $\hat{K}$ have any kind of influence on the gaps for the two-minute settings. Looking at the histograms for these (see appendix B.2), a pattern can be seen. For parameters K_N and K_{AG}, the graphs suggest that higher values of these parameters are more likely to produce instances which will solve to within 5% of optimal in the ten-minute settings. Meanwhile for parameters K_L and $\hat{K}$, for both the two- and ten-minute settings, we see that higher values are more likely to produce instances which will solve to optimal if they solve to within 5%, but that lower values of these penalty factors are more likely to see solutions solving to within 5% in general.

A similar pattern can be seen for the gaps which are able to be proven during the two-minute and ten-minute cost-minimization test runs. In this case, it is for the two-minute settings that all four of the penalty parameters have a significant influence on at least one of the adjusted and unadjusted gaps; while for the ten-minute settings only the long-term penalty K_L and combined penalty $\hat{K}$ have any kind of influence on the proven gaps. Looking at the histograms for these (see appendix B.2), it would appear that for parameters K_N and K_{AG} higher values produce instances which will more likely solve to optimal if they solve to within 5%, but that lower values of these penalty factors are more likely to see solutions solving to within 5% in general. For parameters K_L and $\hat{K}$, it similarly appears that for both the two- and ten-minute settings the higher values of the parameters are more likely to solve to optimal rather than just to within 5% but that lower values are more likely to be solved to within 5% in general.

As well as the costs for the two separate approaches, we can also see from Table 5.5 that there are some significant interactions between the four penalty parameters and the results of cost comparisons between the two approaches. Histograms for the relevant breakdowns of results can be seen in appendix section B.2. Only the agency penalty factor K_{AG} has a significant influence on all three of the cost comparisons (the increase in cost from the two-minute cost-minimization solution to the non-cost-constrained solution, the cheapest minimal-change solution, and the lowest cost change-minimization solution), with the histograms showing a pattern suggesting that higher values of K_{AG} give instances which show a smaller percentage increase in cost, and in particular that no additional penalty (i.e. setting $K_{AG} = 1$) leads to much larger differences in all three of the cost comparisons. The near-term and combined disruption penalties, K_N and $\hat{K}$ show a similar influence on the non-cost-constrained and cheapest minimal-change solution comparisons, with larger values tending to give instances which can be solved to values much closer to the two-minute cost-minimization solution. This can also be seen for the long-term disruption penalty K_L when comparing the two-minute cost-minimization and lowest cost change-minimization solutions.

The final group of Kruskal-Wallis test results to discuss from Table 5.5 are those relating to running time, which can be seen to be subject to a significant interaction from all four of the penalty factors, with the exception of the agency penalty factor K_{AG} which does not appear to have a significant impact on the running time for the ten-minute cost-minimization settings. Examination of the histograms for these (see appendix B.2) shows a general pattern that for higher values of the penalty factors, instances are likely to terminate more quickly under both the two- and ten-minute cost-minimization settings as well as with the change-minimization algorithm.

5.5.2.4 Summarising the effect of parameter values

To summarise the discussion on the effect of parameter values above (section 5.5.2.3), we can say a number of things about how the suggested approaches may operate in practice depending on the true values of our uncertain parameters. Firstly, looking at the factors which affect crew availability, we have seen that if in reality crew absences tend to be longer but less frequent, then we would expect to see fewer changes being found in the two-minute cost-minimization and in the change-minimization solutions, and a greater gap between the non-cost-constrained solution and the cost-minimization solution in terms of number of changes. In this case, we would also expect a smaller number of iterations to be required in the change-minimization algorithm. A similar pattern might be observed if the *time reduction* of the absence probability proved to be a reasonable assumption in reality.

Meanwhile, the factors which relate to costs influence considerably more of the output statistics. If the company wishes to apply disruption factors in order to penalise making changes to the schedule, then in general we could expect the effects to be the following: shorter running times in both approaches, and fewer iterations for the change-minimization approach; fewer changes in the two-minute cost-minimization and lowest cost change-minimization solutions; a greater percentage decrease in changes for both minimal-change and lowest cost change-minimization solutions as compared to the cost-minimization solution; and smaller cost gaps both for the individual cost- and change-minimization approaches and when comparing the two. The additional penalty for using agency crew K_{AG} has a similar influence with regard to running time, iterations and gaps with respect to cost, but has no significant effects on the number of changes. It should be noted that those instances for which the $K_{AG} = 1$ (i.e. there is no additional penalty for agency crew) displayed particularly long running times and particularly large percentage gaps with respect to cost.

5.6 Practical Implications of Results

As noted during this chapter, the assumptions upon which the Task-Based version of the model is based are not realistic for our case study company; however, these assumptions of pre-determined regular working patterns ('tasks') and homogeneous contractual restrictions may well hold at another company. The results given in section 5.5.2 allow us to suggest how, in such a situation, our solution methods for the Task-Based formulation of the Vessel Crew Scheduling problem might be applied in practice.

The proposal here involves the implementation of a decision support tool which would be used as part of an iterative process, with solutions being proposed which can be checked for acceptability by the Planners. When judging acceptability, Planners may take into account the optimality gap, both in percentage terms and real terms, and the number of changes that the solution entails. They may also consider constraints which are not recorded explicitly by the company, such as a desire not to change the assignments for a particular employee or to avoid having two or more particular employees assigned to the same vessel at the same time, but which will be implicitly known by the Planners. Our proposed structure for the iterative process can be seen in Figure 5.13.

As discussed in section 5.5.2.2, the cost-minimization approach will usually produce the cheapest solutions; however, the change-minimization approach will produce solutions which cause less disruption to the current schedule, and will be able to propose multiple solutions to the Planners. Therefore we propose that the first stage in the solution process should be to solve the problem using the two-minute change-minimization settings. The Planner would then be presented with a Pareto-optimal set (in terms of cost and number

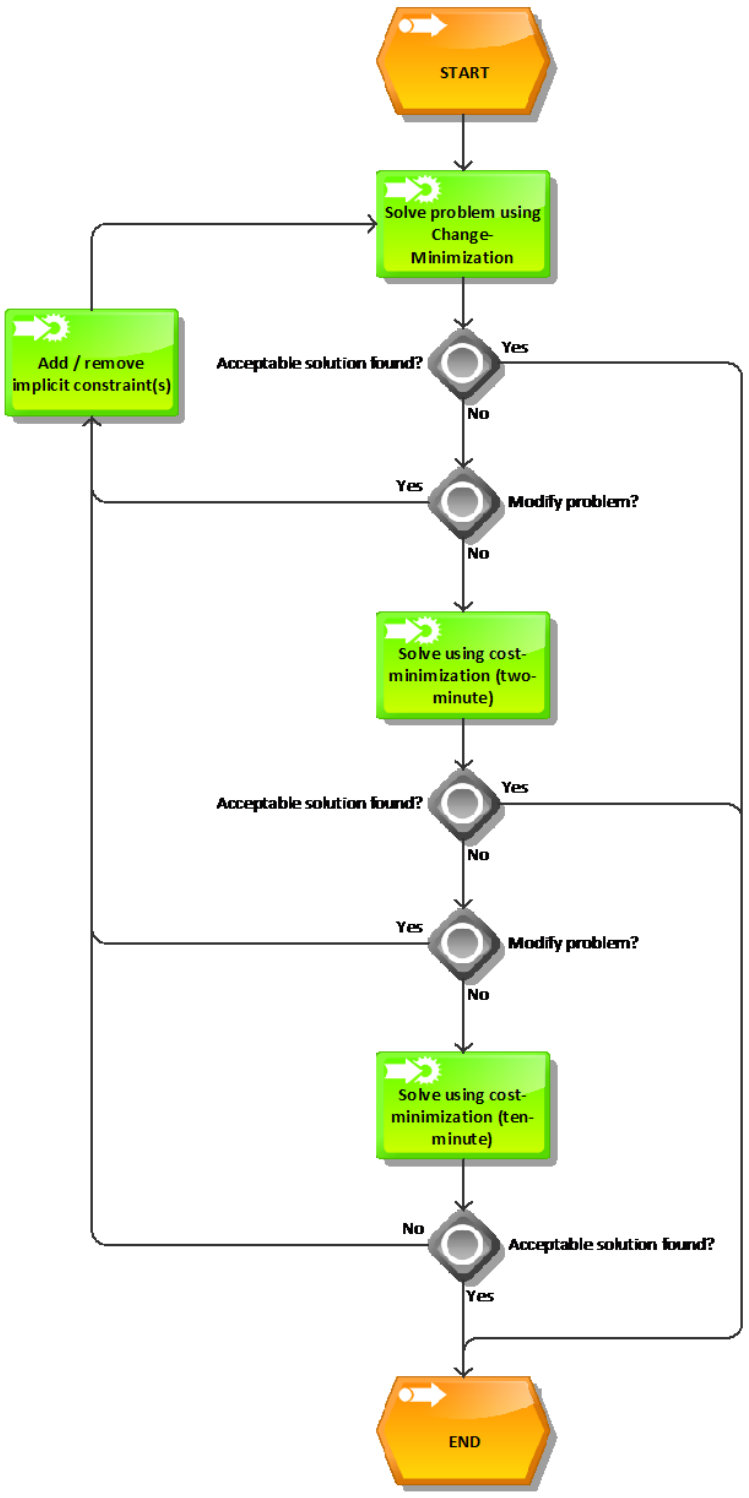

START
Solve problem using Change-Minimization
Acceptable solution found?
Yes
No
Add / remove implicit constraint(s)
Modify problem?
Yes
No
Solve using cost-minimization (two-minute)
Acceptable solution found?
Yes
No
Modify problem?
Yes
No
Solve using cost-minimization (ten-minute)
Acceptable solution found?
No
Yes
END

Figure 5.13: Flowchart showing the process for using the proposed planning tool for the Task-Based problem.

of changes, as illustrated in Figure 5.4) of solutions, including the minimal-change solution, and has a number of options as to how to proceed. The most straightforward option is to accept one of the proposed solutions, in which case the process can terminate. Whether or not this is a desirable option will depend on whether the implicit constraints are satisfied, whether the cost of the solution is acceptable (note that at this stage, no bound will yet have been generated on the solution cost), and whether the Planner has time to carry out further computations.

If the implicit constraints have not been satisfied, the Planner can add these and re-run the change-minimization process. The implicit constraints can either be treated as additional 'hard' constraints which must be satisfied, or a penalty can be introduced for violating them - the most suitable way of handling these can be discussed with a company if this is being implemented in practice. Alternatively, if there are concerns over cost, the cost-minimization programme can be called upon, firstly with the two-minute settings and subsequently if necessary and if time permits using the ten-minute settings. This would have the benefit of generating a bound on the solution cost, and therefore have a means of judging the quality of the solutions with regard to cost.

Note that we recommend that if implicit constraints are added (or, at a subsequent stage are thought to be too restrictive and removed), the process should revert to the change-minimization solution algorithm. This is because changing the implicit constraints essentially creates a new problem, and so the benefits of the change-minimization such as generating multiple solutions should be made use of. The process can carry on as long as necessary, or as long as time permits, to allow the Planner to find an acceptable solution. It may be the case that the Planner experiments with adding implicit constraints but finds that no improvement can be made on a solution found earlier in the process - if this happens at any time, the Planner can of course terminate the process.

5.6.1 Further Work

There are a number of future research steps which could be taken based on the work in this chapter. Not least of these would be to implement in practice the decision support tool which has been proposed above. This will required additional software engineering skills which were not available when this research was carried out, in order to design a suitable user interface and integrate the programme with the company systems. Clearly it will also be necessary to identify a suitable company at which to implement this model, since as we have highlighted it is based on overly-simplifying assumptions about the problem faced by our current client company.

Another piece of further research arises from the discussion of the change-minimization algorithm in section 5.5.2.2. Here it was noted that it may be the case that the method presented in Algorithm 5.2 does not cope well with negative or near-zero costs. This arises from the fact that the sequential reduction of costs is carried out as a percentage, meaning that while the cost is small positive value the decrease of the cost limit will become smaller and smaller. A possible solution to this would be to introduce an additional condition to the cost limit calculation such that once the cost is reduced to a certain value there is a fixed amount by which it is reduced, allowing the reduction to continue and a negative-cost solution to potentially be found. Further investigation would have to be carried out to determine the size of this fixed amount and the point at which it should come into force.

Finally, a clear next step arising from the work on the Task-Based problem is to extend it such that it becomes more relevant to the problem faced by our client company. Discussion of research into this can be seen in Chapter 6 which follows.

Chapter 6

Time-Windows Model

The previous chapter discussed the simplified *Task-Based* formulation of the Vessel Crew Scheduling Problem, which made use of several assumptions. These assumptions however were not realistic for the problem at our company (although they may hold, and therefore be useful, for the scheduling problem at another company). Specifically, there were the assumptions that all roles could be divided up into pre-determined task lengths, and that all employees were allowed to work a pattern which was consistent with these pre-determined tasks. In reality, crew work to various different contracts with their own maximum working and minimum resting periods specified, meaning they cannot all fit to the same pattern of work. In addition, these fixed work patterns were not realistic as it did not account for the flexibility which is available to the Planners in practice, whereby they may wish to ask an employee to work one additional week on board a vessel - increasing the assignment beyond the usual length, but still within their maximum legal or contractual limit.

Removing these assumptions which were required for the *Task-Based* formulation of the problem allows us to create a more complex but more realistic formulation which we call the *Time-Windows* version of the problem. Here, we make use of the principle that crew changes happen on a weekly basis, and we consider crew roles as being divided into one-week *windows*. This means that rather than being assigned a single task and then resting before their next assignment, crew will be assigned multiple windows in the same role consecutively, subject to their contractual obligations.

As with Chapter 5, we begin this chapter by setting out the formulations for this problem, firstly with the 'basic' formulation which assumes there is no partial schedule at the start of the planning process before extending this to the 'Recovery-type' formulation. We then go on to discuss generation of data for this problem and our initial computational results using similar methods to those used for the Task-Based problem. However, due to the increased size and complexity of the Time-Windows problem it will be seen that these results are not particularly promising in terms of what can be used in practice; we therefore discuss alternative solution methods and examine whether some of these may be more useful for solving this more realistic problem.

6.1 Problem Formulation

As indicated at the beginning of this chapter, we will start by describing the formulation of the Time-Windows problem. Firstly, in section 6.1.1 we set out a description of the so-called 'basic' scheduling problem which assumes there is no partial schedule and that all planning is done from scratch. This gives a stepping stone to the more realistic 'recovery-type' formulation which is set out in section 6.1.2.

6.1.1 Basic Problem Formulation

As with the Task-Based formulation, we begin by setting out a 'basic' scheduling formulation for the problem which assumes all scheduling is done from scratch. This allows us to set out the key aspects of the model in a simpler manner, before extending to the recovery-type problem in section 6.1.2 below. We begin by defining the sets, data and decision variables which are necessary here. Note that some of these are identical to those given in section 5.1.1 for the Task-Based problem; however, some of the notation has altered and has a different meaning to that used for the Task-Based problem.

Sets

The following sets and indices are used in the Basic Time-Windows model:

t $\in \{1, \dots, T\}$ is the index for the week-long time windows in the planning horizon of length T weeks.

E_R is the set of all regular employees, with $|E_R| = m$.

E $= E_R \cup \{m+1\}$ is the set of all employees, where $m+1$ is the index used to denote agency employees.

G $\subseteq E_R$ is the set of fixed contract employees.

K is the set of vessels which are being planned for during the planning period.

J is the set of all roles being planned for over the planning period, with $|J| = n$.

V_k $\subseteq J$ is the set of roles being planned for which take place on board vessel $k \in K$, with $\bigcup\limits_{\forall k} V_k = J$ and with $V_k \cap V_{k'} = \emptyset$ for $k' \neq k$.

λ is an index used when counting an employee's number of consecutive working weeks.

Data

We define the data items required by the model as follows:

Ω_i is the estimated number of weeks that employee $i \in G$ will work this financial year *outwith* the current planning horizon.

g_i is the number of guaranteed weeks specified by employee i's contract, $i \in G$.

c_i^U is the effective rate of pay per week for employee $i \in G$, which can be considered an *under-time* rate in the event that the employee works less than g_i weeks in the year.

c_i^O is the weekly rate at which overtime is paid to employee $i \in G$, in the event that the employee works more than g_i weeks in the year.

c_{ikt}^B is the cost of employee i boarding vessel k prior to working on board in period t.

c_{ikt}^D is the cost of employee i departing vessel k prior to period t (i.e. having completed an assignment on board in period $t-1$).

c_{ijt}^W is the cost *directly* associated with employee i carrying out role j in period t (i.e. ignoring travel costs, under-/over-time payments, etc.).

$c_{\lambda ijt}^L$ is the additional cost (either actual cost, or a theoretical *penalty* cost) of employee i carrying out role j for at least a λ^{th} consecutive week in period t.

a_{jt} is an indicator of whether role j requires an employee assigned to it in period t, such that

$$a_{jt} = \begin{cases} 1 & \text{if role } j \text{ requires to be covered in period } t \\ 0 & \text{otherwise} \end{cases}$$

e_{ijt} is an indicator of whether an employee is eligible and available to carry out role j in period t;

$$e_{ijt} = \begin{cases} 1 & \text{if employee } i \text{ can be assigned to role } j \text{ in period } t \\ 0 & \text{otherwise} \end{cases}$$

w_i^{max} is the legal / contractual upper limit on the number of consecutive weeks at sea to which employee $i \in E_R$ can be assigned.

α_j^{max} is the legal / contractual upper limit on the number of consecutive weeks that an individual agency employee can be assigned to role j.

ρ_i is the minimum length (in weeks) of the rest period to which employee $i \in E_R$ is entitled following an assignment offshore.

s_{ik} is used to indicate the location of employee $i \in E_R$ at the start of the planning period, such that

$$s_{ik} = \begin{cases} 1 & \text{if employee } i \text{ is aboard vessel } k \text{ immediately prior to period 1} \\ 0 & \text{otherwise} \end{cases}$$

σ_j is used to indicate the assignments of agency employees at the start of the planning period, such that

$$\sigma_j = \begin{cases} 1 & \text{if an agency employee is assigned to role } j \text{ immediately prior to period 1} \\ 0 & \text{otherwise} \end{cases}$$

$s_{m+1,k}$ is the *number* of agency employees who are on board vessel k immediately prior to period 1. Note that this means $s_{m+1,k} = \sum\limits_{j \in V_k} \sigma_j$ for all vessels $k \in K$.

w_{i0} is the number of consecutive weeks work offshore that employee $i \in E_R$ has been assigned prior to period 1.

α_{j0} is the number of consecutive weeks an agency employee has been assigned to role j prior to period t.

r_{i0} is the number of consecutive weeks without offshore work to which employee $i \in E_R$ is entitled prior to period 1.

Decision variables

There are a number of decision variables which are required for the Time-Windows problem. These are as follows:

x_{ijt} is the main decision variable, indicating each employee's assignment, such that

$$x_{ijt} = \begin{cases} 1 & \text{if employee } i \text{ is allocated to role } j \text{ during time period } t \\ 0 & \text{otherwise} \end{cases}$$

u_i is the number of weeks short of their guaranteed amount that employee $i \in G$ is expected to work during the year.

o_i is the number of weeks over their guaranteed amount that employee $i \in G$ is expected to work during the year.

b_{ikt} indicates employee $i \in E_R$ boarding a vessel, such that

$$b_{ikt} = \begin{cases} 1 & \begin{array}{l}\text{if employee } i \text{ boards vessel } k \\ \text{prior to an assignment in period } t\end{array} \\ 0 & \text{otherwise} \end{cases}$$

$b_{m+1,kt}$ is the *number* of agency employees which board vessel k prior to performing an assignment in period t.

β_{jt} indicates agency crew boarding a vessel to carry out a specific role j, such that

$$\beta_{jt} = \begin{cases} 1 & \text{if an agency employee begins to carry out role } j \text{ in period } t \\ 0 & \text{otherwise} \end{cases}$$

d_{ikt} indicates employee $i \in E_R$ departing from a vessel, such that

$$d_{ikt} = \begin{cases} 1 & \begin{array}{l}\text{if employee } i \text{ departs vessel } k \text{ prior to period } t \\ \text{(i.e. having completed duties there in period } t-1\text{)}\end{array} \\ 0 & \text{otherwise} \end{cases}$$

$d_{m+1,kt}$ is the *number* of agency employees which depart vessel k prior to period t, i.e. having completed duties on board in period $t-1$.

δ_{jt} indicates agency crew departing after carrying out a specific role j, such that

$$\delta_{jt} = \begin{cases} 1 & \begin{array}{l}\text{if an agency employee has finished carrying out role } j \text{ prior to period } t \\ \text{(i.e finishes carrying out the role in period } t-1\text{)}\end{array} \\ 0 & \text{otherwise} \end{cases}$$

w_{it} is the number of consecutive weeks of offshore work that employee $i \in E_R$ has been assigned up to and including week t.

α_{jt} is the number of consecutive weeks that an agency employee has been assigned to role j up to an including week t.

r_{it} is number of consecutive weeks without offshore work to which employee $i \in E_R$ is entitled *after* week t; for example, if $r_{it} = 2$ this means that employee i is entitled to weeks $t+1$ and $t+2$ without any offshore work.

$l_{\lambda ijt}$ indicates if employee i is working at least a λ^{th} consecutive week in role j;

$$l_{\lambda ijt} = \begin{cases} 1 & \text{if the carrying out of role } j \text{ in period } t \text{ is at least} \\ & \text{the } \lambda^{\text{th}} \text{ consecutive working week for employee } i \\ 0 & \text{otherwise} \end{cases}$$

Formulation

Using these quantities defined above, we can now give our basic Time-Windows formulation the crew scheduling problem as follows:

$$\min \sum_{\forall i,k,t} \left(c^B_{ikt} b_{ikt} + c^D_{ikt} d_{ikt}\right) + \sum_{\forall i,j,t} \left(c^W_{ijt} x_{ijt} + \sum_{\forall \lambda} c^L_{\lambda ijt} l_{\lambda ijt}\right) + \sum_{i \in G} \left(c^U_i u_i + c^O_i o_i\right) \tag{6.1}$$

subject to:

$$\sum_{\forall i \in E} e_{ijt} x_{ijt} = a_{jt} \qquad \forall j, t \tag{6.2}$$

$$\sum_{j \in J} x_{ijt} \leq 1 \qquad \forall t, i \in E_R \tag{6.3}$$

$$b_{ik1} \geq \sum_{j \in V_k} x_{ij1} - s_{ik} \qquad \forall k, i \in E_R \tag{6.4}$$

$$b_{ikt} \geq \sum_{j \in V_k} x_{ijt} - \sum_{j \in V_k} x_{ij,t-1} \qquad \forall k, i \in E_R, t \in \{2, \ldots, T\} \tag{6.5}$$

$$d_{ik1} \geq s_{ik} - \sum_{j \in V_k} x_{ij1} \qquad \forall k, i \in E_R \tag{6.6}$$

$$d_{ikt} \geq \sum_{j \in V_k} x_{ij,t-1} - \sum_{j \in V_k} x_{ijt} \qquad \forall k, i \in E_R, t \in \{2, \ldots, T\} \tag{6.7}$$

$$\beta_{j1} - \delta_{j1} = x_{m+1,j1} - \sigma_j \qquad \forall j \tag{6.8}$$

$$\beta_{jt} - \delta_{jt} = x_{m+1,jt} - x_{m+1,j,t-1} \qquad \forall j, t \in \{2, \ldots, T\} \tag{6.9}$$

$$b_{m+1,kt} \geq \sum_{j \in V_k} \beta_{jt} \qquad \forall k, t \tag{6.10}$$

$$d_{m+1,kt} \geq \sum_{j \in V_k} \delta_{jt} \qquad \forall k, t \tag{6.11}$$

$$u_i \geq g_i - \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} x_{ijt}\right) \qquad \forall i \in G \tag{6.12}$$

$$o_i \geq \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} x_{ijt}\right) - g_i \qquad \forall i \in G \tag{6.13}$$

$$w_{it} \geq w_{i,t-1} + \sum_{j \in J} x_{ijt} - w_i^{max} \left(1 - \sum_{j \in J} x_{ijt} \right) \qquad \forall t, i \in E_R \qquad (6.14)$$

$$w_i^{max} l_{\lambda ijt} \geq w_{i,t-1} - w_i^{max}(1 - x_{ijt}) + x_{ijt} - (\lambda - 1) \qquad \begin{array}{c} \forall j, t, i \in E_R, \\ \lambda \in \{1, \ldots, w_i^{max}\} \end{array} \qquad (6.15)$$

$$\alpha_{jt} \geq \alpha_{j,t-1} + x_{m+1,jt} - \alpha_j^{max} \delta_{jt} \qquad \forall j, t \qquad (6.16)$$

$$\alpha_{jt} \geq x_{m+1,jt} \qquad \forall j, t \qquad (6.17)$$

$$\alpha_j^{max} l_{\lambda,m+1,jt} \geq \alpha_{jt} - (\lambda - 1) \qquad \forall j, t, \lambda \in \left\{1, \ldots, \alpha_j^{max}\right\} \qquad (6.18)$$

$$r_{it} \geq r_{i,t-1} - \left(1 - \sum_{j \in J} x_{ijt} \right) \qquad \forall t, i \in E_R \qquad (6.19)$$

$$r_{it} \geq (\rho_i - 1) \sum_{k \in K} d_{ikt} \qquad \forall t, i \in E_R \qquad (6.20)$$

$$\rho_i \left(1 - \sum_{j \in J} x_{ijt} \right) \geq r_{i,t-1} \qquad \forall t, i \in E_R \qquad (6.21)$$

$$x_{ijt} \in \{0, 1\} \qquad \forall i, j, t \qquad (6.22)$$

$$l_{\lambda ijt} \in \{0, 1\} \qquad \forall i, j, t, \lambda \in \{1, \ldots, w_i^{max}\} \qquad (6.23)$$

$$b_{ikt}, d_{ikt} \in \{0, 1\} \qquad \forall k, t, i \in E_R \qquad (6.24)$$

$$\beta_{jt}, \delta_{jt} \in \{0, 1\} \qquad \forall j, t \qquad (6.25)$$

$$b_{m+1,kt}, d_{m+1,kt} \geq 0 \text{ and integer} \qquad \forall k, t \qquad (6.26)$$

$$u_i, o_i \geq 0 \qquad \forall i \in G \qquad (6.27)$$

$$w_{it}, r_{it} \geq 0 \qquad \forall i, t \qquad (6.28)$$

$$\alpha_{jt} \geq 0 \qquad \forall j, t \qquad (6.29)$$

The objective, given by equation (6.1) is to minimize the total cost of all assignments, including travel costs and over- and under-utilization of fixed contract crew. In a similar way as constraints (5.3) and (5.4) for the Task-Based problem, constraints (6.2) and (6.3) ensure respectively that all roles are covered when they are required to be, and that employees cannot be assigned to two different roles at the same time. Note that as before, the multiple assignment constraint does not apply to the agency employee $m + 1$, as we assume that multiple agency crew are available if required in a given time period.

Constraints (6.4) and (6.5) ensure correct definition of the 'boarding' variables b_{ikt} for the regular employees $i \in E_R$ by linking them to the changes in an employees working location from one time period to the next, while constraint sets (6.6) and (6.7) perform the corresponding role for the departure decision variables. The board and depart variables for agency crew are handled jointly by constraints (6.8) and (6.9), which allows us to ensure that if agency crew are assigned to a role in two consecutive weeks but should be carried out by two different people (e.g. because of working period length restrictions), then the appropriate boarding and departing costs are incurred for this crew change. Inequalities (6.10) and (6.11) define the number of crew respectively boarding and departing the vessel based on the boarding and departing for the individual roles. Note that these constraints have no equivalent in the Task-Based problem, since there the boarding of and departure from vessels was contained implicitly within each task.

The subsequent constraints ensure that legal and contractual obligations are met, starting with inequalities (6.12) and (6.13) which in the same way as (5.10) and (5.11) for the Task-Based problem deal with the respective estimated number of weeks under- or over-utilized each fixed contract employee will be in the year. In place of constraints (5.6) and (5.7) in the Task-Based problem, constraint (6.14) calculates the amount of consecutive work done by each employee from one period to the next. Constraint set (6.15) meanwhile determines how many consecutive weeks an employee has spent at sea, in order to (potentially) apply an appropriate penalty cost. The parameter w_i^{max} on the left hand side of this inequality acts as a *big-M* to ensure that cumulative working length w_{it} never exceeds its legal limit for any employee. Similarly, constraint set (6.16) calculates the amount of consecutive work done by the agency employees from one period to the next, but on a role-by-role basis, while inequality (6.17) is required to ensure the tracking variable α_{jt} takes the correct value in the case when a new agency employee joins the role immediately following the departure one. The number of consecutive weeks worked by an agency employee is then linked to the possible penalty costs by constraints (6.18).

In place of (5.8) and (5.9) in the Task-Based formulation, obligations with regard to rest periods are dealt with by inequalities (6.19), calculating the decrease in rest time required by an employee if they do not work in that period, and (6.20) resetting the rest time requirement whenever an employee leaves a vessel, ensuring the legal amount of resting time after finishing a spell of offshore work is respected. Constraint (6.21) ensures that an employee cannot work in a period if it is indicated in the previous period that they were still due some rest time. Finally, expressions (6.22-6.29) ensure the correct definition of all variables.

Note that no equivalent of the experience constraint (5.5) from the Task-Based formulation appears here. The constraint was dropped in order to simplify the problem, and because it was found that it was too difficult to quantify experience mathematically. Instead, it will now be treated as one of the 'implicit' constraints discussed in Chapter 5, which can be applied by Planners using their personal knowledge of their employees when choosing between proposed solutions.

6.1.2 Recovery-type Formulation

As before, the basic formulation given above could used in a situation where planning is done in advance with no need for modifications once assignments have been made. However, we now require to extend this formulation to the recovery-type problem, to account for the need in our case to change and update the existing schedule in response to new information on a regular basis. In order to do this, we must first give some additional definitions, before the formulation is given below.

The existing schedule

Firstly, in a similar way to that described in Section 5.2, we must consider the partial schedule which is known at the start of the planning step, having been decided in the previous planning step. This can be described by the final values of the decision variables discussed in section 6.1.1, denoted as: x^*_{ijt}, u^*_i, o^*_i, b^*_{ikt}, β^*_{jt}, d^*_{ikt}, δ^*_{jt}, w^*_{it}, α^*_{jt}, r^*_{it} and $l^*_{\lambda ijt}$. These are now input data for the current planning phase, and will be used to help evaluate the changes which are required to be made to the schedule.

Decision variables

We next need to define a corresponding new set of decision variables which will represent the new schedule which is to be created in the current week. We denote these as: $\hat{x}_{ijt}$, $\hat{u}_i$, $\hat{o}_i$, $\hat{b}_{ikt}$, $\hat{\beta}_{jt}$, $\hat{d}_{ikt}$, $\hat{\delta}_{jt}$, $\hat{w}_{it}$, $\hat{\alpha}_{jt}$, $\hat{r}_{it}$ and $\hat{l}_{\lambda ijt}$. Clearly, the new schedule must be feasible according to these new variables.

While a new feasible schedule is the main goal for this problem, what is of interest to the planners is the changes which must be made in order to make the transition to this new schedule. We will must therefore define additional decision variables to represent these changes. Taking for example the main assignment variable for the new schedule, $\hat{x}_{ijt}$, we can define a new variable $x^{\pm}_{ijt}$ such that

$$x^{\pm}_{ijt} = \begin{cases} 1 & \text{if there is a change to employee } i\text{'s schedule in week } t \text{ with respect to role } j \\ 0 & \text{otherwise} \end{cases}$$

We can use this to link the previous week's schedule, represented by x^*_{ijt}, and the new schedule, represented by $\hat{x}_{ijt}$, as follows:

$$x^{\pm}_{ijt} = \begin{cases} \hat{x}_{ijt} & \text{if } x^*_{ijt} = 0 \\ x_{ijt} - \hat{x}_{ijt} & \text{if } x^*_{ijt} = 1 \end{cases}$$

A more compact (but mathematically equivalent) way of expressing this is as

$$\hat{x}_{ijt} = x^*_{ijt} + (1 - 2x^*_{ijt})x^{\pm}_{ijt} \quad \forall i, j, t \tag{6.30}$$

Similarly, we can also define decision variables to represent changes to the other factors as follows: $b^{\pm}_{ikt}$, $\beta^{\pm}_{jt}$, $d^{\pm}_{ikt}$, $\delta^{\pm}_{jt}$ and $l^{\pm}_{\lambda ijt}$. These can be linked to their respective corresponding previous schedule values and new schedule variables using the following expressions:

$$\hat{b}_{ikt} = b^*_{ikt} + (1 - 2b^*_{ikt})b^{\pm}_{ikt} \quad \forall k, t, i \in E_R \tag{6.31}$$

$$\hat{\beta}_{jt} = \beta^*_{jt} + (1 - 2\beta^*_{jt})\beta^{\pm}_{jt} \quad \forall j, t \tag{6.32}$$

$$\hat{d}_{ikt} = d^*_{ikt} + (1 - 2d^*_{ikt})d^{\pm}_{ikt} \quad \forall k, t, i \in E_R \tag{6.33}$$

$$\hat{\delta}_{jt} = \delta^*_{jt} + (1 - 2\delta^*_{jt})\delta^{\pm}_{jt} \quad \forall j, t \tag{6.34}$$

$$\hat{l}_{\lambda ijt} = l^*_{\lambda ijt} + (1 - 2l^*_{\lambda ijt})l^{\pm}_{\lambda ijt} \quad \forall \lambda, i, j, t \tag{6.35}$$

Since under-time and over-time are not binary variables, the changes to these can be calculated differently. For these, we define change variables $u^{\pm}_i$ and $o^{\pm}_i$ which will be linked to the previous and new schedule values using the following expressions:

$$u^{\pm}_i = \hat{u}_i - u^*_i \quad i \in G \tag{6.36}$$

$$o^{\pm}_i = \hat{o}_i - o^*_i \quad i \in G \tag{6.37}$$

Note that by these definitions, $u^{\pm}_i$ and $o^{\pm}_i$ could take negative values if there is a *decrease* in under- or over-time.

Note that as there is no cost directly associated with the consecutive working periods (for regular and agency crew) or resting periods (for regular crew only), and no arrangements to be made directly concerning these, we do not track the changes. Therefore there is no need to define or to use the corresponding variables $w^{\pm}_{it}$, $\alpha^{\pm}_{jt}$ and $r^{\pm}_{it}$.

Cost coefficients

As with the recovery-type Task-Based problem previously (see section 5.2.1), the focus for our more realistic recovery-type Time-Windows formulation must be on the cost of changing the assignments of the crew. As before, these costs may comprise the purely financial cost (or saving) of making a change, such as wage payments or transportation costs, or may also include penalty costs in order to reflect the undesirable nature of some options (e.g. changing an assignment at very short notice).

In general, we can say that the following cost elements (whether real or incorporating an intangible element) will be required in the cost calculation of our recovery-type model:

ϕ^W_{ijt} is the cost of changing whether or not employee $i \in E$ works in role j in period t.

ϕ^B_{ikt} is the cost of changing whether or not employee $i \in E_R$ boards vessel k in advance of period t.

ϕ^D_{ikt} is the cost of changing whether or not employee $i \in E_R$ leaves vessel k in advance of period t.

$\phi^{B_A}_{jt}$ is the cost of changing whether or not an agency employee must board in order to carry out role j in period t.

$\phi^{D_A}_{jt}$ is the cost of changing whether or not an agency employee will depart a vessel having carried out role j in period $t-1$.

$\phi^L_{\lambda ijt}$ is the cost of changing whether or not employee $i \in E$ works for at least a λ^{th} consecutive week by carrying out role j in period t.

Formulation

Using the quantities defined above, and with other quantities remaining as defined in section 6.1.1 earlier, we can formulate the recovery-type Time-Windows problem as follows:

$$\begin{aligned}\min \sum_{i\in E_R}\sum_{\forall k,t}\left(\phi^B_{ikt}b^{\pm}_{ikt}+\phi^D_{ikt}d^{\pm}_{ikt}\right)+\sum_{\forall j,t}\left(\phi^{B_A}_{jt}\beta^{\pm}_{jt}+\phi^{D_A}_{jt}\delta^{\pm}_{jt}\right)\\ +\sum_{\forall i,j,t}\left(\phi^W_{ijt}x^{\pm}_{ijt}+\sum_{\forall\lambda}\phi^L_{\lambda ijt}l^{\pm}_{\lambda ijt}\right)+\sum_{i\in G}\left(c^U_i u^{\pm}_i+c^O_i o^{\pm}_i\right)\end{aligned} \tag{6.38}$$

subject to:

$$\sum_{\forall i\in E} e_{ijt}\hat{x}_{ijt}=a_{jt} \qquad \forall j,t \tag{6.39}$$

$$\sum_{j\in J}\hat{x}_{ijt}\le 1 \qquad \forall t, i\in E_R \tag{6.40}$$

$$\hat{b}_{ik1}\ge\sum_{j\in V_k}\hat{x}_{ij1}-s_{ik} \qquad \forall k, i\in E_R \tag{6.41}$$

$$\hat{b}_{ikt}\ge\sum_{j\in V_k}\hat{x}_{ijt}-\sum_{j\in V_k}\hat{x}_{ij,t-1} \qquad \forall k, i\in E_R, t\in\{2,\dots,T\} \tag{6.42}$$

$$\hat{d}_{ik1}\ge s_{ik}-\sum_{j\in V_k}\hat{x}_{ij1} \qquad \forall k, i\in E_R \tag{6.43}$$

$$\hat{d}_{ikt}\ge\sum_{j\in V_k}\hat{x}_{ij,t-1}-\sum_{j\in V_k}\hat{x}_{ijt} \qquad \forall k, i\in E_R, t\in\{2,\dots,T\} \tag{6.44}$$

$$\hat{\beta}_{j1}-\hat{\delta}_{j1}=\hat{x}_{m+1,j1}-\sigma_j \qquad \forall j \tag{6.45}$$

$$\hat{\beta}_{jt}-\hat{\delta}_{jt}=\hat{x}_{m+1,jt}-\hat{x}_{m+1,j,t-1} \qquad \forall j, t\in\{2,\dots,T\} \tag{6.46}$$

$$\hat{u}_i \geq g_i - \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt}\right) \qquad \forall i \in G \tag{6.47}$$

$$\hat{o}_i \geq \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt}\right) - g_i \qquad \forall i \in G \tag{6.48}$$

$$\hat{w}_{it} \geq \hat{w}_{i,t-1} + \sum_{j \in J} \hat{x}_{ijt} - w_i^{max} \left(1 - \sum_{j \in J} \hat{x}_{ijt}\right) \qquad \forall t, i \in E_R \tag{6.49}$$

$$w_i^{max} \hat{l}_{\lambda ijt} \geq \hat{w}_{i,t-1} - w_i^{max}(1 - \hat{x}_{ijt}) + \hat{x}_{ijt} - (\lambda - 1) \qquad \forall j, t, i \in E_R, \lambda \in \{1, \ldots, w_i^{max}\} \tag{6.50}$$

$$\hat{\alpha}_{jt} \geq \hat{\alpha}_{j,t-1} + \hat{x}_{m+1,jt} - \alpha_j^{max} \hat{\delta}_{jt} \qquad \forall j, t \tag{6.51}$$

$$\hat{\alpha}_{jt} \geq \hat{x}_{m+1,jt} \qquad \forall j, t \tag{6.52}$$

$$\alpha_j^{max} \hat{l}_{\lambda,m+1,jt} \geq \hat{\alpha}_{jt} - (\lambda - 1) \qquad \forall j, t, \lambda \in \left\{1, \ldots, \alpha_j^{max}\right\} \tag{6.53}$$

$$\hat{r}_{it} \geq \hat{r}_{i,t-1} - \left(1 - \sum_{j \in J} \hat{x}_{ijt}\right) \qquad \forall t, i \in E_R \tag{6.54}$$

$$\hat{r}_{it} \geq (\rho_i - 1) \sum_{k \in K} \hat{d}_{ikt} \qquad \forall t, i \in E_R \tag{6.55}$$

$$\rho_i \left(1 - \sum_{j \in J} \hat{x}_{ijt}\right) \geq \hat{r}_{i,t-1} \qquad \forall t, i \in E_R \tag{6.56}$$

$$\hat{x}_{ijt} = x^*_{ijt} + (1 - 2x^*_{ijt}) x^{\pm}_{ijt} \qquad \forall i, j, t \tag{6.57}$$

$$\hat{b}_{ikt} = b^*_{ikt} + (1 - 2b^*_{ikt}) b^{\pm}_{ikt} \qquad \forall k, t, i \in E_R \tag{6.58}$$

$$\hat{d}_{ikt} = d^*_{ikt} + (1 - 2d^*_{ikt}) d^{\pm}_{ikt} \qquad \forall k, t, i \in E_R \tag{6.59}$$

$$\hat{\beta}_{jt} = \beta^*_{jt} + (1 - 2\beta^*_{jt}) \beta^{\pm}_{jt} \qquad \forall j, t \tag{6.60}$$

$$\hat{\delta}_{jt} = \delta^*_{jt} + (1 - 2\delta^*_{jt}) \delta^{\pm}_{jt} \qquad \forall j, t \tag{6.61}$$

$$\hat{l}_{\lambda ijt} = l^*_{\lambda ijt} + (1 - 2l^*_{\lambda ijt}) l^{\pm}_{\lambda ijt} \qquad \forall i, j, t, \lambda \in \{1, \ldots, w_i^{max}\} \tag{6.62}$$

$$u_i^{\pm} = \hat{u}_i - u_i^* \qquad i \in G \tag{6.63}$$

$$o_i^{\pm} = \hat{o}_i - o_i^* \qquad i \in G \tag{6.64}$$

$$\hat{x}_{ijt}, x^{\pm}_{ijt} \in \{0, 1\} \qquad \forall i, j, t \tag{6.65}$$

$$\hat{l}_{\lambda ijt}, l^{\pm}_{\lambda ijt} \in \{0, 1\} \qquad \forall i, j, t, \lambda \in \{1, \ldots, w_i^{max}\} \tag{6.66}$$

$$\hat{b}_{ikt}, b^{\pm}_{ikt}, \hat{d}_{ikt}, d^{\pm}_{ikt} \in \{0, 1\} \qquad \forall k, t, i \in E_R \tag{6.67}$$

$$\hat{\beta}_{jt}, \beta^{\pm}_{jt}, \hat{\delta}_{jt}, \delta^{\pm}_{jt} \in \{0, 1\} \qquad \forall j, t \tag{6.68}$$

$$\hat{u}_i, \hat{o}_i \geq 0 \qquad \forall i \in G \tag{6.69}$$

$$\hat{w}_{it}, \hat{r}_{it} \geq 0 \qquad \forall i, t \tag{6.70}$$

$$\hat{\alpha}_{jt} \geq 0 \qquad \forall j, t \qquad (6.71)$$

$$u_i^{\pm}, o_i^{\pm} \text{ are unrestricted} \qquad \forall i \in G \qquad (6.72)$$

Here, the objective (6.38) states that the total cost of all *changes* should be minimized. Note that the agency boarding and departing variables for individual tasks ($\beta_{jt}^{\pm}$ and $\delta_{jt}^{\pm}$) are used rather than those counting the total number of agency crew boarding and departing as in (6.1) previously. This is to account for the possibility that an agency employee is added to one role on a vessel, but an agency assignment is cancelled for another role on the vessel. This would give a net change for the vessel of zero, and hence this step must be taken to ensure that the costs associated with each change are still included in the solution.

Meanwhile, constraints (6.39-6.56) fulfil essentially the same purpose as constraints (6.2-6.21) in the basic formulation, ensuring the feasibility of the schedule and linking the primary assignment decision variables with the boarding, departing, and work and rest resource variables. The difference in this formulation is that we now use the new schedule variables (with the $\hat{\bullet}$ notation) to replace the basic scheduling variables. Notice however that because the boarding and departing of agency crew is now dealt with on a role-by-role basis in the objective, there is no need for an equivalent to constraints (6.10) and (6.11).

As described earlier, this formulation introduces equations (6.57-6.64) to deal with the linking between the values representing the current schedule (denoted by $\bullet^*$), and the sets of decision variables representing the new schedule (denoted by $\hat{\bullet}$) and the differences between the two (denoted by $\bullet^{\pm}$). Finally, (6.65-6.72) ensure the correct definition of all variables, as (6.22-6.29) did for the basic Time-Windows formulation.

6.1.3 A Change-minimization Formulation

As with the Task-Based problem, we will also generate an alternative approach to the cost-minimization problem with the objective to minimize the number of changes in the solution. In this case, we will consider a ‘change’ to take place when there is a change directly to the schedule of employee $i \in E$ at week t with respect to role j - we will not count changes to the other aspects such as boarding, departing and working durations, as these will only change as a consequence of the assignment changes.

Similarly to the Task-Based problem, this could be achieved simply by setting $\phi_{ijt}^{W} = 1$ for all $i \in E$, $j \in J$ and for all t in the planning horizon, and by setting all other cost coefficients $\phi_{ikt}^{B} = \phi_{ikt}^{D} = \phi_{jt}^{B_A} = \phi_{jt}^{D_A} = \phi_{\lambda ijt}^{L} = c_i^U = c_i^O = 0$ for the formulation presented in section 6.1.2 above. However, we again want this formulation to be more useful in practical terms and therefore propose to continue to use the true values of the cost coefficients, thereby allowing the setting of an upper limit Λ on the cost of the changes. As before, we note that there is no guarantee that a solution with a smaller number of changes will also have a lower cost.

We will first discuss the various modifications which must be made to the recovery-type formulation, before presenting the change-minimization formulation.

Objective function

The objective is now to minimize the number of changes, which as stated above takes into account only the direct changes to the assignment and not the knock-on effects to transport arrangements or working durations. We therefore state the new objective function as:

$$\min \sum_{\forall i,j,t} x^{\pm}_{ijt} \tag{6.73}$$

This is, as discussed above, the equivalent of setting $\phi^W_{ijt} = 1$ for all $i \in E$, $j \in J$ and for all t in the planning horizon, and by setting all other cost coefficients $\phi^B_{ikt} = \phi^D_{ikt} = \phi^{B_A}_{jt} = \phi^{D_A}_{jt} = \phi^L_{\lambda ijt} = c^U_i = c^O_i = 0$ in the cost-minimization objective (6.38).

Cost limit

We can then modify the cost-minimizing objective to give a constraint which places an upper limit Λ on the cost:

$$\begin{aligned} &\sum_{i\in E_R}\sum_{\forall k,t}\left(\phi^B_{ikt}b^{\pm}_{ikt} + \phi^D_{ikt}d^{\pm}_{ikt}\right) + \sum_{\forall j,t}\left(\phi^{B_A}_{jt}\beta^{\pm}_{jt} + \phi^{D_A}_{jt}\delta^{\pm}_{jt}\right) \\ &+ \sum_{\forall i,j,t}\left(\phi^W_{ijt}x^{\pm}_{ijt} + \sum_{\forall\lambda}\phi^L_{\lambda ijt}l^{\pm}_{\lambda ijt}\right) + \sum_{i\in G}\left(c^U_i u^{\pm}_i + c^O_i o^{\pm}_i\right) \quad \le \Lambda \end{aligned} \tag{6.74}$$

As with the Task-Based problem, there will be cases where a sensible upper limit will not be known, or may not be desired when minimizing the number of changes is the only concern. We can therefore if required set Λ equal to the highest possible value of the cost function as follows:

$$\begin{aligned} \Lambda = \quad &\sum_{\forall i,j,t}\left(\left|\phi^W_{ijt}\right| + \sum_{\forall\lambda}\left|\phi^L_{\lambda ijt}\right|\right) + \sum_{i\in E_R}\sum_{\forall k,t}\left(\left|\phi^B_{ikt}\right| + \left|\phi^D_{ikt}\right|\right) \\ &+ \sum_{\forall j,t}\left(\left|\phi^{B_A}_{jt}\right| + \left|\phi^{D_A}_{jt}\right|\right) + \sum_{i\in G}\left(366\left(\left|c^U_i\right| + \left|c^O_i\right|\right)\right) \end{aligned} \tag{6.75}$$

Additional constraints

As with the Task-Based formulation, the need for additional constraints arises from the need to ensure that undertime and overtime variables $\hat{u}_i$ and $\hat{o}_i$ for $i \in G$ take the correct values. We must therefore introduce new constraints to supplement inequalities (6.47) and

(6.48). To do this, we introduce a new variable ψ_i, defined for all $i \in G$ as follows:

$$\psi_i = \begin{cases} 1 & \text{if the overtime value for employee } i \text{ is non-negative} \\ 0 & \text{if the undertime value for employee } i \text{ is non-negative} \\ 0 \text{ or } 1 & \text{if } \hat{u}_i = \hat{o}_i = 0 \end{cases}$$

Note that we could alternatively introduce two new variables, ψ_i^u and ψ_i^o, which would equal 1 if respectively undertime and overtime were non-negative, and equal zero otherwise. However, as before we note that by definition at most one of $\hat{u}_i$ and $\hat{o}_i$ can take a positive value for a given employee i, allowing us to use the single variable ψ_i for each employee $i \in G$ to indicate their overtime or undertime status.

Using this variable ψ_i we can supplement inequality (6.47) with the following two expressions to ensure proper definition of $\hat{u}_i$:

$$\hat{u}_i \leq g_i - \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt} \right) + M(1 - \psi_i) \quad \forall i \in G \tag{6.76}$$

$$\hat{u}_i \leq M\psi_i \quad \forall i \in G \tag{6.77}$$

and similarly use the following two expressions to supplement inequality (6.48):

$$\hat{o}_i \leq \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt} \right) - g_i + M\psi_i \quad \forall i \in G \tag{6.78}$$

$$\hat{o}_i \leq M(1 - \psi_i) \quad \forall i \in G \tag{6.79}$$

Note that in the above expressions, M is a suitably large number which is greater than or equal to the maximum possible value for the number of days of under- or over-time. Since these are calculated on an annual basis, we can set $M = 366$.

Formulation

We can now update the formulation given in section 6.1.2 with the new information given above. This gives us the following formulation for the Time-Windows problem with an objective of minimizing the number of changes:

$$\min \sum_{\forall i,j,t} x_{ijt}^{\pm} \tag{6.80}$$

subject to:

$$\begin{aligned}&\sum_{i\in E_R}\sum_{\forall k,t}\left(\phi^B_{ikt}b^{\pm}_{ikt}+\phi^D_{ikt}d^{\pm}_{ikt}\right)+\sum_{\forall j,t}\left(\phi^{B_A}_{jt}\beta^{\pm}_{jt}+\phi^{D_A}_{jt}\delta^{\pm}_{jt}\right)\\&+\sum_{\forall i,j,t}\left(\phi^W_{ijt}x^{\pm}_{ijt}+\sum_{\forall\lambda}\phi^L_{\lambda ijt}l^{\pm}_{\lambda ijt}\right)+\sum_{i\in G}\left(c^U_iu^{\pm}_i+c^O_io^{\pm}_i\right)\quad\le\Lambda\end{aligned}\tag{6.81}$$

$$\sum_{\forall i\in E}e_{ijt}\hat{x}_{ijt}=a_{jt}\qquad\forall j,t\tag{6.82}$$

$$\sum_{j\in J}\hat{x}_{ijt}\le 1\qquad\forall t,i\in E_R\tag{6.83}$$

$$\hat{b}_{ik1}\ge\sum_{j\in V_k}\hat{x}_{ij1}-s_{ik}\qquad\forall k,i\in E_R\tag{6.84}$$

$$\hat{b}_{ikt}\ge\sum_{j\in V_k}\hat{x}_{ijt}-\sum_{j\in V_k}\hat{x}_{ij,t-1}\qquad\forall k,i\in E_R,\ t\in\{2,\dots,T\}\tag{6.85}$$

$$\hat{d}_{ik1}\ge s_{ik}-\sum_{j\in V_k}\hat{x}_{ij1}\qquad\forall k,i\in E_R\tag{6.86}$$

$$\hat{d}_{ikt}\ge\sum_{j\in V_k}\hat{x}_{ij,t-1}-\sum_{j\in V_k}\hat{x}_{ijt}\qquad\forall k,i\in E_R,\ t\in\{2,\dots,T\}\tag{6.87}$$

$$\hat{\beta}_{j1}-\hat{\delta}_{j1}=\hat{x}_{m+1,j1}-\sigma_j\qquad\forall j\tag{6.88}$$

$$\hat{\beta}_{jt}-\hat{\delta}_{jt}=\hat{x}_{m+1,jt}-\hat{x}_{m+1,j,t-1}\qquad\forall j,t\in\{2,\dots,T\}\tag{6.89}$$

$$\hat{u}_i\ge g_i-\left(\Omega_i+\sum_{j\in J}\sum_{t=1}^{T}\hat{x}_{ijt}\right)\qquad\forall i\in G\tag{6.90}$$

$$\hat{u}_i\le g_i-\left(\Omega_i+\sum_{j\in J}\sum_{t=1}^{T}\hat{x}_{ijt}\right)+M\left(1-\psi_i\right)\qquad\forall i\in G\tag{6.91}$$

$$\hat{u}_i\le M\psi_i\qquad\forall i\in G\tag{6.92}$$

$$\hat{o}_i\ge\left(\Omega_i+\sum_{j\in J}\sum_{t=1}^{T}\hat{x}_{ijt}\right)-g_i\qquad\forall i\in G\tag{6.93}$$

$$\hat{o}_i\le\left(\Omega_i+\sum_{j\in J}\sum_{t=1}^{T}\hat{x}_{ijt}\right)-g_i+M\psi_i\qquad\forall i\in G\tag{6.94}$$

$$\hat{o}_i\le M\left(1-\psi_i\right)\qquad\forall i\in G\tag{6.95}$$

$$\hat{w}_{it}\ge\hat{w}_{i,t-1}+\sum_{j\in J}\hat{x}_{ijt}-w^{max}_i\left(1-\sum_{j\in J}\hat{x}_{ijt}\right)\qquad\forall t,i\in E_R\tag{6.96}$$

$$w^{max}_i\hat{l}_{\lambda ijt}\ge\hat{w}_{i,t-1}-w^{max}_i(1-\hat{x}_{ijt})+\hat{x}_{ijt}-(\lambda-1)\qquad\forall j,t,i\in E_R,\ \lambda\in\{1,\dots,w^{max}_i\}\tag{6.97}$$

$$\hat{\alpha}_{jt}\ge\hat{\alpha}_{j,t-1}+\hat{x}_{m+1,jt}-\alpha^{max}_j\hat{\delta}_{jt}\qquad\forall j,t\tag{6.98}$$

$$\hat{\alpha}_{jt} \geq \hat{x}_{m+1,jt} \qquad \forall j,t \tag{6.99}$$

$$\alpha_j^{max}\hat{l}_{\lambda,m+1,jt} \geq \hat{\alpha}_{jt} - (\lambda - 1) \qquad \forall j,t, \; \lambda \in \left\{1,\ldots,\alpha_j^{max}\right\} \tag{6.100}$$

$$\hat{r}_{it} \geq \hat{r}_{i,t-1} - \left(1 - \sum_{j\in J}\hat{x}_{ijt}\right) \qquad \forall t, i \in E_R \tag{6.101}$$

$$\hat{r}_{it} \geq (\rho_i - 1)\sum_{k\in K}\hat{d}_{ikt} \qquad \forall t, i \in E_R \tag{6.102}$$

$$\rho_i\left(1 - \sum_{j\in J}\hat{x}_{ijt}\right) \geq \hat{r}_{i,t-1} \qquad \forall t, i \in E_R \tag{6.103}$$

$$\hat{x}_{ijt} = x^*_{ijt} + (1 - 2x^*_{ijt})x^\pm_{ijt} \qquad \forall i,j,t \tag{6.104}$$

$$\hat{b}_{ikt} = b^*_{ikt} + (1 - 2b^*_{ikt})b^\pm_{ikt} \qquad \forall k,t,i \in E_R \tag{6.105}$$

$$\hat{d}_{ikt} = d^*_{ikt} + (1 - 2d^*_{ikt})d^\pm_{ikt} \qquad \forall k,t,i \in E_R \tag{6.106}$$

$$\hat{\beta}_{jt} = \beta^*_{jt} + (1 - 2\beta^*_{jt})\beta^\pm_{jt} \qquad \forall j,t \tag{6.107}$$

$$\hat{\delta}_{jt} = \delta^*_{jt} + (1 - 2\delta^*_{jt})\delta^\pm_{jt} \qquad \forall j,t \tag{6.108}$$

$$\hat{l}_{\lambda ijt} = l^*_{\lambda ijt} + (1 - 2l^*_{\lambda ijt})l^\pm_{\lambda ijt} \qquad \forall i,j,t, \; \lambda \in \{1,\ldots,w_i^{max}\} \tag{6.109}$$

$$u_i^\pm = \hat{u}_i - u_i^* \qquad i \in G \tag{6.110}$$

$$o_i^\pm = \hat{o}_i - o_i^* \qquad i \in G \tag{6.111}$$

$$\hat{x}_{ijt}, x^\pm_{ijt} \in \{0,1\} \qquad \forall i,j,t \tag{6.112}$$

$$\hat{l}_{\lambda ijt}, l^\pm_{\lambda ijt} \in \{0,1\} \qquad \forall i,j,t, \; \lambda \in \{1,\ldots,w_i^{max}\} \tag{6.113}$$

$$\hat{b}_{ikt}, b^\pm_{ikt}, \hat{d}_{ikt}, d^\pm_{ikt} \in \{0,1\} \qquad \forall k,t,i \in E_R \tag{6.114}$$

$$\hat{\beta}_{jt}, \beta^\pm_{jt}, \hat{\delta}_{jt}, \delta^\pm_{jt} \in \{0,1\} \qquad \forall j,t \tag{6.115}$$

$$\psi_i \in \{0,1\} \qquad \forall i \in G \tag{6.116}$$

$$\hat{u}_i, \hat{o}_i \geq 0 \qquad \forall i \in G \tag{6.117}$$

$$\hat{w}_{it}, \hat{r}_{it} \geq 0 \qquad \forall i,t \tag{6.118}$$

$$\hat{\alpha}_{jt} \geq 0 \qquad \forall j,t \tag{6.119}$$

$$u_i^\pm, o_i^\pm \text{ are unrestricted} \qquad \forall i \in G \tag{6.120}$$

The majority of the constraints are unchanged from the cost-minimizing formulation, with constraints (6.82 - 6.90), (6.93), (6.96 - 6.115) and (6.117 - 6.120) being identical to expressions (6.39 - 6.72) in section 6.1.2 above. As discussed however, the objective function (6.80) is now different, with the objective being to minimize changes, and the previous objective function expression has been incorporated into inequality (6.81) to allow a cost limit to be placed on the solution. Also added to the problem are constraint sets (6.91), (6.92), (6.94) and (6.95), which allow undertime and overtime values to be correctly evaluated; along with expression (6.116) which gives the correct definition of new variable ψ_i.

6.1.4 A Note on the Relative Problem Sizes

Having set out the formulation of the Time-Windows problem, it is perhaps worthwhile to discuss how the size of this problem compares to that of the Task-Based version. This discussion can be made with respect to any crew group, but for simplicity we will deal specifically with the Captains crew group, as this relates directly to the results discussed both earlier in section 5.5 and later in section 6.3.

Firstly we review a number of values required in this discussion:

- There are m regular employees to be scheduled, plus the agency employees with index $m + 1$.
- There are n roles which must be covered. Note that for the Captain crew group there will be only one role per vessel, and therefore there are also n vessels to be covered; however, the number of vessels is not important in this discussion.
- For the Task-Based model, these roles will be divided up into tasks of predetermined length. In Chapter 5 we defined the total number of tasks to be carried out as n_W; however, for clarity here we will redefine this number as ν here.
- The planning period is T weeks long.
- Each employee has a contractually specified longest working time - the maximum of these over all employees is W_i^{max}.
- Since, as noted in section 5.4, the *projects* do not relate to the Captains crew group, and since as noted in section 6.1.1 the experience constraints are ignored in the formal description of the Time-Windows model, we will disregard the number of projects and their requirements from this calculation.

Using these values, it can be seen from the formulation given in section 5.2 that the maximum number of variables in the Task-Based problem can be given by the expression

$$2m + 2\nu\,(T + 1)\,(2m + 1) \tag{6.121}$$

while the maximum number of constraints in the problem can be expressed as

$$m + \nu + 6m\nu\left(T + 1\right) \tag{6.122}$$

Clearly both of these expressions are of order $O\left(m\nu T\right)$.

Meanwhile, from the formulation given in section 6.1.2 it can be calculated that the number of variables in the Time-Windows problem is at most

$$nT + 2m\left(T + 2\right) + 2nT\left(m + 1\right)\left(W_i^{max} + 3\right) \tag{6.123}$$

while the number of constraints will be at most

$$4m + 6nT + 5mT\left(n + 1\right) + 2nTW_i^{max}\left(m + 1\right) \tag{6.124}$$

Both of these expressions are of the order $O\left(mnTW_i^{max}\right)$.

These expressions can be more readily compared if it is noted that there is a relationship between the number of roles n and the number of tasks ν. In the data generating algorithm laid out in section 5.4, we assumed that the number of tasks into which any one role can be divided depends on an estimate of the maximum working time value across all employees, which we will call W^{ave} here. In this case, we can say that the number of tasks which will be derived from a single role will be either $\left\lceil \frac{T}{W^{ave}} \right\rceil$ or $\left\lceil \frac{T}{W^{ave}} \right\rceil + 1$, meaning that ν will lie in the range

$$n\left\lceil \frac{T}{W^{ave}} \right\rceil \leq \nu \leq n\left(\left\lceil \frac{T}{W^{ave}} \right\rceil + 1\right)$$

Using this expression, we would say that the size of the Task-Based problem is of order $O\left(mn\frac{T^2}{W^{ave}}\right)$.

Considering the extreme values of W^{ave}, clearly the minimum value of the estimated maximum working time is one week. This would mean each role divided into T tasks, and give a total number of tasks $\nu = nT$. Alternatively, if $W^{ave} \geq T$ then this would make it possible for each role to comprise only a single task over the T weeks of the planning period, giving a total number of tasks $\nu = n$. Therefore, removing the unknown W^{ave} term we can say that ν will lie in the range

$$n \leq \nu \leq nT$$

As a result, we can now say that this size of the Task-Based problem is in the best case of order $O\left(mnT\right)$, and in the worst case is of order $O\left(mnT^2\right)$.

Comparing the expressions for the size of the problems, can say that the Time-Windows problem is between $O\left(\frac{W_i^{max}}{T}\right)$ and $O\left(W_i^{max}\right)$ times the size of the Task-Based problem.

Alternatively, in terms of the estimated maximum working time W^{ave} we can say that the Time-Windows problem is $O\left(\frac{W_i^{max} \times W^{ave}}{T}\right)$ times the size.

It is interesting to note from this that the size of the Time-Windows problem is not necessarily greater than that of the Task-Based problem - in cases where the value of W^{ave}, and consequently also W_i^{max}, is close to 1 and T is a standard length (i.e. around 13 weeks) then the fraction $\frac{W_i^{max}}{T}$ will be less than 1, indicating that the Time-Windows problem will be smaller than the Task-Based. However, for data which contains values of W^{ave} and W_i^{max} which are larger (and in particular, are a larger proportion of T), the Time-Windows problem will clearly be larger than the Task-Based formulation. This is of course tempered by the benefit of the Time-Windows formulation giving a more realistic representation of the problem, and therefore opening up the possibility of better solutions being found.

6.2 Generating Time-Windows Data

As with the Task-Based problem, before investigating solution methods for the formulations described in Sections 6.1.2 and 6.1.3, datasets had to be created. The datasets generated for the Task-Based model as outlined in section 5.4 were not suitable to be used directly here, since they were constructed on the assumption that all crew had the same contractual conditions, and also since the costs of work and transport were aggregated together into one single task change cost. Instead, the algorithm used to generate these datasets had to be adapted to generate suitable Time-Windows data, based on the same key parameters as before.

As with the Task-Based data, using these parameters along side some randomization, datasets were generated which which while not being strictly ‘real’ would be realistic. Here we go on to outline the procedure for doing this, as well as describing how uncertain elements were once again accounted for.

6.2.1 Procedure for Data Generation

As in section 5.4.1, we now give a step-by-step outline of the process for generating the realistic datasets for the Time-Windows formulation, while full details of the Xpress code used can be seen in the appendix in section E.2.1. As before, there was a degree of uncertainty about how some of the data elements should be generated, and this lead to assumptions having to be made. Some of these assumptions are similar to that which were made for the Task-Based data, while some are specific to the Time-Windows problem. Section 6.2.2 below gives a discussion of the assumptions in more detail, while all the assumptions made are also highlighted in bold within the relevant steps of the procedure.

Note that as before, data is only generated for a single crew group at a time.

Step 1 - Calculate number of roles:

- This is simply the sum of the number required on board each vessel (as per the data given by the company).

Step 2 - Assign vessels regular crew and standard working pattern:

- Count the number of crew assigned to the various contract types
  - *Norwegian* contracts will normally work 2 weeks on, 4 weeks off;
  - *Singapore* contracts will normally work 10 weeks on, 5 weeks off;
  - Otherwise, will be either 4-on-4-off or 5-on-5-off depending on vessel location.
- **Assume that all crew assigned to a given vessel will be of the same contract type (to ensure balance of regular back-to-back working).**
  - *Norwegian* contracts imply three employees are required to cover each role;
  - *Singapore* contracts imply three employees can cover every two roles;
  - Otherwise, two employees will be required to cover each role.
- **Assume also that there are no other constraints on which contract type will be assigned to which vessel (or which location).**
- **Assume that if there is not enough crew to cover all roles with the required number regular employees, then agency employees will be assigned as *regulars* to fill any gaps.**

Step 3 - Determine standard crew-change times and standard assignments:

- Based on the contract type for the regular crew, will know the standard duration for working in each role.
- **Assume an even spread of crew change dates.**
- **If there is more than one of a role required on board a vessel, assume that their change-over times are staggered.**
- At random, assign a length of time each role has remaining on its initial assignment at the start of the planning horizon.
- At random, assign one of the regular employees (as defined above) for each role to be in that role at the start of the planning horizon.

- From here, can work along the timeline, assigning each employee to their regular role for a period of time, with crew-change points as appropriate to the contract type.
- **Assume that these regular patterns have also been observed immediately prior to the planning horizon starting, which therefore allows a calculation of the *work resource* and *rest resource* values at time zero.**
- **Assume that assignments for the final week of the planning horizon have not yet been determined.**

Step 4 - Generate employee availabilities:

- This will be done in the same way as in section 5.4, defining quantities p and q as before and linking them through equation (5.57) - as a reminder, this states

  $$0.008 \times p + 0.992 \times q = 0.008$$

  Day-to-day availabilities can be generated as before, potentially applying the reduction factor $r(d)$ as defined in equation (5.58) as

  $$r(d) = \frac{(d-28)}{(n-28)}$$

  and illustrated in Figure 5.1. *Note that, as before, the values of the p and q are not known, and the use of the reduction factor has not been validated, and therefore these should be subject to the same variation across the datasets as for the Task-Based data.*
- Based on the day-by-day availabilities we can calculate each employee's availability for each week, since an employee is available for a week if and only if they are available over all the days of that week.

Step 5 - Generate costs of changes for regular crew:

- In this case there are four cost types. These relate to: crew boarding a vessel; crew departing a vessel; crew working on a specific role for a period of time; and crew working longer than usual in a given role (known as *extension* cost).
- With respect to boarding and departing:
  - **Assume that these costs will depend on where the crew are normally based and where the vessel is usually operating:**

        - **Crew are European, North American, Asian, Australasian or *other*.**
        - **Vessels operate in Europe, Africa, USA, Brazil, Asia-Pacific, or *other*.**
    - **Assume that if either the boarding or the departing are due to take place after the first four weeks of the horizon then cost of making the change is less severe - the estimated cost would therefore be halved.**
    - **Assume that because of the cost associated with arranging for the change to be made, any saving made by cancelling a flight will not completely cover the cost of an otherwise identical employee being booked in to take the same flight instead.**

- With respect to the working costs:
    - **Assume that if an employee is on a fixed contract and the assignment is within the next four weeks that there is a small penalty / admin cost associated with changing this. However, otherwise there is no additional cost directly associated with them working during a particular week.**
    - **Assume if the employee is not fixed contract then the cost of making the changes depends on the crew nationality - Norwegian crew will have a higher day-rate cost.**

- With respect to *extension* costs:
    - Note: this will only be relevant to crew on *other* contracts on vessels based in Europe, who would normally work four weeks on but could legally work for five.
    - **Assume that the penalty associated with an employee working a longer than usual period on board the vessel is half the working cost (i.e. equivalent to paying time-and-a-half).**

- May also wish to consider the additional penalty cost as defined previously in equation (5.59):

$$c'_{ij} = \begin{cases} K \times c^F_{ij} & \text{if } c^F_{ij} > 0 \\ c^F_{ij}/K & \text{if } c^F_{ij} < 0 \end{cases}$$

  This can be applied in exactly the same manner as described in section 5.4. *Note that again it is not clear what a sensible penalty factor should be here, and hence it should again be varied across the generated datasets.*

Step 6 - Generate costs of changes for agency crew:

- Again, this can be done in a similar way to the regular crew.
- **Assume that all agency crew will be sourced (relatively) locally and so the transportation costs will be low.**
- **Assume that extension costs will not apply to agency crew.**
- **Assume that there is no penalty factor associated with disrupting assignments, as agency crew by definition should be available at short notice - therefore the penalty costs mentioned above do not apply.**
- However, as in section 5.4 we may have a penalty arising from the undesirability of employing agency crew. This was described in equation (5.60) as

$$c'_{m+1,j} = \begin{cases} K \times c^F_{ij} & \text{if } c^F_{m+1,j} > 0 \\ c^F_{m+1,j}/K & \text{if } c^F_{m+1,j} < 0 \end{cases}$$

  Note that as before it is not clear what a sensible penalty factor should be, and therefore the value should be varied across the generated datasets.

Step 7 - Generate terms of fixed contracts:

- Under- and over-rates will depend of crew nationality / contract type, as certain Norwegian contract types are much more expensive.
- **Assume that all fixed-contract employees are contracted to work 26 weeks in the year.**
- **Assume the initial solution (before any cancellations had to be made due to absence / illness) had all fixed-contract employees fully utilised - i.e. all were set to work 26 weeks in the current contractual year.**
- Can therefore calculate the expected working time outwith the planning horizon.
- From this, can calculate the amount of working time to which crew are currently assigned, taking into account that they cannot work weeks for which they are now unavailable.
- Can therefore calculate how many weeks under or over their guaranteed number an employee is in line for under the current solution, taking into account any unavailability.

Step 8 - Write the required information into a data file:

- File heading, and an indication of the values of the parameters used (as discussed above).
- The number of weeks in the planning horizon.
- Employee details:
  - Employee labels.
  - Which of them are on fixed contracts (subset G in the formulation given in section 6.1.2).
  - Details for the fixed contract crew (i.e. under-rate, over-rate, number of guaranteed days, expected working time - c_i^U, c_i^O, g_i and Ω_i respectively in the given formulation).
  - The minimum rest and maximum working periods of the employees.
- Vessel and role details:
  - The vessel labels.
  - The roles which are to be performed on board each vessel.
  - The time windows during which each role is required.
    - **Assume all roles are required at all times.**
- The eligibility matrix, showing whether or not an employee is available each week.
  - **Assume that agency crew will be available and eligible to carry out all tasks.**
- Work and rest *resource* values at time zero (quantities $\hat{w}_{i0}$ and $\hat{r}_{i0}$ respectively in the formulation).
- The initial solution, allowing for availability data. This can be calculated by multiplying together the two binary values indicating the initial assignments and the availabilities. Clearly for agency crew, this will just be a case of printing the initial assignment value.
- The change costs, comprising:
  - The boarding change cost (the quantities ϕ_{ikt}^B for regular crew and $\phi_{jt}^{B_A}$ for agency crew in the formulation).
  - The departing change cost (quantities ϕ_{ikt}^D and $\phi_{jt}^{D_A}$).

  - The working change cost (ϕ^W_{ijt}).
  - The *extension* change cost($\phi^L_{\lambda ijt}$).

6.2.2 Dealing with Assumptions and Uncertainty

As before, there were several assumptions which could not be validated, even though they appear intuitively sensible, and several values whose true value was uncertain. The same approach was taken to tackle this, namely to generate numerous data sets with different values of these uncertain parameters. As before, four factors were identified which should be varied across datasets:

- The values of availability probabilities p and q;
- Whether or not to use the probability *reduction factor* $r(d)$;
- The values of the disruption penalty factor K; and
- The values of the agency penalty factor K_{AG}.

No reason was seen to change the values of these parameters from those which had been used to generate the Task-Based data, and therefore the following groups of values were again used:

- For the availability probabilities p and q, the following pairs of values:
  1. $p = 0.8$ and $q = 0.0016$;
  2. $p = 0.5$ and $q = 0.004$; and
  3. $p = 0.2$ and $q = 0.0065$.
- For the probability reduction factor $r(d)$, either:
  1. Reduce probabilities p and q by multiplying by $r(d)$; or
  2. Do not use reduction factor $r(d)$, and generate data using the unreduced values of p and q.
- For the disruption penalty K, this would again be broken down into K_N for disruptions in the near-term and K_L for disruptions in the longer-term, with K_N and K_L taking the following combinations of values:
  1. $K_N = 1$, $K_L = 1$;
  2. $K_N = 2$, $K_L \in \{1, 2\}$;
  3. $K_N = 5$, $K_L \in \{1, 2, 5\}$; and
  4. $K_N = 10$, $K_L \in \{1, 2, 5, 10\}$.
- For the agency penalty K_{AG}, this would take values in the set $\{1, 2, 5, 10\}$.

6.2.3 Datasets Generated

As with the Task-Based data, the method given here was first used to generate datasets for the Captain crew group. As well as being a good crew group for testing, for the reasons outlined in section 5.4.3, this will also make possible some direct comparisons between the test results for the Task-Based and Time-Windows formulations.

Datasets were generated using each of the 240 different possible combinations of the parameter values described in section 6.2.2 above.

6.3 Initial Computational Results

As noted in section 6.1.4, it was expected that the Time-Windows problem would be much larger than the Task-Based formulation for the equivalent data, and therefore be much more difficult to solve to an acceptable level of accuracy within a reasonable amount of time. However, this could not be assumed to be the case and so tests were carried out to investigate the performance on the Time-Windows model of the solution approaches used on the Task-Based formulation, as discussed in section 5.5.1.

6.3.1 Cost-minimization

As with the Task-Based data sets, the FICO Xpress software was used here, utilising its inbuilt solution methods. As described previously, these comprise solving the LP relaxation of the problem, followed by a heuristics and cutting-planes phase, and then a branch-and-cut procedure. Full details of the code used can be found in appendix section E.2.2.

Preliminary tests were again carried out, which indicated that as anticipated the problem was larger and much more difficult to solve. This was seen in the solution time for the LP relaxation, which for the selected Time-Windows data sets took around 30 to 50 seconds to run; in contrast, the LP relaxations for the Task-Based problem could be solved in under 5 seconds. This indicated that, consequently, at each cutting iteration it would also take significantly longer to evaluate the new solution value. In addition, the cuts and heuristics themselves seemed largely ineffective at producing good integer solutions or good lower bounds on the problem. As a result it was decided to run the programme on the most basic settings - default cutting strategy, and default tolerance of the optimality gap (i.e. within 0.0001%).

Figure 6.1 below shows the percentage gaps to the best known bound (from all test runs) when these settings were carried out with a two-minute time limit. It can be seen that for the majority of instances (135, or 56.25%, of the 240), the gap between the best known bound and best solution found was between 95% and 100% of the solution value; and for another 62 instances (25.83% of the total) the gap was between 90% and 95% of

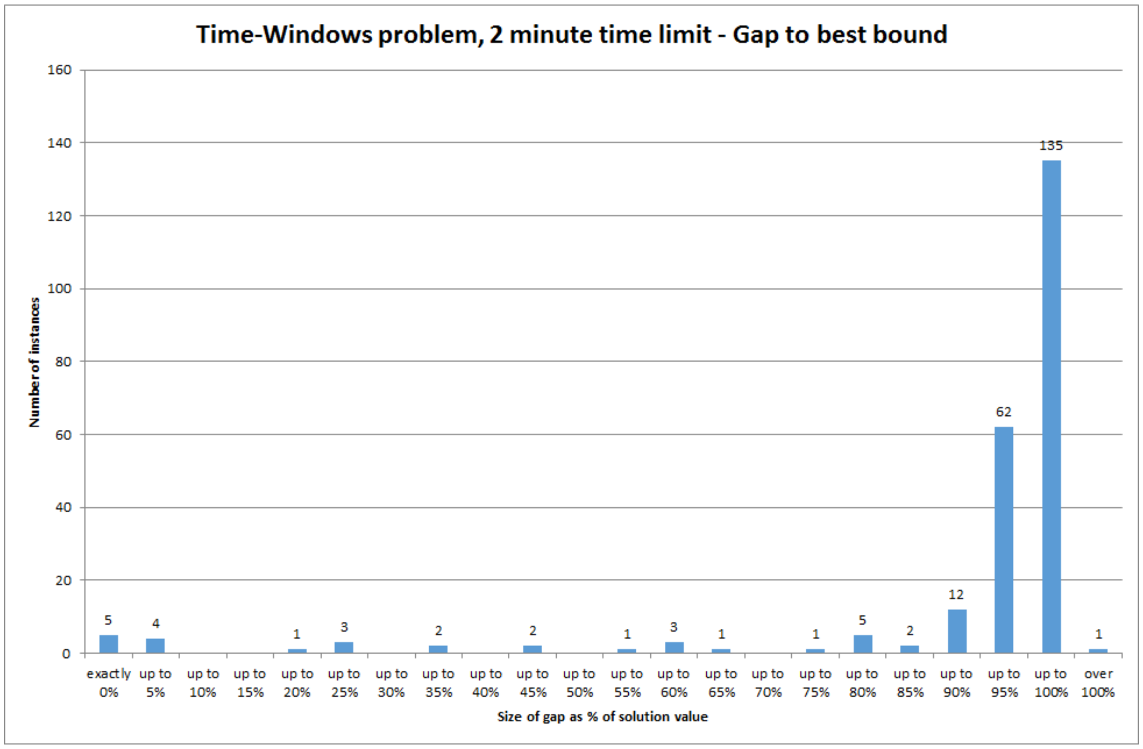


Figure 6.1: Histogram of percentage gaps to best bounds for 'direct' solution of Time-Windows instances using FICO Xpress, 2 minute time limit.

the solution value. Only 42 instances (17.5% of the total) solved even to within 90% of the best bound, while only 5 instances were solved to optimality and 4 others to within 5% of optimal. This can be compared to Figure 5.2 for the two-minute settings of the Task-Based problem. The contrast is obvious, given that the vast majority of the Task-Based instances were solved to optimality (98 instances, ≈ 40.83% of the total) or at least provably to within 5% of the bound (127 instances, ≈ 52.92% of the total) within two minutes, as shown in the upper 'Gap in run' chart of the figure. For the Task-Based problem almost all instances were solved to a good level of optimality within two minutes; for the Time-Windows problem hardly any instances could be solved to an acceptable level.

Investigation was also made into an 'extended' run, similar to that used for the Task-Based problem for particularly difficult problem instances. For this, the time limit was set to one hour (i.e. 3600 seconds); the other settings were however left at their default values. Figure 6.2 below illustrates the gaps to the best known bound (again taking into account all test runs) for the best solutions found within the one hour time limit. We see from this graph that there is a more even spread of gaps across the range, although there is still a greater occurrence of larger gaps, with the most common sizes of gap being between 90%

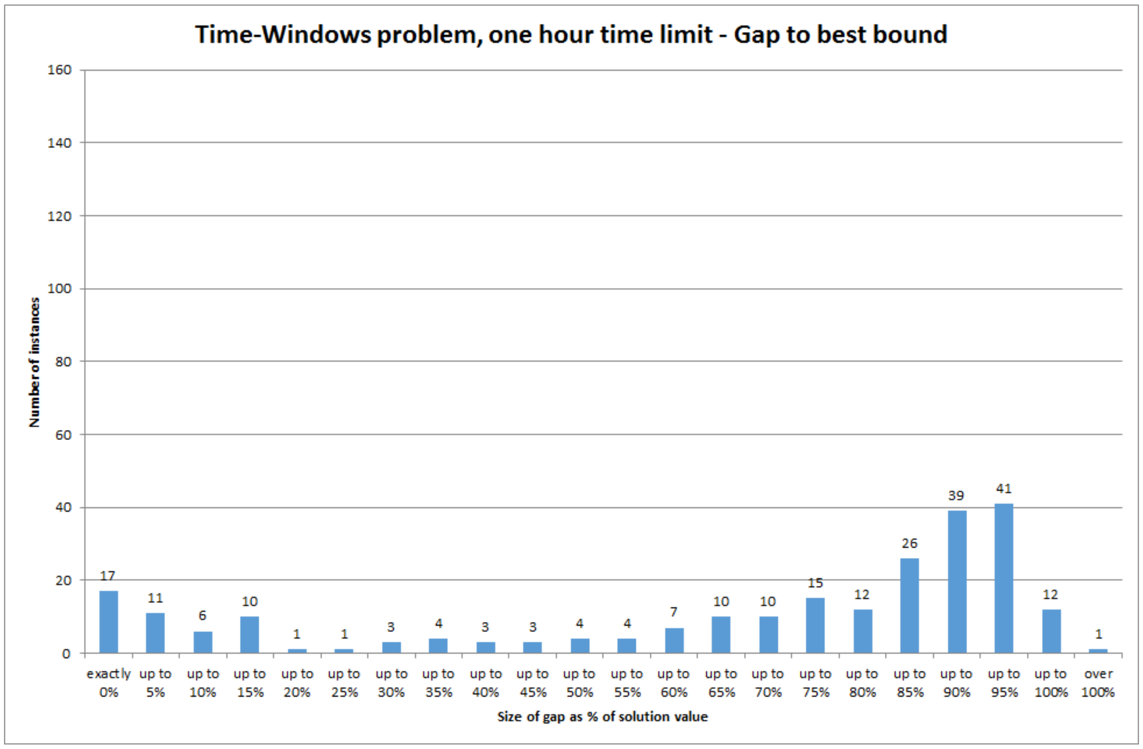


Figure 6.2: Histogram of percentage gaps to best bounds for 'direct' solution of Time-Windows instances using FICO Xpress, 1 hour time limit.

and 95% (41 instances, ≈ 17.08% of the total) and between 85% and 90% (39 instances, 16.25% of the total). It can be seen that with the one hour time limit, more instances solve to within 5% of optimal than with the two minute time limit. In Figure 6.2 we see 17 instances (≈ 7.09% of the total) are solved to optimality for the one-hour settings, while a further 11 (≈ 4.58% of instances) were solved to within 5% of the best bound; this is compared to 5 instances and 4 instances respectively with the two-minute settings.

While clearly there is an improvement in the results for the one-hour approach discussed here, it is still noticeable that the level of accuracy to which these Time-Windows instances can be solved is poor compared to the results for the Task-Based problem. Note that in light of this, it was felt that it would not be particularly informative to carry out an equivalent of the ten-minute settings that were used for the Task-Based problem. The conclusion we can draw from these results is that the 'direct' cost-minimization solution method is not suitable for solving the Time-Windows problem in practice, especially when we consider that Planners are often operating under time pressure and consequently waiting even ten minutes let alone an hour is not really feasible. Instead, it would appear that a more tailored solution method is needed to find good quality solutions in a reasonable amount of time. The subsequent sections of this chapter discuss ideas and results of other solution methods, starting with section 6.4.

6.3.2 Change-minimization

There was also a need to test the change-minimization approach devised for the Task-Based problem on the Time-Windows data sets. Preliminary tests however indicated that the procedure described in Algorithm 5.2 below could not be used here without some modifications. Firstly, while the non-cost-constrained problem solved extremely quickly for the Task-Based formulation, the equivalent for the Time-Windows problem was harder to solve and in some cases could take longer than the usual two-minute time limit to solve to optimality; therefore a time limit had to be placed on this non-cost-constrained problem where none had been used in Algorithm 5.2.

Also requiring to be altered was the way the cost limit was lowered. Because the problem was more difficult to solve, it appeared that fewer iterations could be achieved within the time limit and that therefore a greater initial cost reduction might be desirable in order to bring about a faster initial decrease in the cost. The potential issue with larger percentage reductions was that it would force the cost limit below the minimal value for the instance, making the problem infeasible, and so a more flexible way of reducing the percentage also had to be found. In order to monitor potential infeasible cost limits, this algorithm records a lower bound Λ^{min} on the cost. The algorithm used for the change-minimization approach can be seen in Algorithm 6.1, with full detail of the code implemented in FICO Xpress given in the appendix in section E.2.3.

Note that the algorithm states that the initial value of Λ^{min} should be set equal to the negative of the initial value of Λ. This can be done since the initial value of Λ is an aggregation of all the absolute values of all the costs, i.e. a value that the cost cannot exceed, and therefore the negative of this will put a simple lower bound on the solution cost. The value of this however is very likely to be a considerable distance from the true minimum possible value for the problem. We note also that the means of calculating κ is not given in the algorithm, for the purposes of simplicity. For the Time-Windows problem, this can be done in a similar way to that discussed in Algorithm 5.2 - using the left hand side values of constraint (6.81) in the Time-Windows change-minimization formulation, we can calculate the cost of the solution as

$$\begin{aligned}\kappa = \quad & \sum_{i\in E_R}\sum_{\forall k,t}\left(\phi^B_{ikt}b^{\pm}_{ikt} + \phi^D_{ikt}d^{\pm}_{ikt}\right) + \sum_{\forall j,t}\left(\phi^{B_A}_{jt}\beta^{\pm}_{jt} + \phi^{D_A}_{jt}\delta^{\pm}_{jt}\right) + \\ & \sum_{\forall i,j,t}\left(\phi^W_{ijt}x^{\pm}_{ijt} + \sum_{\forall\lambda}\phi^L_{\lambda ijt}l^{\pm}_{\lambda ijt}\right) + \sum_{i\in G}\left(c^U_i u^{\pm}_i + c^O_i o^{\pm}_i\right)\end{aligned} \tag{6.125}$$

In terms of results for this, we will firstly look at the number of solutions found by

Algorithm 6.1 Algorithm for solving the Time-Windows problem using Change-minimization

```
set iteration time limit $\tau_I = 30$ seconds, and overall time limit $\tau_T = 120$ seconds
set $\Lambda$ at maximum value (see equation (6.75)), and set initial lower bound $\Lambda^{min} = -\Lambda$
set *terminate* = false
solve the problem with cost limit $\Lambda$ and time limit $\tau_I$
**if** no feasible integer solution is found **then**
  set *terminate* = true
**else**
  calculate solution cost $\kappa$ (see equation (6.125))
**end if**
set initial proportion $\pi = 0.81$, and set $\Lambda_0 = -10$, which will be the cost limit if $\kappa = 0$
**while** *terminate* = false **do**
  **repeat**
    calculate new cost limit $\Lambda = \begin{cases} \Lambda_0 & \text{if } \kappa = 0 \\ \kappa - \lfloor \pi\kappa \rfloor & \text{otherwise} \end{cases}$
    **if** new cost limit $\Lambda < \Lambda^{min}$ **then**
      set $\pi \leftarrow \frac{2}{3} \times \pi$
    **end if**
  **until** $\Lambda \geq \Lambda^{min}$
  solve problem with cost limit $\Lambda$ and time limit $\tau_I$
  **if** problem is infeasible **then**
    set lower bound $\Lambda^{min} = \Lambda$
  **else if** no feasible integer solution is found **then**
    **if** $\pi < 0.06$ **then**
      set *terminate* = true
    **else**
      set $\pi \leftarrow \frac{2}{3} \times \pi$
      set $\Lambda_0 = -1$
    **end if**
  **else**
    set $\Lambda_0 = -10$
    calculate solution cost $\kappa$ (see equation (6.125))
  **end if**
  calculate total running time $\tau_R$
  **if** $\tau_R > \tau_T$ **then**
    set *terminate* = true
  **else if** $\tau_R > \tau_T - \tau_I$ **then**
    set $\tau_I = \lfloor \tau_T - \tau_R \rfloor + 1$
  **else**
    $\tau_I$ remains unchanged
  **end if**
**end while**
```

Algorithm 6.1 for each instance. This can be summarised in Figure 6.3 below. As can be

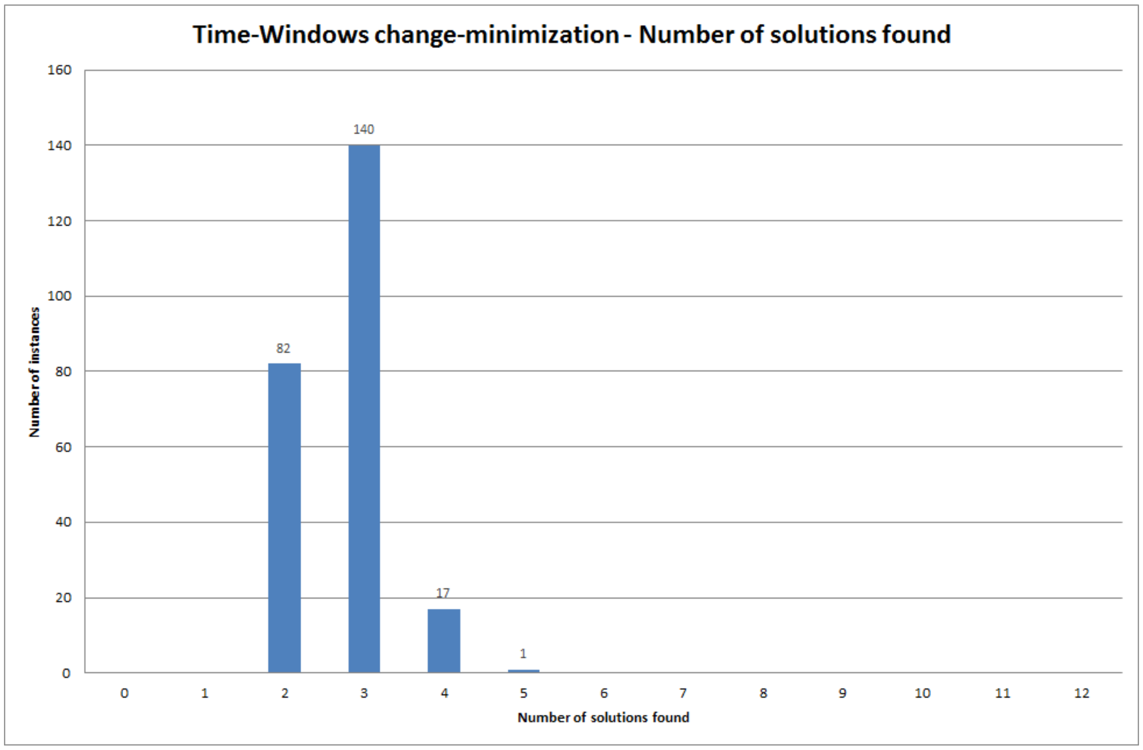


Figure 6.3: Number of solutions found by each instance using algorithm for change-minimization approach.

seen, the number of solutions found is far fewer than for the Task-Based problem, as shown in Figure 5.5. Whereas for the Task-Based algorithm, all instances were provided with at least 5 solutions, for the Time-Windows problem we see that 5 solutions the maximum number that any achieves within the two-minute time limit. Even then, there is only a single instance for which this was possible, with the majority of instances (140 of the 240, $\approx 58.33\%$) resulting in only three solutions. The mean number of solutions found was less than this, at only ≈ 2.74 solutions for the average instance. This indicates that for the Time-Windows problem, if the change-minimization procedure was used then the choice of solutions available to the Planner would be a limited one.

Connected with this, it is interesting to consider the number of iterations carried out for each instance - this is summarised in Figure 6.4. From this, it can be seen that all instances saw five iterations of the algorithm carried out, while the majority of instances (143, i.e. $\approx 59.58\%$ of the total) had 6 iterations of the algorithm. At a maximum, there were 11 iterations carried out for two of the instances, with the average being ≈ 6.62.

Clearly there is an inconsistency between the number of iterations of the algorithm

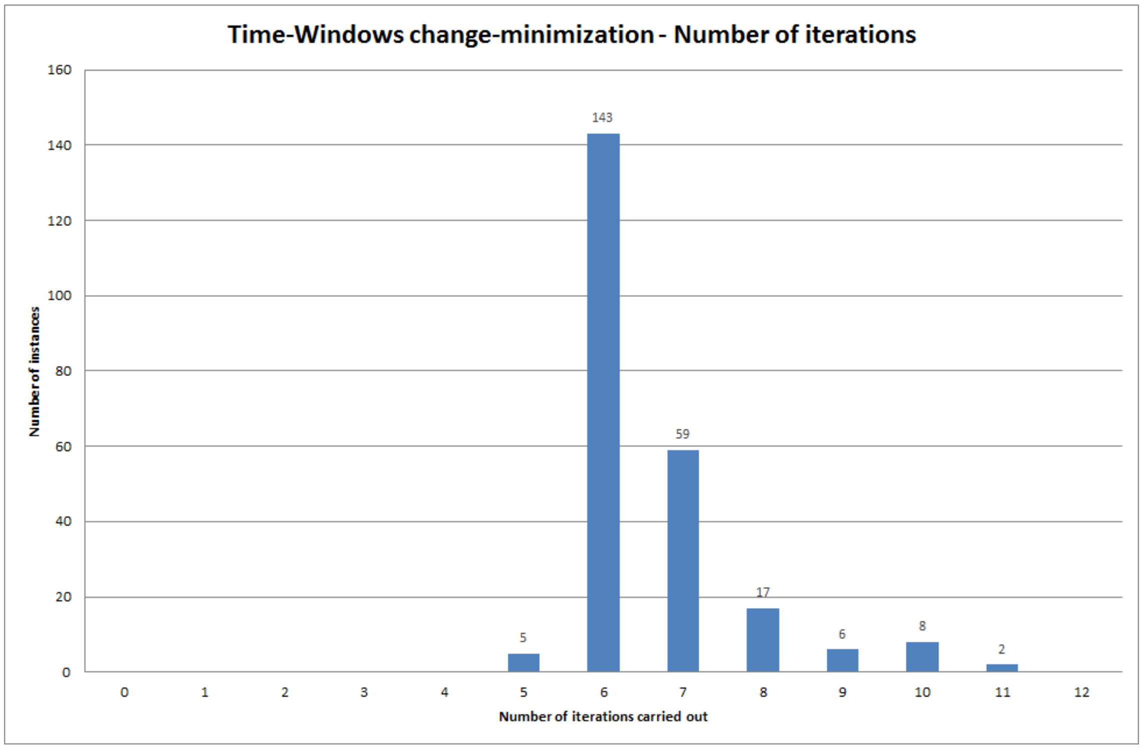


Figure 6.4: Number of iterations of change-minimization algorithm carried out for each instance.

carried out and the number of solutions found for each instance. This arises from the time restriction placed on the algorithm, which means many iterations are terminated before any integer solution can be found. A breakdown of the iterations across all 240 instances can be seen in Table 6.1 below. The table distinguishes between iterations (combined across all instances) for which an optimal solution was found, for which the problem was found to be infeasible, and for which the programme was terminated before it could be completed. For these unfinished iterations, a small number found an integer solution in the time available while the majority did not. As well as being given for all iterations, this breakdown is also given ignoring the results for the non-cost-constrained solution. This is done because for all instances, the non-cost-constrained problem solved to optimality in a matter of seconds; therefore removing these 240 'optimal' instances gives a further indication of how the algorithm performs for instances which have a cost limit set on them. The table shows that only $\approx 40.55\%$ of instances found an optimal solution, with a further $\approx 0.82\%$ finding a solution before terminating the search early. Discounting the non-cost-constrained solutions, only $\approx 30.93\%$ of iterations resulted in an integer solution, while over half the cost-constrained iterations ($\approx 50.22\%$) were terminated before any solution could be found (or infeasibility proved). This accounts for the discrepancy between the

Table 6.1: Breakdown of iterations in Change-minimization approach to Time-Windows problem

		Optimal solution found	Unfinished; integer soln found	Unfinished; no integer soln found	Problem infeasible	TOTAL
All iterations	Count	644	13	677	254	1588
	%	40.55%	0.82%	42.63%	15.99%	100%
Ignore non-cost-constr	Count	404	13	677	254	1348
	%	29.97%	0.96%	50.22%	18.84%	100%

number of iterations carried out and the number of solutions obtained for the instances.

We can also consider the gaps to the best known bound for the solutions found by the change-minimization procedure. Figure 6.5 shows a summary of these gaps (to the best bound found during the 'direct' cost-minimization test runs) for the lowest-cost solutions found by the change-minimization procedure for each instance. The chart shows quite a spread of gap sizes for these solutions, ranging from one instance for which the optimal solution was found, to another instance where the gap between best solution and best bound was over 100% (i.e. the solution value was more than double the gap value). The tendency is towards instances having gaps of over 55%, with the gaps of from 65% to 70% and from 85% to 90% having the most occurrences (26 instances, $\approx 10.83\%$ of the total, in each bracket). On average, the gap between the best bound and lowest cost solution was $\approx 59.78\%$.

It is interesting to compare this graph with the equivalent results for Algorithm 5.2 for the Task-Based problem, as shown in Figure 5.6. In that case, 198 instances (82.5% of the total) were solved to within at most 5% of the best known bound, and for 21 (8.75% of the total) of these the optimal solution was found; for the Time-Windows results given here, the numbers were 8 instances and 1 instance respectively. This again highlights the difficulty of solving the Time-Windows problem, with the change-minimization approach struggling much as the cost-minimization approach does to achieve the same level of results for this as for the Task-Based problem.

Perhaps a fairer comparison however is between the cost-minimization and change-minimization results for the Time-Windows problem. If we look at the two figures given above summarising the gaps to the best bound in the two-minute and one-hour cost-minimization runs (Figures 6.1 and 6.2 respectively), we see that the results shown in Figure 6.5 demonstrate a much better performance in two minutes by the change-minimization than by the cost-minimization approach. The comparison with the one-hour cost-minimization setting is more equal, with the change-minimization results showing fewer instances being solved to optimal or very near to optimal, but also fewer with gaps of 80% or higher.

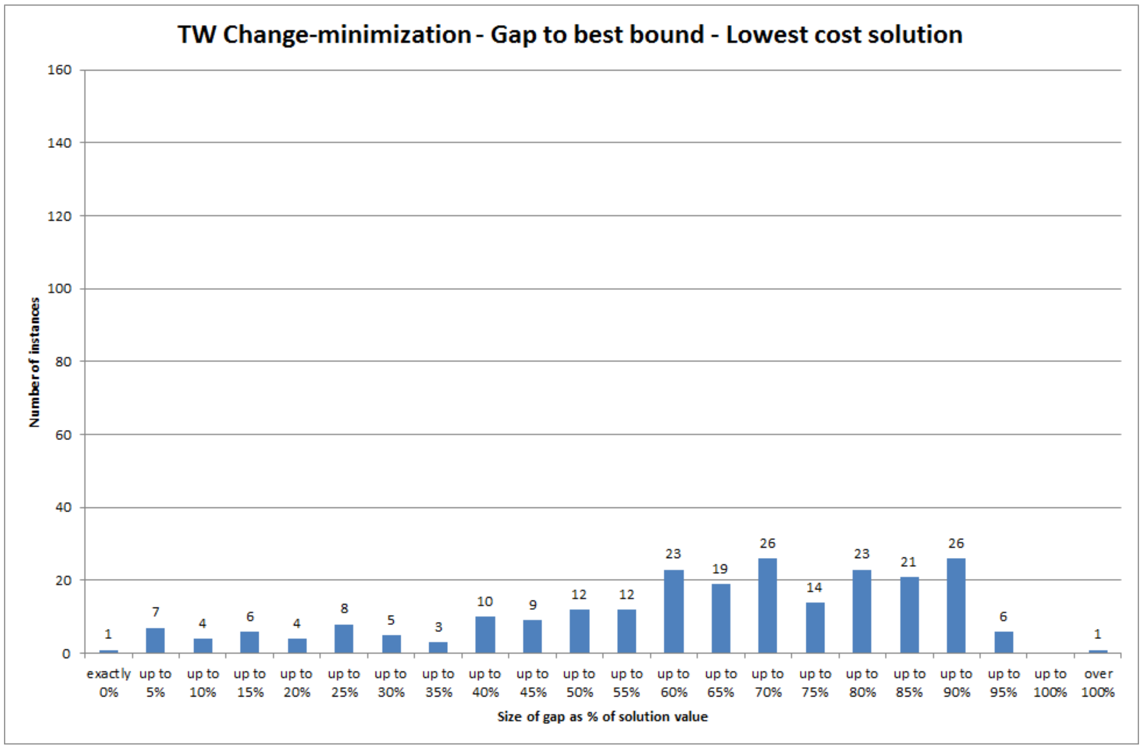


Figure 6.5: Gap to best bound for lowest-cost solutions found in two minute Change-minimization test run for the Time-Windows formulation.

Therefore, while the change-minimization approach may appear less successful for the Time-Windows problem than for the Task-Based, it is still more successful that the cost-minimization approach at producing lower cost solutions for the Time-Windows problem in a short time frame.

Another set of results which can be shown is the quality of the first, i.e. non-cost-constrained, solution which is found by the change-minimization algorithm. These results can be summarised in Figure 6.6 below. As can be seen, the size of gap is over 80% for all instances, with almost all instances falling into the categories of either between 90% and 95% (105 instances, 43.75% of the total) or between 95% and 100% (115 instances, $\approx$ 47.92% of the total) gap to the best bound. Interestingly, to an extent even this non-cost-constrained solution seems to outperform the two-minute cost-minimization approach for solving the Time-Windows problem. As can be seen in Figure 6.1, the two-minute cost-minimization approach solved 26 instances ($\approx$ 10.83% of the total) to within 80% of the best bound while the change-minimization solution had no instances achieving this; however, the 151 instances ($\approx$ 62.92%, of the 240) which resulted in a gap of between 95% and 100% is a greater number than in this category for the non-cost-constrained change-minimization solutions. While this out-performance is not particularly large, it

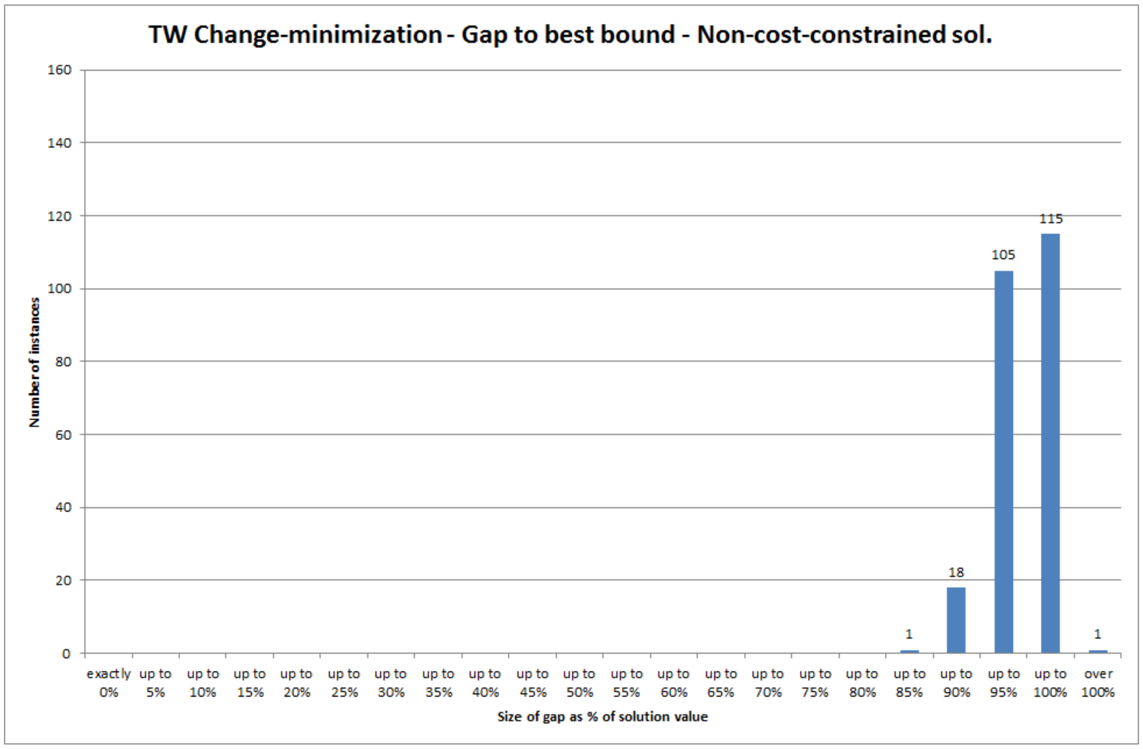


Figure 6.6: Gap to best bound for the non-cost-constrained solutions found in two minute Change-minimization test run for the Time-Windows formulation.

still demonstrates that the change-minimization approach in itself yields solutions of a reasonable quality as compared to the two-minute cost-minimization settings. One possible explanation for this when it was not observed for the Task-Based problem, is that the Time-Windows data represents a stronger link between cost and number of changes.

6.3.3 Summarising the Initial Results

To summarise the results given in sections 6.3.1 and 6.3.2 above, we have seen that neither the cost-minimization nor change-minimization approaches used for the Task-Based problems are as effective when applied to the more complex Time-Windows formulation of the problem. The change-minimization approach is more successful than the cost-minimization over a short time limit at producing lower cost solutions; however it does not produce numerous solutions to the same extent as when applied to the Task-Based problem because of the time required in some iterations to produce a solution. This would have the consequence that Planners could not be presented with a good range of solution options if using the change-minimization algorithm.

The implications of these results are that other solution methods should be investigated to find other approaches which may be able to produce a greater number of solutions, solutions of a higher quality, or solutions in a shorter time frame than those approaches used for the Task-Based problem. The remainder of this chapter discusses some alternative solution methods. Sections 6.4 to 6.7 outline possible solution approaches to the problem, with further computational results described and discussed in sections 6.8 and 6.9.

6.4 Using the Task-Based Formulation as an Approximation

The first alternative solution method considered for the Time-Windows problem involved turning attention back to the simpler Task-Based formulation. As has been discussed, the Task-Based formulation has a clear disadvantage of lacking flexibility and is a less accurate representation of the true nature of the problem than the Time-Windows formulation. However, as shown by our initial computational results in section 6.3 above it had an advantage over the Time-Windows problem of being easier and therefore quicker to solve.

This gave rise to the idea that the Task-Based formulation could be used to quickly find a solution to the Time-Windows problem. It was clear that assumptions would have to be made, in particular with respect to task lengths, in order for the Time-Windows problem data to be fed into a Task-Based formulation. Once a solution had been found (in our case, this was done using the cost-minimization approach outlined in section 5.5.1; however, this could also be done using the change-minimization approach or indeed some other method not considered here), this could be translated back into the Time-Windows problem variables, with role assignments being derived from the task assignments and the other auxiliary variables being calculated from these. With correct assumptions during the process, the solution space for this Task-Based Approximation should be a subset of the Time-Windows problem it represents, and therefore any solutions to this Task-Based problem should be feasible for the original Time-Windows formulation and should have the same objective function value.

6.4.1 Procedure for the Task-Based Approximation approach

The procedure is divided into four parts: converting the data into the Task-Based format, solving the problem, converting the solution to the Time-Windows format, and checking the feasibility. For simplicity, details are not not given here; instead, a description of the procedure, including all assumptions needed to ensure the Task-Based solution will be feasible in the Time-Windows formulation, can be found in the appendix in section C.1. Note that full details of the algorithm as implemented in FICO Xpress can also be found in the appendix, section E.2.4.

6.4.2 Making use of the Task-Based Approximation solution

Analysis of the results produced by this Task-Based Approximation approach is given later in this chapter, in section 6.8. However, even before results were known some thought was given to how useful this approach might be in solving the Time-Windows problem.

When first proposed, it was not known how good the solutions found in this way would be. It was believed a solution could be found quickly, but because of the level of simplification needed to adapt the problem to the Task-Based format it was not expected to provide particularly high quality solutions. A plan was therefore devised that once a solution had been found it would be improved by a heuristic algorithm which would seek to reduce the cost. As well as improving solution quality, this type of approach could result in a system which could provide multiple solutions to the Planners, which would be desirable since the Task-Based Approximation approach described here produces only a single solution which may not be acceptable with regard to implicit constraints. The approach taken to developing the heuristics used are discussed in section 6.5 below.

6.5 A Heuristic Algorithm

As discussed above (section 6.4), the Task-Based formulation can be used as an approximation of the Time-Windows formulation, and can be used to quickly find feasible solutions to this more realistic problem. The quality of this solution, however, is unlikely to be high since the solution space has been limited in the approximation process. It would therefore seem reasonable to investigate ways to improve this solution once it has been found, and so a heuristic algorithm has been designed.

The design of this algorithm was an iterative process, with versions being tested, modified, and re-tested in order to find the most useful procedure. In total, eight distinct prototype versions of the heuristic algorithm were created, with the most promising elements of each being selected for inclusion in the formal testing of the programme. This section proceeds as follows: firstly, sections 6.5.1 and 6.5.2 describe the basic principles of the heuristic, followed by an outline of the eight prototype algorithms in section 6.5.3. Finally, the elements of the final testing version of the algorithm are discussed in sections 6.5.4 and 6.5.5, along with an outline of a possible improvement in section 6.5.6.

6.5.1 Solution Representation

The basic principle of this heuristic is that of a neighbourhood search, where a solution is defined by a set of rosters (i.e. list of assignments). For each regular employee $i \in E_R$, there is a roster Υ_i containing T elements, representing the assignment of employee i in each time window $t \in \{1, \dots, T\}$ under this solution. We will say that the element in Υ_i

representing time t has the value j^i_t. If employee i is assigned a task $j \in J$ at time t then $j^i_t = j$; if employee i is unassigned then j^i_t is given the label *rest*. It is assumed that any role j which cannot be covered by regular crew at time t will be covered instead by agency crew. However, because there may be multiple agency crew required at a given time t, we cannot define a roster for agency crew; instead, we should define a separate set Θ_j for each role $j \in J$. Set Θ_j is defined such that if an agency employee is required in role j at time t, then t will be an element in Θ_j. If no agency employees are required in role j across the planning period, then $\Theta_j = \emptyset$.

Using the information contained in the rosters Υ_i, along with the input data such as initial working time w_{i0} and starting location indicator s_{ik}, it is possible to infer all 'new schedule' decision variables of the main formulation (as given in section 6.1.2) for employee $i \in E_R$: $\hat{x}_{ijt}$, $\hat{o}_i$, $\hat{u}_i$, $\hat{b}_{ikt}$, $\hat{d}_{ikt}$, $\hat{w}_{it}$, $\hat{r}_{it}$ and $\hat{l}_{\lambda ijt}$. Since more than one agency employee could be assigned to a role during the planning horizon, and it is possible that two agency employees could work back-to-back in a role, it is necessary to store additional information about the crew change times for agency crew. In addition to the set Θ_j, we therefore also define a set Ξ_j which will contain all time periods t in advance of which an agency employee either boards or departs role $j \in J$. Using this additional information, as well as the corresponding input data such as initial working time α_{j0} and starting location indicator σ_j we can now similarly infer all 'new schedule' decision variables for agency crew: $\hat{x}_{m+1,j,t}$, $\hat{\beta}_{jt}$, $\hat{\delta}_{jt}$, $\hat{\alpha}_{jt}$, and $\hat{l}_{\lambda,m+1,j,t}$. If the new schedule variables can be inferred from this information, then it follows that by comparing with the existing schedule, we can determine which elements have changed under this new solution, and therefore which costs could be applied. We therefore say that describing any solution using the rosters Υ_i and sets Θ_j and Ξ_j gives enough information to evaluate that solution for feasibility and objective function value.

6.5.2 The Neighbourhood

Using this 'roster' representation of the solution, it is then possible to define the *neighbourhood* as the solutions created by making changes to these rosters. The basic changes which can be made are to increase or decrease the amount of consecutive time an employee works on a particular vessel, or to swap an assignment or group of assignments from one employee to another. In order to facilitate the detailed definition of these changes, we first define the concept of a *block* of work, which is a set of consecutive time periods during which an employee is assigned to the same role.

6.5.2.1 Defining blocks of work

We can divide these blocks into two types: a *planned block* starts during the planning period, i.e. in or after week 1; a *current block* is one which is already in progress when planning is taking place, i.e. it has started in or before week zero and is due to finish at some point after week zero. Formally, we can say a planned block of length $\lambda^B \geq 2$ is a group of elements in roster Υ_i which satisfies the following conditions:

$$j^i_{t^S} = j^i_{t^S+1} = \ldots = j^i_{t^S+\lambda^B-1} \neq rest \tag{6.126}$$

and, if $t^S > 1$, then

$$j^i_{t^S-1} \neq j^i_{t^S} \tag{6.127}$$

or, if $t^S = 1$, then for the vessel $k \in K$ such that $j^i_{t^S} \in V_k$

$$s_{ik} = 0 \tag{6.128}$$

and, if $t^S + \lambda^B - 1 < T$, then

$$j^i_{t^S+\lambda^B-1} \neq j^i_{t^S+\lambda^B} \tag{6.129}$$

We say that this block starts in period t^S and ends in week $(t^S + \lambda^B - 1)$. An isolated single period of assignment $j^i_{t^S} \neq rest$ is also considered to be a block of length $\lambda^B = 1$ if it satisfies condition (6.129) and either condition (6.127) or condition (6.128) above.

A current block by definition takes place at least partially (and possibly wholly) before the planning period, and therefore cannot be identified purely by examining roster Υ_i. Instead we must recognise that an employee i working up to and including week zero will have a non-zero work resource value a time zero (i.e. $w_{i0} > 0$) and will have a non-zero value for one of the starting indicator variables (i.e. $s_{ik} > 0$) for the vessel $k \in K$ on board which the block of work takes place. For some employee i such that $w_{i0} > 0$, we can identify the start time of the block as week $t^S = (1 - w_{i0})$. Given that the block is of length $\lambda^B \geq w_{i0}$, we say that the block ends following week $(\lambda^B - w_{i0})$. Therefore for an employee $i \in E_R$ to have a current block under a given solution, the following conditions must be satisfied:

$$w_{i0} > 0 \tag{6.130}$$

and, if $\lambda^B = w_{i0}$, then for the vessel k such that $s_{ik} > 0$

$$j^i_1 \notin V_k \tag{6.131}$$

otherwise, if $\lambda^B > w_{i0}$, then for the vessel k such that $s_{ik} > 0$

$$j^i_1 \in V_k \tag{6.132}$$

and, if $\lambda^B > w_{i0}$, then

$$j^i_1 = \ldots = j^i_{\lambda^B - w_{i0}} \tag{6.133}$$

and, if $0 < \lambda^B - w_{i0} < T$, then

$$j^i_{\lambda^B - w_{i0}} \neq j^i_{\lambda^B - w_{i0}+1} \tag{6.134}$$

With these blocks identified, the algorithm explores the neighbourhood by making alterations to these blocks either by making an *extension*, or making a *swap.*

6.5.2.2 Extending the blocks

A block can be extended by adding additional working periods such that it either ends later (termed *backward* extension) or begins sooner (termed *forward* extension). Clearly a *current* block cannot be extended forwards, since it has already begun during the planning week zero; however if we consider a current block of length $\lambda^B < T - w_{i0}$ in role role j appearing on roster Υ_i, we can extend it backwards by $0 < \lambda^E \leq T - (\lambda^B - w_{i0})$ periods by setting

$$j^i_{\lambda^B - w_{i0}+1} = \ldots = j^i_{\lambda^B - w_{i0}+\lambda^E} = j \tag{6.135}$$

Note that the upper limits on λ^B and λ^E here ensure that we are only dealing with weeks within the planning period. Similarly, if roster Υ_i contains a planned block in role j starting at time t^S of length $\lambda^B \leq T - t$, then we can extend this block backwards by $0 < \lambda^E \leq T - (t^S + \lambda^B - 1)$ periods by setting

$$j^i_{t^S+\lambda^B} = \ldots = j^i_{t^S+\lambda^B+\lambda^E-1} = j \tag{6.136}$$

Where the upper limits on λ^B and λ^E again ensure that we are only dealing with weeks within the planning period. There is no necessity to place an upper limit on λ^B for carrying out forward extension. In this case, provided the planned block described starts at time $t^S > 1$, it can be extended forwards by $0 < \lambda^E \leq t^S - 1$ periods by setting

$$j^i_{t^S-\lambda^E} = \ldots = j^i_{t^S-1} = j \tag{6.137}$$

It should be noted that whenever a block is extended, another block should be shortened. This block may correspond to another employee $i' \in E_R$ working back-to-back with employee i in role j, or may be a part of the set of agency tasks Θ_j. Since it is not possible for more than one employee to work in a role in a given week (see constraint (6.39) in section 6.1.2), the relevant roster $\Upsilon_{i'}$ or set Θ_j must also therefore be amended to maintain feasibility.

It should also be noted that the extensions described here while being technically possible will not necessarily produce a feasible solution to the problem. In particular, constraints described in section 6.1.2 above relating to working period ((6.50) for $i \in E_R$ and (6.53) for agency crew) and rest period (i.e. (6.54), (6.55) and (6.56)) lengths may not be satisfied. If we wish to maintain feasibility throughout the running of the heuristic algorithm, and discount any infeasible solutions from the neighbourhood, we must check that the proposed extension does not violate these constraints before carrying it out.

It can be seen that no definition is given here for extending blocks of work for agency crew. This is due to the cost structure of the problem, with agency employees typically costing up to double that of regular crew to employee in a role. This makes it highly unlikely that an increase to the amount of agency crew utilised will result in a reduction in the overall cost of the solution, and it was therefore deemed not an efficient use of computational time to examine this option in the algorithm.

6.5.2.3 Swapping blocks

A swap involves exchanging a block of work on one employee's roster for a block occurring at a similar time on another employee's roster. Because this involves altering a whole block, the fixed elements of a *current* block prevent it from being used in swap. Alternatively, if we have identified a *planned* block on roster Υ_i in role j, starting at time t^S and ending at time $(t^S + \lambda^B - 1)$, then this can be swapped with a different *planned* block of role j' on the roster of employee i', starting at time t'^S and of length λ'^B, which satisfies the following:

$$t'^S \geq t^S - 1 \tag{6.138}$$

and

$$(t'^S + \lambda'^B - 1) \leq (t^S + \lambda^B - 1) + 1 \tag{6.139}$$

Note that condition (6.139) can equivalently be stated as $(t'^S + \lambda'^B) \leq (t^S + \lambda^B) + 1$. Together, these conditions require the swapping block to start not more than one period before and end not more than one period after the initially selected block. To carry out the swap, we simply transfer the elements of each block from one employee's roster to the other. However, as with the extensions described above, while this may give a technically valid solution in terms of the solution representation, there are also feasibility concerns to be addressed. Therefore we must check for violations of the working and rest period constraints (i.e. (6.50) for $i \in E_R$ and (6.53) for agency crew; and (6.54), (6.55) and (6.56) respectively).

In the case of a swap, it is also possible to exchange the selected block with a block of agency work. To search for a block of agency work in a role j we examine set Θ_j along with the additional crew change information in set Ξ_j. We say that set Θ_j contains a block of length $\lambda^B \geq 1$, starting at time t^S, if it contains elements satisfying the following conditions:

$$t \in \Theta_j \quad \text{for all} \quad t^S \leq t \leq (t^S + \lambda^B - 1) \tag{6.140}$$

and, if $t^S > 1$, then

$$(t^S - 1) \notin \Theta_j \quad \text{or} \quad t^S \in \Xi_j \tag{6.141}$$

and, if $t^S + \lambda^B - 1 < T$, then

$$(t^S + \lambda^B) \notin \Theta_j \quad \text{or} \quad (t^S + \lambda^B) \in \Xi_j \tag{6.142}$$

Note that agency crew blocks are only considered to be *planned* blocks - if $t^S = 1$ then this is taken to be the block start time regardless of the value of σ_j.

An agency block can be examined for swapping with the selected block if it satisfies conditions (6.138) and (6.139) as for a regular crew block, i.e. if it starts not more than one period before and ends not more than one period after the selected block. Also as with the regular crew block, it is necessary to make additional checks to ensure that feasibility is maintained when carrying out a swap with agency crew.

6.5.3 Preliminary Versions of the Algorithm

Having described the basic building blocks of the heuristic algorithm in the preceding sections, we can now describe the various intermediate steps taken towards the creation of the final test version. As might be expected, the development of the heuristic algorithm was an iterative process, with various ideas being implemented, and then modified in response to preliminary test results. This section outlines this process, giving details of what was implemented in each of the eight preliminary versions of the heuristic algorithm, and an outline of the preliminary results. Note that while the final version of the algorithm was coded and tested using C++, it was more expedient to implement these earlier versions of the algorithm using the FICO Xpress-MP software. While the structure and substance of the algorithm is unaffected by this difference, it is clear that C++ results in a more time-efficient algorithm. Therefore, while the preliminary results reported in this section are comparable to each other, there is little comparison which can be made between these and the final results given later in section 6.8.

6.5.3.1 Version 1

This first version of the algorithm provided the basis from which the others were developed. The algorithm begins by reading in an initial solution and calculating the cost. It then proceeds with an iteration by taking each employee $i \in E_R$ in turn, according to a random ordering, and searching for blocks on roster Υ_i in chronological order from $t = 1$ to $t = T$. When a block of work in role j is found, it is considered firstly for backward extension, then for forward extension, and finally for a swap. The extension length in this first version of the algorithm is limited to $\lambda^E = 1$.

As indicated previously (section 6.5.2), the aim is to maintain feasibility at every step. Repairs can be made to some infeasibilities which might be caused by an extension. In particular, it is possible that an extension will shorten a rest period between two blocks such that it is less than ρ_i, the minimum rest period required for employee i. In this case, the first element of the subsequent block (for backward extensions), or the final element of the preceding block (forwards extension) can be re-labelled as *rest*, and the role in that time period can be assigned to an agency employee instead. This cannot be done for all restrictions however - those relating to working period lengths or involving fixed data such as an employee's status at time zero (location, rest resource value) must be obeyed. Therefore, before any extension is carried out, in addition to checking the technical criteria (conditions (6.135), (6.136) and (6.137) above), we must also confirm that the following are satisfied:

- The block length is less than the employee's maximum permitted working time, i.e.

$$\lambda^B < w_i^{max} \tag{6.143}$$

- That role j is required in the period in question, and the employee is eligible to be assigned to the role in that, i.e. for a backward extension:

$$a_{j,t^S+\lambda^B} = 1 \quad \text{and} \quad e_{i,j,t^S+\lambda^B} = 1 \tag{6.144}$$

  or, for a forward extension:

$$a_{j,t^S-1} = 1 \quad \text{and} \quad e_{i,j,t^S-1} = 1 \tag{6.145}$$

- The extension period does not interfere with any rest periods required by the employee at the start of the planning period, i.e. for a backward extension:

$$t^S + \lambda^B > 1 \quad \text{or} \quad r_{i0} = 0 \tag{6.146}$$

or, for a forward extension:

$$t^S - 1 > r_{i0} \tag{6.147}$$

- For a forwards extension, must also account for possible rest period requirements relating to the employee i's location at the start of the planning period. This involves checking that if $\sum\limits_{k \in K : j \notin V_k} s_{ik} = 1$ then

$$t^S - 1 > \rho_i \tag{6.148}$$

  or, if $\sum\limits_{k \in K : j \in V_k} s_{ik} = 1$, then

$$t^S - 1 > \rho_i \quad \text{or} \quad t^S \leq 2 \tag{6.149}$$

When considering this block for a swap, a second block must be found meeting criteria (6.138) and (6.139) described earlier. To do this, firstly all agency task sets $\Theta_{j'}$ for $j' \neq j$ are examined to find a suitable block - this block would start at time t'^S and be of length λ'^B. As with the extension procedure, it is necessary to make checks to ensure feasibility is maintained. An agency block is only acceptable for a swap if it satisfies the following, which relate to infeasibility which cannot be repaired:

- The agency employee must be eligible to be assigned the selected block of work, i.e.

$$e_{m+1,j,t} = 1 \quad \text{for all} \quad t^S \leq t \leq (t^S + \lambda^B - 1) \tag{6.150}$$

- The selected employee i must similarly be eligible to be assigned the agency block of work, i.e.

$$e_{ij't} = 1 \quad \text{for all} \quad t'^S \leq t \leq (t'^S + \lambda'^B - 1) \tag{6.151}$$

- The agency block must be no longer than the maximum working period of employee i, i.e.

$$\lambda'^B < w_i^{max} \tag{6.152}$$

- The block must not interfere with the rest period required by employee i from the start of the planning period, i.e.

$$t'^S > r_{i0} \tag{6.153}$$

Repairable infeasibility may arise from the fact that employee i requires a rest period of length $\geq \rho_i$ weeks either side of this block which has been swapped into roster Υ_i. Any non-rest assignments which must be displaced by this requirement can be assigned to an agency employee.

Having examined the agency sets for swappable blocks, the rosters of regular employees $i' \neq i$ are now examined. This is done in the same random order as is being used to find selectable blocks above. As with the agency blocks, once we find a regular employee block in role j' starting at time t'^S and of length λ'^B there are certain criteria which must be satisfied in order to avoid unrepairable infeasibility. These comprise criteria (6.151), (6.152) and (6.153) from above, and also the following:

- Similar to condition (6.150) above, the swapping employee i' must be eligible to be assigned the originally selected block, i.e.

$$e_{i'jt} = 1 \quad \text{for all} \quad t^S \leq t \leq (t^S + \lambda^B - 1) \tag{6.154}$$

- Similar to condition (6.152) above, the originally selected block length must not exceed the maximum working time of swapping employee i', i.e.

$$\lambda^B < w_{i'}^{max} \tag{6.155}$$

- Similar to condition (6.153) above, the originally selected block must not interfere with the rest periods due to employee i' at the start of the planning period, i.e.

$$t^S > r_{i'0} \tag{6.156}$$

- As with the forwards extension, the addition of role j to roster $\Upsilon_{i'}$ in the required time periods must not violate any rest period requirements relating to the location of employee i' at the start of the planning period. In this case, we require that if $\sum\limits_{k \in K: j \notin V_k} s_{i'k} = 1$ then

$$t^S > \rho_{i'} \tag{6.157}$$

  or, if $\sum\limits_{k \in K: j \in V_k} s_{i'k} = 1$, then

$$t^S > \rho_{i'} \quad \text{or} \quad t^S = 1 \tag{6.158}$$

Infeasibility which can be repaired again relates to the need for minimum-length rest periods to be observed at either side of the blocks. In this case, as well as ensuring that employee i has a rest period of length $\geq \rho_i$ weeks either side of this block which has been swapped into their roster, we must similarly ensure employee i' has a rest period of length $\geq \rho_{i'}$ weeks either side of the originally selected block, which now appears on their roster $\Upsilon_{i'}$. As before, any non-rest assignments which must be displaced in order to achieve this will be assigned to the relevant agency set.

Having described the mechanics of extensions and swaps in this version of the algorithm, we can now return to discussing its overall operation. As stated earlier, the algorithm examines each block it finds firstly for backward extension, then for forward extension, and finally for a swap. When a solution is found within the neighbourhood, its objective function value (i.e. cost) is then calculated. This is done by inferring the relevant variable values, as described in section 6.5.1 and using them to evaluate the objective function quantity expressed in (6.38). Note that this step includes finding the optimal elements of the sets Ξ_j to minimize the cost of crew changes for each agency allocation set Θ_j. As well as calculating cost, the solutions feasibility is also checked - using the inferred variable values, the left and right hand sides of constraints (6.39 - 6.64) can be compared. If the solution found is feasible, its cost is compared with that of the solution at the start of the iteration (referred to as the *current* solution) - if the new solution cost is an improvement on the current solution, then this new solution is accepted, the iteration ends, and a new one begins with a new search for blocks of work. The old solution is stored on a tabu list (of length 1) so that it cannot be returned to at the next iteration.

If the solution does not improve on the current solution cost, then the algorithm continues to look for the next extension or swap, or if it has exhausted all possibilities with this block then to look for the next block. At all times during the iteration, the non-improving, non-tabu solution with the lowest cost will be stored - this is referred to as the *candidate* solution for this iteration. Note that in order to check if a solution is tabu, all rosters Υ_i and agency sets Θ_j are compared between the two solutions. If at the end of the iteration (i.e. with all blocks in rosters Υ_i for all $i \in E_R$ examined) no improving solution has been found, the candidate solution is accepted as the new solution, and a new iteration can begin, with the previous solution being recorded as tabu.

This process continues until one of the stopping criteria is reached. These are:

1. no usable block can be found during an iteration; or

2. 20 iterations have been carried out; or

3. the cumulative running time at the end of an iteration exceeds 600 seconds.

The resulting programme was very slow to run, with an average of approximately

8.66 iterations being achieved within the time limit in the preliminary tests on 82 of the datasets. In the vast majority (76 instances, $\approx 92.3\%$ of those tested) the algorithm was able to find some kind of improvement on the initial solution found using the Task-Based Approximation, but only 26 of these ($\approx 31.7\%$ of tested instances) achieved better than a 5% improvement when taken as a percentage of the best known bound for the instance. There were also a small number of occurrences of infeasible solutions being proposed which were highlighted by the feasibility check subroutine. All these findings were used to inform changes made to the programme in future versions, as discussed below.

6.5.3.2 Version 2

The first revised version of the algorithm was on the whole very similar to Version 1. However there were some changes in response to the findings described above. One of the main changes was to examine only a quarter of all employees' rosters (selected randomly) for usable blocks at each iteration. It was hoped that improving solutions could still be found, but that this would save time at each iteration that would otherwise have been largely wasted. The acceptance criteria were also modified, with a solution only being accepted automatically if it improved on the best solution found so far, rather than just improving on the current solution. This was brought in to encourage the algorithm to find better improvements following the acceptance of a non-improving solution, as under the improve-on-current rule it may otherwise take several iterations to see another improvement on the best solution found so far. Note that while this may lead to more employees having to be examined at each iteration, the effect of this should be mitigated by the reduction in the number of employees to be examined during the iteration.

In addition to these changes, the feasibility check procedure was updated in an attempt to improve efficiency. In this version, a constraint set would be checked if and only if no infeasibility had been detected in the preceding constraint sets. This would prevent the programme carrying out unnecessary checks once a solution had been found to be infeasible. Related to this, the infeasibilities detected in the Version 1 test runs were found to stem from a missing check which should take place when the programme is looking for blocks to swap in to employee i's roster. Similar to conditions (6.157) and (6.158), it is necessary to ensure that employee i can be assigned to role j' at the required times without compromising rest period requirements arising from the starting location of employee i. This means we require that if $\sum\limits_{k \in K : j' \notin V_k} s_{ik} = 1$ then

$$t'^{S} > \rho_i \qquad (6.159)$$

or, if $\sum\limits_{k \in K : j' \in V_k} s_{ik} = 1$, then

$$t'^S > \rho_i \quad \text{or} \quad t'^S = 1 \tag{6.160}$$

With the exception of this addition, all conditions described in section 6.5.2 and for Version 1 above are applied as before.

Overall, these changes were expected to decrease the time taken for each iteration, and therefore the stopping criterion for number of iterations was increased from 20 to 50. The other two stopping criteria remained unchanged from Version 1.

Test runs were carried on on 99 of the problem instances, with small improvements being shown as compared to Version 1. The algorithm now completed an average of approximately 16.35 iterations within the time limit, and although it appeared that there was a substantial increase in the number of *candidate* solutions being accepted (an average of 9.1 per instance, up from an average of 1.4), there was also an increase in the percentage of instances for which the heuristics made some improvement (94 of the 99 instances, which is $\approx 94.9\%$ of those tested, up from $\approx 92.3\%$). The scale of improvement from the initial solution was lower however, with only 23 instances ($\approx 23.2\%$ of those tested) seeing an improvement of greater than 5%, taking the improvement as a percentage of the best known bound for the instance. With regard to infeasibility, the additional checks appeared to have been successful as no infeasible solutions were detected for any of the test runs.

6.5.3.3 Version 3

The main focus of the modifications made to create Version 3 was increasing the efficiency of the algorithm by cutting out unnecessary calculations. The main method of doing this was to record which employees and roles were affected by the movement from the current solution to one of its neighbours. Since all costs relate in some way to an employee (regular or agency), we can break down the cost of the current solution according to each employee $i \in E_R$, and further break down the agency crew cost according to the roles $j \in J$ and record these separately. Essentially this means each roster Υ_i and each pair of agency sets Θ_j and Ξ_j have an individual cost associated with them. This allows us, when a neighbour solution is found, to recalculate only the costs relating to the rosters and sets which have been altered in that move. Since only a maximum of two regular employees plus the agency employees can be changed during an extension or swap, it was anticipated that this would save a large amount of unnecessary calculation time.

Similarly, noting which rosters and agency sets have changed allows some streamlining of the feasibility check procedure. All constraints in the set (6.39), ensuring that all roles are covered when they are required to be, will be checked. However, constraints in the other sets will only be checked if they relate to an employee or an agency role which has been altered; otherwise, they will be assumed to have the same values as in the current solution, and therefore still be feasible. The constraints which still must be checked are:

- If the roster of employee $i \in E_R$ has been changed then for that employee we should check overlapping task constraint (6.40), boarding and departing constraints (6.41 - 6.44), undertime and overtime constraints (6.47) and (6.48), work and rest resource constraints (6.49 - 6.50) and (6.54 - 6.56), constraints linking the old and new schedules (6.57 - 6.59) and (6.62 - 6.64) and constraints defining the domain of the variables (6.65 - 6.67) and (6.69 - 6.70)

- If the allocation set for agency employees for role $j \in J$ has changed then for that role (and, where applicable, for $i = m+1$) we should check agency boarding and departing constraints (6.45) and (6.46), working time constraints (6.51 - 6.53), constraints linking the old and new schedules (6.57) and (6.60 - 6.62) and constraints defining the domain of the variables (6.65 - 6.66) and (6.68).

Some modifications were also made to the way that possible swaps were searched for. In order to cut down the number of calculations carried out in an iteration, Version 3 of the algorithm was set up to record when an employee had been examined to be a *swapping* employee for a given *selected* employee. This combination was then ruled out from being examined again until there was a change to the schedule of one or other of these employees. While this may have the effect of missing out a small number of possible swaps, the overall effect should be to prevent a substantial number of swaps being unnecessarily investigated at each iteration.

The swap procedure was also modified in order to increase the size of the neighbourhood, in the hope that this would increase the scope for good solutions to be found. In this version, it is now permissible to 'swap' the selected block (that is, employee i's assignment to role j for λ^B periods starting at time T^S) with a rest period to which another employee (agency or regular crew) is currently assigned. In the agency crew case, this amounts to assigning the block to agency set Θ_j, which cannot by definition contain any assignment for the duration of the block if the current solution is feasible with respect to role covering constraints (6.39). Assuming the technical criteria (section 6.5.2) are satisfied, this possibility will be examined following examination of sets $\Theta_{j'}$ for $j' \neq j$. As with swaps with other agency crew, condition (6.150) needs to be satisfied in order to avoid unrepairable infeasibility, but note that no repairable infeasibilities can arise from this swap.

A swap with a resting regular employee i' will be examined after the search for a swapping block on roster $\Upsilon_{i'}$ has been completed. The swap can only be carried out if there are no working assignments on roster $\Upsilon_{i'}$ for the duration of the block, that is if

$$j_t^{i'} = rest \quad \text{for all} \quad t^S \leq t \leq (t^S + \lambda^B - 1) \tag{6.161}$$

The other conditions described above for a swapping employee (conditions (6.154), (6.155), (6.156), (6.157) and (6.158)) must also be satisfied, along with the technical criteria. The swap can then be carried out by setting $j_t^{i'} = j$ and $j_t^i = rest$ for all t such that $t^S \leq t \leq (t^S + \lambda^B - 1)$. Repairable infeasibility may arise from assigning role j to employee i' in this time period - in order to correct this, we must ensure that employee i' is allowed their minimum rest period $\rho_{i'}$ either side of the block, with any displaced tasks being assigned to agency crew.

Three additional relatively minor changes were also made for Version 3 of the algorithm. The order of precedence for solution acceptance was changed, so that a neighbour would have to pass the tabu check before it could be accepted on the grounds of solution value. While this would have no bearing on accepting a solution which improved on the best solution found (since this cannot by definition be tabu), this would have an impact if the rule was changed back to accepting a solution which improved on the current solution. In that case, by performing the tabu check first, the possibility of cycling around a local optimum might be reduced.

The other two minor changes were in response to the streamlining of the algorithm. As each iteration should now be much faster to run, the proportion of employees searched for usable blocks at each iteration was increased from a quarter to a third, in order to again increase the search space. The overall time limit was reduced for the test runs however, being set now at 300 rather than 600 seconds.

The test runs for this version were carried out on 198 instances, with marked improvements being shown in terms of the time efficiency of the algorithm. Despite the halving of the time limit, the programme completed an average of ≈ 29.87 iterations for each instance, almost double that of Version 2 and meaning an average iteration was taking just over 10 seconds to run. One instance reached the iteration limit of 50 iterations and terminated after 284 seconds, taking an average of under 6 seconds per iteration. Only one of the 198 instances saw the heuristics achieve no improvement on the initial solution, while 93 instances ($\approx 47.0\%$ of those tested) now achieved an improvement of at least 5% and 37 instances ($\approx 18.7\%$ of those tested) an improvement of at least 10%, taking the improvement as a percentage of the best bound for that instance.

The best solution for an average instance was found just before the 25th iteration, indicating that the ability to carry out more iterations in the time limit was of benefit here. However, on average just over 5 iterations were carried out after the best solution was found, and in some cases this was much greater, with 6 instances being worked on for 20 iterations or more without further improvement. This indicated that the algorithm may still be getting stuck in local optima, and that further diversification or an expansion of the neighbourhood was required. In addition, it was observed that for one instance, six infeasible solutions had been proposed, showing that the conditions on feasibility set out for the three versions so far were not exhaustive.

6.5.3.4 Version 4

The primary improvement made for this version was the introduction of a means of diversifying the search, by bringing in a random *kick* which under certain circumstances would try to move the search to another area of the solution space. The kick would be activated if there had been no improvements to the best solution for eight or more iterations and either

- A kick had previously been carried out and it was ten or more iterations ago; or
- The current cost was equal to the best cost found so far; or
- There had been five or more iterations where the current solution (irrespective of how close to the best solution it is) has not been reduced.

If any of these conditions were met, then for the next iteration the solution acceptance criteria were altered. Tabu solutions were still ruled out, and any solution which improved on the best found so far would still be accepted; but otherwise, each solution generated from the neighbourhood would be accepted with probability 0.7 regardless of its value. It was understood that this might lead to some large increases in the cost, but it was hoped that a kick would provide scope to descend to a better solution in a different area of the solution space. It should be noted that, under this definition, an iteration in which a 'kick' solution was accepted is included in the iteration count.

In order to increase the diversity further, during a 'kick' iteration the recording of which swapping employees had been examined with respect to the selected employees was suspended - *all* employees would be examined (in turn) at this point. In addition, another attempt to increase diversification of the search was made by increasing the size of the tabu list. Now, in addition to the immediately previous solution, all solutions found which have a value equal to the best solution value would be recorded as tabu.

The infeasibilities detected during the test runs of Version 3 were dealt with by a modification to condition (6.158), relating to the location of swapping employee i' at the start of the planning period. In the case that $\sum\limits_{k \in K: j \in V_k} s_{i'k} = 1$, we should now specify that if the selected block starts at time $t^S = 1$ then we require

$$\lambda^B + w_{i'0} \leq w_{i'}^{max} \tag{6.162}$$

and otherwise (i.e. if $t^S > 1$) we require that

$$t^S > \rho_{i'} \tag{6.163}$$

Note that the counterpart condition to this, (6.157) which applies if $\sum\limits_{k \in K: j \notin V_k} s_{i'k} = 1$, has been left unaltered.

Small changes were also made to the stopping criteria for the test runs on this version, with the time limit being increased by 30 seconds to 330 seconds and the iteration limit being increased from 50 to 100 iterations. The resulting Version 4 of the algorithm was tested on 113 problem instances.

The results showed that the average time spent on an iteration was slightly higher at a little under 13 seconds, as might be expected given the extra potential for tabu solutions and the increased number of tabu checks. The level of improvement on the initial solution was similar to Version 3, with 3 instances ($\approx 2.65\%$ of those tested) showing no improvement, 52 instances ($\approx 46.0\%$ of those tested) showing an improvement of 5% or more as a percentage of the best bound for the instance, and 19 instances ($\approx 16.8\%$ of those tested) showing an improvement of 10% or more. This would indicate that the added kick procedure was being largely ineffective.

Indeed, it was seen that in 73 instances ($\approx 64.6\%$ of those tested), the kick was not used. In those for which it was implemented, the best solution seemed to be found before the kick took place in most cases, suggesting that the kick did not enable better solutions to be found in most cases. Further investigation showed that in some instances the kick produced large increases in cost, as was expected. The algorithm was subsequently able to reduce the solution cost back towards the best known, but not below it.

This was supported by the recording of the number of solutions found which were equal to the best known - over the 113 instances tested, the average was just over 4 solutions; however, in 28 instances where one kick took place the average was ≈ 5.57, and for 12 instances where two kicks were used the average was exactly 7, with four of these instances having as many as 10 distinct best solutions. One possible explanation was that the kick solution was too similar to the solutions around it, which is reasonable given that it comes from the neighbourhood. It was therefore decided that a more effective kick would see a greater random element applied, so that the new solution following the kick was not in the neighbourhood of the current solution.

6.5.3.5 Version 5

As indicated above, it was felt necessary based on the preliminary tests of Version 4 to update the kick procedure in the algorithm. This is the main update carried out for

Version 5. The new method for shifting the search into a new area of the solution space is to generate a random block of work, defined by randomly selecting a role j, a start time t^S and a duration λ^B. This random block is then inserted into the roster of a randomly selected employee $i' \in E_R$ if this can be done feasibly; if it cannot, or the resulting solution is tabu, a new (j, t^S, λ^B, i') combination is generated, and this is repeated until a feasible and non-tabu combination is found.

The role j and employee i' are generated such that there is an equal probability of selecting any element of sets J and E_R respectively. The duration cannot exceed the maximum working time of employee i', and therefore takes an integer value such that $1 \leq \lambda^B \leq w_{i'}^{max}$ with equal probability. The start time must then be generated such that the block does not extend beyond the end of the planning horizon, and therefore t^S will be assigned a random integer value which satisfies $1 \leq t^S \leq (T - \lambda^B + 1)$ with equal probability.

In order to be feasible, the proposed block must pass certain checks which will avoid any unrepairable infeasibility. These checks are very similar to those described for Versions 1 and 4 for inserting a block into swapping employee i''s roster when carrying out the swap procedure. Specifically we must ensure:

- Employee i' is eligible to be assigned the generated block, which is exactly condition (6.154);
- The block does not interfere with rest periods due to employee i' at the start of the planning horizon, which is exactly condition (6.156); and
- The block does not interfere with rest periods due to employee i' arising from their location at time zero, which is exactly conditions (6.157), (6.162) and (6.163).

If these conditions are satisfied, we can proceed to reconfiguring the rosters to include this new block. Any assignments which appear on roster Υ_i within the duration of the block must be assigned to the agency set, clearing space to assign the block to the selected employee. This may cause several infeasibilities which can and should be repaired. Firstly, the block of work must be removed from the employee(s) to whom it was previously assigned - if this was to a regular employee, we replace j with the *rest* label for the duration of the block; if it was assigned to agency, we can simply remove the relevant time periods from the set Θ_j (note that the crew change set Ξ_j will be re-evaluated when the cost is calculated). We must also ensure, similar to the swap procedure, that the rest period requirements of employee i are met. This means ensuring a minimum of $\rho_{i'}$ periods of rest either side of the newly inserted block on roster $\Upsilon_{i'}$, with any displaced assignments being assigned instead to agency crew.

Once these updates have been carried out, a tabu check is carried out, the cost is calculated, and a final feasibility check is conducted. If the tabu and feasibility checks are passed, the 'kick' solution is accepted as the new current solution, and the algorithm continues as before; otherwise, a new random block for a random employee must be generated. Note that a kick solution is not necessarily part of the neighbourhood, and that applying the kick no longer constitutes an iteration as it did under the terms of Version 4.

Some other smaller modifications to the programme were also made, not least to the rules for implementing a kick. We now apply the kick if there has been no improvement to the best solution for four or more iterations (down from eight or more in Version 4) and either

- A kick has not yet been carried out (new rule from Version 4); or
- A kick had previously been carried out and it was twelve or more iterations ago (up from ten or more in Version 4); or
- The current cost was equal to the best cost found so far (no change from Version 4); or
- There had been five or more iterations where the current solution (irrespective of how close to the best solution it is) has not been reduced (no change from Version 4).

It was hoped that these new rules would allow the kick to be activated sooner when it was required, but subsequently allow the algorithm sufficient time to explore the new are of the solution space before applying it again.

There was also a redefinition of the tabu list for this version, with the equal best solutions still being recorded but again being allowed to be repeated. This was with the intention of cutting down on unnecessary comparisons, as the kick procedure should be more useful in diversifying the search. Meanwhile, the iteration limit remained at 100, although note that this now does not include kicks, with the time limit being set to 240 seconds for the test runs on this version.

In this instance, time was available to test Version 5 on all 240 problem instances available. The average time taken per iteration was reduced again, to a little over 11 seconds, but so too was the percentage of instances for which the heuristics made no improvement - 13 of the 240 instances, $\approx$ 5.42%, fell into this category. The overall improvement on solutions appeared to have reduced, with now only 81 instances (= 33.75% of those tested, down from $\approx$ 46.0% for Version 4) achieving an improvement of 5% or more, and only 23 instances ($\approx$ 9.6% of those tested, down from $\approx$ 16.8%) achieving an improvement of 10% or more, as a percentage of the best known bound for the instance.

One possible conclusion for this is that the kick is not being as successful as anticipated, although alternatively it could be that the size of the neighbourhood is not adequate to allow for solutions to be improved sufficiently.

In terms of the kick implementation rules, the results were in line with the reasoning behind the changes to these. The kick was implemented in over half of the instances (124 instances, $\approx 51.67\%$ of the total), but far fewer instances activated the kick more than once (9 of the 240, 3.75%, compared with 12, $\approx 10.62\%$, of the 113 instances tested for Version 4). Additionally, having had no infeasible solutions proposed for Version 4, this version saw an average of one infeasible solution proposed for every 5 instances. Investigation revealed this to be due to some missing calculations with respect to the kick procedure, when removing the role from the employee to whom it was assigned in the current schedule.

6.5.3.6 Version 6

For Version 6 it was decided to expand the definition of the neighbourhood to increase the likelihood of finding improving solutions. While in all previous versions, extensions were examined of length $\lambda^E = 1$ only, this version saw the neighbourhood expanded to allow multi-period extensions. The technical criteria for extension set out in section 6.5.2 (i.e. conditions (6.135), (6.136) and (6.137)) must still be adhered to, but the feasibility conditions (6.143) to (6.149) discussed earlier for $\lambda^E = 1$ must now be revised. Instead we now set out conditions which will allow us to determine a maximum number of periods $\Lambda^E \geq 0$ for which a given block can be extended in the given direction, such that the extension will be feasible for all (integer) values of λ^E in the range $0 \leq \lambda^E \leq \Lambda^E$. The quantity Λ^E will be the largest non-negative integer which satisfies the following conditions:

- The length of the extended block (i.e. block length λ^B plus extension length λ^E) must not exceed the employee i's maximum permitted working time. We therefore replace condition (6.143) with

$$\Lambda^E \leq \max\left\{w_i^{max} - \lambda^B, 0\right\} \tag{6.164}$$

- The role j must be required, and employee i must be eligible to be assigned to the role, for all extending periods. For clarity, we will replace the earlier conditions (6.144) and (6.145) with separate conditions for requirement of the role and the employee eligibility respectively. Firstly, for requirement of the role, we must have for a backward extension that

$$\Lambda^E \leq \max_{0\leq\gamma\leq T-(t^S+\lambda^B-1)} \left\{\gamma : \sum_{\lambda^E=0}^{\gamma} a_{j,(t^S+\lambda^B+\lambda^E-1)} = \gamma + 1\right\} \tag{6.165}$$

and for a forward extension that

$$\Lambda^E \leq \max_{0 \leq \gamma \leq (t^S - 1)} \left\{ \gamma : \sum_{\lambda^E = 0}^{\gamma} a_{j,(t^S - \lambda^E)} = \gamma + 1 \right\} \tag{6.166}$$

Similarly, for employee eligibility, we require for a backward extension that

$$\Lambda^E \leq \max_{0 \leq \gamma \leq T - (t^S + \lambda^B - 1)} \left\{ \gamma : \sum_{\lambda^E = 0}^{\gamma} e_{i,j,(t^S + \lambda^B + \lambda^E - 1)} = \gamma + 1 \right\} \tag{6.167}$$

and for a forward extension that

$$\Lambda^E \leq \max_{0 \leq \gamma \leq (t^S - 1)} \left\{ \gamma : \sum_{\lambda^E = 0}^{\gamma} e_{i,j,(t^S - \lambda^E)} = \gamma + 1 \right\} \tag{6.168}$$

- The extension period must not interfere with any rest periods required by the employee at the start of the planning period. For a backward extension we can restate condition (6.146) as

$$\Lambda^E \leq \begin{cases} 0 & \text{if } t^S + \lambda^B = 1 \text{ and } r_{i0} > 0 \\ T - (t^S + \lambda^B - 1) & \text{otherwise} \end{cases} \tag{6.169}$$

and for a forward extension, we can replace condition (6.147) with

$$\Lambda^E < t^S - r_{i0} \tag{6.170}$$

- As before, for a forwards extension there are limits to be placed on Λ^E relating to the possible rest period requirements connected to the location of employee i at the start of the planning period. As before, we break this down into two parts, and say that if $\sum_{k \in K : j \notin V_k} s_{ik} = 1$ then in place of condition (6.148) we require

$$\Lambda^E < t^S - \rho_i \tag{6.171}$$

or, if $\sum_{k \in K : j \in V_k} s_{ik} = 1$, then in place of condition (6.149) we require

$$\Lambda^E \leq \max_{0 \leq \gamma \leq (t^S - 1)} \left\{ \gamma : t^S - \lambda^E > \rho_i \text{ or } t^S - \lambda^E \leq 1 \text{ for all } 0 \leq \lambda^E \leq \gamma \right\} \tag{6.172}$$

If $\Lambda^E = 0$ this indicates that the block cannot be extended in the specified direction; otherwise, all possible extensions to the block are examined, starting from the maximum value $\lambda^E = \Lambda^E$ and decreasing to $\lambda^E = 1$. The reasoning behind this was the anticipation that a multi-period extension has the potential to improve the solution by a greater amount than a single period extension, and therefore it would be advantageous to examine the higher values of λ^E first. Note that, as before, repairable infeasibility may arise from a multi-period extension if the gap between two blocks of work is fewer than ρ_i periods. In this case, elements of the subsequent (for backwards extension) or previous (forwards extension) may be displaced (i.e. re-labelled as *rest*) and re-assigned to agency crew in order to ensure rest requirements are satisfied.

Some other minor adjustments were made to the algorithm from Version 5. A small efficiency improvement was made to the tabu check and best solution comparison functions, allowing the check to finish as soon as any difference between the compared solutions was found rather than having to compare all crew rosters and agency sets. In addition, the acceptance criterion has been changed back to an improvement on the current, as opposed to best, solution. It was hoped that this would lead to faster improvements, especially following the implementation of the random kick procedure.

The implementation rules for the kick procedure were also modified again, with the kick now being applied if the best solution was found four or more iterations ago and

- A kick has not yet been carried out (no change from Version 5); or
- A kick had previously been carried out and it was twenty or more iterations ago (up from twelve or more in Version 5); or
- The current cost was equal to the best cost found so far (no change from Version 4); or
- There had been four or more iterations where the current solution (irrespective of how close to the best solution it is) has not been reduced (down from five or more in Version 5).

It was hoped that this modification would allow the algorithm more time to improve on the solution value following a kick, but would also activate more quickly if the algorithm became trapped in a local optimum.

Finally, a correction had to be made to the way in which rosters were updated following a kick. Specifically, repairable infeasibilities which had not been accounted for would be created when a kick block was generated starting at time $t^S = 1$ in role j, where an employee i was currently assigned to the role and would already be on board the vessel at time zero. In this case, removing role j from roster Υ_i at time 1 necessitates the clearing of a minimum rest period of length ρ_i at the start of the planning period. In general, further steps were taken to ensure correct rest period lengths were given either side of any working tasks remaining on roster Υ_i after the kick block had been removed, with any displaced assignments going to agency crew.

Version 6 of the heuristics was tested on 229 problem instances, with results broadly similar to those for Version 5. Again, 13 instances ($\approx 5.68\%$ of those tested) saw no improvement on the initial solution, while 71 instances ($\approx 31.0\%$ of those tested, down from 33.75%) showed an improvement of 5% or more and 24 instances ($\approx 10.5\%$ of those tested, up from $\approx 9.6\%$) saw an improvement of 10% or more, as a percentage of the best known bound for the instance. The average time per iteration was reduced slightly to ≈ 10.9 seconds, probably as a result of the change to the acceptance rule described above. However, clearly this was balanced out by the need to examine additional extension lengths when searching the neighbourhood. Interestingly, a number of these multi-period extensions were accepted as improving solutions, even though the overall degree of improvement does not seem significantly better.

The kick meanwhile was implemented in more instances than previously - 139 of those tested, $\approx 60.7\%$ of the total (up from $\approx 51.67\%$ for Version 5) - and in fact saw more instances in which the kick was activated more than once (17, $\approx 7.42\%$, of those tested, up from 3.75% for Version 5). This was unexpected, as the new kick implementation rules were designed to reduce the frequency of a second kick. However, these results may show that in some instances the kick quickly lead to another local optimum and was therefore required again for this reason. Finally, no infeasible solutions were proposed for any of the instances, suggesting the steps described above to deal with repairable infeasibility had been successful.

6.5.3.7 Version 7

The main alteration carried out for Version 7 was to attempt an improvement to the extensions which, it was hoped, would allow better solutions to be generated. As discussed previously, it may for some extensions be necessary to remove other assignments from the extending employee's roster, roster Υ_i, in order to maintain feasibility with respect to rest period lengths. Up until now, any assignments displaced from Υ_i for this reason were assigned instead to agency crew. However, as agency crew are generally more expensive it may be more economical to re-assign the displaced assignments to an employee who is already carrying out that role during an adjacent time period.

Taking for example a backwards extension, if there are any non-rest assignments in the period from $t = (t^S + \lambda^B + \rho_i)$ to $t = (t^S + \lambda^B + \rho_i + \lambda^E - 1)$, i.e. following the extended block, then these elements must be re-labelled as *rest* and the roles reassigned. Note that elements in the period from $t = (t^S + \lambda^B)$ to $t = (t^S + \lambda^B + \rho_i - 1)$, i.e. immediately following the block *before* it was extended, will already be labelled *rest* assuming roster Υ_i is feasible in the current solution. Any non-rest assignments in this period will therefore constitute the beginning of the subsequent block on employee i's roster, which begins at time t'^S, and we will say the number of weeks of this block which are disrupted is λ^D. If we say that these assignments are to a role j', then we can say that previously these time periods would have been placed in set $\Theta_{j'}$. However, we now search for an employee i' who is assigned to role j' at time $(t'^S - 1)$ and in chronological order assign as many of the displaced periods as possible to them.

In the case that $i' = m + 1$, i.e. it is agency crew that is assigned to to role j' at time $(t'^S - 1)$, then all relevant time periods from $t = t'^S$ to $t = (t'^S + \lambda^D - 1)$ can simply be allocated to the set $\Theta_{j'}$. However, if $i' \in E_R$, then certain conditions must be checked in order to avoid creating unrepairable infeasibility. Before we can do this, we must identify the details of the block on roster $\Upsilon_{i'}$ which we are, effectively, looking to extend. As stated above, the block will be in role j' and will end at time $(t'^S - 1)$, and we will say that it is of length λ'^B (and therefore begin at time $t'^S - \lambda'^B$). The approach then becomes similar to that described for Version 6 with regard to multi-period extensions - we must find the maximum number of displaced assignments which can be *extended* onto the block of work of employee i'. We will call this number Λ'^E, which must lie in the range $0 \leq \Lambda'^E \leq \lambda^D$, and which will take the largest integer value satisfying the following:

- Similar to condition (6.164) earlier, the total length of the extended block (i.e. block length λ'^B plus extension length Λ'^E) must not exceed the maximum permitted working time of employee i'. I.e.

$$\Lambda'^E \leq \max \left\{ w_{i'}^{max} - \lambda'^B, 0 \right\} \tag{6.173}$$

- The employee i' must be eligible to be assigned the role in all relevant time periods, i.e. equivalently to condition (6.167) earlier we require that

$$\Lambda'^E \leq \max_{0 \leq \gamma \leq \lambda^D} \left\{ \gamma : \sum_{\lambda^E = 0}^{\gamma} e_{i',j',(t'^S + \lambda^E - 1)} = \gamma + 1 \right\} \tag{6.174}$$

  Note that there is no requirement to check that the role is required, since it is assumed that the role would not have appeared on roster Υ_i if it were not required. There is therefore no equivalent of condition (6.165) in this case.

It is of course possible that extending the adjacent employee's block will, in turn, create infeasibility with respect to their minimum rest periods. In this case, the infeasibility can be repaired by again removing subsequent assignments which interfere with the rest period - these will be assigned to agency crew instead. It was considered to continue the process of attempting to assign these displaced assignments to a more suitable employee; however, it appeared that benefit of this would be small compared to the effort of coding and the computational time required to find the correct employee and calculate the new rosters.

Clearly, the equivalent case can be defined for the forwards extension. In this case, it is the period before the extended block on employee i's schedule which we must examine. The elements in the period from $t = (t^S - \rho_i)$ to $t = (t^S - 1)$, i.e. immediately preceding the block before it was extended, will already be labelled as *rest* assuming roster Υ_i is feasible in the current solution. However, any non-rest assignments in the period from $t = (t^S - \lambda^E - \rho_i)$ to $t = (t^S - \lambda^E - 1)$, such that $t > 0$, must be reassigned. These assignments constitute the end of the previous block on employee i's roster, which we will say ends at time t^F. As above, we say that the number of disrupted weeks is λ^D and that the role involved is j'. We therefore in this case search for an employee i' who is assigned to role j' at time $(t'^F + 1)$, and in *reverse* chronological order assign as many of the displaced periods as possible to them.

As before, it may be the case that $i' = m + 1$, i.e. agency crew, meaning all relevant time periods from $t = (t'^F - \lambda^D + 1)$ to $t = t'^F$ can simply be allocated to the set $\Theta_{j'}$. However, if $i' \in E_R$, then certain conditions must again be satisfied in order to avoid creating unrepairable infeasibility. We can identify the details of the block on roster $\Upsilon_{i'}$ which we are going to extend forwards - as stated, it takes place in role j' and starts at time $(t'^F + 1)$, and we will say that it is of length λ'^B, and therefore ends at time $(t'^F + \lambda'^B)$. We then must find the maximum number of displaced assignments which can be added to the start of the block - again, we call this number Λ'^E, which must lie in the range $0 \leq \Lambda'^E \leq \lambda^D$, and which will take the largest integer value satisfying the following:

- As above, the total block length must not exceed $w_{i'}^{max}$, and so condition (6.173) applies here as well as to the backward extension.

- Similar to condition (6.174) for the backward extension, and equivalent to condition (6.168) earlier, we require that i' is eligible to be assigned the role in all relevant time periods, i.e.

$$\Lambda'^E \leq \max_{0 \leq \gamma \leq \lambda^D} \left\{ \gamma : \sum_{\lambda^E = 0}^{\gamma} e_{i',j',(t'^F - \lambda^E + 1)} = \gamma + 1 \right\} \tag{6.175}$$

  Note that as with the backward extension there is no requirement to check that role j' is required in these time periods, and therefore no equivalent of condition (6.166) in this case.

- For the forwards extension, we must also ensure the periods being added to the adjacent block do not interfere with rest periods required by the employee at the start of the planning period. Therefore, similar to condition (6.170) earlier, we require that

$$\Lambda'^E \leq t'^F - r_{i'0} \tag{6.176}$$

- There is also the requirement to check the rest period requirements relating to the location of employee i' at the start of the planning period. Equivalent to condition (6.171) earlier, if $\sum_{k \in K : j' \notin V_k} s_{i'k} = 1$ then we require

$$\Lambda'^E \leq t'^F - \rho_{i'} \tag{6.177}$$

or, if $\sum_{k \in K : j' \in V_k} s_{i'k} = 1$ then, equivalent to condition (6.172) earlier, we require

$$\Lambda'^E \leq \max_{0 \leq \gamma \leq \lambda^D} \left\{ \gamma : t'^F - \lambda^E \geq \rho_{i'} \text{ or } t'^F - \lambda^E < 1 \text{ for all } 0 \leq \lambda^E \leq \gamma \right\} \tag{6.178}$$

As with the backward extension, it may be that that extending the adjacent employee's block forwards will create infeasibility with respect to their minimum rest periods. As discussed above, this is repaired be removing prior assignments which interfere with the rest period, and instead assigning them to agency crew.

No other changes were made to the algorithm, except to make some small changes to the termination criteria. For this version, two different criteria were used. Firstly, a maximum of 100 iterations or 240 seconds was applied, in order to give a direct comparison with Versions 5 and 6 - this was termed Version 7a. In addition to this, it was also decided to investigate the performance of the algorithm over a longer period of time, and therefore a Version 7b test run was made with a limit of 1000 iterations and 600 seconds per instance. For both settings, all 240 problem instances were tested.

The results for Version 7a showed a slight improvement on some of the metrics as compared to Version 6, with only 10 instances ($\approx 4.17\%$ of those tested, down from $\approx 5.68\%$ in Version 6) showing no improvement in the solution value through application of the heuristics. For those that did show improvement, 79 instances ($\approx 32.9\%$ of those tested, up from $\approx 31.0\%$ in Version 6) achieved a 5% improvement or better, and 26 instances ($\approx 10.8\%$ of those tested, up from $\approx 10.5\%$ in Version 6) achieved an improvement of 10% or better (as a percentage of the best known bound for the instance). As would be expected with more time available, these measurements were better for Version 7b, with 8 instances ($\approx 3.33\%$ of the total) showing no improvement, 99 instances ($= 41.25\%$ of the total) showing an improvement of 5% or better, and 45 instances ($= 18.75\%$ of the total) an improvement of 10% or better.

Interestingly, there appeared to be more high-valued improvements for the Version 7 runs than had been achieved by Version 6, with a maximum improvement of $\approx 45.25\%$, and five other instances with an improvement of 20% or better for Version 7a. A possible conclusion of this is that, while the new neighbourhood settings have only helped improve a small number of the instances, in a handful of cases it has allowed significant improvement to be achieved. Also interesting was the variation observed when the level of improvement for Versions 7a and 7b were compared pairwise for each instance. Here it was found that while overall the the longer time limit achieved better results, there were in fact 78 instances (= 32.5% of the total) for which the shorter time limit achieved a better solution. This may indicate that the random element of the programme has a more significant influence than anticipated.

Finally, some infeasible solutions were again proposed during both the Version 7 runs. Investigation showed that this related to some checks which had been omitted from the backwards extension which had, for some reason, not become apparent previously. One possible explanation for them not having shown up before is that different solution areas were being explored under the Version 7 settings than previously, because of the new settings for block extensions. As discussed below, these checks would be included in Version 8.

6.5.3.8 Version 8

As indicated above, there were some infeasible solutions proposed during some of the test runs of Version 7. It emerged that this was due to an anomaly created when certain blocks were extended backwards by $\lambda^E \geq \rho_i$, meaning that the extended block would run into the subsequent block. It was therefore necessary to check that this new combined block did not have length exceeding w_i^{max}, and if it did then to remove any assignments over the maximum length from roster Υ_i with a view to assigning them to the adjacent employee as described above.

While this correction was important to make, the main alteration for this version of the algorithm was to the order in which employees were examined when searching for selectable blocks or employee able to take part in a swap. In all previous versions the ordering was random, but in light of the apparent variation caused by this randomness, it was decided to attempt to encourage the algorithm to focus on a specific area of the neighbourhood where appropriate. In order to achieve this, the algorithm now records the iteration at which each employee's schedule was last changed, with all employees' schedules considered to have changed following a kick. At the start of the first iteration, or immediately following a kick, the employee list is sorted at random. However, if neither of these conditions are true then the following is done:

- If the last iteration improved on the current solution value (irrespective of the *best* solution value found so far), priority is given to those employees whose schedules have changed most recently, with any ties broken at random. This should mean that while the solution value is improving the algorithm will intensify the search in the area where improvement is being found.

- If however no improving solution can be found at a given iteration, the algorithm will order the employees such that those whose schedule has changed least recently appear first in the list, again with any ties broken at random. This means that when the benefit of the intensification appears to have diminished, the algorithm will diversify again to look for improvements which may have been neglected previously.

It was hoped that this would bring a greater stability to the performance of the algorithm, and would overall allow better solutions to be found.

In addition to these changes, the kick implementation rules were once again modified slightly, having been the same in Versions 6 and 7. Previously, one of the implementation conditions stated the kick would be implemented if "the current cost is equal to the best cost found so far". This however implies that the kick would be implemented as soon this equal-best solution was reached, and so the rule is therefore updated to state

- The current cost is equal to the best cost found so far **and at least one non-reducing iteration has just been carried out**.

This allows the algorithm a single iteration to improve the current solution beyond the best found so far before deciding that a local optimum appears to have been reached. Other than this change, the kick implementation rules remained the same as in Version 6 and 7.

As with Version 7, two different termination criteria were used to test Version 8 - a Version 8a used a maximum of 100 iterations and 240 seconds, in order to compare with Version 5, 6 and 7a; Version 8b used a maximum of 1000 iterations and 600 seconds, the same settings as Version 7b and giving an indication of longer-term performance. In addition, in order to gain a better understanding of the effects of randomness on the performance, both Versions 8a and 8b were run twice each. All 240 instances were tested in the first run of both versions; however, due to time constraints the second test run had to be cut short, resulting in 219 instances being tested on the second run of Version 8a, and 218 instances on the second run of Version 8b.

Overall, there was a noticeable improvement both in terms of the number of instances for which the initial solution was improved upon, and the amounts by which this improvement was achieved. The two shorter time limit runs saw 5 and 8 instances respectively ($\approx 2.08\%$ and $\approx 3.65\%$ of those tested, respectively) for which there was no improvement, down from $\approx 4.17\%$ for Version 7a, while the longer time limit runs both resulted in 7 instances ($\approx 2.92\%$ and $\approx 3.21\%$ of those tested, respectively), down from $\approx 3.33\%$ in Version 7b. In terms of degree of improvement, the shorter Version 8a found a 5% or better improvement on the initial solution for 36.25% of instances for the first run, and $\approx 39.7\%$ of instance for the second, up from $\approx 32.9\%$ for Version 7a. Similarly, an improvement of 10% of better was found respectively in $\approx 15.4\%$ and $\approx 14.2\%$ of instances tested, up from $\approx 10.8\%$ in Version 7a. The longer Version 8b runs achieved an improvement of 5%

or better in $\approx 42.1\%$ and $\approx 45.9\%$ of instances respectively, up from 41.25% for Version 7b; and an improvement of 10% or better in $\approx 22.1\%$ and $\approx 21.1\%$ of instances for the two respective runs, up from 18.75% in Version 7b.

The random element seems less of a factor here, with a large majority of instances displaying a similar percentage improvement in solution value when comparing both the two Version 8a runs and the two Version 8b runs. Comparing the Version 8a with the Version 8b results, these appear more consistent with the pattern of having broadly similar results for the two settings, but with a better level of improvement gained from the longer time and iteration limits. This suggests that using the more specialised ordering of the employees not only leads to better solutions being found, but also leads to more consistency of results.

There were still a small number of infeasible solutions found on the four test runs. However, these were small in number and so it was decided not to expend additional effort investigating these and adding additional code to prevent these infeasible solutions being proposed. Even if these infeasible proposals could be eliminated, it would still be necessary to run the feasibility checking procedure in case of any other omissions, therefore any benefit of adding further checks in the calculation steps would be marginal.

6.5.3.9 Summary of equations used

It may be noted that there are numerous equations set out here in section 6.5.3 which are used to check the feasibility of extensions and swaps. It can be seen that some of those in place for earlier versions of the algorithm are modified and superseded by equations used in later versions. We therefore provide a list of all those which are relevant to being taken forward to the formal testing phase of the heuristic development (discussed in more detail in sections 6.5.4 and 6.5.5 below.

This list does not include any of the technical criteria set out in section 6.5.2, as these were unchanged through the preliminary process and must hold at all times. However, the feasibility conditions which had evolved through the preliminary process amounted to the following by the time Version 8 was developed:

- To check maximum extension length of a block, must find the largest value of Λ^E which satisfies the following:
  - Block length: condition (6.164).
  - Role is required: condition (6.165) for a backward extension; condition (6.166) for a forward extension.
  - Employee eligible: condition (6.167) for a backward extension; condition (6.168) for a forward extension.

  - Initial rest periods: condition (6.169) for a backward extension; condition (6.170) for a forward extension.
  - Rest periods relating to employee's location at time zero (forward extension ONLY): conditions (6.171) and (6.172).

- For the adjacent employee disrupted by the extension, to check the maximum length of the knock-on extension must find the largest value of $\Lambda'^E \leq \lambda^D$ which satisfies the following:
  - Block length: condition (6.173).
  - Employee eligible: condition (6.174) for a backward extension; condition (6.175) for a forward extension.
  - Initial rest periods (forward extension ONLY): condition (6.176).
  - Rest periods relating to employee's location at time zero (forward extension ONLY): conditions (6.177) and (6.178).

- To check if a working block can be placed into selected employee's schedule during a swap:
  - Eligible: condition (6.151).
  - Length of swapping block: condition (6.152).
  - Rest period at time zero: condition (6.153).
  - Rest periods relating to employee's location at time zero: conditions (6.159) and (6.160).

- To check if a rest block can be placed into selected employee's schedule during a swap:
  - Roster contains only rest periods in period of interest: condition (6.161).

- To check if a working block can be inserted into the swapping or kicked employee's schedule:
  - Eligible: condition (6.154).
  - Length of selected block (for swap ONLY): condition (6.155).
  - Rest period at time zero: condition (6.156).
  - Rest periods relating to employee's location at time zero: conditions (6.157), (6.162) and (6.163).

- To check if a working block can be placed into the agency employees' schedule:
  - Eligible: condition (6.150).

6.5.4 Elements Selected for Formal Testing

Having set out in section 6.5.3 the different settings and other elements of the algorithm examined during the design process, we can now describe those taken forward to be coded in C++ for the formal testing. The preliminary tests described above, as well as some calibrating runs carried out in C++, gave an indication of what may or may not be useful, but could not be considered to be a rigorous investigation into the settings which should be used to achieve the best results in practice. Therefore elements which appeared promising during preliminary tests were selected in order that combinations of these could be examined in conjunction.

In all, the aspects of the design process discussed in section 6.5.3 can be broken down in twelve distinct categories which were altered across the eight preliminary versions of the algorithm. Not all of these give rise to different test settings, with some categories demonstrating one clear setting (or group of settings) which outperforms all others examined. However for completeness, and in order to provide a clear summary of the different preliminary versions, all twelve categories are discussed below. In addition, it is known that the initial solution can have an impact on the performance of a heuristic algorithm; therefore it was also decided to vary the initial solution which was input into the algorithm - this is also discussed in more detail below.

6.5.4.1 Number of employees to examine per iteration

Version 1 of the heuristic examined all employees at each iteration, which appeared to involve wasted effort, while Version 2 cut this to one quarter of all employees, which may have been too restrictive. The majority of the preliminary versions examined a third of all employees at each iteration, which appeared to be suitable in that it allowed the algorithm to explore useful areas of the neighbourhood without taking an unreasonable amount of time. Early experience with C++ highlighted the speed at which the algorithm runs on this platform as opposed to with FICO Xpress, and indicated that useful results may in fact be obtainable if all employees were examined at each iteration. Therefore it was decided to use this setting as a comparison with this reduced number, meaning the two settings used for this category were:

1. All employees.

2. One third of all employees.

6.5.4.2 Order in which to examine the employees

The default setting for the majority of test runs with regard to this was a random order. However, as described for Version 8, an alternative ordering method was designed which would assist intensification and diversification of the search, as appropriate. This more helpful ordering did indeed appear to give a benefit in terms of results, although there is no proper evidence which would allow us to say this with certainty. The formal tests therefore use two settings for this category:

1. Random.
2. The more tailored ordering, as follows:
   - if a kick has just been carried out or we are at the first iteration, sort at random;
   - else, if the last iteration improved the current solution value, sort employees with those most recently changed first (with any ties broken at random);
   - else, sort employees with those who have been changed least recently first (with any ties broken at random).

   Note that we should consider all employees' schedules to have changed following a kick.

6.5.4.3 Definition of the neighbourhood - extensions

The definition of an 'extension' to a block evolved considerably during the development of the algorithm. Initially, the algorithm would only examine a single period extension either forwards or backwards, but this was later adapted to allow multi-period extensions to blocks. If any later or earlier blocks (respectively for backward or forward extensions) had to be disrupted to maintain feasibility, then initially these disrupted assignment weeks were allocated to agency crew; however, this was later updated to try to allocate the tasks to a more suitable employee, i.e. one who was assigned to an 'adjacent' block in the current solution.

Results of the preliminary tests seem consistent with the intuitive belief that the later developments in this area allow more promising changes to be explored. Specifically, the multi-period extension setting also includes the possibility of a single period extension, while allocating a role to an 'adjacent' employee will in general be more cost-effective than allocating to agency crew. Clearly the more complex definitions of the extension will take more time to run; however this appears to be outweighed by the benefit in terms of promising solutions. Therefore, the only definition used in the formal testing from this category was:

1. Examine each possible length of extension, starting with the maximum, for both backward and forward directions. Any earlier / later blocks which must be disrupted from 'extending' employee's schedule should be used to 'extend' other employees'

working periods where possible (with any tasks being removed as a knock-on effect being allocated to agency crew).

6.5.4.4 Definition of the neighbourhood - swaps

The definition of a swap move was also altered during the development of the algorithm, but to a lesser extent than for extensions. Initially, a selected block could only be swapped with another block which started no more than one period before it and ended no more than one period after it. This definition was subsequently extended to also include examining a swap with an employee who had no assignment during the period of interest (be that a regular or agency employee). As with the extensions, there seems to be a benefit of the inclusion of the unoccupied employees, while the additional computational time needed in this case is negligible. Therefore there is again only one option used for formal testing:

1. Find block to swap which starts no more than one period before and ends no more than one period after the selected block, or swap with unoccupied employee (either regular or agency).

6.5.4.5 Solution acceptance criteria

Two specific criteria were experimented with during the development of the algorithm. Firstly, the question of whether to accept the first solution which improves on the current solution or improves on the best found so far; and secondly, whether or not a tabu check should be carried out at this stage, with any tabu solution being rejected regardless of the improvement it gives. the combination of these gave four different settings which were examined in the preliminary versions of the algorithm.

With regard to the second question, results so far suggest that the tabu check is necessary in order to ensure the algorithm does not cycle, and is generally effective in doing this. However, the answer to the question of improvement on current versus improvement on best is less clear. The two rules are equivalent when the current cost is equal to the best found so far, but will cause the algorithm to act differently if the current cost is higher. Under these circumstances, the 'improve on best' rule will take longer to find an acceptable change but will ensure either a new best solution will be found, or the best available change is carried out otherwise. On the other hand, the 'improve on current' rule will generally lead to iterations being carried out more quickly, although more iterations might be needed to bring the solution value back down to the best found so far. Preliminary tests suggested that the 'improve on current' rule is overall better than the 'improve on best', but it was decided to test them formally against each other. Therefore the two options chosen for the formal testing were:

1. First non-tabu solution which improves on best solution value.

2. First non-tabu solution which improves on current solution value.

6.5.4.6 Action if no solution accepted by end of iteration

The only option used during the development process was to accept the best non-tabu solution found during the iteration if no solution meeting the acceptance criteria had been found by the end of that iteration. Intuitively this also appears sensible, and therefore the only option in this category used for formal testing was:

1. Select best non-tabu solution found otherwise.

6.5.4.7 Definition of tabu solutions

Throughout the eight preliminary versions of the algorithm set on in section 6.5.3, two different definitions have been used for the tabu list - the immediate previous solution only, and the immediate previous solution along with all solutions of equal value to the best found so far. This inclusion of the equal-best solutions was intended to encourage diversification in the search and to rule out the possibility of cycling almost completely. However, preliminary results appeared to show that this was unnecessary - Version 3 for example, which treats only the immediate previous solution as tabu, outperforms Version 4 overall - and suggest that the additional comparisons required may be a waste of computational time. It was therefore decided to use only one setting in this category for formal testing:

1. Immediate previous solution only.

6.5.4.8 Definition of the random kick

The random kick took two general forms in the preliminary versions of the algorithm. Initially, the 'kick' took the form of accepting a non-tabu, non-improving solution with a 0.7 probability, in order to move the search away from a local optimum. However, this was later changed to redefine the 'kick' as a solution from (most likely) outwith the neighbourhood, by selecting at random an employee, role, start time and duration and inserting this into the schedule. This updated version of the kick appeared to be more useful that the initial definition, and was intuitively more sensible with regard to the objective of the kick - to move the search away from the current neighbourhood. Interestingly, Version 3 of the algorithm had no random kick in operation yet still has competitive results against some of the later preliminary versions. With this in mind, two settings with respect to the random kick were taken forward to the formal testing phase:

1. No kick used.

2. Randomly select employee, role, start time and duration which can be inserted into schedule while maintaining feasibility and giving a non-tabu solution. Note, this means that change is not necessarily part of the neighbourhood. 'Kick' done between iterations, so does not add to iteration count.

6.5.4.9 Activation of the random kick

Overall, four different activation settings for the random kick were used in the preliminary tests, in addition to the versions where no kick was used at all. These activation settings were:

- If best solution was found **eight** or more iterations ago, and a) a kick has already been carried out and it was **ten** or more iterations ago; or b) the current cost is equal to the best cost found so far; or c) there have been **five** or more iterations where the current solution (irrespective of how close to the best solution it is) has not been reduced.

- If best solution was found **four** or more iterations ago, and a) no kick has yet been carried out; or b) a kick has already been carried out and it was **twelve** or more iterations ago; or c) the current cost is equal to the best cost found so far; or d) there have been **four** or more iterations where the current solution (irrespective of how close to the best solution it is) has not been reduced.

- If best solution was found **four** or more iterations ago, and a) no kick has yet been carried out; or b) a kick has already been carried out and it was **twenty** or more iterations ago; or c) the current cost is equal to the best cost found so far; or d) there have been **four** or more iterations where the current solution (irrespective of how close to the best solution it is) has not been reduced.

- If best solution was found **four** or more iterations ago, and a) no kick has yet been carried out; or b) a kick has already been carried out and it was **twenty** or more iterations ago; or c) the current cost is equal to the best cost found so far **and this is not an improvement** on the previous iteration; or d) there have been **four** or more iterations where the current solution (irrespective of how close to the best solution it is) has not been reduced.

The structure of the activation rules has changed little across the different versions, with the main alterations being with regard to the numbers of iterations (as highlighted above). The results appear to support the belief that the added 'no kick has yet been carried out' rule ensures that we do not have to wait too long for the first kick when it is required. Similarly, it appears that the ‘and this is not an improvement on the previous iteration’ addition to the current cost equal to best cost clause should allow (at least to an extent) for the possibility that this new equal best solution is not in fact a local minimum and gives the algorithm a chance to explore this possibility.

The main uncertainty remaining from the preliminary tests is how many iterations should be allowed to pass before implementing a kick - too few and the chance to move to a more promising solution might be lost; too many and time is wasted. Therefore, in addition to the setting where no kick is used at all, two settings have been selected for formal testing. The first of these reflects the best options seen during the preliminary tests, while a second contained an increase in the number of iterations in order to reflect the additional number which would be achieved by C++ implementation of the algorithm over a similar length of time. Including the option of no kick being implemented, the settings chosen for formal testing are:

1. No kick used.

2. If best solution was found **four** or more iterations ago, and a) no kick has yet been carried out; or b) a kick has already been carried out and it was **twenty** or more iterations ago; or c) the current cost is equal to the best cost found so far **and this is not an improvement** on the previous iteration; or d) there have been **four** or more iterations where the current solution (irrespective of how close to the best solution it is) has not been reduced.

3. If best solution was found **eight** or more iterations ago, and a) no kick has yet been carried out; or b) a kick has already been carried out and it was **forty** or more iterations ago; or c) the current cost is equal to the best cost found so far **and this is not an improvement** on the previous iteration; or d) there have been **eight** or more iterations where the current solution (irrespective of how close to the best solution it is) has not been reduced.

6.5.4.10 Recording of best solutions

Even though the consideration of the equal-best solutions was rejected in the discussion of the tabu check above, it is still necessary to question whether or not they should be recorded at all. Some of the preliminary versions of the algorithm recorded only the first solution found with this ‘best’ value, while others recorded all solutions found with that value. Clearly, recording all solutions of a given value takes more computational time, but given that one of the objectives of the problem overall is to provide *multiple* useful solutions to the planners there is an obvious benefit to doing this. Therefore for the formal testing stage the only setting used for this category was

1. Record all solutions found of that value.

6.5.4.11 Stopping criteria

Numerous stopping criteria were applied over the course of the preliminary test, ranging from a maximum of 20 up to a maximum of 1000 iterations and a time limits ranging from 240 to 600 seconds, as well as the condition that the algorithm should terminate if no usable block is found during an iteration. It should be noted that the 'no usable block' criterion is redundant except in the most extreme of circumstances, and is increasingly unlikely to be required when the neighbourhood definition is expanded; however it does no harm to leave it in place just in case these circumstances arise. The iteration limits were similarly redundant in the preliminary test run, since they were deliberately set higher than the programme would be expected to reach in the time available; it was therefore decided for the formal testing to remove the iteration limit altogether. In terms of the time limit, it was decided to test the programme over a realistic time that could be of benefit to the company. Originally, this was considered to be 5 or 10 minutes; however, the speed at which C++ is able to run compared to FICO Xpress lead us to believe a 2 minute time limit should be able to produce useful results. In order to monitor results available at earlier times, solution values after 1 minute were also recorded. The setting used for the formal testing was therefore:

1. No usable block found; or 120 seconds. No limit on iterations. Solution value after 60 seconds recorded to monitor progress and to give comparison at an earlier time point.

6.5.4.12 Efficiency of the algorithm

It should be acknowledged that the early preliminary versions of the algorithm were not as efficient as they might have been. This was then improved by reducing the computations required for the cost calculation and feasibility check, and recording the examined swaps to reduce occurrences of re-examination. Subsequently, improvements were also made to reduce the effort needed to compare solutions with the tabu list or the repeated best solutions. Clearly all possible efficiency savings should be included in the formal testing, and so we say the setting used for this category in the formal tests is

1. Efficiency improved for cost calculation, feasibility checks, and re-examining for swaps and tabu and repeat of best checks.

6.5.4.13 Initial solution

All the preliminary versions of the algorithm were tested using the Task Based Approximation as the initial solution, as discussed above at the start of section 6.5. However, a later development discussed below in section 6.6 was to use a heuristic method to create an initial solution, based on similar principles to the heuristic described here. It was was therefore decided to carry out the formal testing using this heuristic initial solution in addition to the Task Based Approximation, giving two sets of inputs to the heuristic testing:

1. Task Based Approximation as initial solution.
2. Heuristic initial solution

6.5.4.14 Overall plan

The settings described throughout this section give the following variations for the formal testing phase:

- Two options for the number of employees to examine at each iteration:
    1. All employees.
    2. One third of all employees.
- Two options for orderings in which to examine employees:
    1. Random.
    2. The more tailored ordering, as described in section 6.5.4.2.
- Two options for solution acceptance criteria:
    1. First non-tabu solution which improves on best solution value.
    2. First non-tabu solution which improves on current solution value.
- Three options for random kick settings:
    1. No kick used.
    2. Two different sets of kick activation criteria, as described in section 6.5.4.9.
- Two options for initial solutions:
    1. Task Based Approximation as initial solution.
    2. Heuristic initial solution

A full-factorial combination of these settings were carried out for the formal test runs, giving a total of 48 combinations for each test instance. The matter of repetition was also considered, given the random variation which would be present in the running of the programme. However, it was decided that the effect of random variation would be reduced by the repetition inherent in the full-factorial test design, and so a single run of each combination for each instance would be acceptable. One concern in this decision was the overall running time of the tests, with a single non-repeating run taking a total of 48 x 240 x 2 minutes = 384 hours of computational time.

This section concludes with a description of the complete procedure used in the formal testing, as well as an idea for a modification which could make the algorithm more generalisable to other settings. Results of the test runs are discussed later, in section 6.8.

6.5.5 Formal Test Version of the Algorithm

Having set out the various components comprising the heuristic algorithm throughout section 6.5, as well as the process followed to develop it, it would seem sensible to draw this together into a summary of the algorithm used in the formal testing phase. In this section we will lay out the specific components of the programme and describe the function of each, referencing where appropriate conditions and other criteria described above.

Firstly, we present the algorithm in diagram form, split between Figures 6.7 and 6.8. The flowchart shows the outline of the algorithm, with the detail of procedures such as **'Evaluate backwards'**, **'Evaluate forwards'** or **'Random kick'** hidden for simplicity. Detail of these hidden procedures and an expanded version of this flowchart are given in the appendix, section C.2.

It can be seen that the main part of the programme begins at the top left of Figure 6.7, progressively entering each of the three main loops of the algorithm: starting a new iteration with a newly sorted employee list; taking each of these employees in turn; and examining each block on the employee's schedule in turn. The **'Evaluate block'** procedure can be seen to work progressively through checking for backwards extensions, forward extensions and swaps in turn, and breaking off to implement a new solution if one is found which meets the automatic acceptance criteria. At the end of the iteration, if any solutions have been found which do not meet the automatic acceptance criteria, then the best of these (the 'candidate') is implemented. Termination criteria are then checked, and if the algorithm continues then the kick implementation criteria are also checked before moving onto the next iteration.

A detailed step-by-step description of the algorithm, starting with the main programme and moving deeper into the sub-programmes used, can be found in the appendix in section C.3, while the full code of the algorithm as implemented in C++ can be found in appendix

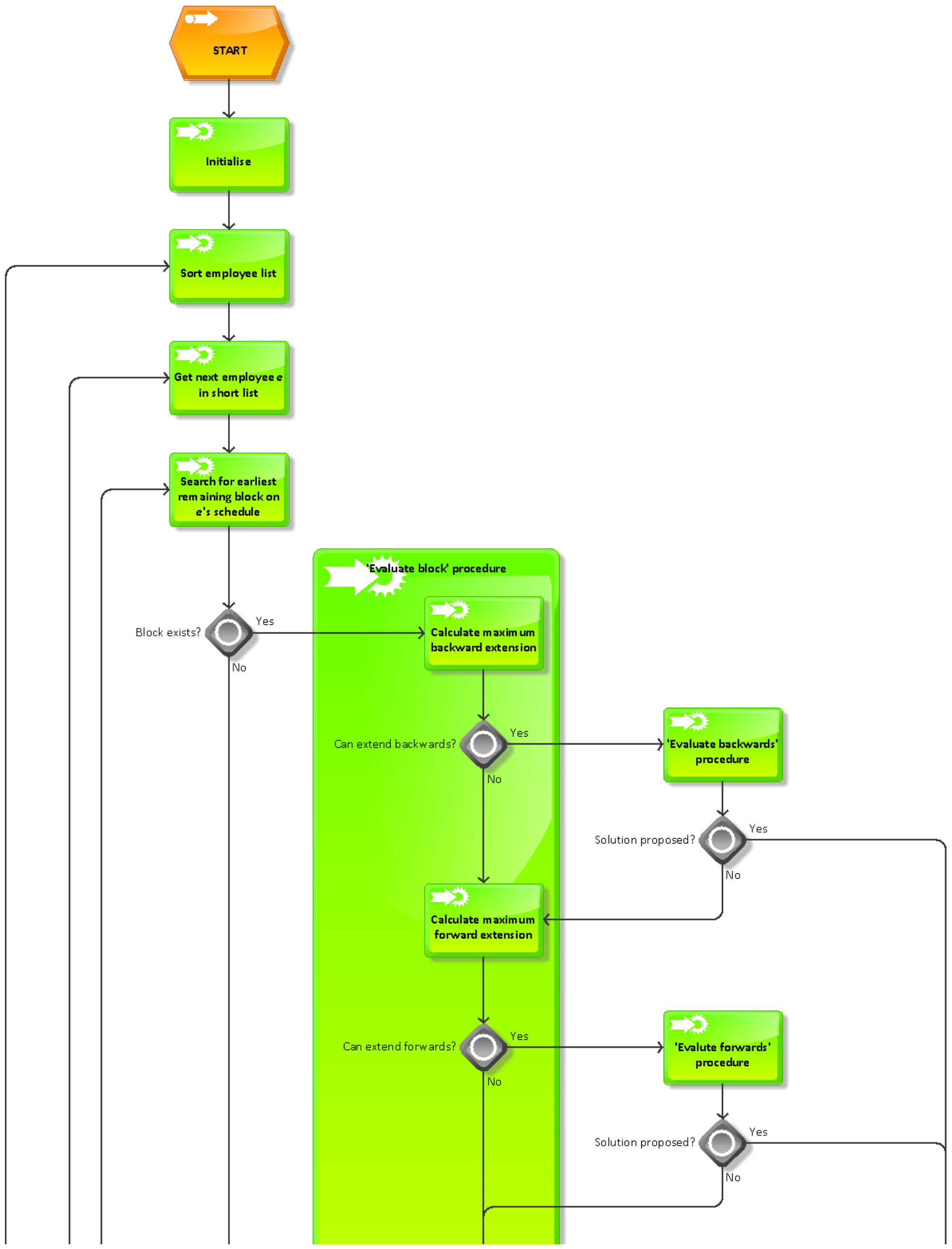
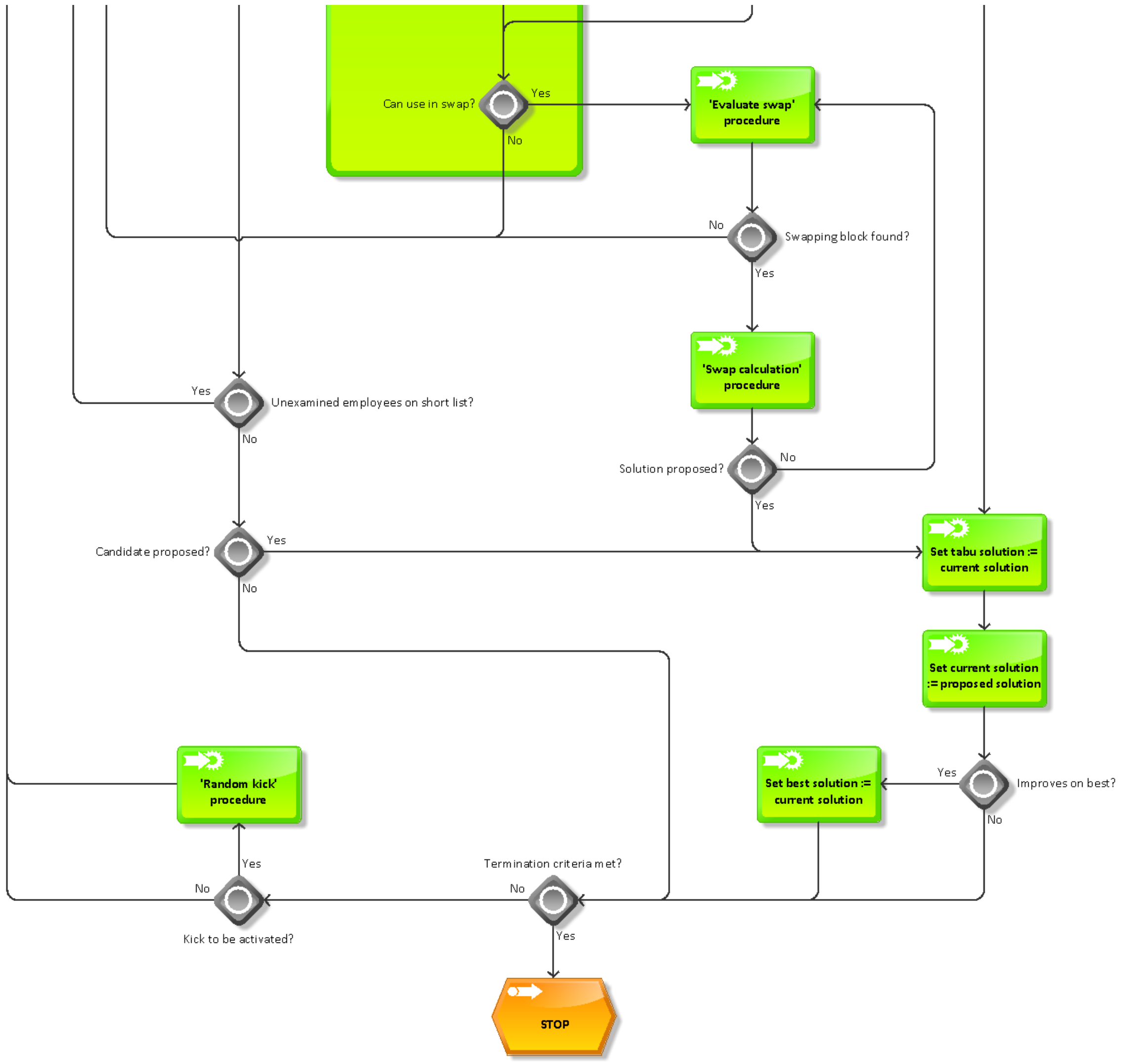

START
Initialise
Sort employee list
Get next employee e in short list
Search for earliest remaining block on e's schedule
Block exists?
Yes
No
'Evaluate block' procedure
Calculate maximum backward extension
Can extend backwards?
Yes
No
'Evaluate backwards' procedure
Solution proposed?
Yes
No
Calculate maximum forward extension
Can extend forwards?
Yes
No
'Evalute forwards' procedure
Solution proposed?
Yes
No

Figure 6.7: Flowchart outlining the formal test version of the heuristic algorithm - upper section.



Can use in swap?
Yes
No
'Evaluate swap' procedure
No
Swapping block found?
Yes
'Swap calculation' procedure
Yes
Unexamined employees on short list?
No
Solution proposed?
No
Yes
Candidate proposed?
Yes
No
Set tabu solution := current solution
Set current solution := proposed solution
'Random kick' procedure
Set best solution := current solution
Yes
Improves on best?
No
Yes
Termination criteria met?
No
No
Kick to be activated?
Yes
STOP

Figure 6.8: Flowchart outlining the formal test version of the heuristic algorithm - lower section.

section E.2.5. Note that while this formal test version is a product of the development process described above, the actual coding of this algorithm into C++ was carried out by fellow PhD student Seda Sucu.

6.5.6 Possible Modification

The elements of the heuristics selected for testing, as discussed in sections 6.5.4 and 6.5.5 above, were believed to be the best known for the problem and for the available data. However, after the testing phase had taken place, it was realised that some small modifications could be made which would make the problem more generalisable.

Specifically, this related to the stopping criterion discussed in section 6.5.4 that the programme should terminate if no usable block was found. As noted in the discussion of this element of the programme, the chance of this occurring with our company's data is negligible. This is because of the possible swap which is considered for all blocks whereby an unoccupied agency employee is assigned the block (and the selected employee gains a rest period in return). The nature of the problem description is such that an agency employee will almost certainly be available for every block, and indeed in the data that was generated for the Time-Windows problem it was assumed that agency crew would be available and eligible to carry out all tasks (see **Step 8** of the procedure given in section 6.2.1).

However, while it is true that if even one of the examined blocks can be worked by an agency employee then this stopping criterion will not be activated, it was realised that it was still theoretically possible for the criterion to be required. This could occur if the problem description at this company changes, or indeed if this algorithm were to be generalised to another setting where it may be the case that agency employees are less readily available. In addition, it was considered that terminating under these conditions may not be as sensible as first believed. Originally, this criterion was used in the belief that if no usable block was found then it meant the algorithm was at an optimum and could be terminated with a good solution. However, we later realised the possibility that this optimum would be a local one only, and for that matter in cases where only a fraction of employees are being examined at each iteration it was possible that a potential improvement had been missed.

In light of this, we propose a modification to the algorithm discussed above, with relation to the action taken if no usable block can be found during an iteration. This modification has not been investigated as part of this work, but is given as a suggestion for future work which can be carried out either with new data, or with an entirely other company at which the problem circumstances are different. In order to improve the algorithm under these new circumstances, we propose to make the following changes to the algorithm as described in section 6.5.5 above and set out in detail in section C.3 of the appendix:

1. We introduce a new list of employees which we will call the 'examination list'. At the start of a new iteration, this examination list will be assigned the contents of the 'short' list. This will be done in a new step in the **'Main programme'**, inserted between **Step 3(b)** and **Step 3(c)**.

2. The **'Find usable block'** procedure will then proceed through all the employees on the examination list, rather than all those on the short list - this means an alteration to the beginning of **Step 2** of this procedure.

   These first two modifications can also be illustrated in the flowchart, with the step 'Get next employee *e* in short list' replaced as shown in Figure 6.9 below.

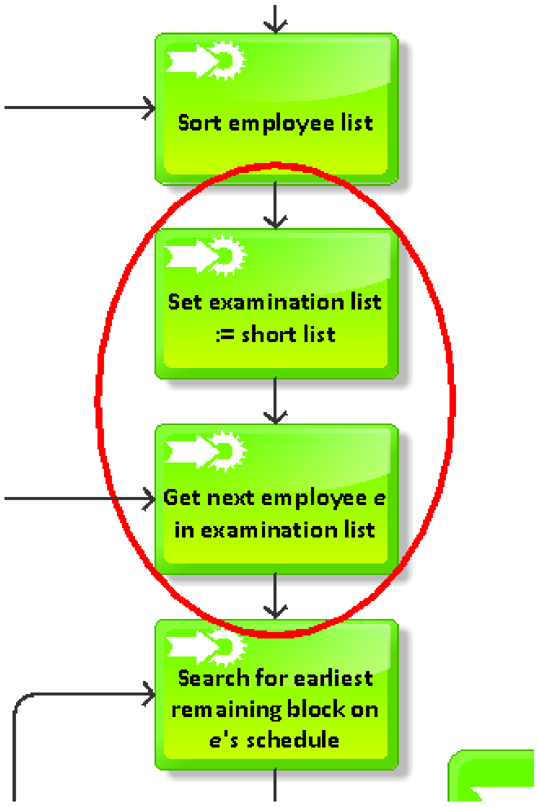

Sort employee list
Set examination list := short list
Get next employee e in examination list
Search for earliest remaining block on e's schedule

Figure 6.9: Detail of modified flowchart, showing addition of 'examination list'.

3. If once all employees on the examination list have been considered there has been no solution automatically accepted, this is no longer necessarily the end of the iteration. Instead, we will add a new step ahead of **Step 3** of the **'Find usable block'** procedure which will state:

"if no update has been done and no 'candidate' solution exists, then if the examination list does not contain all regular employees extend the list to include all regular employees; for each as yet unexamined employee i in the examination list, repeat **Step 2**."

The programme would then progress to **Step 3** as before. This change can be illustrated in Figure 6.10 below, which shows how the flowchart would be modified to take account of this alteration.

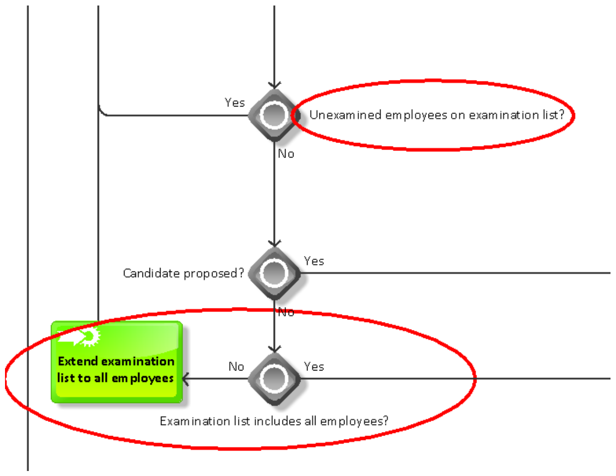

Yes
Unexamined employees on examination list?
No
Candidate proposed?
Yes
No
Extend examination list to all employees
No
Yes
Examination list includes all employees?

Figure 6.10: Detail of modified flowchart, showing other effects of adding 'examination list'.

4. At the end of the iteration, having considered *all* regular employees if necessary and potentially having made no update, the **'Find usable block'** procedure ends and we return to the main programme to check the termination criteria (**Step 3(e)** of the **'Main programme'**). We propose to remove 'no usable block found' from the stopping criteria, as set out in section 6.5.4.

5. Instead, we will add 'no usable block found' as one of the kick activation criteria checked for in **Step 3(f)** of the **'Main programme'**.

These changes will mean that, if the programme reaches a point where no usable block is found, then additional time is given to enable it to find an improving solution. If none can still be found, we would no longer accept that no further improvement is possible, and instead 'kick' the programme to another part of the search space and continue to attempt to improve the solution while the time limit permitted. These changes can also be shown in diagram form, updating the flowcharts shown in Figures 6.7 and 6.8 earlier - details of this are split between Figures 6.11 and 6.12.

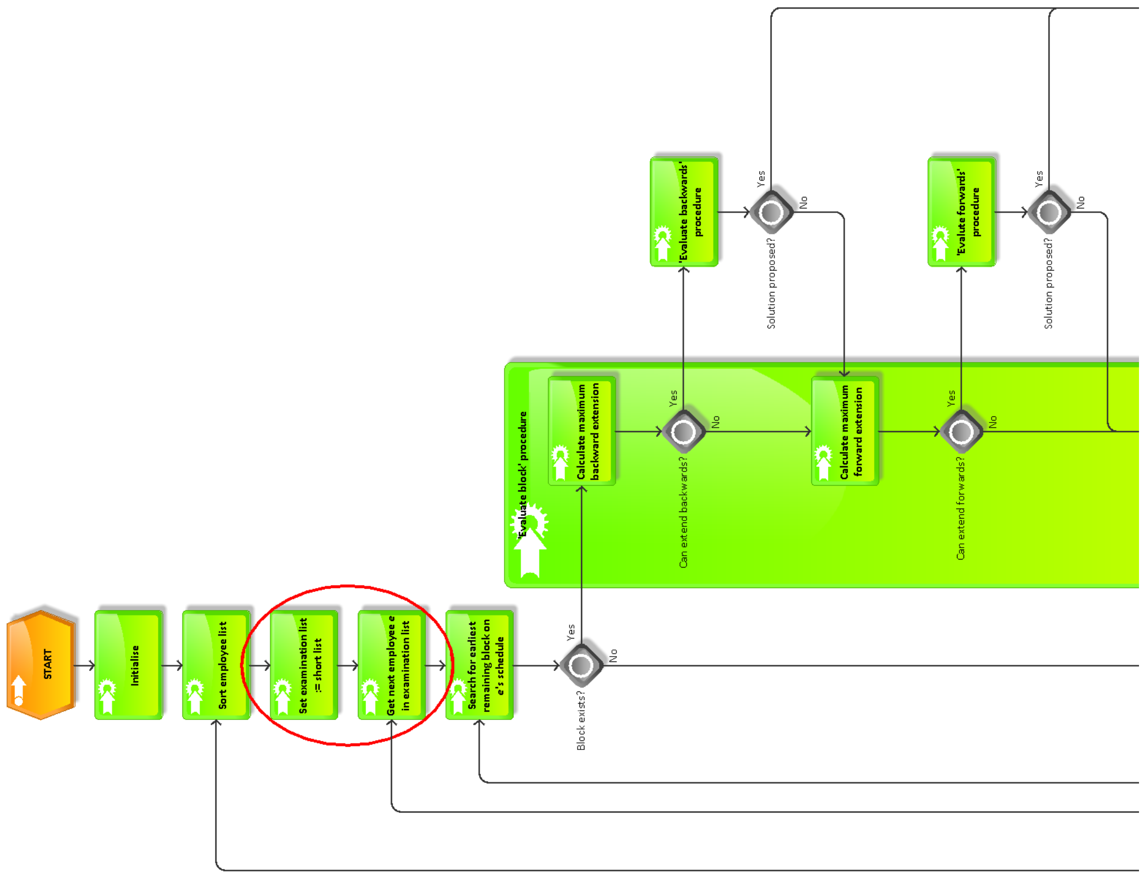

START
Initialise
Sort employee list
Set examination list := short list
Get next employee e in examination list
Search for earliest remaining block on e's schedule
Block exists?
Yes
No
'Evaluate block' procedure
Calculate maximum backward extension
Can extend backwards?
Yes
No
'Evaluate backwards' procedure
Solution proposed?
Yes
No
Calculate maximum forward extension
Can extend forwards?
Yes
No
'Evalute forwards' procedure
Solution proposed?
Yes
No

Figure 6.11: Proposed flowchart outlining modified heuristic algorithm - upper section.

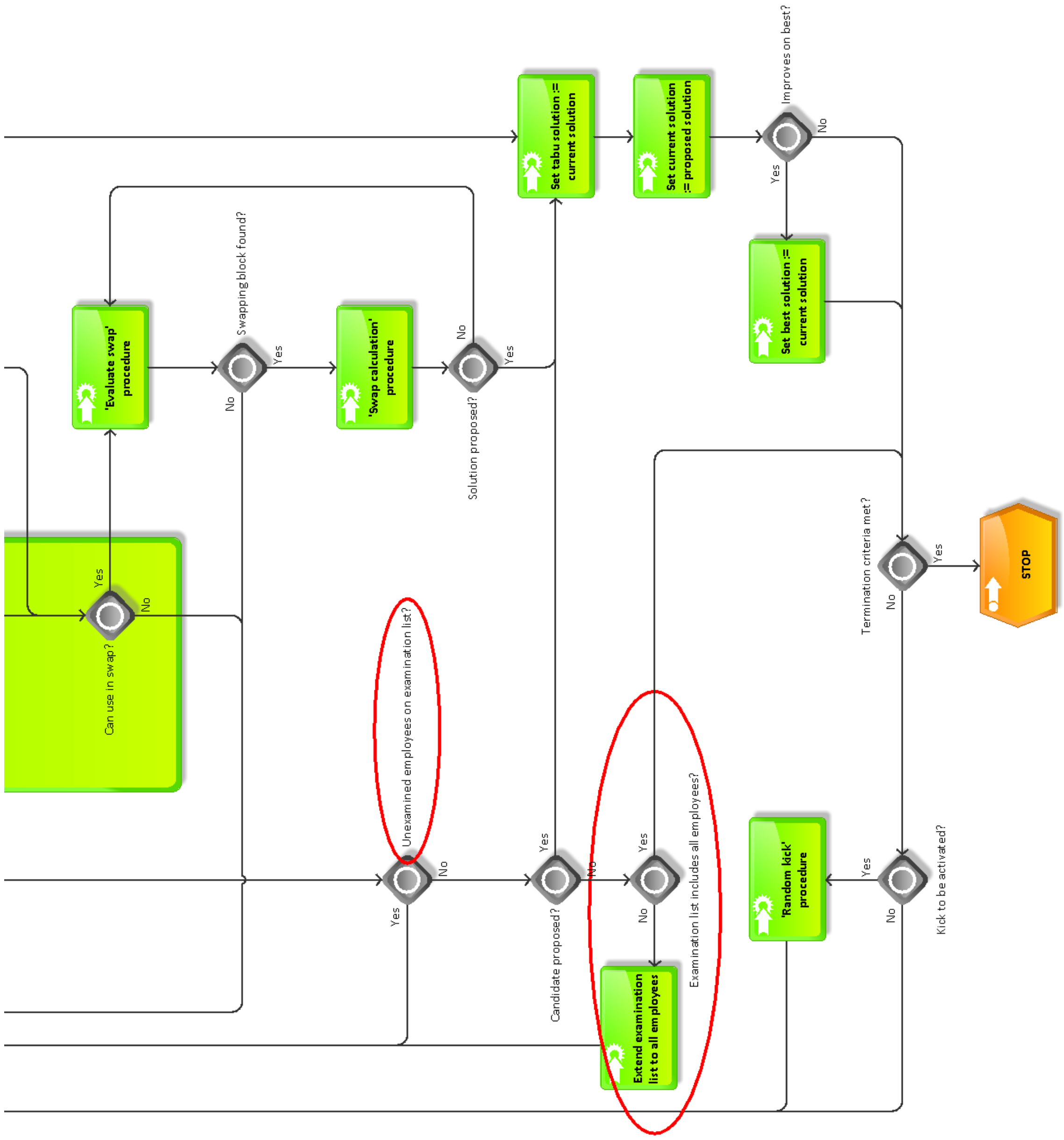

Set tabu solution := current solution
Set current solution := proposed solution
Improves on best?
Set best solution := current solution
'Evaluate swap' procedure
Swapping block found?
'Swap calculation' procedure
Solution proposed?
Can use in swap?
Unexamined employees on examination list?
Candidate proposed?
Examination list includes all employees?
Extend examination list to all employees
'Random kick' procedure
Kick to be activated?
Termination criteria met?
STOP
Yes
No

Figure 6.12: Proposed flowchart outlining modified heuristic algorithm - upper section.

6.6 Using Heuristics to obtain an Initial Solution

Having developed the heuristic procedure as described in section 6.5 above, it became apparent that a mechanism based on the same principles could be used to find an initial solution to the problem. As with using the Task-Based formulation as an approximation (section 6.4), it could not be predicted in advance the quality of these initial solutions; however, it may have the advantage over the Task-Based Approximation approach in that it could be readily programmed in an open-source language such as C++, rather than relying on commercial solvers such as FICO Xpress.

Below we set out our proposed procedure for finding an initial solution using heuristics, before going on to discuss how this new algorithm fits in with the testing scheme described in section 6.5.4 above. Note that the full code of the algorithm as implemented in FICO Xpress can be found in appendix section E.2.6.

6.6.1 Algorithm for a Heuristic Initial Solution

As a reminder, there are two situations in which the scheduling process will be required in our problem. The first of these is at the start of a new planning week, when planners will have in place a partial schedule covering the first 12 weeks of the 13-week planning horizon, which will have been created during the previous planning week. The problem in this case is to update the schedule to cover all required roles in Week 13, as well as fixing any other infeasibilities which may have arisen in the meantime. Alternatively, after the scheduling process has been carried out for the planning week, new information may arrive which makes the current schedule infeasible. In this case, the problem is only to fix these infeasibilities.

This procedure uses some of the definitions and principles of the heuristic algorithm described in section 6.5 earlier, and attempts to strike a balance between making the smallest number of changes, and not making expensive changes. The algorithm identifies vacancies, which can be considered to exist in blocks, and looks to fill these. The cheapest way to do this is often to extend an employee's assignment backwards or forwards from before or after the vacant block if this is possible. If this is not possible, the algorithm will seek to fill the gap with another employee. However, as it is rarely cost effective to have employees working in short blocks (because of the transportation costs), if the vacant block is particularly short then efforts may be made to clear a larger block which will be cheaper to fill, even if this creates more changes. Agency crew will generally only be used as a last resort, unless it is clearly the cheapest option for that vacancy.

As with the main heuristic algorithm, the details of the algorithm are not given here. Instead, a step-by-step procedure, starting with the main programme and working inwards, can be found in the appendix in section C.4, while full details of the FICO Xpress code used can be found in appendix section E.2.6.

6.6.2 Implications for Testing Plans

The heuristic method described above provides an alternative solution method for the Time-Windows problem, and the quality of these solutions will be assessed as part of the results given in section 6.8. The primary contribution of the above algorithm is however that it provides an alternative method of finding an initial solution which can be used in the main heuristic algorithm.

Before testing was carried out it was not known how the quality of these initial solutions would compare to those found by the Task-Based Approximation approach, and in any case a better quality initial solution is no guarantee of better solutions being found by the heuristic algorithm. Therefore as indicated in section 6.5.4.13 above, we propose to use this heuristic initial solution as well as the Task-Based Approximation, and as a result double the number of permutations which would otherwise have been tested for each dataset.

6.7 Possible Alternative - Column Generation

An alternative approach to those discussed in sections 6.3 to 6.6 for solving the Time-Windows formulation is to use Column Generation. This is a widely used technique in the literature on crew scheduling and crew recovery problems, especially when applied to airline crew problems, and essentially operates by considering only a section of the problem at a time. This has the advantage of reducing the problem size, freeing memory space and reducing the amount of time needed to search for optimal solutions as compared to the branch-and-cut approach used by the default FICO Xpress approach (as discussed in section 6.3). It is also an exact method, and the algorithm if allowed to do so will eventually find the true optimal solution to the problem.

There are also potential drawbacks to the approach. In particular, depending on the problem such an approach can still be time consuming to run, which may preclude it from being applied in our problem setting, or may at least necessitate modifications. It was also considered that the time needed to code and fine-tune a Column Generation algorithm would be prohibitive within the time limits of this PhD project. Therefore the solution approach presented here is given as a suggestion for future work, rather than as a method which has been implemented and tested.

6.7.1 Proposed Approach

Column generation operates by breaking down the problem into numerous *sub-problems*, linked together by a *master-problem*. Often for crew scheduling applications, the sub-problems are solved to find rosters for individual employees which will potentially be helpful for minimizing the objective function - these are termed *reduced cost* rosters and in more general Linear Programming terms can be thought of as reduced-cost columns. The role of the master-problem in this case is to select a subset of the available rosters such that roles are covered and the objective is minimized. The algorithm is iterative, with new reduced-cost rosters being searched for after each solution of the master-problem, and terminating with an optimal solution when no reduced-cost rosters can be found.

As stated earlier, there may be concerns with the running time of a Column Generation algorithm, which is of particular interest in our case when we are looking to find solutions quickly. Attention has however been given in the literature to the need to speed up the solution processes. For example, when solving a train driver recovery problem Rezanova and Ryan (2010) use the technique of initially considering a reduced problem, focussing only on affected time points and crew members and those who may be useful in the solution. In their case, if no solution can be found for the restricted problem then other time windows or other employees can be added to the problem. Note that in our case, since agency crew are (almost) always available, it is highly unlikely that *no* solution will be found; therefore, to use this technique we would require a different criterion to trigger expansion of the problem, such as a cost limit for example. This technique is not explored further here, but is left as a suggestion for future work.

Alternatively, Gamache et al. (1999) propose an algorithm for an airline crew scheduling problem which speeds up the solution process by only solving the LP relaxation of the master problem at each iteration, and periodically running a branch-and-bound heuristic to fix some variable values and reduce the problem size. This can be seen in Algorithm 6.2 below. While this may speed up the solution process, the technique of fixing certain

Algorithm 6.2 Outline solution algorithm for Gamache et al. (1999)

```
define an initial set of columns (i.e. rosters)
repeat
  repeat
    solve an LP relaxation of the master-problem
    use the sub-problem to search for reduced cost columns
  until no reduced cost columns are identified
  use a branch-and-bound-type heuristic to fix the values of certain variables {this re-
  duces the size of the master problem}
  eliminate any columns rendered incompatible
until an integer solution is found
```

variable values does prevent the algorithm from finding a guaranteed optimal solution. This however is not a serious problem in our setting, as discussed previously. In addition, the method described also leaves open the possibility of producing multiple solutions from one run of the algorithm, and therefore the Column Generation approach we propose here will adapt the ideas of the Gamache et al. (1999) approach.

6.7.2 Formulating the Master-Problem

Using the ideas set out above, we will now discuss how a master-problem could be created from the formulation presented for the Recovery-Type Time-Windows problem as given in section 6.1.2. We must begin by giving some additional definitions, before given the formulation.

6.7.2.1 Additional definitions

Firstly, since we are concerned with allocating rosters rather than individual weeks in a given task, the allocation decision variables $\hat{x}_{ijt}$ are no longer appropriate. Instead, we will say that at a given stage of the algorithm we have a current set P of rosters being considered. Then for all employees $i \in E$ and rosters $p \in P$, we can define a new decision variable y_{ip} which should take the following values:

$$y_{ip} = \begin{cases} 1 & \text{if employee } i \text{ is assigned to roster } p \\ 0 & \text{otherwise} \end{cases}$$

For the purposes of ensuring all costs are accounted for, we must also identify some subsets of P to relate rosters specifically to agency crew. We must ensure that all roles to which agency crew are assigned at time zero are assigned a roster (even if this is an empty roster), in order that this employee's departure from the role is taken into account. Therefore, for all $j \in J$ such that $\sigma_j = 1$ we say that there is a set of rosters P_j where $P_j \subseteq P$.

In addition, we must also define some data about these rosters. These would be calculated either as part of the sub-problem or as a subsequent step before the next solution of the master-problem, as discussed below. Firstly, we must have a cost of assigning employee i to roster p, which we will define as c^R_{ip}. Secondly, to know whether all roles are covered as required, we must know which roles each roster covers, which is given by

$$z^*_{pjt} = \begin{cases} 1 & \text{if roster } p \text{ includes performing role } j \text{ in time period } t \\ 0 & \text{otherwise} \end{cases}$$

for all rosters $p \in P$, roles $j \in J$ and times $t \in \{1, \ldots, T\}$. Finally, we need to identify which employees can be assigned to a given roster:

$$\epsilon_{ip} = \begin{cases} 1 & \text{if employee } i \text{ is eligible to be assigned to roster } p \\ 0 & \text{otherwise} \end{cases}$$

which is defined for all employees $i \in E$ and rosters $p \in P$. Note that by definition $\epsilon_{m+1,p} = 1$ for all rosters p belonging to one of the sets P_j.

6.7.2.2 Formulation

Using these values, and with other quantities as defined in sections 6.1.1 and 6.1.2 previously, we can state the formulation of the master problem as follows:

$$\min \sum_{\forall i,p} c^R_{ip} y_{ip} \tag{6.179}$$

subject to:

$$\sum_{\forall i,p} \epsilon_{ip} z^*_{pjt} y_{ip} = a_{jt} \qquad \forall j, t \tag{6.180}$$

$$\sum_{p \in P} y_{ip} \leq 1 \qquad \forall i \in E_R \tag{6.181}$$

$$\sum_{p \in P_j} y_{m+1,p} = \sigma_j \quad \forall \text{j such that } \sigma_j = 1 \tag{6.182}$$

$$y_{ip} \in \{0, 1\} \qquad \forall i, p \tag{6.183}$$

Where the objective function (6.179) looks to minimize the overall cost of assigning the rosters to the employees, with all costs (transportation costs, working costs, penalty costs, and undertime and overtime costs) embedded in the new c^R_{ip} cost coefficients. Equations (6.180) ensures all roles are covered when they require to be, in much the same way as equations (6.39) in the recovery-type cost-minimization problem; while constraint set (6.181) ensures that each employee can be assigned at most one roster in a similar way to forbidding the assignment of overlapping tasks with constraints (6.40) previously. Note that we assume employee $m+1$ (i.e. the agency employee) *can* be assigned multiple rosters - this is because we assume that multiple agency crew are available, and hence more than one can be brought in if more than one is required in a given time period.

Constraint set (6.182) has no equivalent in the recovery-type formulations given earlier, and has been added to ensure that each agency employee present in the schedule at the start of the planning period is assigned a roster. This is necessary even if the agency employee is departing ahead of the first week and has no further assignments, since this departure must be taken into account in the cost calculation. Finally, constraints (6.183) ensure the correct definition of decision variables.

6.7.3 The Sub-Problem

The aim of the sub-problem is to find rosters which firstly are feasible, and which can be added to set P to potentially allow a better solution to be found when the master-problem is next solved. The mathematical formulation of the sub-problem will comprise the constraints not covered by the master-problem, i.e. those pertaining to roster feasibility; however, we will only consider one employee at a time, reducing the number of constraints and therefore the problem size. Firstly however we must give some additional definitions.

6.7.3.1 Additional definitions

We must firstly define new decision variables for the sub-problem. The decision is now not (directly) whether an employee works in a role, boards a vessel, etc. at time t, but whether the roster under construction involves working, boarding, etc. Therefore, for a new roster p' we will have:

$z^W_{p'jt}$ as the allocation decision variable, indicating the assignments contained in roster p', such that

$$z^W_{p'jt} = \begin{cases} 1 & \text{if roster } p' \text{ entails working in role } j \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

$z^B_{p'kt}$ indicates whether roster p' involves boarding a vessel, such that

$$z^B_{p'kt} = \begin{cases} 1 & \begin{array}{l}\text{if roster } p' \text{ requires employee to board vessel} \\ k \text{ prior to an assignment in period } t\end{array} \\ 0 & \text{otherwise} \end{cases}$$

$z^D_{p'kt}$ similarly indicates whether roster p' involves departing a vessel, such that

$$z^D_{p'kt} = \begin{cases} 1 & \begin{array}{c}\text{if roster } p' \text{ requires employee to depart vessel } k \text{ prior to period } t \\ \text{(i.e. having completed duties there in period } t-1)\end{array} \\ 0 & \text{otherwise} \end{cases}$$

6.7.3.2 Formulation for feasible rosters

We can now set out the constraints which describe a feasible roster. The feasibility conditions however will be different for a regular employee $i \in E_R$ as opposed to an agency employee $i = m + 1$. For a regular employee $i \in E_R$ a new roster p' is feasible if it satisfies the following:

$$\sum_{j \in J} z^W_{p'jt} \leq 1 \qquad \forall t \qquad (6.184)$$

$$\sum_{k \in K} (z^B_{p'kt} + z^D_{p'kt}) \leq 1 \qquad \forall t \qquad (6.185)$$

$$s_{ik} + z^B_{p'k1} = z^D_{p'k1} + \sum_{j \in V_k} z^W_{p'j1} \qquad \forall k \qquad (6.186)$$

$$\sum_{j \in V_k} z^W_{p'j,t-1} + z^B_{p'kt} = z^D_{p'kt} + \sum_{j \in V_k} z^W_{p'jt} \qquad (6.187)$$

$$\hat{w}_{it} \geq \hat{w}_{i,t-1} + \sum_{j \in J} z^W_{p'jt} - w_i^{max} \left(1 - \sum_{j \in J} z^W_{p'jt} \right) \qquad \forall t \qquad (6.188)$$

$$w_i^{max} \hat{l}_{\lambda ijt} \geq \hat{w}_{i,t-1} - w_i^{max}(1 - z^W_{p'jt}) + z^W_{p'jt} - (\lambda - 1) \quad \forall j, t, \lambda \in \{1, \ldots, w_i^{max}\} \, (6.189)$$

$$\hat{r}_{it} \geq \hat{r}_{i,t-1} - \left(1 - \sum_{j \in J} z^W_{p'jt} \right) \qquad \forall t \qquad (6.190)$$

$$\hat{r}_{it} \geq (\rho_i - 1) \sum_{k \in K} z^D_{p'kt} \qquad \forall t \qquad (6.191)$$

$$\rho_i \left(1 - \sum_{j \in J} z^W_{p'jt} \right) \geq \hat{r}_{i,t-1} \qquad \forall t \qquad (6.192)$$

$$z^W_{p'jt} \in \{0, 1\} \qquad \forall j, t \qquad (6.193)$$

$$z^B_{p'kt}, z^D_{p'kt} \in \{0, 1\} \qquad \forall k, t \qquad (6.194)$$

$$\hat{l}_{\lambda ijt} \in \{0, 1\} \quad \forall j, t, \lambda \in \{1, \ldots, w_i^{max}\} \, (6.195)$$

$$\hat{w}_{it}, \hat{r}_{it} \geq 0 \qquad \forall t \qquad (6.196)$$

Where constraint set (6.184) ensures that the roster cannot contain more than one assignment at the same time point, and in conjunction with constraints (6.181) in the master-problem above, ensuring that employees cannot be given overlapping assignments as expressions (6.40) did in the recovery-type problem given in section 6.1.2. Constraints (6.185 - 6.187) are a rewritten form of constraints (6.41 - 6.44) in the original formulation, which is necessary to ensure that the board and depart variables $z^B_{p'kt}$ and $z^D_{p'kt}$ take *exactly* correct values rather than being greater than or equal it. Meanwhile, the other constraints (6.188 - 6.192) perform the same function as constraints (6.49), (6.50) and (6.54 - 6.56) in the original formulation. Finally, expressions (6.193-6.196) ensure the correct definition of all variables.

Meanwhile, a roster for agency crew must satisfy similar equations. However, in order to make correct calculations of working times, and therefore costs, it will be necessary to consider agency crew rosters on a role-by-role basis. Therefore, we say that a new agency crew roster p' relating specifically to role j' (which takes place on board vessel k', with $j' \in V_{k'}$) will be feasible if it satisfied the following:

$$z^B_{p'k'1} - z^D_{p'k'1} = z^W_{p'j'1} - \sigma_{j'} \tag{6.197}$$

$$z^B_{p'k't} - z^D_{p'k't} = z^W_{p'j't} - z^W_{p'j',t-1} \qquad \forall t \in \{2, \ldots, T\} \tag{6.198}$$

$$\hat{\alpha}_{j't} \geq \hat{\alpha}_{j',t-1} + z^W_{p'j't} - \alpha^{max}_{j'} z^D_{p'k't} \qquad \forall t \tag{6.199}$$

$$\hat{\alpha}_{j't} \geq z^W_{p'j't} \qquad \forall t \tag{6.200}$$

$$\alpha^{max}_{j'} \hat{l}_{\lambda,m+1,j't} \geq \hat{\alpha}_{j't} - (\lambda - 1) \quad \forall t, \lambda \in \left\{1, \ldots, \alpha^{max}_{j'}\right\} \tag{6.201}$$

$$z^W_{p'j't} \in \{0,1\} \qquad \forall t \tag{6.202}$$

$$z^B_{p'k't}, z^D_{p'k't} \in \{0,1\} \qquad \forall t \tag{6.203}$$

$$\hat{l}_{\lambda,m+1,j't}, \in \{0,1\} \quad \forall t, \lambda \in \left\{1, \ldots, \alpha^{max}_{j'}\right\} \tag{6.204}$$

$$\hat{\alpha}_{j't} \geq 0 \qquad \forall t \tag{6.205}$$

Where constraints (6.197 - 6.201) perform essentially the same role as (6.45), (6.46) and (6.51 - 6.53) in the recovery-type formulation in section 6.1.2, save that they relate to a specific role j'. Meanwhile, expressions (6.202 - 6.205) ensure the correct definition of all variables. Note that for the role j' to which roster p' relates, if $\sigma_{j'} = 1$ then we should add this new roster to the set $P_{j'}$, as defined earlier, as well as to set P when we return to the master-problem; if $\sigma_{j'} = 0$ then the roster should only be added to the main roster set P.

6.7.3.3 Calculating roster costs

As discussed above (section 6.7.2), there will be a cost c^R_{ip} associated with assigning employee $i \in E$ to roster $p \in P$. Note that we do not include the cost calculation as part of the sub-problem solution process directly, and can instead calculate $c^R_{ip'}$ once a new roster p' has been constructed. This can be done because the rosters generated in the sub-problem need to be *useful* for the master-problem, and therefore some consideration other than cost must be taken in order to generate a suitable range of feasible rosters. A suggested solution method is discussed in section 6.7.3.4 below.

Before that, however, we can show how the cost of a newly constructed rosters p' can be calculated for employee $i \in E$. This must be divided into three separate calculations: for a day-rate employee $i \in \{E_R \setminus G\}$; for a fixed contract employee $i \in G$; and for an agency employee $i = m + 1$.

Firstly, for a day-rate employee $i \in \{E_R \setminus G\}$ (i.e. a regular employee who is not on a fixed contract), we can use the values of the decision variables found for roster p' which satisfy equations (6.184 - 6.196) above. We will refer to the values found for this particular roster using the $\bullet^*$ notation, i.e. as $z^{W*}_{p'jt}$, $z^{B*}_{p'kt}$, $z^{D*}_{p'kt}$ and $\hat{l}^*_{\lambda ijt}$. These values can be used to evaluate the 'change' variables (denoted by $\bullet^{\pm}$) using the following equations:

$$z^{W*}_{p'jt} = x^*_{ijt} + (1 - 2x^*_{ijt})x^{\pm}_{ijt} \qquad \forall j, t \tag{6.206}$$

$$z^{B*}_{p'kt} = b^*_{ikt} + (1 - 2b^*_{ikt})b^{\pm}_{ikt} \qquad \forall k, t \tag{6.207}$$

$$z^{D*}_{p'kt} = d^*_{ikt} + (1 - 2d^*_{ikt})d^{\pm}_{ikt} \qquad \forall k, t \tag{6.208}$$

$$\hat{l}^*_{\lambda ijt} = l^*_{\lambda ijt} + (1 - 2l^*_{\lambda ijt})l^{\pm}_{\lambda ijt} \quad \forall j, t, \lambda \in \{1, \ldots, w^{max}_i\} \tag{6.209}$$

which have been adapted from equations (6.57 - 6.59) and (6.62) in the recovery-type formulation given earlier. Using the values of $x^{\pm}_{ijt}$, $b^{\pm}_{ikt}$, $d^{\pm}_{ikt}$ and $l^{\pm}_{\lambda ijt}$ calculated from these equations, we can evaluate the cost $c^R_{ip'}$ of assigning employee i to roster p' as:

$$c^R_{ip'} = \sum_{\forall k,t} \left(\phi^B_{ikt} b^{\pm}_{ikt} + \phi^D_{ikt} d^{\pm}_{ikt}\right) + \sum_{\forall j,t} \left(\phi^W_{ijt} x^{\pm}_{ijt} + \sum_{\forall \lambda} \phi^L_{\lambda ijt} l^{\pm}_{\lambda ijt}\right) \tag{6.210}$$

which has been adapted from the cost-minimization objective (6.38) given previously.

The necessary calculations are similar for fixed contract crew $i \in G$, except that we have an additional step of calculating undertime and overtime and the associated cost. Taking the values of the allocation variable $z^{W*}_{p'jt}$ which satisfies equations (6.184 - 6.196) above, we can use the following equations to calculate the undertime or overtime which would be applicable if employee i was assigned to rosters p':

$$\hat{u}_i = \max\left\{g_i - \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} z^{W*}_{p'jt}\right), 0\right\} \tag{6.211}$$

$$\hat{o}_i = \max\left\{\left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} z^{W*}_{p'jt}\right) - g_i, 0\right\} \tag{6.212}$$

Note that these have been adapted from equations (6.47) and (6.48) in the recovery-type formulation. The values calculated for $\hat{u}_i$ and $\hat{o}_i$ can then be used in equations (6.63) and (6.64) to calculate the undertime and overtime change, i.e.:

$$u^{\pm}_i = \hat{u}_i - u^*_i \tag{6.213}$$

$$o^{\pm}_i = \hat{o}_i - o^*_i \tag{6.214}$$

Then, using equations (6.206 - 6.209) from above to calculate $x^{\pm}_{ijt}$, $b^{\pm}_{ikt}$, $d^{\pm}_{ikt}$ and $l^{\pm}_{\lambda ijt}$ in the same way as for day-rate crew, we can calculate the cost $c^R_{ip'}$ of assigning employee i

to roster p' as:

$$c^R_{ip'} = \sum_{\forall k,t} \left(\phi^B_{ikt} b^{\pm}_{ikt} + \phi^D_{ikt} d^{\pm}_{ikt}\right) + \sum_{\forall j,t} \left(\phi^W_{ijt} x^{\pm}_{ijt} + \sum_{\forall \lambda} \phi^L_{\lambda ijt} l^{\pm}_{\lambda ijt}\right) + \left(c^U_i u^{\pm}_i + c^O_i o^{\pm}_i\right) \tag{6.215}$$

which, similarly to equation (6.210) for day-rate crew, has been adapted from the cost-minimization objective (6.38).

Meanwhile, for agency crew the calculation steps are a little different. In this case, we will have found a set of values $z^{W*}_{p'j't}$, $z^{B*}_{p'k't}$, $z^{D*}_{p'k't}$ and $\hat{l}^*_{\lambda,m+1,j't}$ which define a roster p' (for a given role j' and vessel k') by satisfying equations (6.197 - 6.205). These values can be used to calculate the 'change' variable (denoted by $\bullet^{\pm}$) values, using the following equations:

$$\begin{aligned}
z^{W*}_{p'j't} &= x^*_{m+1,j't} + (1 - 2x^*_{m+1,j't})x^{\pm}_{m+1,j't} && \forall t && (6.216)\\
z^{B*}_{p'k't} &= \beta^*_{j't} + (1 - 2\beta^*_{j't})\beta^{\pm}_{j't} && \forall t && (6.217)\\
z^{D*}_{p'k't} &= \delta^*_{j't} + (1 - 2\delta^*_{j't})\delta^{\pm}_{j't} && \forall t && (6.218)\\
\hat{l}^*_{\lambda,m+1,j't} &= l^*_{\lambda,m+1,j't} + (1 - 2l^*_{\lambda,m+1,j't})l^{\pm}_{\lambda,m+1,j't} && \forall t, \lambda \in \left\{1, \ldots, \alpha^{max}_{j'}\right\} && (6.219)
\end{aligned}$$

which have been adapted from equations (6.57) and (6.60 - 6.62) above. As with the regular crew, we can then use the values of $x^{\pm}_{m+1,j't}$, $\beta^{\pm}_{j't}$, $\delta^{\pm}_{j't}$ and $l^{\pm}_{\lambda,m+1,j't}$ calculated from these equations to calculate the cost $c^R_{m+1,p'}$ of assigning an agency employee $i = m+1$ to roster p' as follows:

$$c^R_{m+1,p'} = \sum_{\forall t} \left(\phi^{B_A}_{j't} \beta^{\pm}_{j't} + \phi^{D_A}_{j't} \delta^{\pm}_{j't}\right) + \sum_{\forall t} \left(\phi^W_{m+1,j't} x^{\pm}_{m+1,j't} + \sum_{\forall \lambda} \phi^L_{\lambda m+1,j't} l^{\pm}_{\lambda m+1,j't}\right) \tag{6.220}$$

which, as before, is adapted from the cost-minimization objective (6.38).

Therefore, using the method set out above we can calculate the cost of assigning any employee $i \in E$ to any newly generated roster p' for which they are eligible to be assigned.

6.7.3.4 Solving the sub-problem

There are several potential approaches to solving the sub-problem. For example, a MIP solver such as FICO Xpress could be applied directly to the formulation (with a suitable objective) to find rosters. This however could potentially run into the same problems as were faced when trying to apply the cost-minimization approach as described earlier (section 6.3.1), with each roster taking a length of time to find which is prohibitive especially when considering that the sub-problem must be run multiple times during the solution process. Instead, in a manner again similar to that used by Gamache et al. (1999), we suggest using a network representation, and generating rosters using a resource-constrained shortest path problem.

For this network representation, we must introduce arcs which will represent roles to which an employee can be assigned during a time window, as well as the boarding or departing of an employee to a vessel at a particular point in time. Note that these arcs will be directed arcs. The nodes meanwhile represent time-vessel points, meaning that there will be a node for each vessel $k \in K$ at each time point during the planning horizon. There will also be a dummy *rest* node at each time point, used when an employee is going onto or coming out from a rest period. These will be connected by *rest* arcs, allowing a path through the network if there are time periods for which the roster contains no working tasks. A path through the network therefore will represent a roster. More formally, we can construct the network as follows:

1. Create a source node θ^R and a sink node θ^S.

2. Create the time-vessel points:

   (a) If we are constructing a roster for regular crew $i \in E_R$, then for all vessels $k \in K$ and all time points $t \in \{0, \ldots, T\}$, create a node θ_{kt}.

   (b) If we are constructing a roster for agency crew $i = m + 1$, then for all time points $t \in \{0, \ldots, T\}$, create a single node $\theta_{k't}$.

3. For all time points $t \in \{0, \ldots, T\}$ also create a dummy rest node θ_{0t}.

4. Create the working arcs:

   (a) If we are constructing a roster for regular crew $i \in E_R$, then for each time period $t \in \{1, \ldots, T\}$ and each vessel k, create an arc $(\theta_{k,t-1}, \theta_{kt})$ for each task $j \in V_k$ such that $a_{jt} = 1$ and $e_{ijt} = 1$ (i.e. such that the role is required in time period t, and employee i is eligible to carry it out).

   (b) If we are constructing a roster for agency crew $i = m+1$, then for each time period $t \in \{1, \ldots, T\}$ create an arc $\left(\theta_{k',t-1}, \theta_{k't}\right)$ corresponding to role j' provided $a_{j't} = 1$ and $e_{m+1,j't} = 1$ (i.e. such that the role is required in time period t, and agency employees are eligible to carry it out).

   Selection of one of these arcs during construction of a path through the network will correspond to setting $z^W_{p'jt} = 1$ for the roster p' being considered.

5. Similarly, create a dummy rest arc $(\theta_{0,t-1}, \theta_{0t})$ for all times $t \in \{1, \ldots, T\}$.

6. Create boarding arcs:

(a) If we are constructing a roster for regular crew $i \in E_R$, then for each time period $t \in \{0, \dots, T-1\}$ and each vessel k, create an arc $(\theta_{0t}, \theta_{kt})$ corresponding to the employee boarding a vessel prior to carrying out a task in period $t+1$.

(b) If we are constructing a roster for agency crew $i = m+1$, then for each time period $t \in \{0, \dots, T-1\}$, create an arc $(\theta_{0t}, \theta_{k't})$ corresponding to an agency employee boarding vessel k' prior to carrying out a task in period $t+1$.

Selection of one of these arcs during construction of a path will correspond to setting $z^B_{p'k,t+1} = 1$ for the roster p' being considered. Note the offset of the time period indices between the network representation and the IP variables.

7. Similarly, create departure arcs:

(a) If we are constructing a roster for regular crew $i \in E_R$, then create an arc $(\theta_{kt}, \theta_{0t})$ for each time $t \in \{0, \dots, T-1\}$ and each vessel k.

(b) If we are constructing a roster for agency crew $i = m+1$, then create an arc $(\theta_{k't}, \theta_{0t})$ for each time $t \in \{0, \dots, T-1\}$.

These arcs correspond to the roster requiring an employee departing vessel k ahead of time period $t+1$, and therefore selecting such an arc during construction of a path will correspond to setting $z^D_{p'k,t+1} = 1$ for the roster p' being considered.

8. Create starting point arcs from the source node θ^R:

(a) If we are constructing a roster for regular crew $i \in E_R$, then if $s_{ik} = 1$ for some vessel k create an arc $\big(\theta^R, \theta_{k0}\big)$; or, if $s_{ik} = 0$ for all $k \in K$ then create an arc $\big(\theta^R, \theta_{0,0}\big)$.

(b) If we are constructing a roster for agency crew $i = m+1$, then if $\sigma_{j'} = 1$ then create an arc $\big(\theta^R, \theta_{k'0}\big)$; or, if $\sigma_{j'} = 0$ then create an arc $\big(\theta^R, \theta_{0,0}\big)$.

9. Finally, create ending arcs to the sink node θ^S:

(a) If we are constructing a roster for regular crew $i \in E_R$, then for each vessel $k \in K$, create an arc $\big(\theta_{kT}, \theta^T\big)$, and also create an arc $\big(\theta_{0T}, \theta^T\big)$.

(b) If we are constructing a roster for agency crew $i = m+1$, then create an arc $\big(\theta_{k'T}, \theta^T\big)$ and an arc $\big(\theta_{0T}, \theta^T\big)$.

Having constructed the network in this way, we must assign a cost value to each of these arcs. Using the dual values from the current master-problem solution, it is possible to calculate a dual cost associated with each arc. Details of this are not given here, but can be found in the paper by Gamache et al. (1999). The objective of the problem can then be stated as minimizing the sum of the dual costs of the selected arcs, such that the arcs form a path through the network from source to sink and the corresponding roster is feasible. The sum of these dual costs represent the ‘reduced cost’ of the roster.

For regular crew, the formation of a path is covered by equations (6.184 - 6.187), while for agency crew, this is covered by equations (6.197) and (6.198). Of greater concern are the constraints governing feasibility rules of the roster - equations (6.188 - 6.196) for regular crew; or (6.199 - 6.205) for agency crew. This can be dealt with by considering working length and rest requirements as ‘resources’ which can be aggregated along the selected path, and must take values within a certain range at all times. Paths can be constructed dynamically from the source node, ensuring these resources constraints are satisfied at all points. Such an algorithm could potentially be used to construct multiple reduced-cost rosters at a time for each employee.

6.7.4 Future Work

As indicated earlier, this section is given as an outline for possible future work. A possible reformulation has been given which could allow the recovery-type Time-Windows problem to be solved using a Column Generation approach. A solution method for both the master-problem and sub-problem has been proposed, but we acknowledge there may be other more efficient ways to approach this. Future work here could involve examining other potential decompositions of the problem, as well as more in-depth analysis and testing of possible solution methods to determine whether Column Generation can be applied here in an efficient, useful manner.

6.8 Further Computational Results

We now present the results of the computational tests on the Task-Based Approximation approach, the Heuristic algorithm, and the Heuristic Initial Solution approach discussed respectively in sections 6.4, 6.5 and 6.6. In this section, we discuss and compare the performance of the two different initial solution methods, which can of course be used as solution methods in themselves, in terms of the key measures of solution quality and running time. We also examine the test runs involving the Heuristic algorithm, and discuss the relative merits of the various settings used including the influence of the different initial solutions. Finally, in section 6.8.5 we examine the effects of the parameter values used during data generation, as in section 5.5.2.3 for the Task-Based problem.

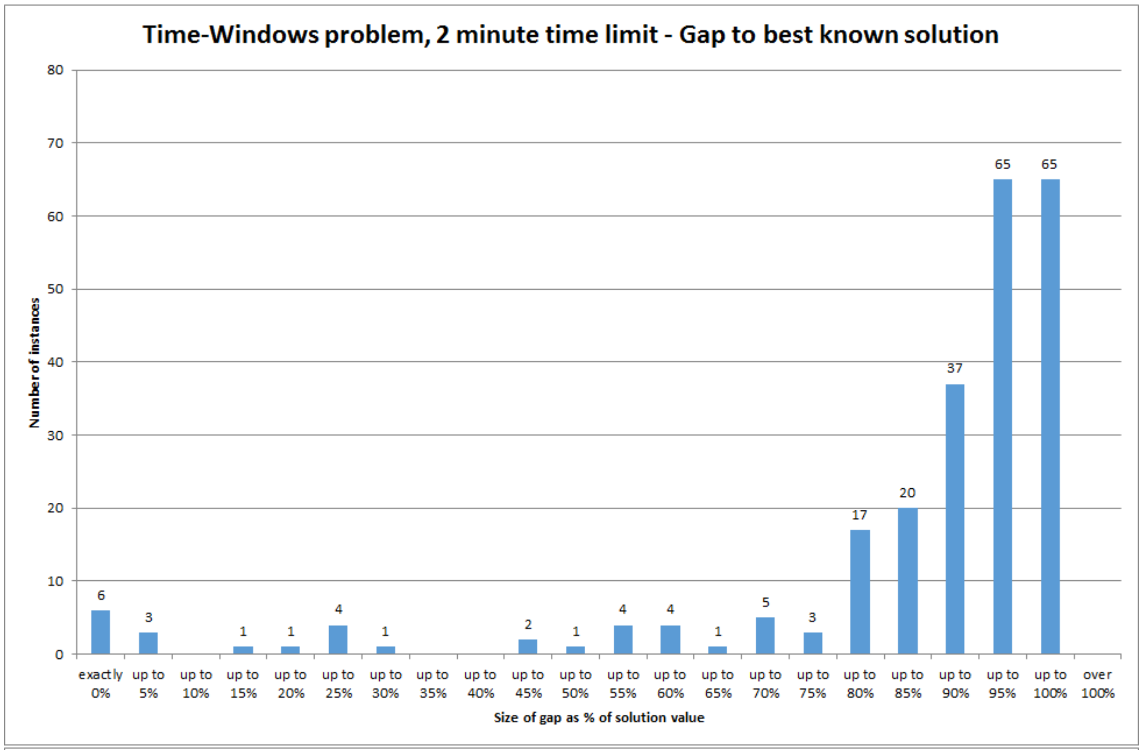
We begin however with some additional results relating to the initial computations presented in section 6.3, which will allow more detailed comparisons between the approaches used.

6.8.1 Additional analysis of Initial Results

Carrying out the computational tests as described in sections 6.4, 6.5 and 6.6, as well as the preliminary tests involved in developing these approaches, created numerous solutions for each instance in addition to those found in the initial computational tests. This allows for a measure of solution quality to be introduced as an alternative to the gap to the best known bound - the gap to the best known solution across all solution approaches used. This may be a more informative measure, since it is possible that the optimality gaps observed in the 'direct' test runs arise not because a better solution has not been found but because enough of the branch-and-bound tree has not been explored in the time available to sufficiently improve the best known bound. The gap to best known solution measure allows us to make a more practical judgement of solution quality, by measuring the solution found against the best which was actually achievable across all approaches.

We can begin by looking at the gap to the best known solution from the solutions found by the 'direct' solution approaches as described in section 6.3.1. Similar to the gap to best bound results presented in Figure 6.1 and Figure 6.2 for the two-minute and one-hour settings respectively, these results can be summarised in histograms as shown in Figure 6.13. We can see from the upper graph in the figure that even when compared to the best known solution rather than best bound, the two-minute settings obtain solutions with large gaps from the possible optimum. In this case, we see that 65 instances ($\approx 27.08\%$ of the total) had a solution with a gap of between 95% and 100% of the solution value away from the best known solution. Another 65 instances were found to have a gap of between 90% and 95%, meaning a majority of instances (130, or $\approx 54.17\%$ of the total) have a gap of over 90% from the best known solution. By comparison, the results with respect to the best known bound (Figure 6.1) had shown 135 instances (56.25% of the total) with a gap of between 95% and 100% of the solution value, and 62 instances (25.83% of the total) with a gap of between 90% and 95%. At the other end of the graph, it can be seen that the two-minute approach achieved the best known solution (i.e. a gap of zero) for 6 instances (2.5% of the total), although from Figure 6.1 it was already known that 5 of the instances had solved to optimality.

The lower graph in Figure 6.13 meanwhile shows the performance of the one-hour settings with reference to the best known solutions. As can be seen, 53 instances ($\approx 22.08\%$ of the total) achieved their best known solution in this test run, with 17 of these ($\approx 7.09\%$ of the total; and nearly one third of the 53) known to be optimal from the comparison with the best known bound in Figure 6.2. Of the instances which did not find the best known solution in the one-hour 'direct' test run, we see that there are more results towards the upper end of the graph with 119 instances ($\approx 49.58\%$ of the total) having a gap of between 45% and 85%. However, the spread can be considered to be quite even when we see that



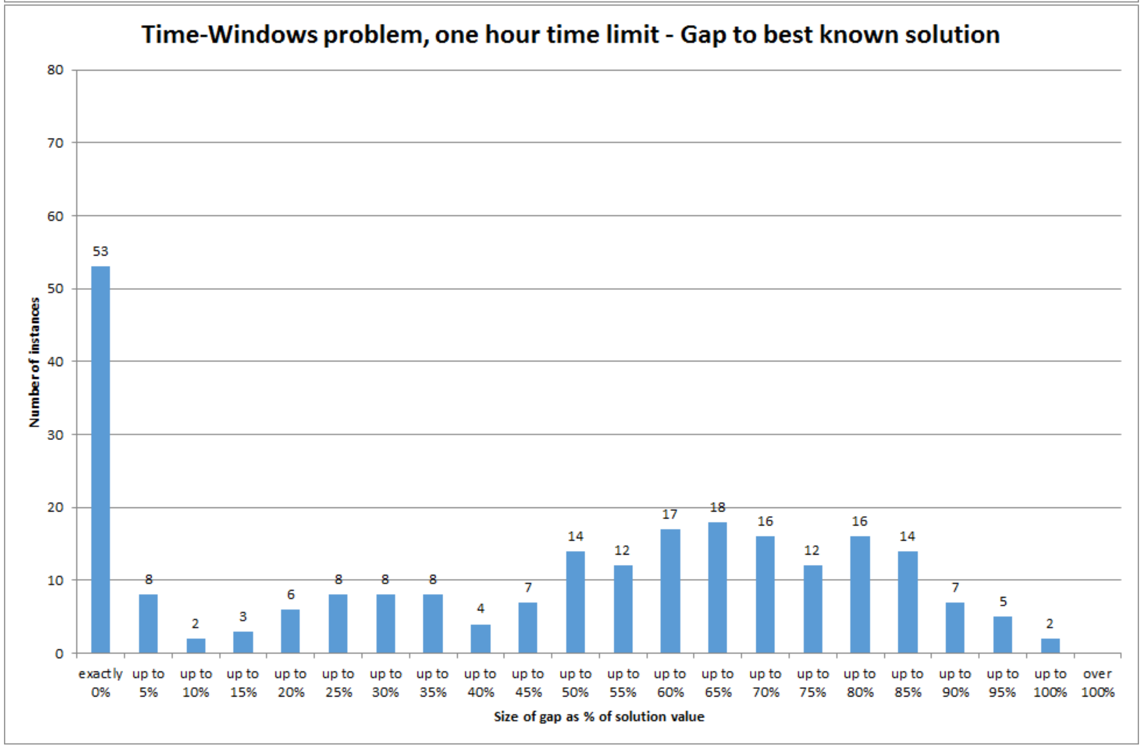


Figure 6.13: Histogram of percentage gaps to best known solution for 'direct' solution of Time-Windows instances using FICO Xpress, 2 minute time limit (top) and 1 hour time limit (bottom).

the most frequently occurring gap (other than the 53 instances with a gap of zero) is from 60% to 65% which accounts for only 18 instances (7.5% of the total).

We can also look at the gap to the best known solution for the solutions found using the change-minimization approach, as discussed in section 6.3.2. In that section, we gave results for the gap to the best known bound for the lowest cost solution found by the change-minimization approach (Figure 6.5); correspondingly, we can summarise the gaps to the best known solution for the lowest cost solutions in Figure 6.14. We can see from this

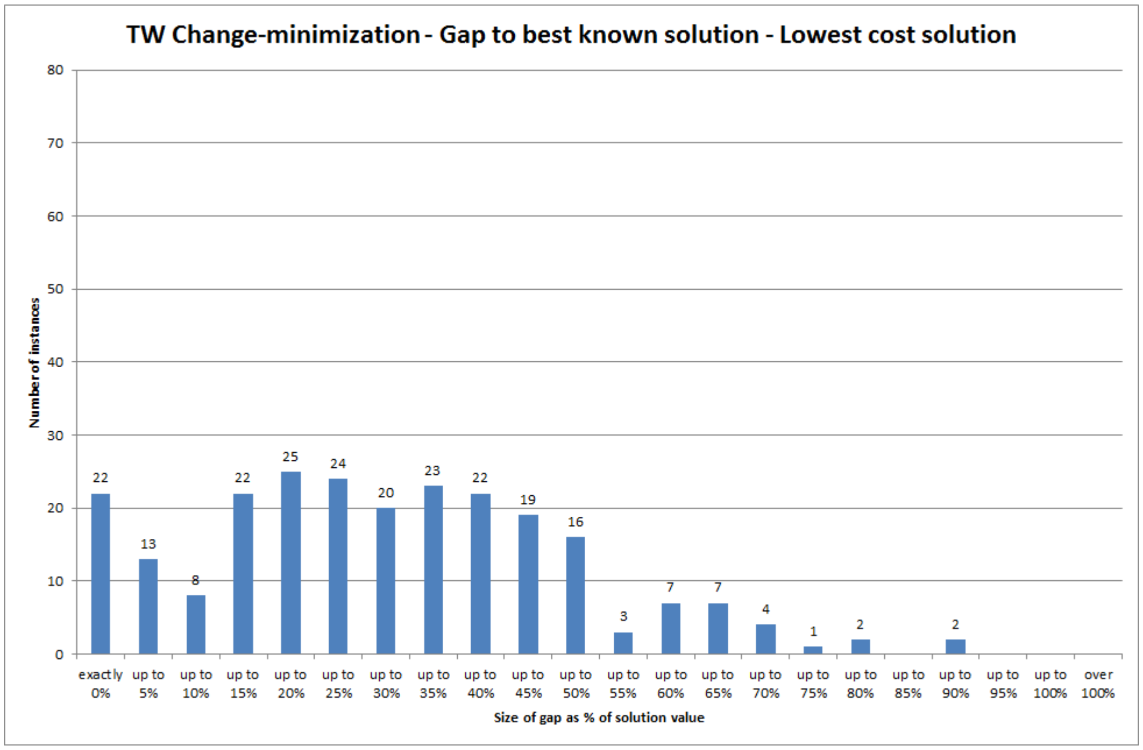


Figure 6.14: Gap to best known solution for lowest-cost solution found in two minute Change-minimization test run for the Time-Windows formulation.

graph that 22 of the instances ($\approx$ 9.17% of the total) achieved their best known solution under this method, of which only one of these is known to be an optimal solution from the comparison to the best known bound. While the tendency when comparing to the best known bound was to have gaps towards the upper end of the scale, we see that the sizes of gaps to the best known solution tend to be at the lower end of the scale. Including those with a gap of zero, nearly half of the instances (114, i.e. 47.5% of the total) had a lowest cost solution within a 25% gap of the best known solution. Exactly 100 instances ($\approx$ 41.67% of the total) saw a gap of between 25% and 50%, while only 26 instances ($\approx$ 10.83% of the total) saw a gap greater than 50%, with the largest value being a gap of

≈ 87.53%.

Finally, we can show the gap between the non-cost-constrained solution found by the change-minimization method and the best known solution overall. These can be summarised in Figure 6.15. As might be expected, and similar to the gap to the best bound

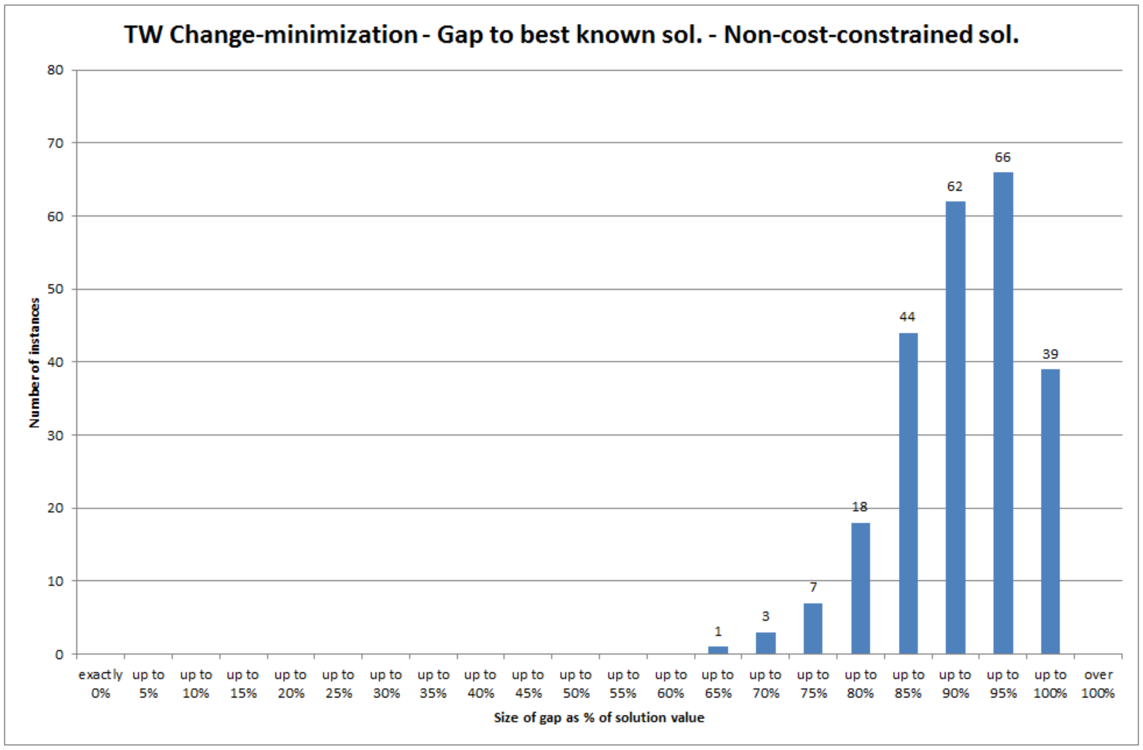


Figure 6.15: Gap to best known solution for non-cost-constrained solution found in two minute Change-minimization test run for the Time-Windows formulation.

results shown in Figure 6.6, the gaps displayed in this graph are all very large. The smallest gap between the non-cost-constrained solution and the best known solution is ≈ 60.56%, while the mean value is ≈ 88.00%. We can see that 39 instances (16.25% of the total) fall into the category of a 95% to 100% gap, while the most common category is a gap of 90% to 95% which accounts for 66 instances (27.5% of the total), closely followed by 62 instances (≈ 25.83% of the total) in the 85% to 90% category.

6.8.1.1 Summary of additional analysis

In summary, there are no especially significant implications of these additional comparisons in themselves; instead, these mainly provide a basis for additional comparison with the results presented in the remainder of section 6.8. It is worth noting however that the two-minute cost-minimization and non-cost-constrained change-minimization solutions are in general a long way from the best solution values obtained across all methods. Of the four results given here, the one-hour cost-minimization approach produces the highest number of instances obtaining their best known solution; however, the lowest cost change-minimization solutions have a higher number of instances which are closer to the best known solution suggesting it is actually performing with more consistency with regard to this measure.

6.8.2 The Task-Based Approximation approach

We can now move on to discussing the results of the further computational tests themselves. We first discuss the results of using the Task-Based formulation to simplify and solve the Time-Windows problem, as discussed in section 6.4. We can judge the success of this approach using similar measures to those used previously - the gaps between the solution and the best known bound and best known solution for the instance; the improvement in solution cost relative to the 'direct' and change-minimization approaches; the running time; and the number of changes. Also discussed for other solution methods is the number of solutions found; however for this particular approach, only one solution was generated for each instance and therefore no further results are given for this measure. We do note though that in principle this solution approach could be adapted so that it could produce multiple solutions.

The first measure we discuss is the gap to the best known bound for each instance, which is summarised in a histogram in Figure 6.16. It can be seen that the results are grouped towards the upper end of the graph, with over half the instances (124 out of the 240, i.e. $\approx 51.67\%$) showing a gap to the best bound greater than 85%. We note that there is a single instance which has a gap of greater than 100%, which correspondingly is the only instance for which the best bound found (in the one-hour direct run, as discussed in section 6.3.1) was negative. Only 12 instances (5% of the total) found a solution with a gap to the best bound not greater than 50%, with the smallest gap being $\approx 16.20\%$. The mean gap size was $\approx 81.25\%$, with the median quite close to this at $\approx 85.23\%$.

Similar to this, as discussed in section 6.8.1 we can also judge the solution cost in terms of the gap to the best known solution for each instance. This can be summarised in a histogram as in Figure 6.17. This graph still has the instances mainly towards the upper end, but less so than when comparing to the best known bound. We need to take into account gaps down to over 65% before we cumulatively have included more than half the instances (125, or $\approx 52.08\%$ of the total). A total of 20 instances ($\approx 8.33\%$ of the total) found a solution with a gap of less than 50% to the best known solution, with the smallest gap being $\approx 11.81\%$. Calculating the average gap sizes for this measure gave a mean gap of $\approx 65.54\%$ and a median of $\approx 65.66\%$.

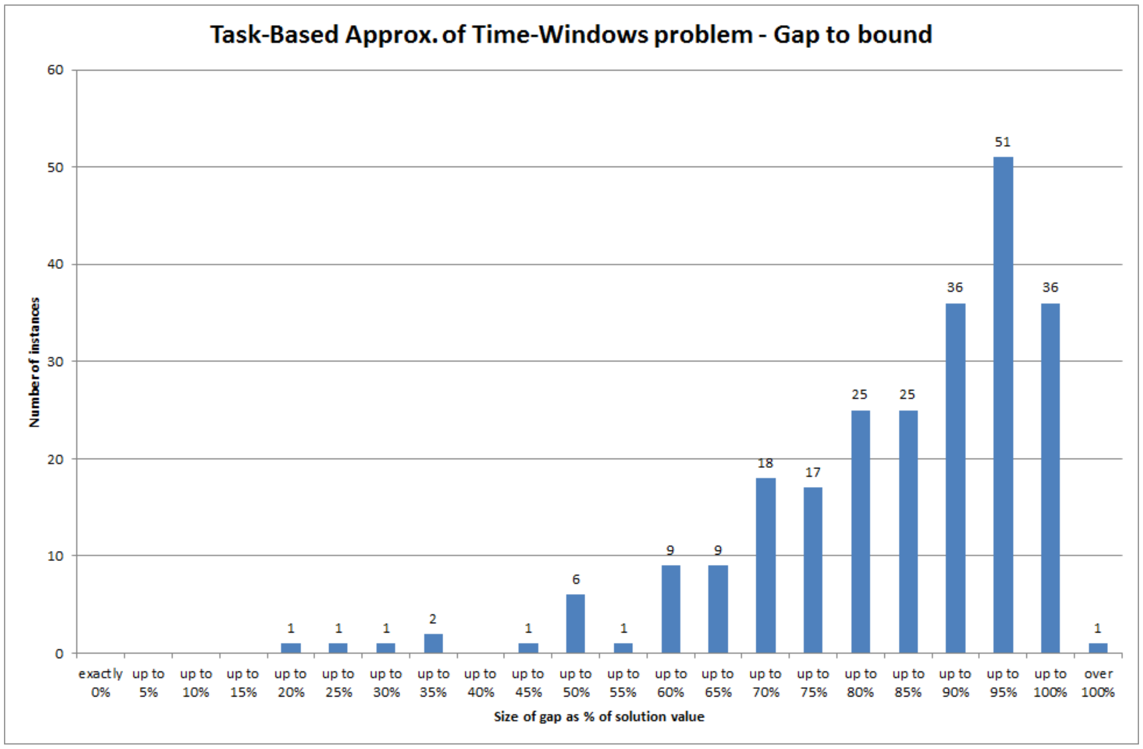


Figure 6.16: Histogram of percentage gaps to best known bounds for solutions found using the Task-Based formulation as an approximation.

To put these gaps into context, we can compare the solution values found using the Task-Based Approximation to those using the methods discussed in section 6.3. Firstly, we compare with the solutions found using 'direct' solution approach, with both the two-minute and one-hour settings. This comparison can be considered in terms of the improvement shown by the Task-Based Approximation solution, i.e. subtracting the Task-Based Approximation solution value from the 'direct' approach solution value, and represented as a percentage of the respective direct approach solution value. A summary of these percentage improvements can be seen in the histograms in Figure 6.18. The upper graph shows that for the majority of instances (216 out of the 240, i.e. 90%), the Task-Based Approximation approach found a solution which was an improvement on the two-minute direct approach solution. In fact, for exactly half of the instances (i.e. 120 out of 240), the solution found using the Task-Based Approximation improved on the two-minute direct solution by over 70%. Meanwhile, 23 of the instances ($\approx 9.58\%$) show a negative improvement, implying that the value of the Task-Based Approximation solution was greater than that found by the two-minute direct approach. Over half of these (12 instances, 5% of the total) had an 'improvement' of less than -100%, which shows the Task-Based Approxima-

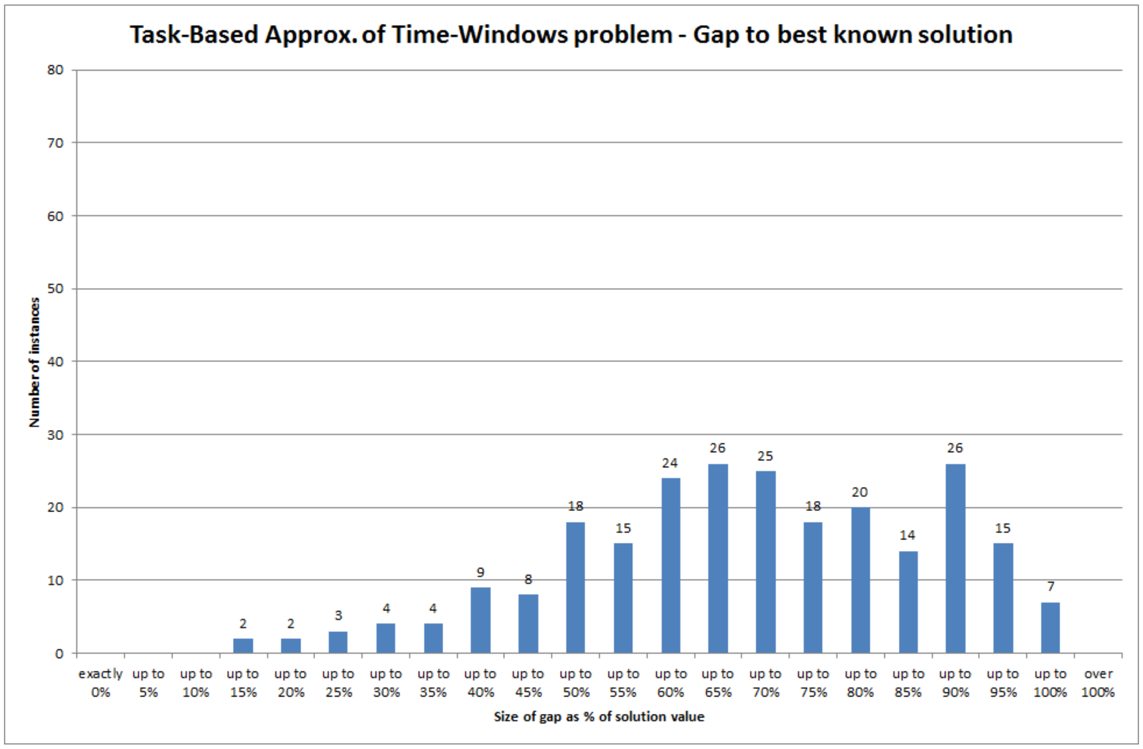


Figure 6.17: Histogram of percentage gaps to best known bounds for solutions found using the Task-Based formulation as an approximation.

tion solution value was more than double the two-minute direct approach value. Finally, at the centre of the x-axis it can be seen that a single instance resulted in the identical solution value for both approaches.

As might be expected, a much different pattern is observed in the lower histogram of Figure 6.18, comparing the Task-Based Approximation solution value to the solution found by the 'direct' approach when given a one-hour time limit. Here, only 67 instances ($\approx$ 27.92% of the total) show a positive improvement, with smaller percentage improvements having a greater frequency than large improvements (for example, 47 of the 67 instances which show positive improvement have an improvement of less than 50%). The remainder of the instances result in a negative improvement, showing that for the most part the Task-Based Approximation solution has a higher value than that found by the one-hour direct approach. For 92 instances ($\approx$ 38.33% of the total, and over half of the 173 which have a negative improvement) the improvement is less than -100%, indicating that for these instances the value of the Task-Based Approximation solution is more than twice that of the solution found using the one-hour direct approach.

We should also compare the Task-Based Approximation solutions with the solutions

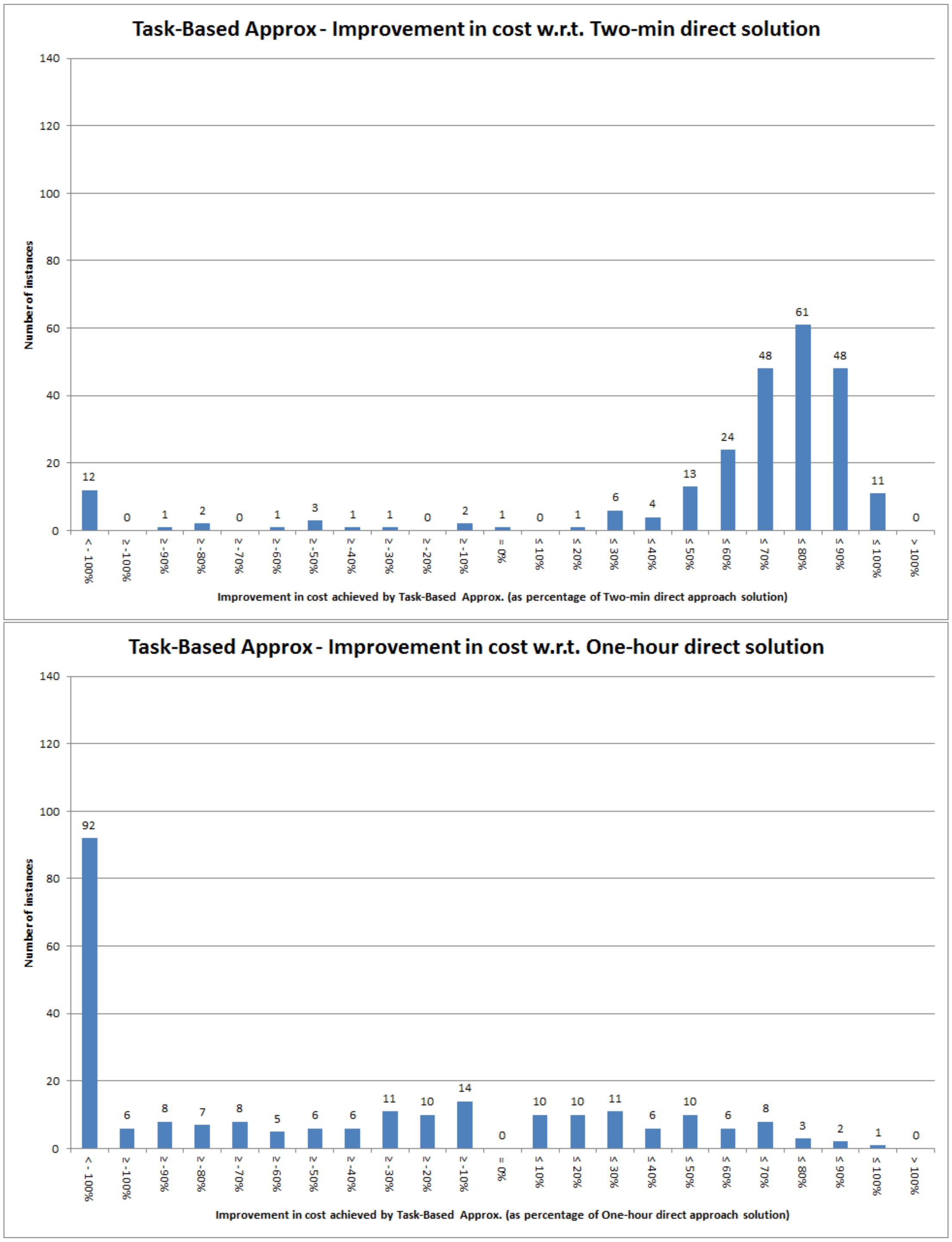


Figure 6.18: Histograms summarising improvement in the solution for each instance using the Task-Based Approximation approach compared to the solution cost for the two-minute (top) and one-hour (bottom) ‘direct’ approach, shown as a percentage of the respective direct approach solution value.

found by the change-minimization approach, again as discussed in section 6.3. Similarly to above, we can calculate an improvement shown by the Task-Based Approximation approach by subtracting its solution value from the (two-minute) change-minimization solution value for each instance. Dividing by the change-minimization value gives a percentage improvement for each instance, which can be summarised in Figure 6.19. The image here is even

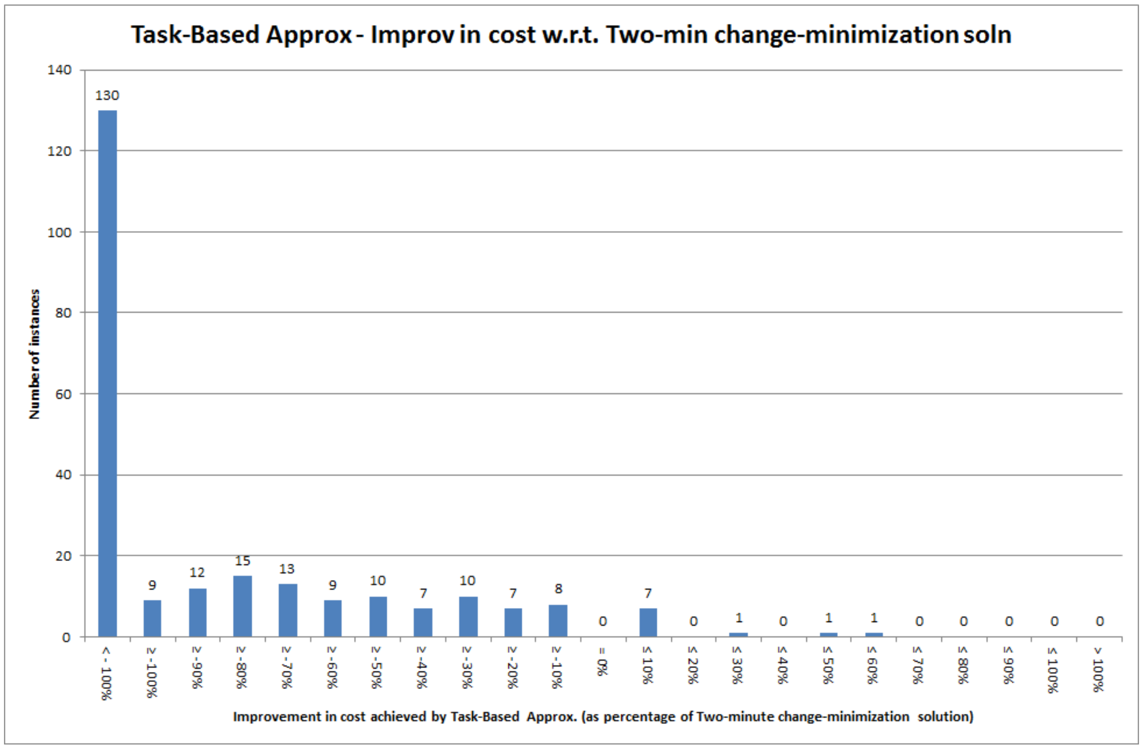


Figure 6.19: Histogram summarising improvement in the solution for each instance using the Task-Based Approximation approach compared to the solution cost for the two-minute change-minimization approach, shown as a percentage of the change-minimization solution value.

more extreme than that shown by the lower graph of Figure 6.18, with only 10 instances ($\approx 4.17\%$ of the total) where the Task-Based Approximation approach was able to find a solution which improved on that found by the change-minimization approach. Of these, 7 of the instances resulted in an improvement of less than 10%. Of the other instances, only 42 (17.5% of the total) show an improvement of greater than -50%, i.e with the value of the Task-Based Approximation solution no more than half as much again (or one-and-a-half times) the value of the change-minimization solution; while 58 instances ($\approx 24.17\%$ of the total) show an improvement of between -50% and -100%. The majority of instances however (130 instances, $\approx 54.17\%$ of the total) show an improvement of less than -100%, indicating that for over half the instances the Task-Based Approximation solution has a cost more than double that of the change-minimization solution.

The results relating to the solution cost indicate that while the Task-Based Approximation tends to outperform the 'direct' solution approach with a two-minute time limit, it is in turn largely outperformed by the change-minimization approach. It also tends to find inferior solutions to the one-hour direct approach setting, although this is of less concern since a one-hour time limit is not of use in practice. An advantage to the Task-Based Approximation approach can be seen however when running time is taken into account. The running times for each instance, including the time to convert the problem into the Task-Based format and convert the solution back to the Time-Windows representation, can be summarised in Figure 6.20. Note the jump in the scale of the graph at the upper end of

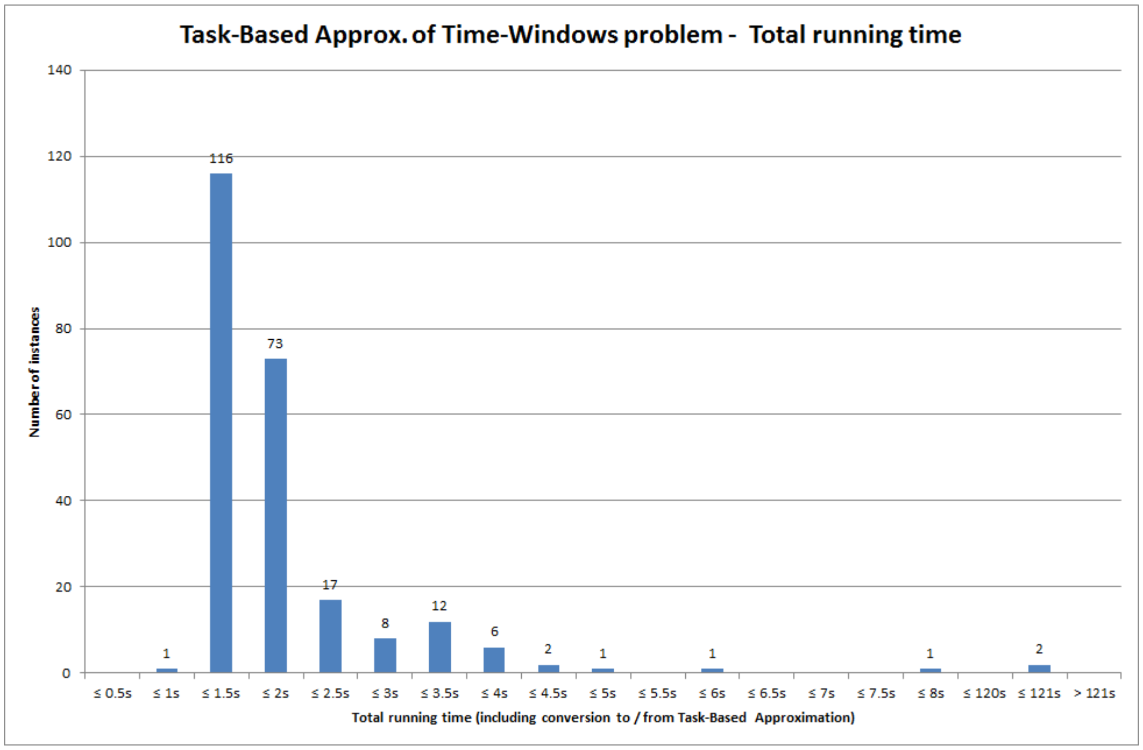


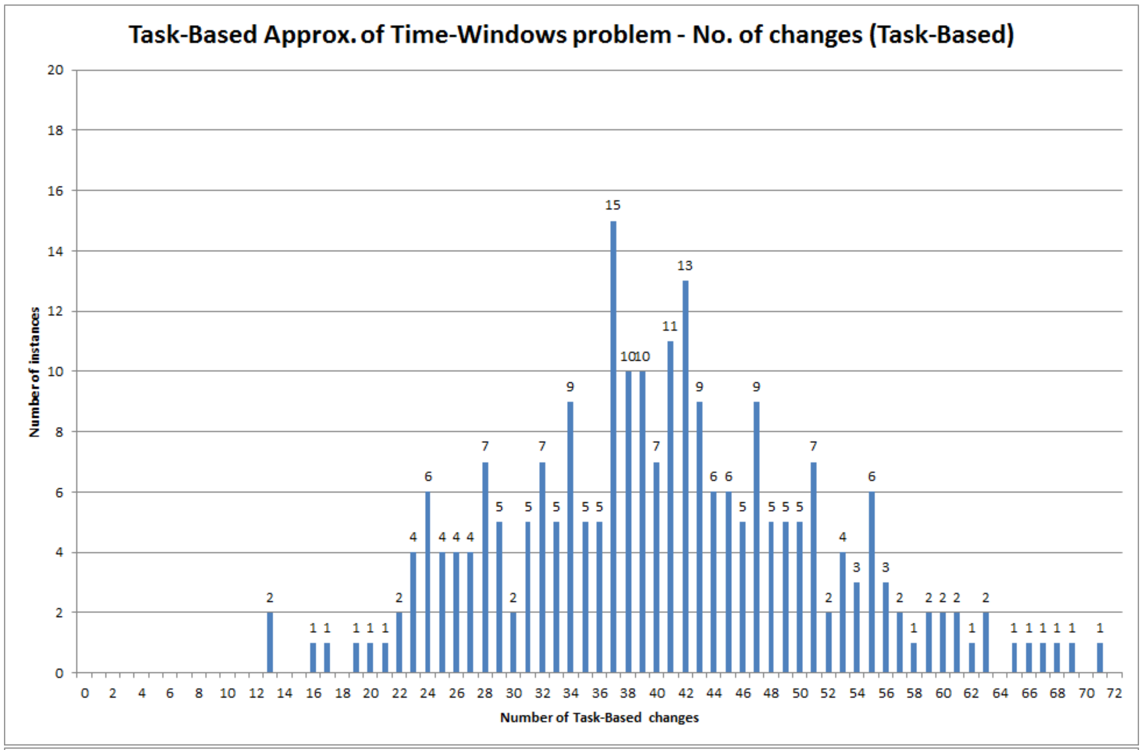
Figure 6.20: Histogram summarising total running time to find a Task-Based approximation solution for each instance, including conversion time.

the x-axis. As can be seen, for only 2 instances did the solver require the full two-minute time limit; for nearly half of the instances (117, 48.75% of the total), the entire solution process was completed within 1.5 seconds, and an additional 73 instances ($\approx$ 30.42% of the total) were solved within 2 seconds. All instances other than the two which required the full time limit were converted, solved, and converted back within 8 seconds. This is noteworthy as it shows the potential of the Task-Based Approximation approach to find solutions very quickly, even if the quality of the solutions (in terms of cost) is inferior to the change-minimization approach.

In an attempt to understand the outlying results, further investigation was carried out into the two datasets which took the full two-minute time limit. This revealed that for both instances the FICO Xpress solved had been terminated before an optimal solution to the Task-Based version of the problem had been found - instead, there was still a gap of $\approx 0.24\%$ and $\approx 1.74\%$ respectively between the best solution found and the best bound calculated for the Task-Based formulation of the problem. Clearly these gaps are relatively small, and indeed had there been an acceptable optimality gap of 5% (or even 2%) set for this approach rather than the default of 0.0001% then these instances would have been considered solved to within tolerance of optimality. It is also possible that the solutions found for these instances were the optimal for the formulation, and that the apparent failure to find the optimal solution was instead a result of the bound not being sufficiently increased.

Looking at the time the solutions were found revealed that they were found after approximately 2 seconds and approximately 24 seconds respectively, meaning the solver spent respectively around 118 seconds and around 96 seconds attempting to find cheaper solutions. The fact that it failed to do this in the time available in either case adds weight to the suggestion that these solutions may be the optimal for the Task-Based Approximation of these instances. Regardless of whether or not a better solution exists in the Task-Based representation for either of these instances, it is interesting to note the gaps between the solutions found by this method and the best known bound for these instances overall (i.e. in the Time-Windows representation) are respectively $\approx 66.85\%$ and $\approx 78.45\%$, i.e. below the average (both mean and median) gap across all 240 instances, as discussed in Figure 6.16. Similarly, these solutions have a gap to the best solution found by any method of respectively of $\approx 38.41\%$ and $\approx 57.00\%$ which is also below the mean and median gaps discussed at Figure 6.17 above.

Finally, it may be interesting to look at the number of changes proposed by the Task-Based Approximation solutions. Here there are two measures of numbers of changes: the number of tasks for which changes are required in the Task-Based version of the solution; and the number of weeks for which an employee has a change to their assigned role under the Time-Windows version of the solution. The count of these can be summarised in the two histograms in Figure 6.21. The upper of the two histograms shows the number of changes in the Task-Based representation of the solution. It can be seen that the values range from a minimum of 13 to a maximum of 71, and follow a roughly Normal-shaped curve over this range. The modal value is 37 changes, with 15 instances (6.25% of the total) obtaining this result, while the mean is ≈ 40.36 and the median value is 40 changes. It is interesting to compare this chart to Figure 5.10 which discusses the number of changes in



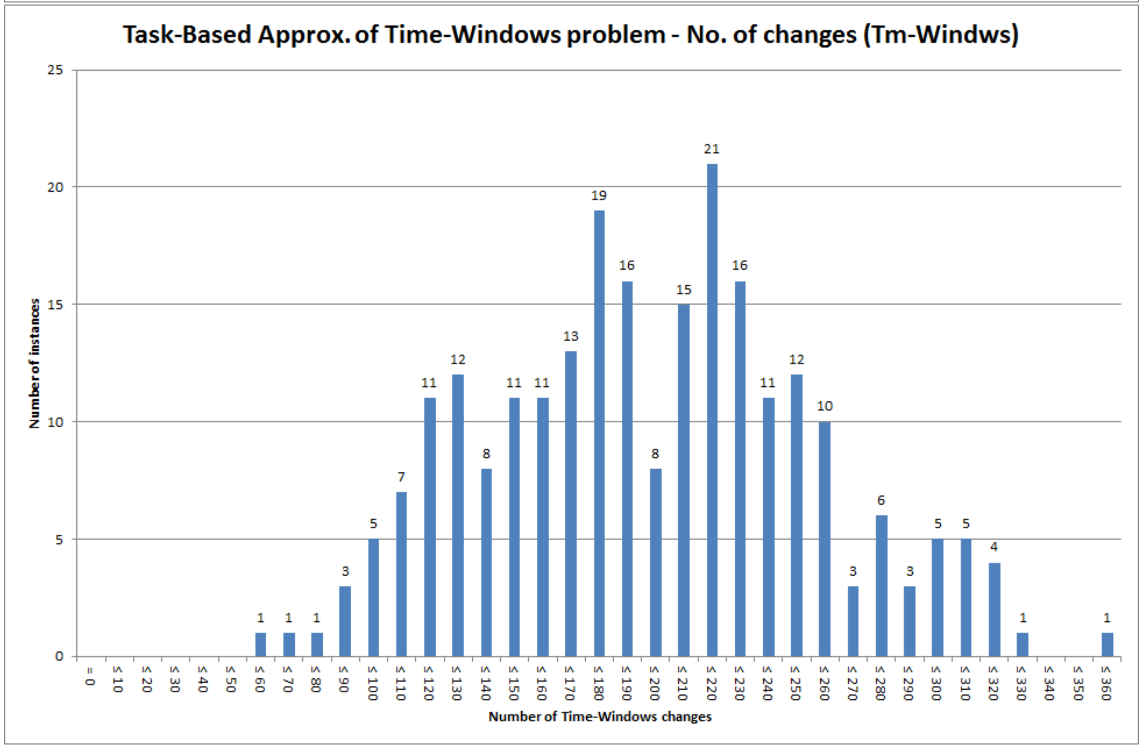


Figure 6.21: Histograms summarising the number of changes in the Task-Based representation (top) and Time-Windows representation (bottom) of the solution for each instance.

the lowest-cost solution to the Task-Based problem found using the change-minimization approach. The results shown here in Figure 6.21 have a slightly larger range, but have a similar maximum and a similar shape to the histogram. Recall that, although the datasets for the Task-Based and Time-Windows problems were generated separately and are therefore different, they do depend on similar underlying assumptions and the same basic parameters. Therefore there is some merit to observing the similarities in these plots, and to observe that while the Task-Based change-minimization approach actively aims to reduce changes, the Task-Based Approximation of the Time-Windows problem does not. Hence we can argue that it has achieved a (broadly) similar number of changes without this being the explicit goal, which adds weight to the suggestion made in section 6.3.2 that the Time-Windows problem may have a stronger link between the number of changes and the cost of the solution.

The lower plot meanwhile shows the number of changes in the solution when represented in the Time-Windows format. The graph displays a similar shape to that Task-Based representation, although it has a more bi-modal appearance and covers a much larger range of between 58 and 355 changes. The wider range is not unexpected, since each task equates to multiple weeks, with an average task length across all tasks and all instances of ≈ 4.24 weeks. The most common numbers of changes are 171 to 180 (19 instances, $\approx 7.92\%$ of the total) and 211 to 220 (21 instances, 8.75% of the total), while the mean number is ≈ 192.30 changes and the median is 191 changes.

6.8.2.1 Summary of Task-Based Approximation results

We now summarise the results relating to the Task-Based Approximation of the Time-Windows problem. In terms of solution cost, the values have a large gap from both the best known bound and best known solution, with mean gaps of $\approx 81.25\%$ and $\approx 65.54\%$ respectively. Comparing the solution values to the two-minute 'direct' solution approach showed that the Task-Based Approximation method achieved a better solution for the vast majority (90%) of instances. The Task-Based Approximation was however only able to beat the one-hour direct solution in $\approx 27.92\%$ of the instances, while comparison to the lowest cost change-minimization solution indicated that the Task-Based Approximation method obtained a cost of more than double the change-minimization solution for over half ($\approx 54.17\%$) of the instances. It is worth noting however than in all three of these comparisons, there were at least some instances where the Task-Based Approximation found a better solution and at least some where the solution was worse, indicating that no one approach is better than the other in all cases.

One of the clear advantages of the Task-Based Approximation approach is its speed, with nearly half of the instances (48.75%) being solved within 1.5 seconds. Two outlying instances required the full two-minute time limit, but investigation revealed that these could have terminated much earlier if a more generous optimality gap had been applied (e.g. 2% rather than the default of 0.0001%), or that the solution value would have been the same if the solver had terminated after say 30 seconds rather than 120 seconds. This means that, while the solution quality may not be the best in terms of cost, the method could be very useful in practical terms by giving a result very quickly. Connected with this, we observed that only one solution was produced by the method as implemented here; however, the approach could be adapted to output multiple results, a process which could potentially be aided by the fast running time allowing multiple solution runs to be carried out. This work has not been carried out during this research project and is left as a suggestion for future work.

Finally, in terms of numbers of changes we observed that the Task-Based representation of the solution, with range of 13 to 71 changes, mode of 37 changes, mean of ≈ 40.36 and median of 40 changes produces a graph of quite similar range and shape to that for the lowest-cost change-minimization solutions obtained for the Task-Based problem. The changes in the Time-Windows representation unsurprisingly cover a much larger range of between 58 and 355 changes, with a mean of ≈ 192.30 changes and a median 191 changes. This graph appeared to have two modes, in the ranges of 171 to 180 changes and 211 to 220 changes. The significance of the numbers of changes in the Time-Windows representation will be more clear when we compare them to results from the Heuristic initial solution and general Heuristic methods below.

6.8.3 The Heuristic Initial Solution approach

We now go on to look at the results obtained by using a heuristic algorithm to generate initial solutions for the Time-Windows problem, as discussed in section 6.6. As with the Task-Based Approximation approach, we use the following measures to evaluate the solutions: the gaps between the solution and the best known bound and best known solution for the instance; the improvement in solution cost relative to the 'direct' and change-minimization approaches; the running time; and the number of changes. In addition, we can also look at the number of iterations performed by the Heuristic initial solution algorithm on each instance.

The results presented in this section refer to the Heuristic initial solutions which were used in the testing of the main Heuristic algorithm, discussed in section 6.8.4 below. However, since running times were found to be relatively fast for this approach, it was felt there was an opportunity to look into the effect of the randomized elements involved in the algorithm, specifically the ordering of the employee list and role list as described in the procedure in sections C.4.2.3 and C.4.2.4. Therefore, as well as the results of the single run reported in this section, ten further randomized runs were carried out for each instance. From these, it was possible to calculate a minimum, maximum and average value for each of the metrics discussed here. A non-randomized run was also carried out, keeping the employees and roles in the same order as they appear in the datasets.

Results for these additional runs suggested that we would not expect to see a large variation in the outputs due to the randomization, and that the single randomized run results were representative of the other runs carried out. We therefore do not report the multiple randomized run or non-randomized run results here. Instead, for completeness, for each of the figures presented for the single randomized run below, there are two corresponding figures to be found in Appendix D:

1. One first showing the single randomized run results for the metric in question, along side the average of that metric across the ten random runs and the result for the non-randomized run.
2. A second showing the counts for that metric for its minimum and maximum values across the ten randomized runs.

It should also be noted that the multiple randomized runs produced numerous different solutions for most instances, but that unfortunately it was not possible to say precisely how many distinct solutions were generated. Since as with the Task-Based Approximation approach each individual test run only produces a single solution, we can as before give no further results with regard to number of solutions generated. However, we note that this approach could be adapted to produce multiple solutions, either by multiple random runs or perhaps some other means, and flag this as something which may potentially be investigated as future work.

In terms of the results for this single randomized run, we first look at the gap to the best known bound for each instance, which can be summarised in Figure 6.22. We can see that the results are grouped towards the upper end of the graph, with nearly half of the instances (111 out of the 240, i.e. 46.25%) showing a gap to the best bound greater than 85%. As before, we note that there is a single instance which has a gap of greater than 100%, being the only instance for which the best known bound was negative. It is noticeable that this graph is very similar to Figure 6.16 shown for the Task-Based Approximation solution, and indeed as with that graph we see here that 12 instances (5% of the total) found a solution with a gap to the best bound not greater than 50%. The smallest gap however is lower for this approach, with one instance only $\approx 1.38\%$ away from the best known bound. The mean gap size was $\approx 79.81\%$ and the median $\approx 83.85\%$, only a little lower than the figures for the Task-Based Approximation approach ($\approx 81.25\%$ and $\approx 85.23\%$ respectively).

We can also look at the quality of the solution cost in relation to the best known solution, from all test runs carried out, for each instance. The gaps between the Heuristic

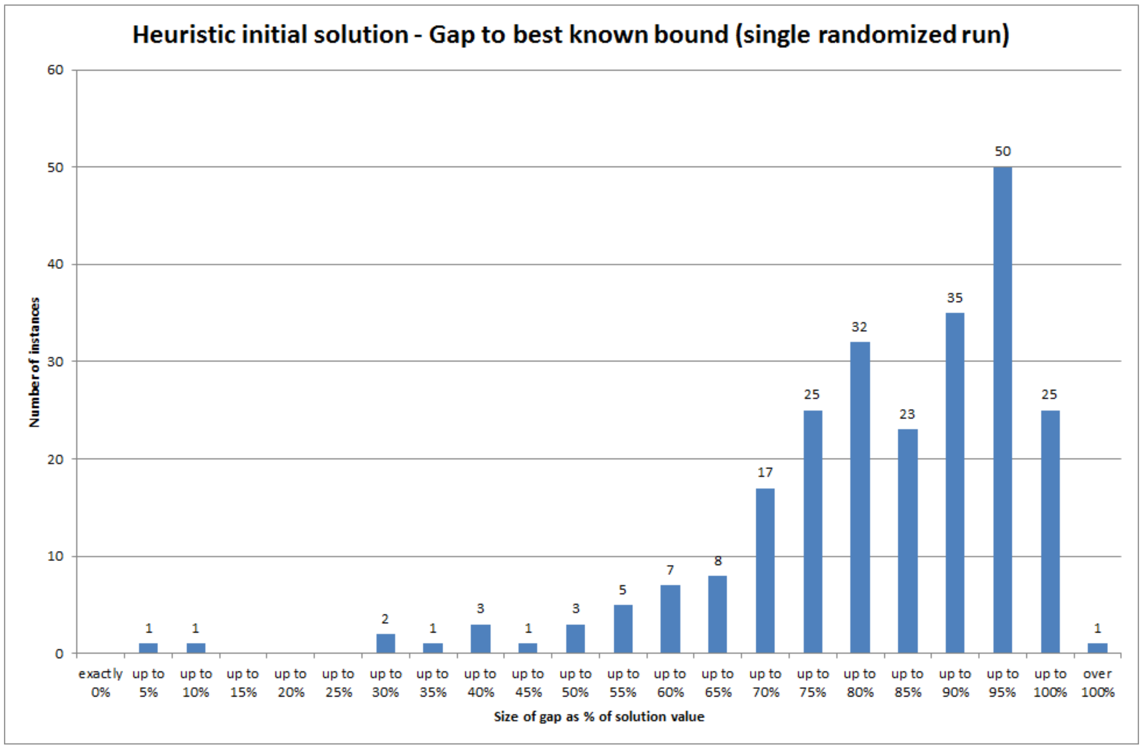


Figure 6.22: Histogram of percentage gaps to best known bounds for solutions found using the Heuristic initial solution approach.

initial solution cost and the best known solution for each instance are summarised in Figure 6.23. As with the Task-Based Approximation results, the gaps displayed here are still mainly towards the upper end of the scale but are clearly smaller than the gaps observed when comparing the solution to the best known bound. There are 115 instances ($\approx 47.92\%$ of the total) for which the gap to the best known solution was over 65%, compared to 125 instances ($\approx 52.08\%$ of the total) for the Task-Based Approximation solution, as shown in Figure 6.17. For the Heuristic initial solution, we see that 48 instances (20% of the total) had a solution within a 50% gap of the best known solution, including one instance which was solved to within 5% and one for which the best known solution was actually achieved by this approach. The mean gap to the best known solution was $\approx 64.00\%$ while the median was $\approx 63.51\%$, as compared to $\approx 79.81\%$ and $\approx 83.85\%$ respectively for the gap to the best bound. These gaps are slightly less than the average gaps to the best known solution for the Task-Based Approximation, which had a mean of $\approx 65.54\%$ and median $\approx 65.66\%$.

We will next compare the costs of the solutions found using the Heuristic initial solution approach to those using the 'direct' approach set out in section 6.3. As before, we consider

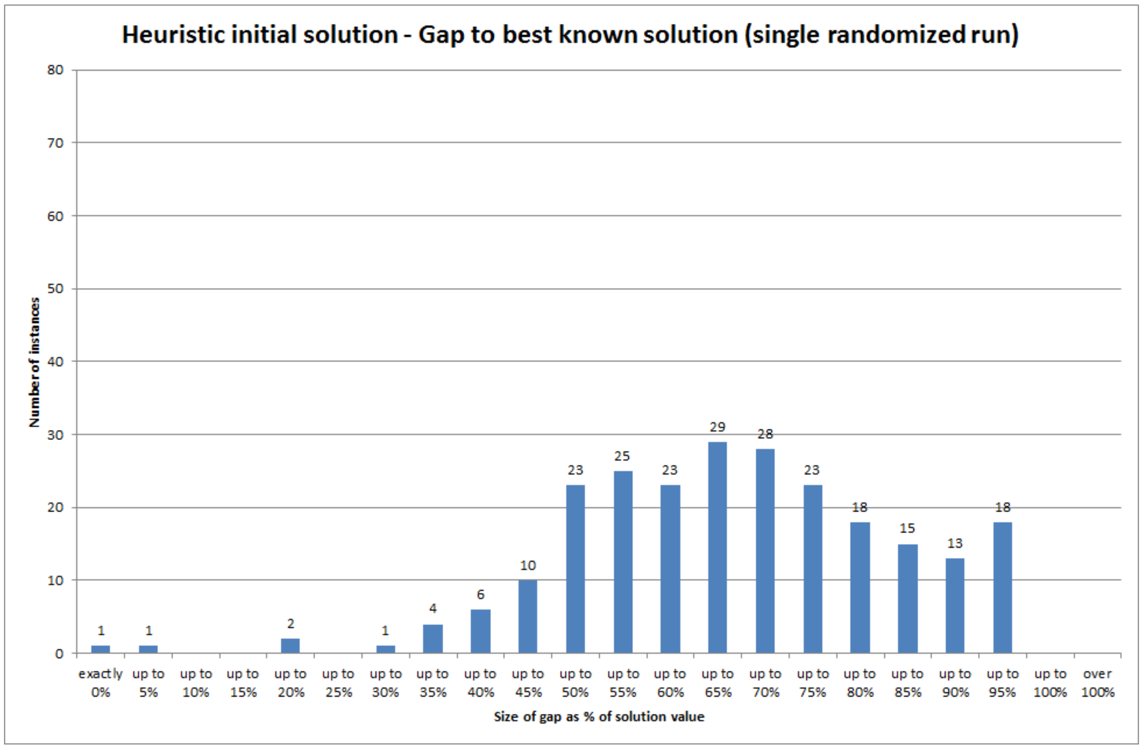
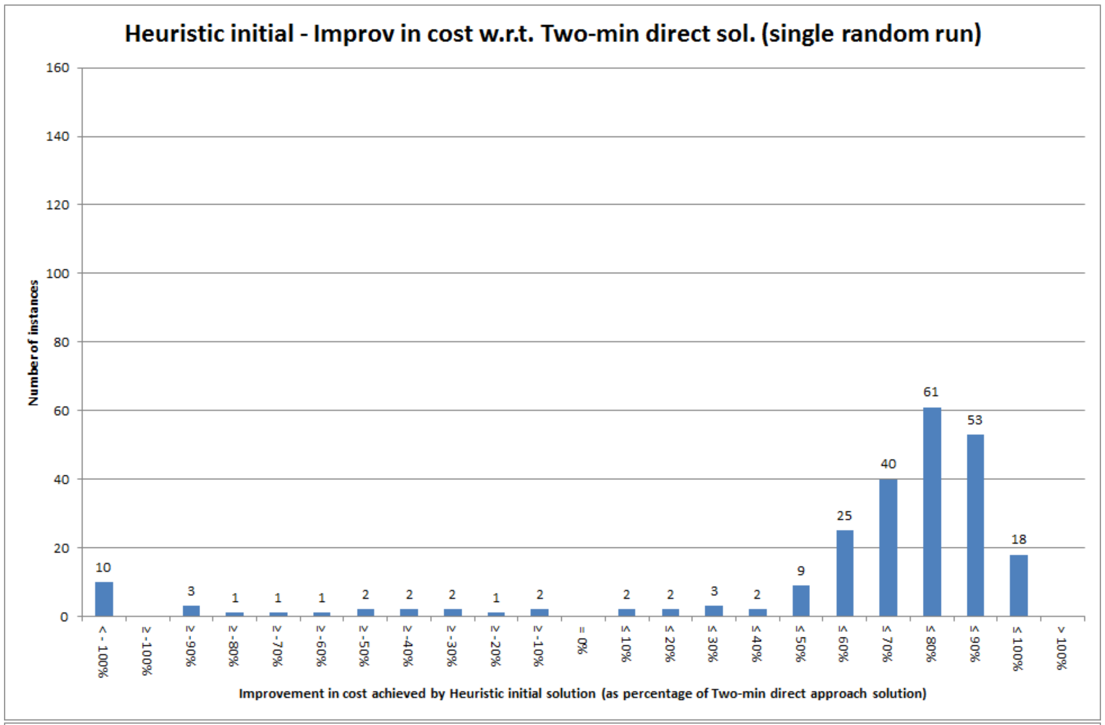


Figure 6.23: Histogram of percentage gaps to best known bounds for solutions found using the Heuristic initial solution approach.

the improvement achieved by the Heuristic initial solution method by subtracting this solution value from the direct approach value and representing it as a percentage of this 'direct' value. These percentage improvements are summarised for both the two-minute and one-hour settings of the direct approach in Figure 6.24. In the upper graph, we see that for the vast majority of instances (215 out of the 240, i.e. $\approx 89.58\%$), the Heuristic initial solution value was an improvement on that found using the two-minute direct approach. For over half of the instances (132, i.e. 55%), the Heuristic initial solution value was a better than 70% improvement on the two-minute direct solution; for the Task-Based Approximation approach as discussed in Figure 6.18, this count was 120 instances (i.e. 50%). Meanwhile, it can be seen that 25 instances ($\approx 10.42\%$ of the total) show a negative improvement indicating the Heuristic initial solution value was in fact larger than that found by the two-minute direct approach. Of these, 10 instances ($\approx 4.17\%$ of the total) have an improvement of less than -100%, i.e. have a Heuristic initial solution value more than double that of the two-minute direct approach value. We note that these values are quite similar to those seen for the Task-Based Approximation results, which had 23 instances with negative improvement and 12 instances with improvement less than -100%.



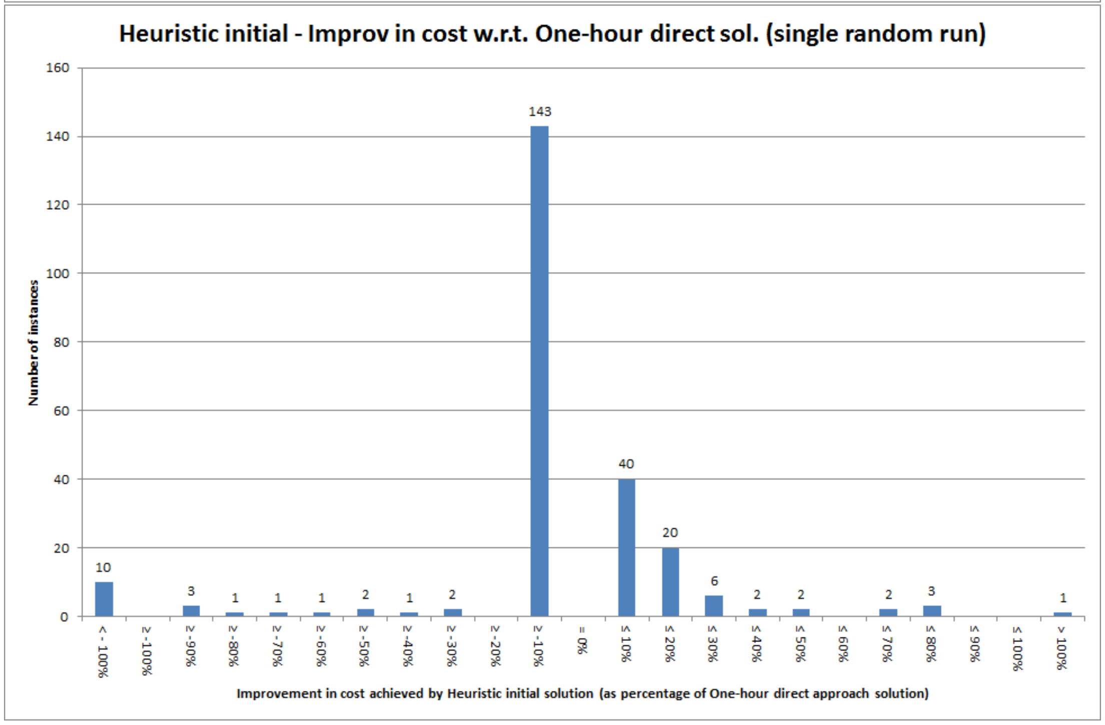


Figure 6.24: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the two-minute (top) and one-hour (bottom) 'direct' approach, shown as a percentage of the respective direct approach solution value.

The pattern of the graph overall is also quite similar to the equivalent chart in Figure 6.18.

Meanwhile, an entirely different pattern is observed in the lower histogram of Figure 6.18, which shows the improvement made by the Heuristic initial solution on the solution value found by the one-hour setting of the direct approach. Here, 76 instances ($\approx 31.67\%$ of the total) show a positive improvement, with most of the improvements in the lower range of percentages - for example, 40 instances ($\approx 16.67\%$ of the total) have a positive improvement of 10% or less. The remaining 164 instances ($\approx 68.33\%$ of the total) show a negative improvement, but the vast majority of these (143 instances, i.e. $\approx 59.58\%$ of the 240 total instances) show a negative improvement not less than -10%, which can be interpreted as the Heuristic initial solution value being within a 10% increase of the one-hour direct solution value. There are however still ten instances with an improvement of less than -100%, i.e. where the Heuristic initial solution value is more than double the one-hour direct solution value. This is a noticeable contrast to the lower graph of Figure 6.18 for the Task-Based Approximation, for which 92 instances ($\approx 38.33\%$ of the total) showed an improvement of less than -100%, and with numbers fairly evenly spread across the range of percentages otherwise. Instead, we find here that a large number of Heuristic initial solution values are relatively close to the one-hour direct solution values, with 183 instances (76.25% of the total) having a Heuristic initial solution value that is within $\pm 10\%$ of the one-hour direct solution cost.

The final cost-related comparison is to compare the Heuristic initial solution value to the lowest-cost change-minimization results, as discussed in section 6.3. As with the comparison to the 'direct' solution approach, we make the comparison as an improvement calculated by subtracting the Heuristic initial solution value from the (two-minute) change-minimization solution value for each instance, and representing this as a percentage of the change-minimization solution value. These percentage improvements can then be summarised in a histogram, as shown in Figure 6.25. The graph shown here is more in keeping with what we observed for the Task-Based Approximation in Figure 6.19, with only 6 instances (2.5% of the total) showing a positive improvement by the Heuristic initial solution approach on the solution found by the change-minimization method. Only 40 instances ($\approx 16.67\%$ of the total) which have a negative improvement have an improvement greater than -50%, i.e. where the Heuristic initial solution cost is no more than one-and-a-half times the change-minimization solution value. This is very similar to the 42 instances in this category for the Task-Based Approximation approach. In keeping with the Task-Based Approximation comparison with the change-minimization approach, and in contrast to the comparison with the one-hour direct solution approach shown in Figure 6.24, we see in Figure 6.25 that over half the instances (125, $\approx 52.08\%$ of the total) show an improvement of less than -100% from the change-minimization solution to the Heuristic initial solution.

We next turn our attention to the running time of the Heuristic initial solution ap-

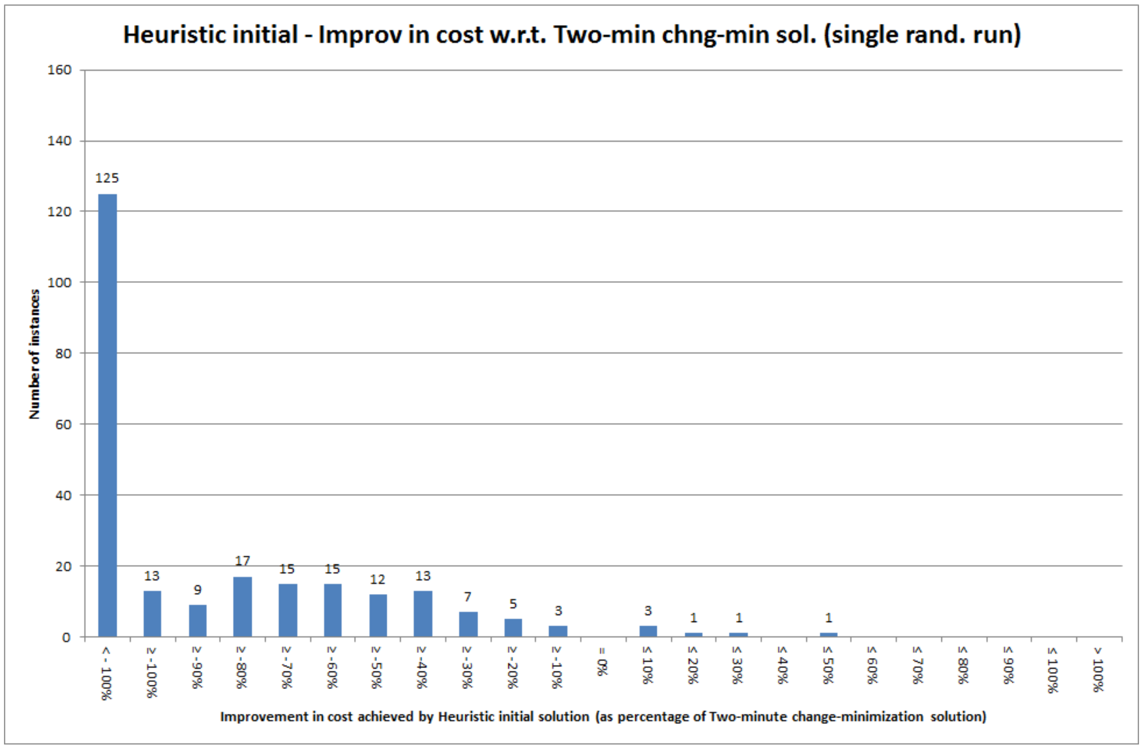


Figure 6.25: Histogram summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the two-minute change-minimization approach, shown as a percentage of the change-minimization solution value.

proach. This can be summarised in Figure 6.26. We see the shape of this graph is roughly Normal, with perhaps a slightly longer tail towards the upper values. The running time range from ≈ 7.44 seconds to ≈ 39.28 seconds, with the mean ≈ 20.32 seconds and the median ≈ 19.66 seconds. It can be noted that only two instances ($\approx 0.83\%$ of the total) were solved within 8 seconds, the upper limit on the running time of the Task-Based Approximation approach for all but the two outlying instances. However, while not as fast as the Task-Based Approximation it should also be noted that all instances were solved within 40 seconds, which still makes this method potentially useful in practical terms when a solution is required quickly.

It is important also to recognise here that for expediency in obtaining the Heuristic initial solution results for use in the main Heuristic algorithm, running time was not of primary importance during these test runs. Therefore, rather than implementing the procedure described in section 6.6.1 in C++ these tests were carried out using the FICO Xpress software. It would be anticipated that if implemented in the more efficient C++ (or similar) programming language, the time taken to find a Heuristic initial solution for any

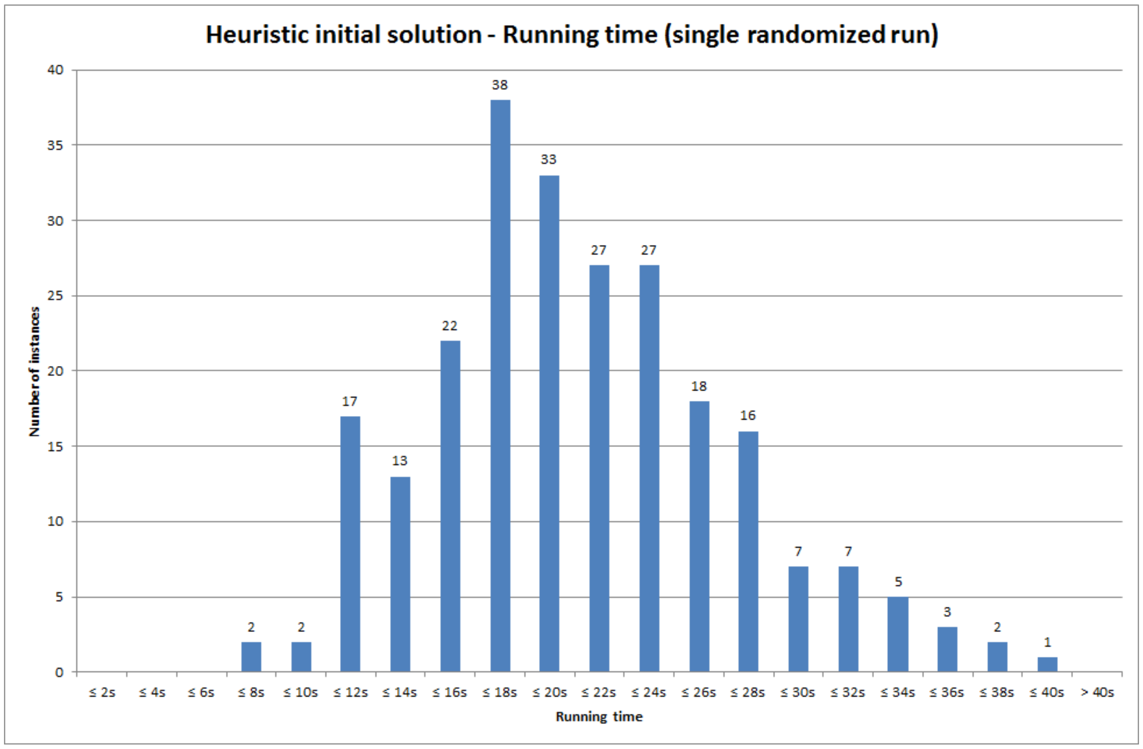


Figure 6.26: Histogram summarising total running time of the Heuristic initial solution algorithm for each instance.

instance would be greatly reduced, thus improving the expected usefulness of this approach in terms of solution time. We would not however expect the change of implementation language to affect any of the other metrics discussed here. A re-coding and re-testing of the Heuristic initial solution algorithm is a potential avenue for future research, perhaps in conjunction with a project to implement a practical decision support tool within the company.

Linked to the running time is the number of iterations required by the algorithm to find the initial solution. This is summarised for each instance in Figure 6.27. The number of iterations required will depend on a number of factors, such as the number of vacancies in the current schedule, or whether vacant blocks must be increased in size to make them more cost-effective to cover. Since more than one vacancy can potentially be filled at each iteration, the order in which roles and employees are considered in the algorithm may also have a bearing. We can see from the graph here that there are 2 instances ($\approx 0.83\%$ of the total) which are solved in a single iteration, while at the other extreme 1 instance requires 8 iterations to solve. Between this, the graph follows a roughly normal curve, with 100 instances ($\approx 41.67\%$ of the total) taking the median and modal value of 4 iterations. The

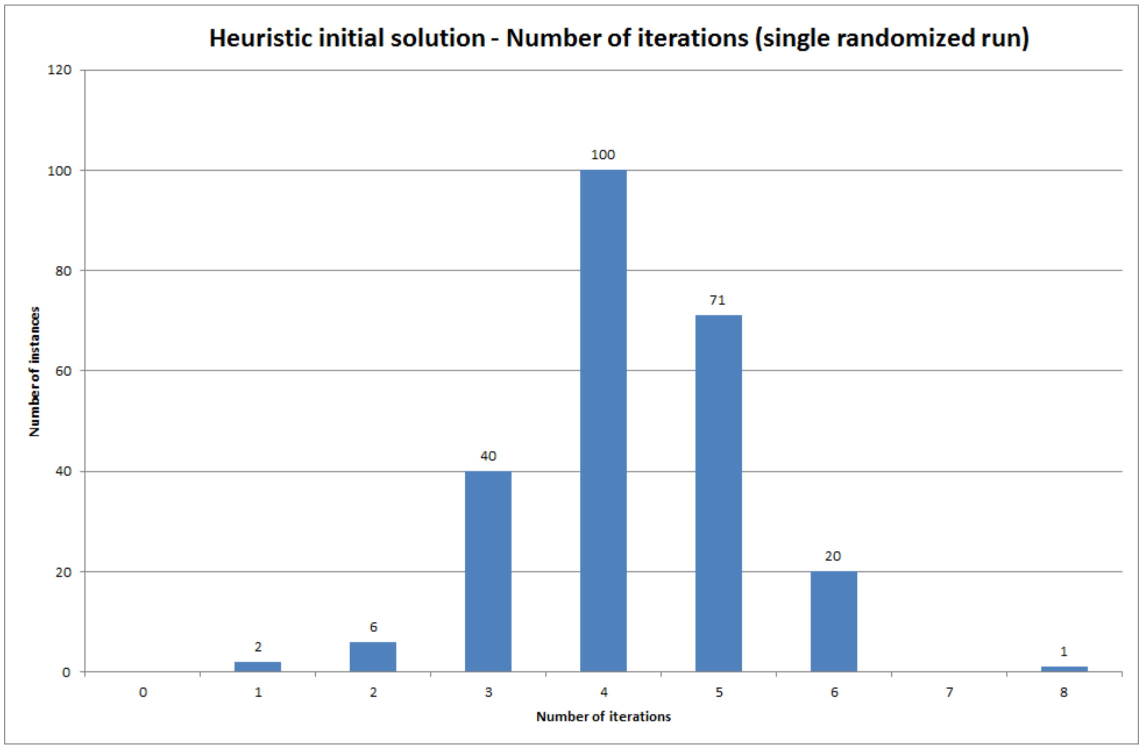


Figure 6.27: Histogram summarising the number of iterations carried out by the Heuristic initial solution algorithm for each instance.

mean was slightly higher than this, at ≈ 4.24.

Finally, we can look at the number of changes to the current schedule in the solutions found using the Heuristic initial approach. Unlike the changes in the Task-Based Approximation solutions, we only have one measure for changes for the Heuristic initial solutions. This relates to the number of changes in the Time-Windows version of the solution, as shown previously in the lower chart of Figure 6.21, found by summing the assignment change variables $x^{\pm}_{ijt}$ across all employees $i \in E_R$, roles $j \in J$ and all times t. The number of changes for this solution approach is summarised in Figure 6.28. It is noticeable that the number of changes shown in this chart is much less than in the Task-Based Approximation solution - here the range goes from a minimum of 25 changes to a maximum of 134, compared to a range of 58 to 355 in the results in the previous section. That there is a difference is not surprising, although the scale of the difference might not have been expected, when we consider the contrasting approaches used. The Heuristic initial solution approach focusses on making a small number of changes provided that very expensive changes can largely be avoided; on the other hand, the Task-Based Approximation, by grouping the assignments into tasks, will be almost certain to require more changes since making a

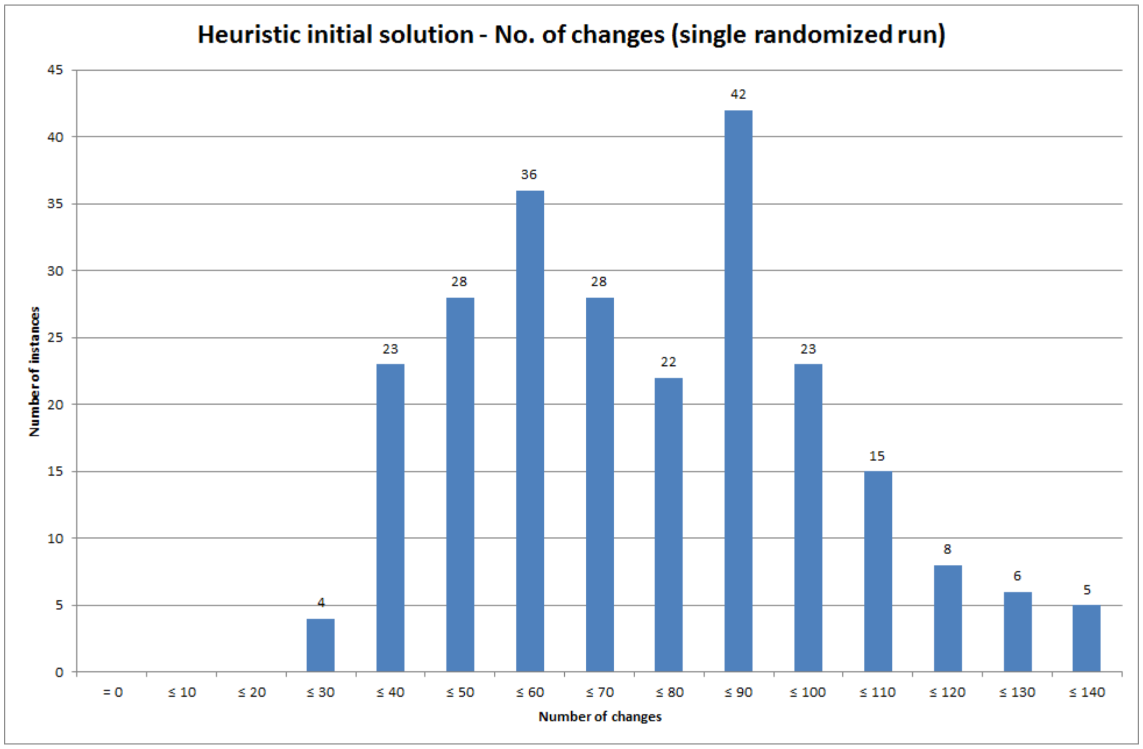


Figure 6.28: Histogram summarising the number of changes in the Heuristic initial solution for each instance.

change to a single task affects multiple assignments in the Time-Windows representation. The graphs shown in the lower half of Figure 6.21 and in Figure 6.28 may have different scales, but do seem to have a similar bi-modal shape, with the Heuristic initial solution results having a peak of 42 instances (17.5% of the total) reporting 81 to 90 changes, with a slightly lower second peak at 51 to 60 changes accounting for 36 instances (15% of the total). The mean number of changes here was ≈ 71.60 changes, with the median equal to 70 changes; this is compared to a mean of ≈ 192.30 changes and median of 191 changes for the Task-Based Approximation solutions.

6.8.3.1 Summary of Heuristic Initial Solution results

We can now summarise the results presented in section 6.8.3 relating to finding an initial solution to the Time-Windows problem using a Heuristic approach. In terms of solution cost, the gaps from the Heuristic initial solutions to the best known bound and best known solution were generally large, averaging $\approx 79.81\%$ and $\approx 64.00\%$ respectively. However, the method performed well on these measures for a small number of instances, with one instance achieving a gap of just $\approx 1.38\%$ to the best bound, and equalling the best known solution during this test run. Comparing the solution values to the initial computational results showed that the Heuristic initial solution approach was able to achieve a better solution than the two-minute ‘direct’ approach on $\approx$ 89.58% of instances, but was able to improve on the two-minute change-minimization solution in only 2.5% of cases. It was interesting to see that the Heuristic initial solutions were in general quite close to the one-hour ‘direct’ solution values, with 76.25% of instances recording a solution within $\pm$10% of the one-hour direct cost. As with the Task-Based Approximation approach, it should be noted that in all three of these comparisons there were both positive and negative improvements observed for at least some instances, indicating that there are no guarantees that the Heuristic initial approach will always be better or worse that the ‘direct’ or change-minimization approaches.

In terms of running time, instances were solved in between $\approx$ 7.44 seconds and $\approx$ 39.28 seconds, with an mean time of $\approx$ 20.32 seconds. While this could in itself be fast enough for the company’s needs, we noted that these results were produced using FICO Xpress and that for this kind of algorithm an implementation in a language such as C++ would be more efficient. Such an implementation would also be more useful in practice if it were done in an open source software, and could pave the way for a more time-efficient adaptation of the algorithm such that multiple solutions could be generated. An investigation into this is left as a proposal for future work.

Linked to running time, the number of iterations required for each instance ranged between 1 and 8, with a median and mode of 4 iterations (accounting for 100 instances, $\approx$ 41.67% of the total) and a mean of $\approx$ 4.24 iterations. Meanwhile, the number of changes recommended in the solutions found were notably fewer than those in the Task-Based Approximation, with a range of 25 to 134 changes in contrast to 58 to 355 as shown in the previous section. The mean number of changes was $\approx$ 71.60, but the shape of the graph seemed consistent with a bi-modal distribution with peaks in the 81 to 90 changes range (17.5% of instances) and the 51 to 60 changes range (15% of instances).

We note that these results relate only to the single randomized test run, the results from which were used as the initial solutions in the testing of the main Heuristic algorithm. However, some additional test runs (ten randomized and a single non-randomized) were carried out, with graphical results given in Appendix D which correspond to those results shown above in Figures 6.22 to 6.28.

6.8.3.2 Comparing the Heuristic Initial and Task-Based Approximation solutions

We will also look further at the results given in section 6.8.3.2 with respect to how they compare to those for the Task-Based Approximation solution. In terms of the solution cost, it was observed that the gaps to the best known bound and best known solution were in general lower for the Heuristic initial solution (Figures 6.22 and 6.23) than for the Task-Based Approximation (Figures 6.16 and 6.17). Similarly, the Heuristic initial solution appeared to compare a little more favourably with the two-minute direct and change-minimization results, although the broad conclusion was the same as for the Task-Based Approximation: that it was generally superior to the two-minute direct approach, and generally inferior to the change-minimization method.

The most significant difference between the two came when comparing to the one-hour direct approach solutions, where the Heuristic initial solutions appeared to be on the whole quite close to the one-hour solutions, while the Task-Based Approximation solutions saw a number of negative improvements of less than -100%. The best way to evaluate the two solutions however is to compare the two directly, and therefore we calculate the improvement that the Heuristic initial solution makes on the Task-Based Approximation solution. Representing this as a percentage of the Task-Based Approximation solution value, we can summarise these percentage improvements in Figure 6.29. Note that graphs

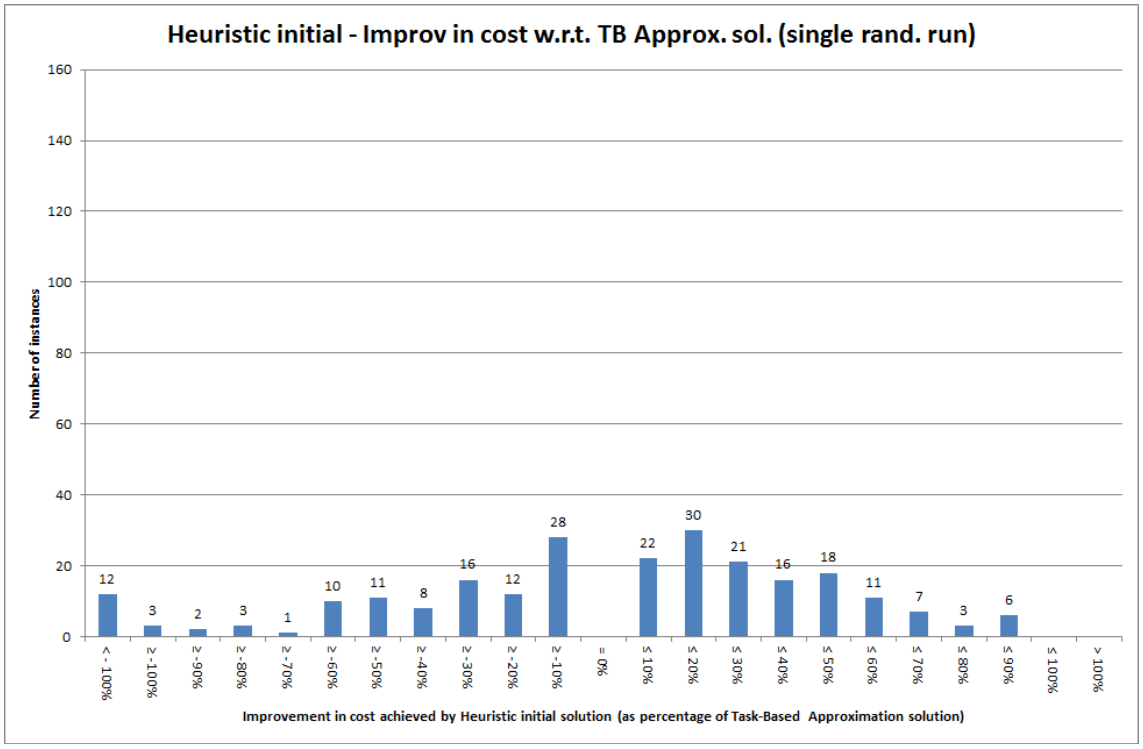


Figure 6.29: Histogram summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the Task-Based Approximation solution, shown as a percentage of the Task-Based Approximation solution value.

corresponding to this for the additional randomized and the non-randomized run are shown in Figure D.9 in Appendix D. As can be seen from the graph here, there is a certain amount of variation, with 134 instances ($\approx 55.83\%$ of the total) having a better solution from the Heuristic initial solution method, but the remainder seeing a lower cost solution from the Task-Based Approximation approach. The percentage changes appear to be grouped towards zero and taper towards the extremities of the graph, with 50 instances ($\approx 20.83\%$ of the total) showing a Heuristic initial solution value within $\pm 10\%$ of the Task-Based Approximation cost and a further 42 instances (17.5% of the total) within $\pm 20\%$. Therefore we can say that, while across the 240 test instances the Heuristic initial solution produces cheaper solutions than the Task-Based Approximation, variations from instance to instance mean that it is not consistently outperforming in terms of cost.

In terms of number of changes however, the Heuristic initial solution method appears to greatly outperform the Task-Based Approximation approach. As mentioned above, this is to an extent expected given the nature of the two methods, with the Heuristic initial solution approach being explicitly designed to try to limit the number of changes while trying to avoid single expensive alterations to the existing schedule. Meanwhile, in terms of running time, it would appear that both methods are capable of finding a solution within a relatively short length of time, although the Task-Based Approximation approach appears to be faster from the results given here. However, it is believed that a more efficient implementation of the Heuristic initial solution algorithm would be possible which would most likely negate the advantage seen here and make the two methods similar in terms of running time (at least to a practical level of comparison).

The overall conclusion of this comparison therefore is that the Heuristic initial solution approach is superior to the Task-Based Approximation approach in terms of the solutions it can produce. However, since neither were designed to be the source of a final solution to the Time-Windows problem, it remains to be seen which can lead to better solutions when the main Heuristic algorithm is applied. As discussed previously, both initial solutions were used in the Heuristic algorithm test runs, and below in section 6.8.4 we examine whether the choice of initial solution has any significant bearing on the final result or on the level of improvement the algorithm is able to make.

6.8.4 The Heuristic Algorithm

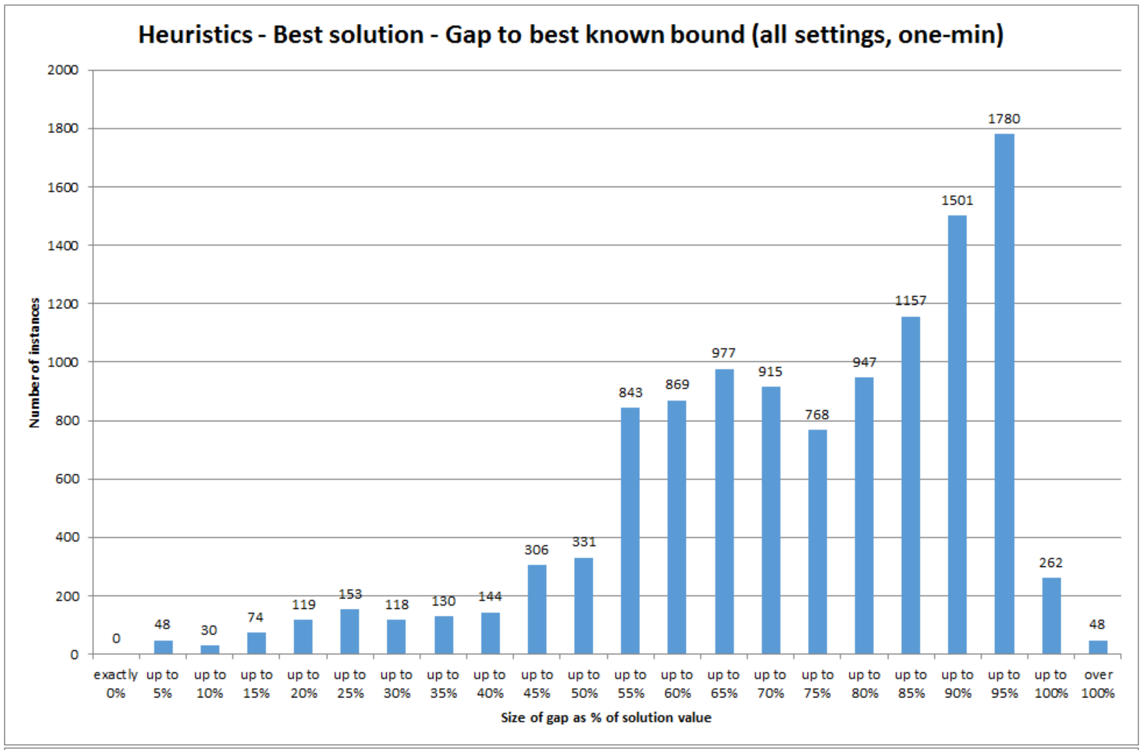
We can now examine the detailed results of tests on the Heuristic algorithm itself. As outlined in section 6.6.2, these Heuristics take as their initial solution either the Task-Based Approximation solution or the Heuristic initial solution and seek to improve them using the methods discussed in section 6.5. As a reminder, the plan given in that section was to test various settings in the heuristic algorithm, which gave rise to 48 different combinations (including the two alternative initial solutions). In this results section we not only present, as before, various metrics of solution quality, but also analyse these metrics to determine whether the different settings have any influence on the performance of the algorithm. Please see section 6.8.4 for a summary of these results.

6.8.4.1 Gap to the best known bound

We begin, as before, by looking at the gaps between the solutions found and the best known bound for each instance. As stated in section 6.5.4.11, solution details were recorded after one minute of the test run as well as at the end of the two-minute running time. Therefore, we present two graphs here for the gaps to the best known bound, summarising the gaps for all instances and all combinations of settings for both the solutions found within one minute and that found after two minutes. This is given in Figure 6.30.

Perhaps the most striking observation to be made about the two charts shown here is that they are very similar in shape, suggesting that there is a degree of similarity between the quality of solutions found within one minute and those found in two minutes. In general, the tendency is towards large gaps, with only 821 of the 11520 total test runs ($\approx 7.13\%$) finding a solution within 40% of the best known bound within one minute, and only 823 test runs ($\approx 7.14\%$ of the total) concluding after two minutes with a solution within 40% of the best bound. Similarly, within one minute only 1458 test runs ($\approx 12.66\%$ of the total), and within two minutes only 1482 test runs ($\approx 12.86\%$ of the total), found a solution that was within 50% of the best known bound. There is then a jump in both graphs, with a large number of test runs resulting in solutions with a gap which is greater than 50% and within 80% - these numbers are 5319 ($\approx 46.17\%$ of the total) for the one-minute results, and 5346 ($\approx 46.41\%$ of the total) for the two-minute. The highest single category in both charts is a gap greater than 90% and up to 95%, which accounts for 1780 test runs ($\approx 15.45\%$ of the total) for the one-minute gaps, and 1745 test runs ($\approx 15.15\%$ of the total) for the two-minute results. Calculation of the average gaps showed the means to be $\approx 71.02\%$ for the one-minute and $\approx 70.84\%$ for the two-minute results, while median gaps were $\approx 74.65\%$ for the one-minute and $\approx 74.48\%$ for the two-minute results.

These results can be compared to the initial computational results given in section 6.3 earlier. Looking first at the direct solution approaches, it can be seen by comparing with Figure 6.1 that overall the Heuristic methods perform much better than the two-minute direct approach, which saw 135 of the 240 instances (i.e. 56.25%) with a gap to the best known bound of between 95% and 100%, although this direct approach did also find (and prove) the optimal solution for 5 instances. The comparison with the one-hour direct approach results given in Figure 6.2 is more even, with a similar proportion of instances falling into the most common categories of a gap between 90% and 95% and a gap



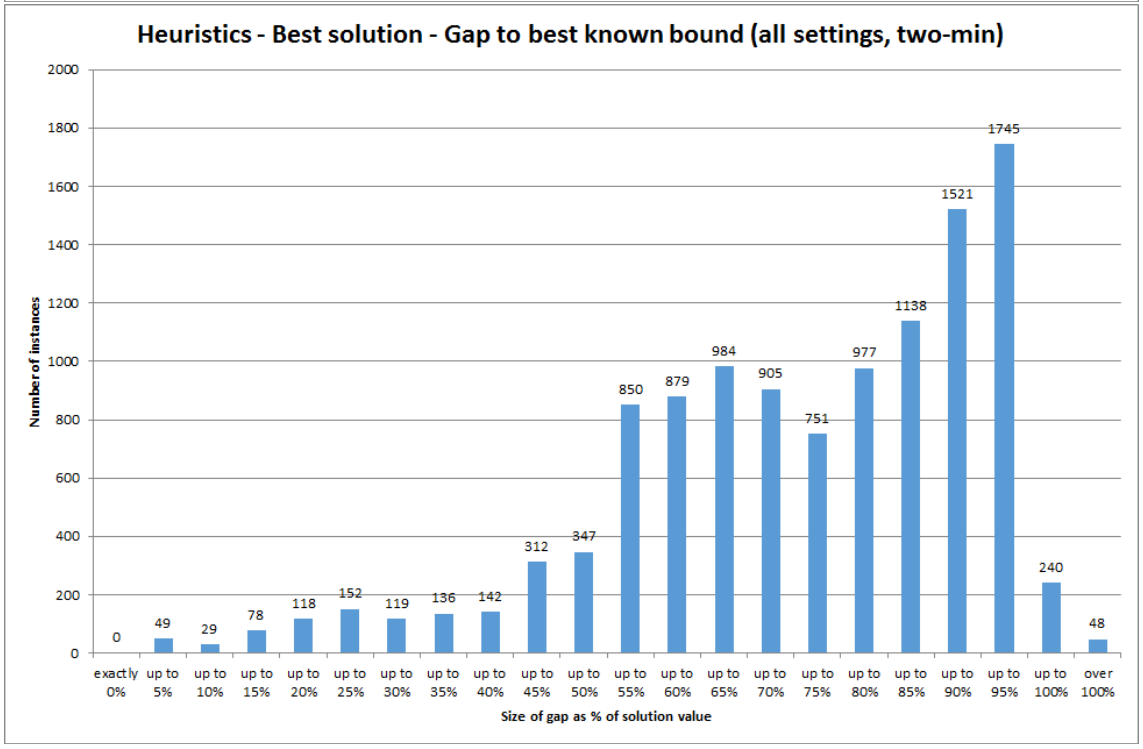


Figure 6.30: Gap to the best known bound from the solution found after one minute (top) and two minutes (bottom), across all test instances and all combinations of settings.

of between 85% and 90%. The one-hour direct solution had proportionately many more instances at the lower end of the scale, with ≈ 7.09% of instances being solved to optimality and a further ≈ 4.58% to within a 5% gap of the best known bound. Comparing with the lowest-cost change-minimization results given in Figure 6.5, we see that in general the change-minimization approach is outperforming the heuristics, with a smaller percentage of instances with results at the higher end of the chart and a mean gap of ≈ 59.78%. The overall pattern of the results is similar however, with the majority of the instances being solved with a gap of greater than 50% to the best known bound. The comparison with the non-cost-constrained change-minimization solution, shown in Figure 6.6, is however much more clear cut. While the Heuristic results above show 6777 test runs (≈ 58.83% of the total) for the one-minute and 6828 test runs (≈ 59.27% of the total) for the two-minute results solving to within 80% of the best known bound, the non-cost-constrained change-minimization had no instances which fell into these categories.

Having discussed the combined results for all heuristic settings, we now examine whether there is an influence made by the setting choices, as described in section 6.5.4.14, on the quality of the solutions produced. We do this using a similar approach to that used in section 5.5.2.3 to examine the effect of the values of the data generation parameters. It is clear that the graphs shown in Figure 6.30 are not consistent with the assumption of the data being Normally distributed; therefore, we must use the non-parametric Kruskal-Wallis test rather than the F-test to investigate the influence of the settings. The p-values obtained from these tests are shown in Table 6.2, with those significant at the 5% level marked with an asterisk. As can be seen, neither the order in which employees are examined (a

Table 6.2: Influence of heuristic settings on gap to best known bound (Kruskal-Wallis test).

Setting type	p-value for one-minute result	p-value for two-minute result
Employees to examine	0.058	0.015*
Employee order	0.662	0.540
Acceptance criteria	0.841	0.653
Kick settings	0.047*	0.006*
Initial solution	< 0.001*	< 0.001*

random order or the ‘smarter’ ordering) nor the difference in acceptance criteria used have a significant influence on the gap to the best known bound found by the algorithm. Meanwhile, the number of employees examined at each iteration appears to have a significant bearing on the gap to best bound for the solutions found after two minutes, but not with relation to the solutions found after one minutes. Significant for both the one-minute and two-minute results are both the kick settings and the type of initial solution used.

We cannot, however, determine from these values alone the manner of the influences and which settings produce more favourable results. Examination of graphs showing the breakdown by the influencing factors reveals that the results follow approximately the same pattern for each of the subgroups (these graphs are not given here, but instead appear in the appendix in section D.2.1). Here, we consider only the central measures of mean and median for each of these subgroups. Looking first at the breakdown of the two-minute results by the number of employees to examine at each iteration, we have that

- When all employees are examined, the mean gap is $\approx 71.22\%$ and the median is $\approx 74.93\%$;
- When only one third are examined, the mean gap is $\approx 70.46\%$ and the median is $\approx 73.97\%$.

This shows that a narrower gap is achieved when only one third of employees are examined at each iteration. This could be explained by the fact that this option shortens the iteration length, allowing more iterations to be carried out within the time available. If this is the case, it would suggest there is more value to carrying out a greater number of iterations rather than searching for the best possible improvement at each step, and this would be consistent with the statistical insignificance of the changing acceptance criteria.

Looking next at the breakdown by the kick activation setting used, for the one-minute results we have that

- When no kick is used, the mean gap is $\approx 70.41\%$ and the median is $\approx 73.87\%$;
- When the kick is activated after 4 or more iterations without improvement, the mean gap is $\approx 71.58\%$ and the median is $\approx 75.46\%$;
- When the kick is activated after 8 or more iterations without improvement, the mean gap is $\approx 71.07\%$ and the median is $\approx 74.58\%$.

Meanwhile, for the two-minute results we have that

- When no kick is used, the mean gap is $\approx 70.03\%$ and the median is $\approx 73.55\%$;
- When the kick is activated after 4 or more iterations without improvement, the mean gap is $\approx 71.51\%$ and the median is $\approx 75.37\%$;
- When the kick is activated after 8 or more iterations without improvement, the mean gap is $\approx 70.97\%$ and the median is $\approx 74.51\%$.

This shows that, contrary to what might be expected and to the intention of the kick procedure, it is in fact the test runs which do not employ a kick which appear to find the best solutions. Of those which do, it appears to be more beneficial to wait longer before applying a kick as the 8-or-more-iteration setting seems to result in a lower gap on average than the 4-or-more-iteration setting. Some further investigation into this suggested that in those test runs where a kick was applied, the best solution was often found *before* the kick was applied, meaning that the kick was not providing a means of reaching better solutions as hoped. A possible explanation for this is the amount of time allowed for the algorithm to improve on the solution after a kick has been carried out. As a large number of iterations are possible, there is perhaps scope to increase the length of time without improvement before a kick is activated, which may provide the time the algorithm requires to improve toward a new best solution. Increasing this setting is therefore suggested as a future improvement which could be investigated further.

Finally, if we break down the results according to the initial solution, for the one-minute results we have that

- When the Task-Based Approximation is used, the mean gap is $\approx 73.80\%$ and the median is $\approx 77.40\%$;
- When the Heuristic initial solution is used, is $\approx 68.24\%$ and the median is $\approx 71.25\%$.

Meanwhile, for the two-minute results we have that

- When the Task-Based Approximation is used, the mean gap is $\approx 73.59\%$ and the median is $\approx 77.18\%$;
- When the Heuristic initial solution is used, is $\approx 68.08\%$ and the median is $\approx 71.08\%$.

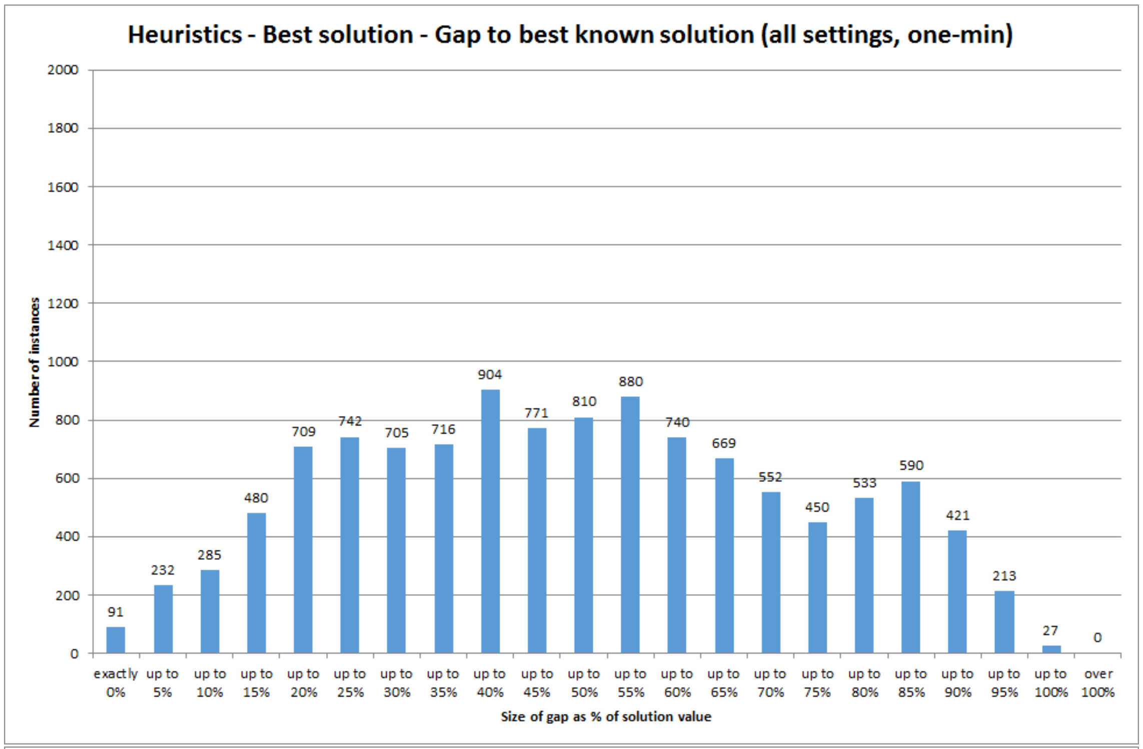
This demonstrates a clear difference between the gaps to the best bound for the two categories, with the test runs starting from the heuristic initial solution achieving a markedly smaller gap. This is not surprising, since as discussed in section 6.8.3.2 the heuristic initial solutions tend to have a lower cost that the solutions found using the Task-Based Approximation method. A more interesting question arising from this would be whether the final solution quality relates directly to the initial solution quality, or whether the algorithm is able to make better improvements on one type of initial solution than the other. This is discussed in section 6.8.4.3, where we examine in detail the improvements made by the algorithm on the initial solution.

6.8.4.2 Gap to the best known solution

Linked to the analysis of the gaps to the best known bound, we now turn our attention to the gap between the solutions found and the best known solution for each instance. As discussed in section 6.8.1 previously, this takes into account the solutions found by all the approaches described in this chapter (including preliminary testing), and gives a more practical indication of solution quality. As above, we begin by presenting two graphs which summarise the gaps to the best known solution overall for all instances and all combinations of settings for both the solutions found within one minute and within two minutes. This is given in Figure 6.31.

Firstly, we note as before that the two charts shown are very similar in shape, which is consistent with the observation about Figure 6.30 that there is probably a degree of similarity between the quality of solutions found within one minute and those found in two minutes. It can be seen that the size of gaps here are more evenly spread than the gaps to the best known bound, with values a little more clustered towards the centre and reducing towards the tails. The most common category in both charts is that of a gap greater than 35% and within 40%, accounting for 904 test runs ($\approx$ 7.85% of the total) for the one-minute and 907 test runs ($\approx$ 7.87% of the total) for the two-minute results. This is followed by the greater than 50% and within 55% category, containing 880 items ($\approx$ 7.64% of the total) for the one-minute and 875 ($\approx$ 7.60% of the total) for the two-minute results. Combining the counts between these two categories, we have that the (approximate) centre of each graph with gaps of over 35% but within 55% of the best known solution account for 3365 ($\approx$ 29.21%) of the one-minute and 3359 ($\approx$ 29.16%) of the two-minute results. In terms of particularly good solutions, we see that within one minute 91 test runs ($\approx$ 0.79% of the total) equalled the best known solution, while a further 232 ($\approx$ 2.01% of the total) found a solution within 5% of the best known; for the two-minute time limit, these numbers increase to 112 results ($\approx$ 0.97%) equalling the best known solution, and a further 243 ($\approx$ 2.11%) within 5% of it. Calculation of the average gaps showed the means to be $\approx$ 46.45% for the one-minute and $\approx$ 46.04% for the two-minute results; median gaps were quite similar, at $\approx$ 45.68% for the one-minute and $\approx$ 45.22% for the two-minute results.

As above, we can compare these results to the supplemental results given in section 6.8.1 for the initial solution methods. We first compare with the direct solution approach results given in Figure 6.13. It would seem clear that the Heuristic methods perform much better than the two-minute direct approach (top graph in Figure 6.13), where the majority of instances (130 out of the 240, i.e. $\approx$ 54.17%) had a gap to the best solution of over 90%. However, the two-minute direct method also equalled the best known solution on 6 instances, which as 2.5% of the 240 test runs is a better percentage than the Heuristics were able to achieve. Comparing with the one-hour direct approach (bottom graph in Figure 6.13), we are reminded that 53 instances ($\approx$ 22.08% of the total) were solved to their best known solution by this method. However, for those instances which did not the peak category was a gap of 60% to 65%, which is greater than the majority of heuristic results. This would suggest that the direct approach is more volatile that the heuristics - potentially finding better solutions, but not capable of doing so consistently. Comparing with the lowest-cost change-minimization results given in Figure 6.14 is consistent with the



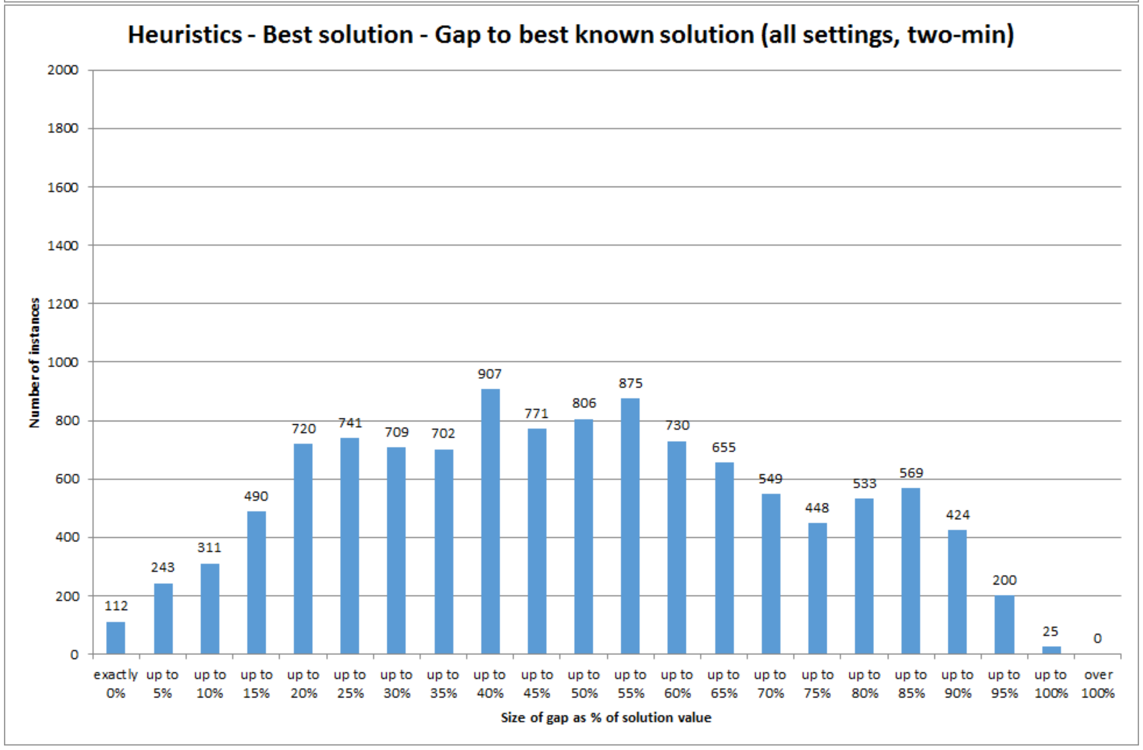


Figure 6.31: Gap to the best known solution overall from the solution found after one minute (top) and two minutes (bottom), across all test instances and all combinations of settings.

comparison of the gaps to the best known bound, suggesting that in general the change-minimization approach is outperforming the heuristics. Nearly half the instances (114, i.e. 47.5% of the total) had a lowest cost change-minimization solution within a 25% gap of the best known solution, compared to 2539 test runs ($\approx$ 22.04%) for the one-minute and 2617 test runs ($\approx$ 22.72%) for the two-minute heuristic results. Finally, comparing with the non-cost-constrained change-minimization solution shown in Figure 6.15 shows that better results are clearly achieved by the heuristics. The non-cost-constrained results have minimum gap of $\approx$ 60.56% and a mean gap of $\approx$ 88.00% to the best known solution; the majority of test runs achieve a gap of under 60% for the heuristics, with the mean gap almost half that of the non-cost-constrained change-minimization solutions.

The above paragraphs have discussed the combined results for all heuristic settings; however we now investigate whether the different settings used, as described in section 6.5.4.14, have an influence on the gaps to the best known solutions. It is believed that the distributions of the data shown in the graphs in Figure 6.31 approximate sufficiently to a Normal distribution that we can perform our Analysis of Variance in this case using the F-test. The p-values obtained from these tests are shown in Table 6.3, with those significant at the 5% level marked with an asterisk. As with the gap to best known bound,

Table 6.3: Influence of heuristic settings on gap to best known solution (F-test).

Setting type	p-value for one-minute result	p-value for two-minute result
Employees to examine	0.004*	< 0.001*
Employee order	0.541	0.410
Acceptance criteria	0.780	0.503
Kick settings	0.001*	< 0.001*
Initial solution	< 0.001*	< 0.001*

neither the order in which employees are examined nor the acceptance criteria used have a significant influence on the gap to the best known solution. The three other settings - the number of employees to examine at each iteration, the kick settings and the initial solution used - on the other hand all have a statistically significant influence on the size of the gap to the best known solution. For all three of these settings, the influence is significant for both the one-minute and the two-minute results.

In order to determine the manner of these influences, we must again look in detail at the breakdown of the results for the different values of these settings. Graphs of these breakdowns are shown in the appendix in section D.2.2, and can be seen to follow roughly the same pattern for each subgroup. Here, we again give only the central measures of mean and median for each subgroup in order to evaluate the nature of the influence. Looking first at the gaps to the best known solution broken down by the number of employees to examine at each iteration, we have for the one-minute results that

- When all employees are examined, the mean gap is $\approx 47.09\%$ and the median is $\approx 46.63\%$;
- When only one third are examined, the mean gap is $\approx 45.81\%$ and the median is $\approx 45.07\%$.

And for the two-minute solutions, we have that

- When all employees are examined, the mean gap is $\approx 46.88\%$ and the median is $\approx 46.13\%$;
- When only one third are examined, the mean gap is $\approx 45.19\%$ and the median is $\approx 44.61\%$.

This shows that solutions tend to be found nearer to the best known solution when only one third of employees are examined at each iteration, which is consistent with the findings relating to the gap to the best known bound. As before, this could be attributed to the shorter iteration length possible when examining a reduced number of employees, which in turn may provide more opportunity for the algorithm to find better solutions.

Looking next at the breakdown according to the kick activation setting used, for the one-minute results we have that

- When no kick is used, the mean gap is $\approx 45.39\%$ and the median is $\approx 44.88\%$;
- When the kick is activated after 4 or more iterations without improvement, the mean gap is $\approx 47.49\%$ and the median is $\approx 46.91\%$;
- When the kick is activated after 8 or more iterations without improvement, the mean gap is $\approx 46.48\%$ and the median is $\approx 45.46\%$.

And for the gaps after two minutes, we have that

- When no kick is used, the mean gap is $\approx 44.58\%$ and the median is $\approx 44.07\%$;
- When the kick is activated after 4 or more iterations without improvement, the mean gap is $\approx 47.33\%$ and the median is $\approx 46.62\%$;
- When the kick is activated after 8 or more iterations without improvement, the mean gap is $\approx 46.20\%$ and the median is $\approx 45.03\%$.

This shows the same pattern as was observed for the gap to the best known bound - the best solutions appear to be found when a kick is not applied at all, while the 8-or-more-iterations rule gives better results on average than then a kick can be applied after

4-or-more iterations without improvement. As suggested above, further investigation into kick settings may be desirable, with a longer time perhaps being required after a kick to make improvement before the next one is applied. This is left as a proposal for future research into improvements.

Finally, breaking down the results according to the initial solution used, we have for the one-minute gaps that

- When the Task-Based Approximation is used, the mean gap is $\approx 51.19\%$ and the median is $\approx 50.81\%$;
- When the Heuristic initial solution is used, is $\approx 41.72\%$ and the median is $\approx 40.24\%$.

And for the two-minute results, we have that

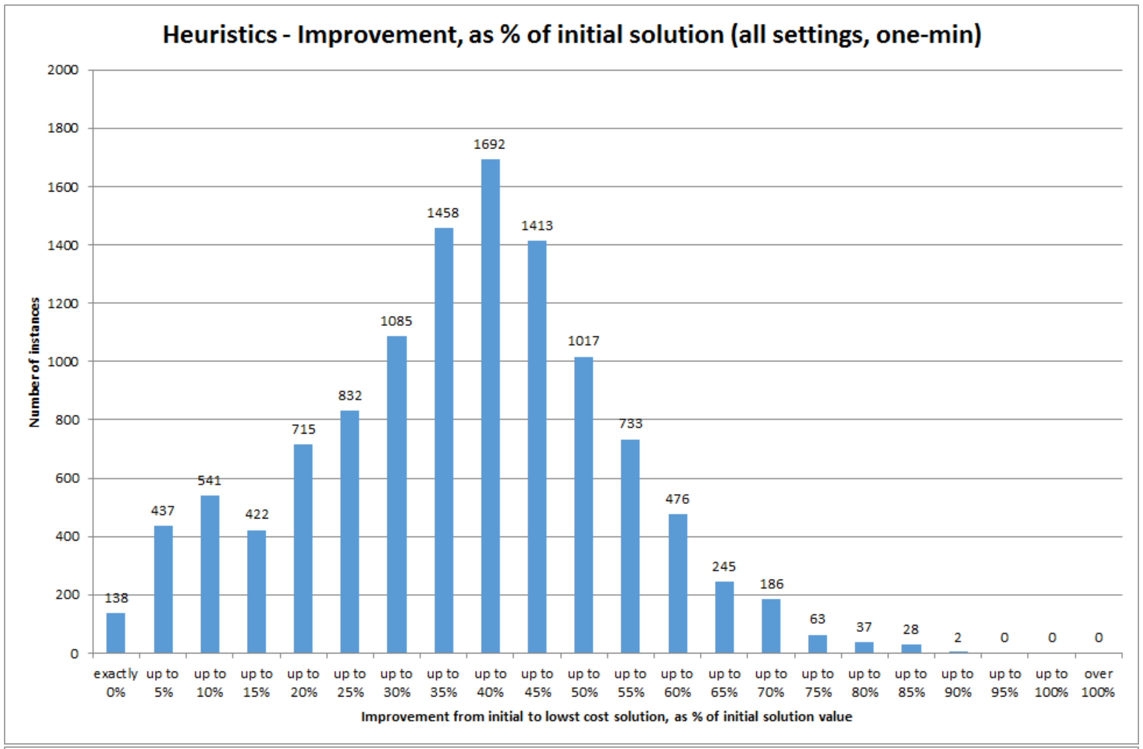
- When the Task-Based Approximation is used, the mean gap is $\approx 50.73\%$ and the median is $\approx 50.52\%$;
- When the Heuristic initial solution is used, is $\approx 41.34\%$ and the median is $\approx 39.78\%$.

The difference here is even more marked than for the gaps to the best known bounds, with the test runs starting from the Heuristic initial solution achieving an average gap to the best known solution of around 10 percentage points better than those runs using the Task-Based Approximation as the initial solution. The question remains however as to whether this can be attributed solely to the Heuristic initial solution itself having a lower cost than the Task-Based Approximation solution, or whether the type of initial solution has a bearing on the amount of improvement achieved by the algorithm. This is discussed in section 6.8.4.3, which follows.

6.8.4.3 Improvement on initial solution

Having discussed in sections 6.8.4.1 and 6.8.4.2 above the quality of the solutions produced by the heuristic algorithm, we now examine the amount of improvement achieved by the algorithm from the initial solution through to the solution found after one and two minutes. The reduction in cost achieved can be represented as a percentage of the cost of the initial solution, and is summarised in histograms for both the one-minute and two-minute solutions for all instances and all combinations of settings as shown in Figure 6.32.

As may be expected having seen the results with regard to the gap to the best known bound and best known solution, it can be seen here that the two graphs are very similar, again showing a similarity between the level of improvement possible within one minute compared to within two minutes. Both charts are roughly Normal in shape, with a peak in the category of over 35% and up to 40% improvement on the initial solution, which accounts



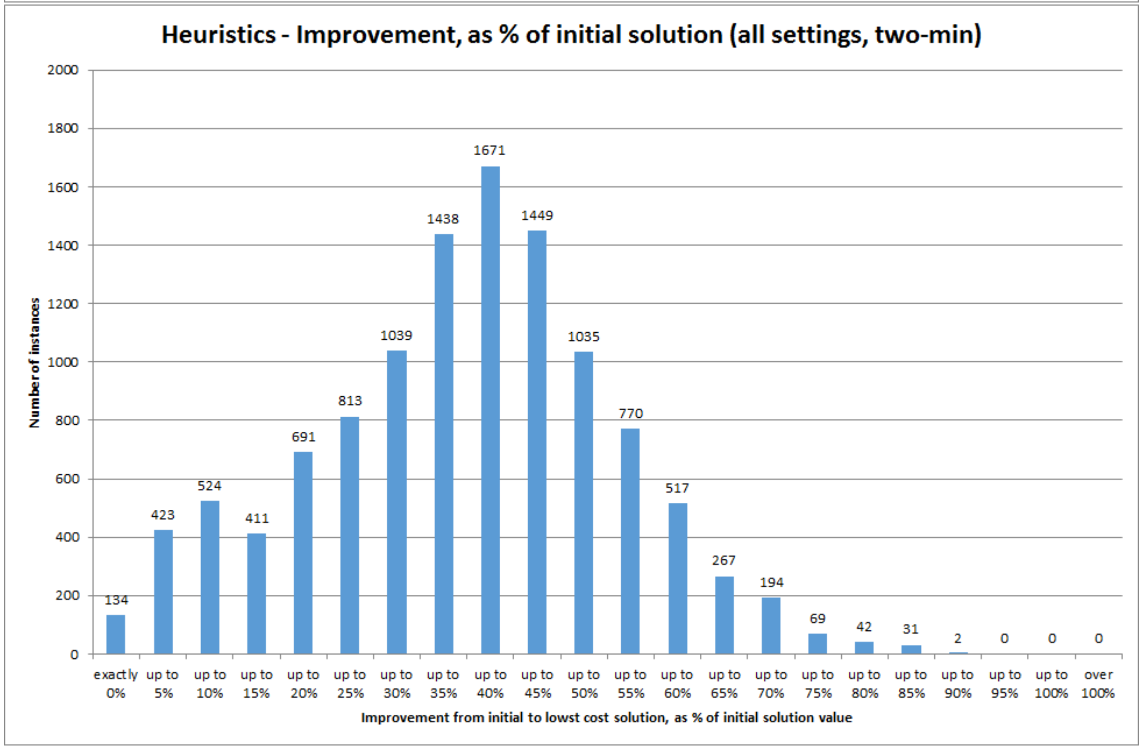


Figure 6.32: Improvement on the initial solution achieved within one minute (top) and two minutes (bottom) given as a percentage of the initial solution value, across all test instances and all combinations of settings.

for 1692 test runs ($\approx$ 14.69% of the total) in the one-minute results and 1671 test runs ($\approx$ 14.51% of the total) in the two-minute results. The mean improvements lie just outwith this category, being $\approx$ 34.18% for the one-minute and $\approx$ 34.69% for the two-minute results; however, the median improvement percentages do lie within the central range, at $\approx$ 35.33% and $\approx$ 35.90% for the one-minute and two-minute results respectively. The concentration of data points around the central category is high, with nearly two-fifths of the test runs showing an improvement in the central three categories - 4563 results ($\approx$ 39.61% of the total) for the one-minute runs and 4558 ($\approx$ 39.57% of the total) for the two-minute lie in the range of greater than 30% and up to 45% improvement on the initial solution. The maximum improvement achieved was $\approx$ 88.08% in both the one-minute and two-minute time limits, while at the other end of the scale it can be seen that the algorithm was able to find no improvement on the initial solution within one minute in 138 cases ($\approx$ 1.20% of the total), with 134 of these ($\approx$ 1.16% of the total) achieving no improvement within two minutes either.

We now turn to the question of whether the different settings used (as described in section 6.5.4.14) have an influence on the degree of improvement that the algorithm is able to achieve. Looking at the graphs in Figure 6.32, it would appear that the distribution of the data is sufficiently close to Normal that the F-test can be used for the Analysis of Variance. The p-values obtained from these tests are shown in Table 6.4, with those significant at the 5% level marked with an asterisk. As might be expected given the analysis of the

Table 6.4: Influence of heuristic settings on improvement on initial solution (F-test).

Setting type	p-value for one-minute result	p-value for two-minute result
Employees to examine	< 0.001*	< 0.001*
Employee order	0.142	0.077
Acceptance criteria	0.548	0.169
Kick settings	< 0.001*	< 0.001*
Initial solution	< 0.001*	< 0.001*

gaps to the best known bound and the best known solution, both the number of employees to examine at each iteration and the kick settings used display a significant influence on the amount of improvement achieved over both one and two minutes. As before, the order in which employees are ordered and the acceptance criteria used have no significant effect on the improvement for either set of results. Interestingly, the initial solution used has a statistically significant influence on the amount of improvement achieved over both one minute and two minutes. While a potential explanation for the significance of the initial solution on the gaps (Tables 6.2 and 6.3) was the difference in initial solution values, the improvement measure controls for this. Therefore, this finding would suggest that there is some alternative explanation. This will become more clear below when we examine the results breakdown.

Turning now to the breakdown of the results according to these settings, we again present graphs of these breakdowns in the appendix in section D.2.3, while here we will give only the central measures of mean and median for each subgroup. Looking first at the improvement achieved broken down by the number of employees to examine at each iteration, we have for the one-minute results that

- When all employees are examined, the mean improvement is $\approx$ 33.31% and the median is $\approx$ 34.38%;
- When only one third are examined, the mean improvement is $\approx$ 35.05% and the median is $\approx$ 36.05%.

And for the two-minute solutions, we have that

- When all employees are examined, the mean improvement is $\approx$ 33.63% and the median is $\approx$ 35.04%;
- When only one third are examined, the mean improvement is $\approx$ 35.74% and the median is $\approx$ 36.67%.

Consistent with the findings relating to the gap measures, this shows that those test runs where only one third of employees are examined at each iteration are able to make a greater improvement on the initial cost than those tests run using the ‘all employees’ setting.

The similar pattern is observed in the breakdown according to the kick activation settings. For the one-minute results we have that

- When no kick is used, the mean improvement is $\approx$ 35.33% and the median is $\approx$ 36.39%;
- When the kick is activated after 4 or more iterations without improvement, the mean improvement is $\approx$ 32.92% and the median is $\approx$ 33.88%;
- When the kick is activated after 8 or more iterations without improvement, the mean improvement is $\approx$ 34.27% and the median is $\approx$ 35.63%.

While for the improvement after two minutes, we have that

- When no kick is used, the mean improvement is $\approx$ 36.20% and the median is $\approx$ 37.23%;
- When the kick is activated after 4 or more iterations without improvement, the mean improvement is $\approx$ 33.16% and the median is $\approx$ 34.09%;

- When the kick is activated after 8 or more iterations without improvement, the mean improvement is $\approx 34.70\%$ and the median is $\approx 36.07\%$.

As before, we see that on average the best improvements are achieved when no kick is applied, while in those cases where the kick is used it appears more beneficial to allow a longer time between each kick. As discussed previously, it is believed that this allows the algorithm more time to improve after each kick, and that future work could be carried out to determine whether an even longer time between kicks could in fact be beneficial.

Finally, we break down the improvement measure according to the initial solution used. For the one-minute results we have that

- When the Task-Based Approximation is used, the mean improvement is $\approx 29.85\%$ and the median is $\approx 30.33\%$;
- When the Heuristic initial solution is used, the mean improvement is $\approx 38.50\%$ and the median is $\approx 39.17\%$.

And for the two-minute results, we have that

- When the Task-Based Approximation is used, the mean improvement is $\approx 30.51\%$ and the median is $\approx 31.05\%$;
- When the Heuristic initial solution is used, the mean improvement is $\approx 38.86\%$ and the median is $\approx 39.63\%$.

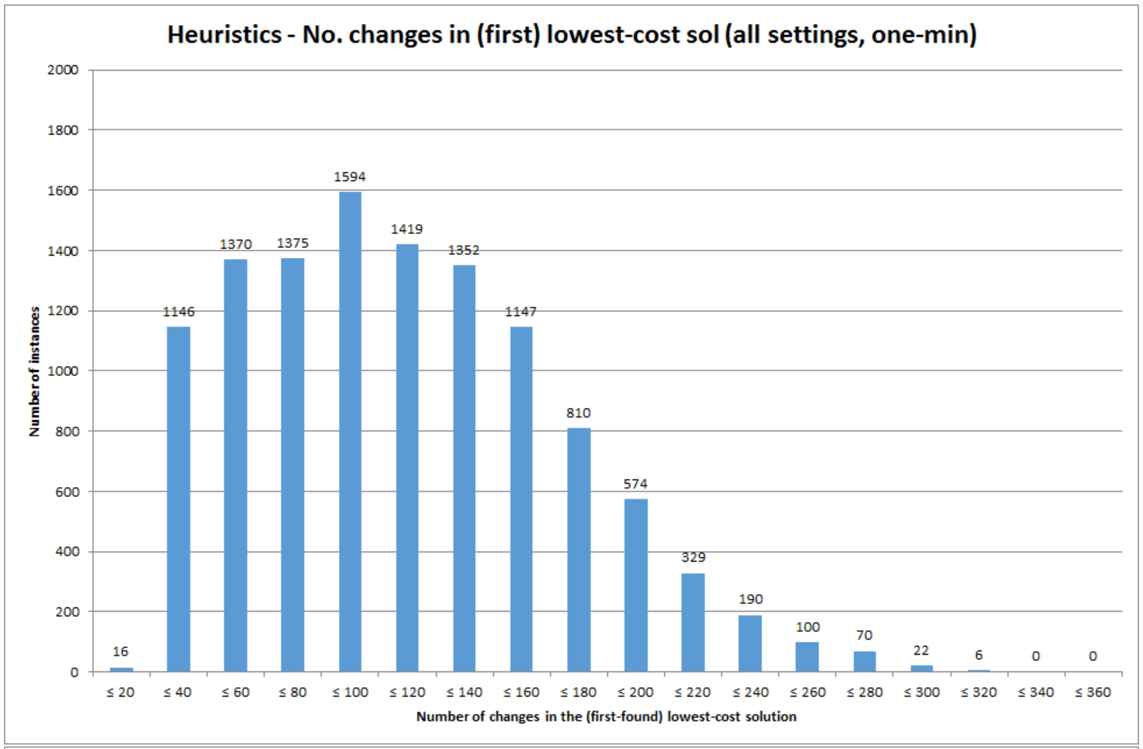
It can be seen here that there is markedly better improvement in the instances which begin from the Heuristic initial solution as opposed to the Task-Based Approximation solution. Bearing in mind that the Heuristic initial solutions have, in general, a lower cost than the Task-Based Approximation solutions this is particularly notable, since the improvement is therefore a greater percentage of a smaller cost. This can perhaps be attributed to the nature of the Heuristic initial solution method, whereby solutions are created which make as far as possible a small number of changes to the existing schedule; the Task-Based Approximation approach on the other hand creates more disruption by artificially imposing 'tasks' on the schedule. Consequently, it may be easier for the algorithm to modify the Heuristic initial solution such that costs are reduced but that the solution overall is still fairly similar to the existing schedule. This conjecture can be examined further by considering the number of changes in the solutions found, as discussed in section 6.8.4.4 below.

6.8.4.4 Number of changes

Having discussed in detail in the quality of the solutions found in terms of cost, we now present the results with regard to the number of changes in the solutions found. Clearly, multiple solutions are generated in each running of the heuristic algorithm; here however we focus on the number of changes in the lowest-cost solution found by the algorithm, and specifically in cases where more than one distinct solution was found with this value we give the result relating to the first solution found with this value. It is believed that this gives a reasonable representation of the number of changes given the closer relationship which appears to be present between cost and number of changes in the Time-Windows version of the problem (see section 6.3.2). As before, we present two graphs for the number of changes in the first-found lowest-cost solution within one minute and within two minutes, summarised for all instances and all combinations of settings. This is given in Figure 6.33.

As with the cost graphs in Figure 6.30 and Figure 6.31, we notice that the two graphs shown here are very similar; this is not particularly surprising, given that we know the costs of the solutions found after one and two minutes are similar, and that we believe number of changes to be closely linked to cost. We can see from these graphs that a small number of the first-found lowest-cost solutions (16 of the solutions ($\approx 0.14\%$ of the total) found within one minute, and 15 of the solutions ($\approx 0.13\%$ of the total) within two minutes) entail 20 or fewer changes. There is then a considerable jump in both of the graphs, with a large number of cases having a number of changes in the next seven categories - 9403 test runs ($\approx 81.62\%$ of the total) for the one-minute results and 9268 cases ($\approx 80.45\%$ of the total) for the two-minute results have a number of changes greater than 20 and less than 160. This comes to a peak in the greater than 80 and up to 100 changes category, which accounts for 1594 one-minute results ($\approx 13.84\%$ of the total) and 1522 two-minute results ($\approx 13.21\%$ of the total). Both graphs exhibit a long tail, with maximum values of 318 changes for the one-minute and 308 changes for the two-minute results. This can be seen to have the effect of pulling the mean upwards to a higher value than the median - calculating these, we have that the mean number of changes is ≈ 109.22 for the one-minute and ≈ 111.79 for the two-minute results, while the median number of changes is 104 for the one-minute and 107 for the two-minute solutions.

It would seem clear from the graphs in Figure 6.33 that the distribution of the number of changes is not a Normal one. We therefore cannot use the Analysis of Variance, and must instead use the non-parametric equivalent Kruskal-Wallis test to investigate what influence the different settings used (as described in section 6.5.4.14) have on the number of changes in the first-found lowest-cost solution. The p-values from these tests are shown in Table 6.5, with those which are significant at the 5% level marked with an asterisk. The results here are similar to those with relating to cost above, with three of the settings - number of employees to examine at each iteration, kick setting and initial solution - having a statistically significant influence on the number of changes in the solution of interest. Meanwhile, acceptance criteria once again shows no significant effect. In contrast



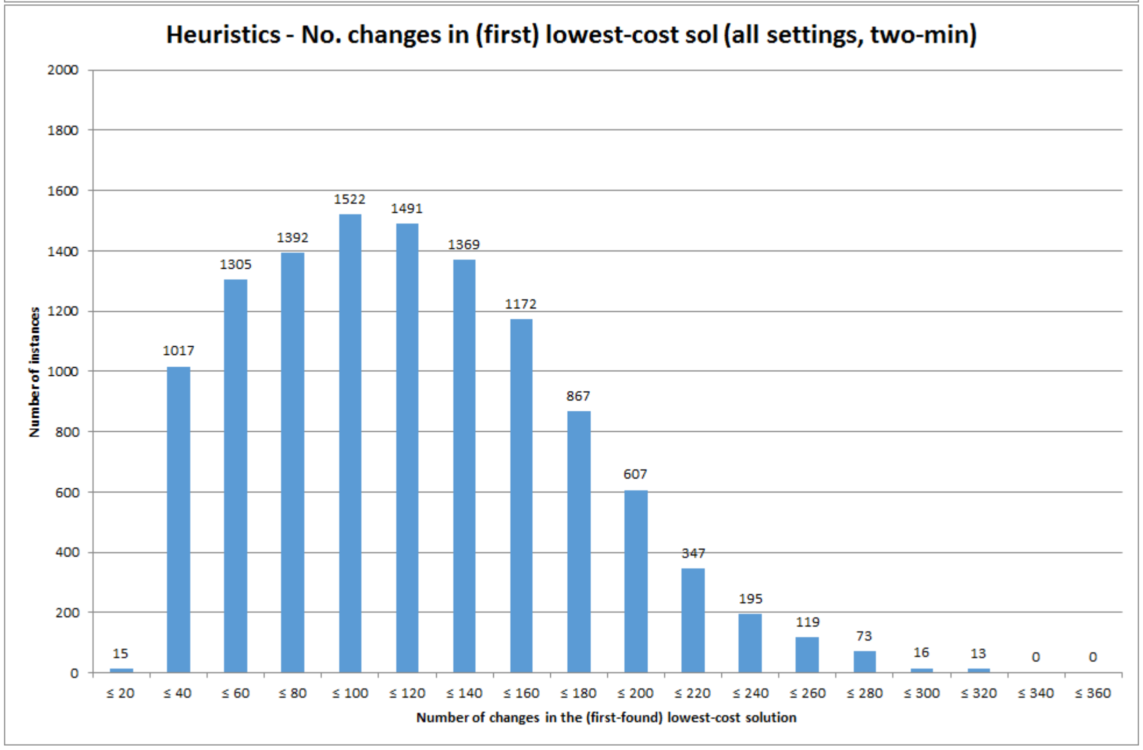


Figure 6.33: Number of changes in the first solution found with lowest cost as at one minute (top) and two minutes (bottom), across all test instances and all combinations of settings.

Table 6.5: Influence of heuristic settings on number of changes in first solution found with lowest cost (Kruskal-Wallis test).

Setting type	p-value for one-minute result	p-value for two-minute result
Employees to examine	< 0.001*	< 0.001*
Employee order	0.168	0.035*
Acceptance criteria	0.417	0.169
Kick settings	< 0.001*	< 0.001*
Initial solution	< 0.001*	< 0.001*

to previous results however, we see that the order in which the employees are examined at each iteration does have a significant influence on the number of changes in the first lowest-cost solution found within two-minutes. It does not though have an influence with respect to the one-minute results.

As before, we now give details of the results breakdown with reference to these significant settings. Graphs of these can be found in the appendix in section D.2.4. In some cases, particularly Figure D.26 relating to the initial solution used, the broken down results display quite marked differences; therefore as well as reporting the mean and median as for previous measures we also where relevant give the minimum and maximum values and comment separately on the distribution of the results.

Looking first at the number of changes in the first lowest-cost solution found broken down by the number of employees to examine at each iteration, we have for the one-minute results that

- When all employees are examined, the number of changes ranges from 0 to 306, with mean ≈ 106.63 and median 101;
- When only one third are examined, the number of changes ranges from 0 to 318, with mean ≈ 111.08 and median 107.

While for the solutions found after two minutes, we have that

- When all employees are examined, the number of changes ranges from 0 to 308, with mean ≈ 108.43 and median 103;
- When only one third are examined, the number of changes ranges from 0 to 307, with mean ≈ 115.16 and median 110.

This indicates that on average a smaller number of changes are found in the solutions when all employees are examined at each iteration. This is in contrast to the findings relating to cost, which found that cheaper solutions could be found when only one third of employees were examined at each iteration. A possible explanation for this is that examining all employees allows the best possible neighbourhood solution to be found at each iteration, which will be unlikely to entail a greater number of changes than the solution which would be implemented if only one third of employees had been considered. This difference at each iteration will cause the number of changes in the 'current' solutions to diverge as the algorithm progresses, causing the difference seen here. This is supported by the observation that the difference in means and difference in medians are both greater in the two-minute results breakdown than in the one-minute results.

We next look at the breakdown of changes according to the ordering of employees at each iteration. This is only significant for the results after two minutes, and for these we have that

- When employees are ordered at random, the number of changes ranges from 0 to 307, with mean ≈ 110.53 and median 105;
- When the tailored ordering rule is used, the number of changes ranges from 0 to 308, with mean ≈ 113.06 and median 108.

This shows that the 'smarter', more tailored ordering rule actually leads on average to an increased number of changes in the first lowest-cost solution. This is an unintended consequence, since the objective of this rule had been to focus in on specific employees which could enable a greater cost saving to be made. In the event, a possible explanation is that by focussing on employees for which there have already been changes, the algorithm becomes more likely to choose a solution which increases the number of changes for that employee and therefore the number of changes overall. It should also be noted that while the objective of the tailored ordering was to help reduce solution cost, there was no significant difference (as shown in Tables 6.2, 6.3 and 6.4) in cost between test runs which used this and those which ordered the employees purely at random.

Looking next at the breakdown according to the kick activation setting used, for the one-minute results we have that

- When no kick is used, the number of changes ranges from 0 to 310, with mean ≈ 105.27 and median 99;
- When the kick is activated after 4 or more iterations without improvement, the number of changes ranges from 2 to 300, with mean ≈ 113.29 and median 108;
- When the kick is activated after 8 or more iterations without improvement, the number of changes ranges from 0 to 318, with mean ≈ 109.08 and median 104.

And for the two-minute change results we have that

- When no kick is used, the number of changes ranges from 0 to 306, with mean ≈ 105.92 and median 101;
- When the kick is activated after 4 or more iterations without improvement, the number of changes ranges from 3 to 308, with mean ≈ 117.59 and median 112;
- When the kick is activated after 8 or more iterations without improvement, the number of changes ranges from 1 to 307, with mean ≈ 111.87 and median 106.

The results here are more in keeping with the cost-related results, with the test runs involving no kick resulting in a lower number of changes on average. This is more in line with what is to be expected, since by its nature the application of kick is likely to produce a new solution with a greater number of changes; even if the algorithm is then able to reduce the cost to find a new best solution, the number of changes is unlikely to return to its pre-kick level. The cases which apply the kick after 4 or more iterations without improvement will have potential for carrying out more kicks in the available time than those which use the 8-or-more-iterations rule, and it is likely to be the effect of this which means the 4-or-more-iteration cases have the higher average number of changes.

Finally we can break down the results according to which type of initial solution was used. For the one-minute results we have that

- When the Task-Based Approximation is used, the number of changes ranges from 6 to 318, with mean ≈ 136.28 and median 136;
- When the Heuristic initial solution is used, the number of changes ranges from 0 to 263, with mean ≈ 82.15 and median 79.

And for the number of changes in the solution after two minutes, we have that

- When the Task-Based Approximation is used, the number of changes ranges from 4 to 308, with mean ≈ 138.38 and median 139;
- When the Heuristic initial solution is used, the number of changes ranges from 0 to 275, with mean ≈ 85.20 and median 82.

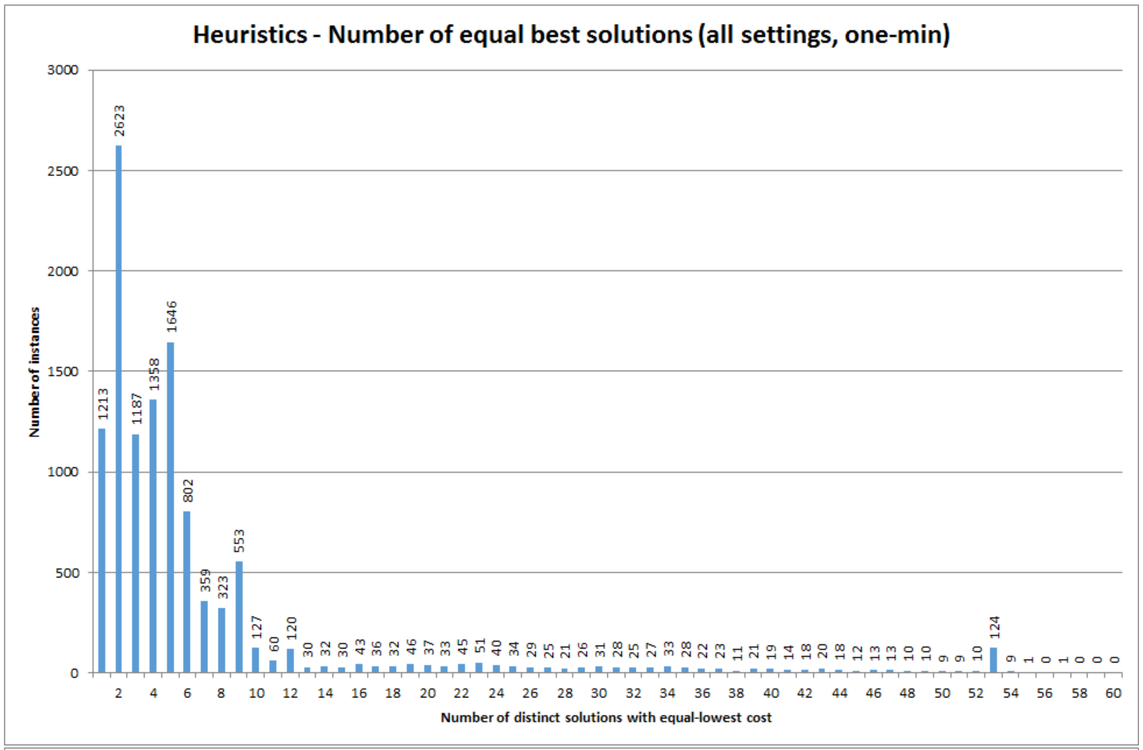
The graphs given in Figure D.26 show that there is a considerable difference in the distributions of the numbers of changes depending on the initial solution used. For those which use the Heuristic initial solution, the distribution is similar in appearance to the overall graphs of Figure 6.33 although with a much lower mean and median, and a much shorter tail (and consequently much lower maximum value). On the other hand, the results for those test runs which use the Task-Based Approximation as the initial solution a distribution more approximating to Normal, centred around a much higher average value (with mean and median very similar) and with a wider range of values overall.

In light of this discrepancy between the results obtained when the different initial solutions are used, it is interesting to compare the number of changes in the one- and two-minute Heuristic approach solutions with the number of changes in the initial solutions themselves. Looking at the number of changes in the Task-Based Approximation solutions, given in the lower graph of Figure 6.21, we see that the number of changes here range from 58 to 355 changes, with a mean of ≈ 192.30 and a median of 191 changes. The number of changes described above for when these solutions are improved using heuristics is around 55 changes fewer on average, showing that the algorithm does indeed reduce the number of changes as well as the cost. The number of changes in the Heuristic initial solution meanwhile are shown in Figure 6.28, and can be seen to cover a much smaller range of 25 to 134 changes, with a mean of ≈ 71.60 and median equal to 70 changes. Notably, this indicates that for the average case the number of changes will in fact increase when the cost is improved using the heuristic algorithm, although only by around 12 changes. This is perhaps explained by the fact that the Heuristic initial solution is designed to be a low-change solution, and therefore additional changes may be required in order to make savings and reduce the cost.

6.8.4.5 Number of best solutions

One of the other key considerations is the number of solutions which are produced by the algorithm. In particular, the heuristic algorithm is likely to produce multiple solutions which have the same cost value but which may be distinct in terms of certain hard-to-model considerations that the Planner may wish to apply. We count the number of equal-best solutions found after one minute and after two minutes, and summarise them for all instances and all combinations of settings in Figure 6.34.

As with all the previous combined graphs, it is noticeable that the one-minute and two-minute results follow a very similar pattern. This is perhaps less surprising than for the costs, since costs are meant to reduce over time. With equal-best solutions, there is no particular reason to believe that the number will be dependent on time; rather, it is more likely to depend on the length of time between the first lowest-cost solution being found and the time that the algorithm terminates. It can be seen that the best solution after one minute is unique in 1213 cases ($\approx 10.53\%$ of the total), and in 1173 cases ($\approx 10.18\%$ of the total) the best solution after two minutes is unique. The most common value is two equal-best solutions, which accounts for 2623 of the one-minute test runs ($\approx 22.77\%$ of the total), and 2554 test runs ($\approx 22.17\%$ of the total) in the two-minute results. More generally, the data can be seen to be grouped predominantly towards the lower end of the charts, with 8027 ($\approx 69.68\%$ of the total) of the one-minute and 7879 ($\approx 68.39\%$ of the total) of the two-minute cases resulting in 5 or fewer equal-best solutions, and 2037 ($\approx 17.68\%$



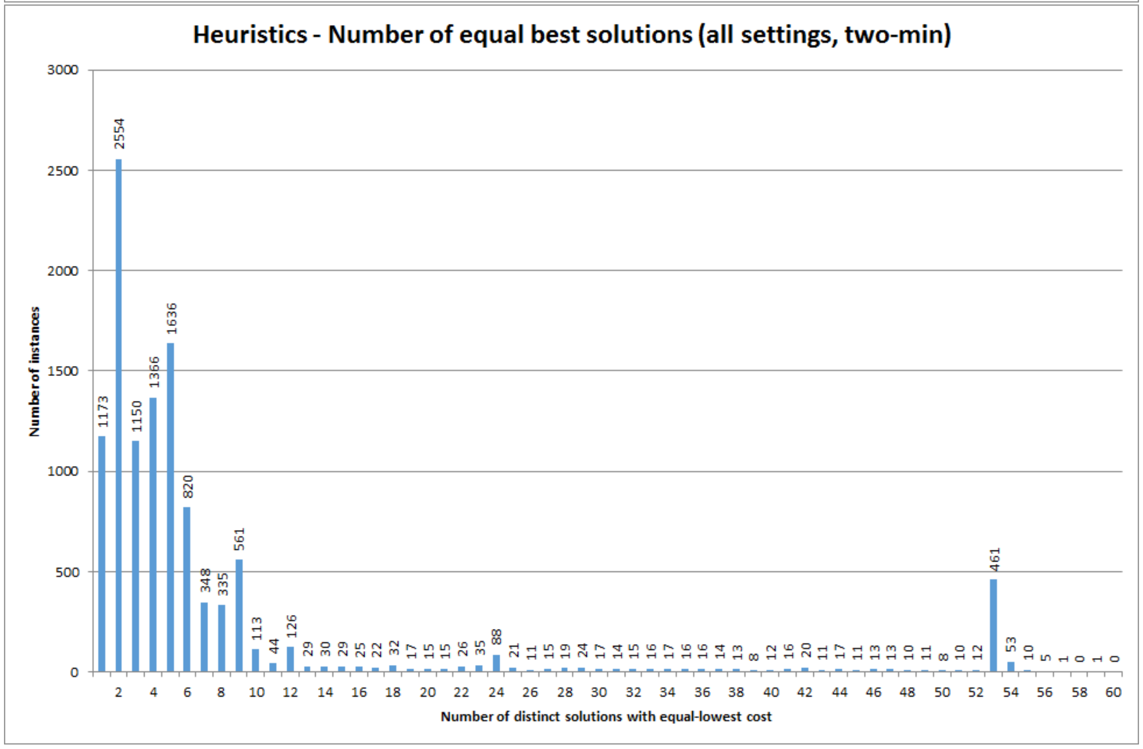


Figure 6.34: Number of distinct solutions with equal-best cost found within one minute (top) and two minutes (bottom), across all test instances and all combinations of settings.

of the total) of the one-minute and 2064 ($\approx$ 17.92% of the total) of the two-minute test runs producing between 6 and 9 equal-best solutions. Only 1456 of the one-minute cases ($\approx$ 12.64% of the total) produced 10 or more equal-best solutions, with the maximum being 55; similarly, 1577 of the two-minute test runs ($\approx$ 13.69% of the total) resulted in 10 or more equal-best solutions, with the maximum here being 59. Curiously, both graphs display a high number of cases producing exactly 53 distinct solutions of equal-best value - 124 cases ($\approx$ 1.08% of the total) for the one-minute results, and 461 ($\approx$ 4.00% of the total) of the two-minute results. Overall, the mean number of equal-best solutions after one minute was mean ≈ 6.74, with the median equal to 4; after two minutes, the mean was a little higher at ≈ 7.96, with the median still equal to 4.

We can now look at the possible effect on the number of equal-best solution of the different settings used, as described in section 6.5.4.14. It is very clear from Figure 6.34 that the data is not Normally distributed in this case, and therefore we must use the Kruskal-Wallis test. The p-values obtained from these tests are shown in Table 6.6, with those which are significant at the 5% level marked with an asterisk. It can be seen that

Table 6.6: Influence of heuristic settings on number of equal-best solutions (Kruskal-Wallis test).

Setting type	p-value for one-minute result	p-value for two-minute result
Employees to examine	< 0.001*	< 0.001*
Employee order	< 0.001*	< 0.001*
Acceptance criteria	0.826	0.941
Kick settings	< 0.001*	< 0.001*
Initial solution	< 0.001*	< 0.001*

this measure is influenced in both the one-minute and two-minute results by four of the five setting types. The number of employees examined at each iteration, the order in which employees are examined, the kick settings and the initial solution used all have a statistically significant effect. The acceptance criteria however does not have any significant influence.

As with our other results, graphs showing the breakdown of the number of equal-best solutions according to these settings are given in the appendix in section D.2.5. Here, we give the basic descriptive information such as mean and median, along with brief discussion of the graphs where appropriate. Looking first at the number of employees to examine at each iteration, we have for the one-minute results that

- When all employees are examined, the mean number of equal-best solutions is ≈ 7.87 and the median is 5;

- When only one third are examined, the mean number of equal-best solutions is ≈ 5.62 and the median is 3.

While for the two-minute test runs, we have that

- When all employees are examined, the mean number of equal-best solutions is ≈ 9.09 and the median is 5;
- When only one third are examined, the mean number of equal-best solutions is ≈ 6.83 and the median is 3.

This shows that when all employees are examined at each iteration, the algorithm is more likely to find more distinct equal-best solution. This is not surprising, as the more employees are examined the wider the search area becomes. Note though that these equal best solutions are achieved at the expense of solution cost - as we saw in section 6.8.4.2, examining all employees (as opposed to only one third) tends to lead to solutions further away from the best known solution cost.

Looking next at the breakdown of the occurrence of equal-best solutions by the employee ordering used, we have for the one-minute results that

- When employees are ordered at random, the mean number of equal-best solutions is ≈ 8.29 and the median is 5;
- When the tailored ordering rule is used, the mean number of equal-best solutions is ≈ 5.19 and the median is 3.

And for the two-minute results that

- When employees are ordered at random, the mean number of equal-best solutions is ≈ 10.24 and the median is 5;
- When the tailored ordering rule is used, the mean number of equal-best solutions is ≈ 5.68 and the median is 3.

Here, we see that the random ordering of employees leads to a much higher number of equal-best solutions on average than using the 'smarter' tailored ordering rule. This might be explained by the diversification element of the tailored ordering, which encourages the search to move away from local optima when these are found. In other words, when a new best solution is found and then repeated, it then selects a new ordering and focusses on a new group of employees for examination, thereby moving to a different part of the search space. This might explain the prevalence of cases with exactly two solutions of equal-best value.

Next, breaking the results according to the kick activation settings gives some clear distinctions. For the one-minute results we have that

- When no kick is used, the mean number of equal-best solutions is ≈ 12.12, the median is 5, and the maximum is 57;
- When the kick is activated after 4 or more iterations without improvement, the mean number of equal-best solutions is ≈ 3.49, the median is 4, and the maximum is 9;
- When the kick is activated after 8 or more iterations without improvement, the mean number of equal-best solutions is ≈ 4.62, the median is 4, and the maximum is 12.

And for the number of equal-best solutions after two minutes, we have that

- When no kick is used, the mean number of equal-best solutions is ≈ 15.75, the median is 6, and the maximum is 59;
- When the kick is activated after 4 or more iterations without improvement, the mean number of equal-best solutions is ≈ 3.51, the median is 4, and the maximum is 9;
- When the kick is activated after 8 or more iterations without improvement, the mean number of equal-best solutions is ≈ 4.63, the median is 4, and the maximum is 12.

It can be clearly seen that the effect of having no kick procedure is to produce far more equal-best solutions on average. It is particularly relevant to observe the maximum values, which show that across all test runs which do make use of the kick the maximum number of equal-best solutions was just 12. All values greater than this are attributable to the 'no kick' group. This is probably because of the effect of the kick to move the search away from a local optimum but in general fail to allow suitable opportunity to return to a value near to this. This is supported by the observation that the cases where longer is allowed for the solution to improve after a kick - i.e. the 8-or-more-iteration setting - have on average and at a maximum a greater number of equal best solutions than those which use the 4-or-more-iteration setting. This adds further weight to the call above for further investigation into a longer gap between kicks to allow the algorithm more time for improvements.

Finally, we can see the breakdown of numbers of equal-best solutions according to the initial solution type. For the one-minute results we have that

- When the Task-Based Approximation is used, the mean number of equal-best solutions is ≈ 6.88 and the median is 4;
- When the Heuristic initial solution is used, the mean number of equal-best solutions is ≈ 6.60 and the median is 4.

And for the two-minute results we have that

- When the Task-Based Approximation is used, the mean number of equal-best solutions is ≈ 8.37 and the median is 4;

- When the Heuristic initial solution is used, the mean number of equal-best solutions is ≈ 7.55 and the median is 4.

We see that while there is a difference between means, the difference is small compared to the differences caused by the other settings. Those test runs which made use of the Task-Based Approximation as the initial solution show a slightly higher average number of equal-best solutions than those which use the Heuristic initial solution. As with the employees examined at each iteration, this is in contrast to the findings with regard to solution cost which indicated the Heuristic initial solution allowed the algorithm to find lower-cost solutions. We can perhaps conclude that at times there is a trade-off between finding lower-cost solutions, and finding multiples of these.

6.8.4.6 Number of iterations

Linked to the number of equal-best solutions found is the number of iterations carried out. While the number of distinct solutions found overall was not able to be recorded, it is reasonable to suppose that in general the more iterations that are carried out, the more solutions have been found. Some of these, while not being equal in cost to the best solution found may be close. Therefore we look at the number of iterations carried out by the algorithm both within one minute and within two minutes, and can summarise these for all instances and all combinations of settings in Figure 6.35.

It is noticeable that these two charts follow a similar pattern of a large number of cases towards the lower end, but not at the extreme, and tapering off to the right. The scale however is obviously different, with the two-minute results more spread out and with a lower peak than the top graph showing the one-minute results. For the one-minute results, we see the most common number of iterations is in the range of 51 to 75, accounting for 2959 test runs ($\approx 25.69\%$ of the total); for the two-minute results, the range of 101-125 iterations is the most common, with 1943 cases ($\approx 16.87\%$ of the total) contained in it. A high proportion of the test runs, 8932 of the 11520 or $\approx 77.53\%$, record 150 or fewer iterations with one minute; doubling this, we find 9609 cases ($\approx 83.41\%$ of the total) for which up to 300 iterations are recorded within two minutes. Overall, the maximum number of iterations within one minute is 341, with the mean equal to ≈ 114.05 and a median of 107 iterations; for the two-minute results, the maximum number of iterations is 661, while the mean is ≈ 204.10 and the median is 186. It is noticeable that the number of iterations performed in two minutes appears less than double that of the one-minute value. A possible reason for this is that as the algorithm progresses, it becomes harder to find improving

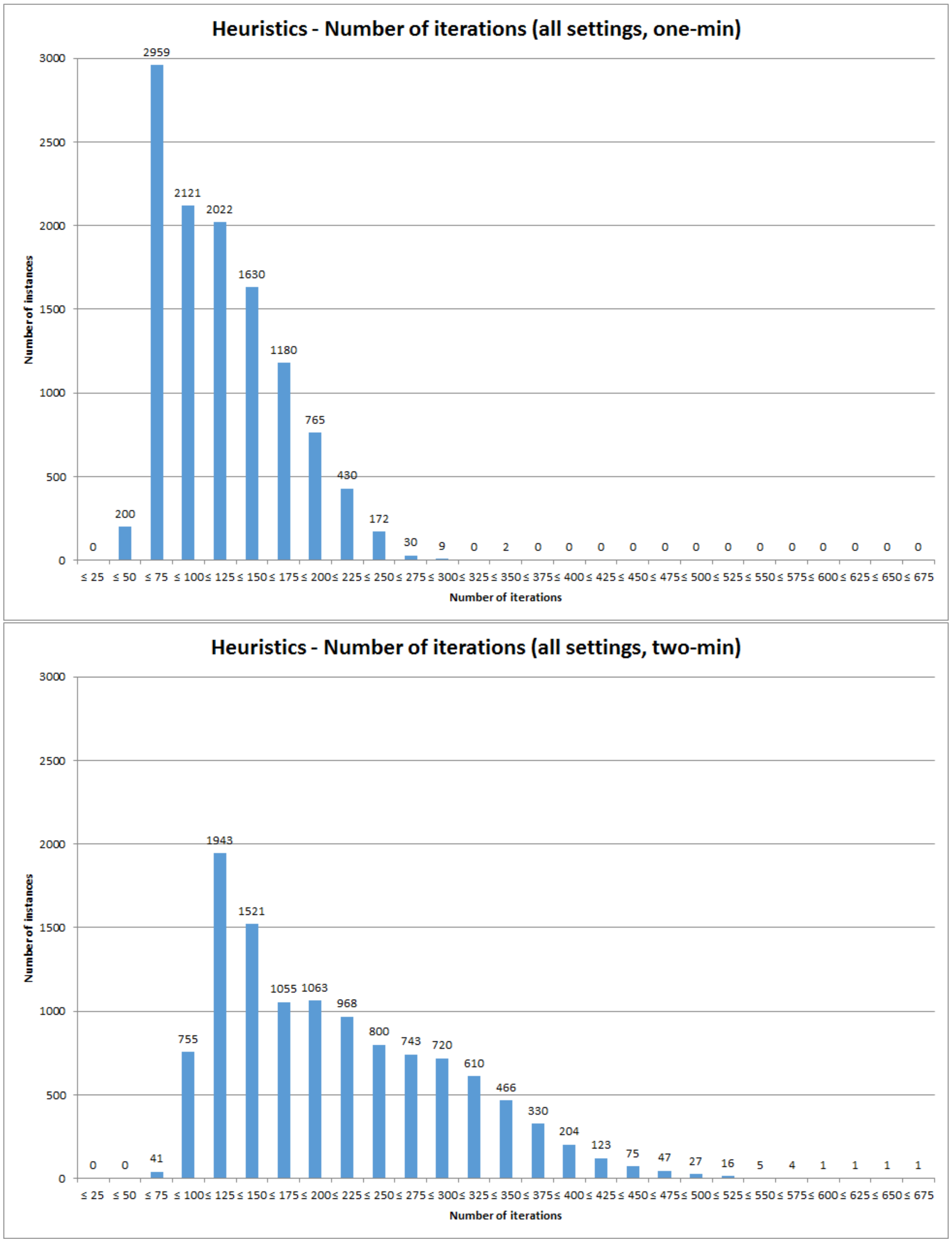


Figure 6.35: Number of iterations carried out by the algorithm within one minute (top) and two minutes (bottom), across all test instances and all combinations of settings.

moves, meaning that more employees must be examined before a change is accepted. This would mean an increase in the iteration length as the algorithm progressed.

Looking now at the influence that the different settings used (as described in section 6.5.4.14) may have on the number of iterations, we note that the graphs in Figure 6.35 do not conform to the assumptions about Normality which are required to utilise the F-test. Instead, we use the Kruskal-Wallis test. The p-values obtained from this are shown in Table 6.7, with those significant at the 5% level marked with an asterisk. The significant

Table 6.7: Influence of heuristic settings on number of iterations carried out (Kruskal-Wallis test).

Setting type	p-value for one-minute result	p-value for two-minute result
Employees to examine	< 0.001*	< 0.001*
Employee order	< 0.001*	< 0.001*
Acceptance criteria	< 0.001*	< 0.001*
Kick settings	0.381	< 0.001*
Initial solution	0.051	0.394

settings here are different from those identified for other measures. For the first time, we have that the initial solution type does not have a significant influence at all, while the kick settings appear significant for the number of iterations carried out within two minutes but not within one minute. The acceptance criteria show a significant influence on both one- and two-minute results here having not previously done so for cost gaps, improvement or equal-best solutions. As before, we also have that both the number of employees examined and the order in which this is done have a significant effect on the number of iterations.

We will now look to break down the results given in Figure 6.35 according to the different settings. Graphs showing the breakdown are given in the appendix in section D.2.6, while here we will give relevant summary information about the data. Looking first at the number of employees to examine at each iteration, we have for the one-minute results that

- When all employees are examined, the mean number of iterations is ≈ 76.94 and the median is 73, with a range of 35 to 171 iterations;
- When only one third are examined, the mean number of iterations is ≈ 151.16 and the median is 145, with a range of 67 to 341 iterations.

While for the two-minute time limit we have that

- When all employees are examined, the mean number of iterations is ≈ 132.96 and the median is 127, with a range of 59 to 271 iterations;

- When only one third are examined, the mean number of iterations is ≈ 275.25 and the median is 267, with a range of 120 to 661 iterations.

It is unsurprising to see that a greater number of iterations can be achieved when only one third of employees are examined at each iteration rather than all of them. It may be expected that examining one third of the employees would allow three times as many iterations to be carried out, but this ignores considerations about the lengths of the iterations. The more employees are examined, the more likely it is that a solution will be found which meets the acceptance criteria before the maximum number of employees have been considered; when fewer employees are examined, it is more likely that no solution will meet the automatic acceptance criteria, and the iteration will end with the best available candidate being accepted instead. Hence while the difference here is clear and statistically significant, it is not the case that number of employees and number of iterations are directly proportionate.

Looking next at the order in which employees are examined, we see for the one-minute data that

- When employees are ordered at random, the mean number of iterations is ≈ 108.04 and the median is 106, with a range of 35 to 228 iterations;
- When the tailored ordering rule is used, the mean number of iterations is ≈ 120.06 and the median is 108, with a range of 36 to 341 iterations.

While for the two-minute results we have that

- When employees are ordered at random, the mean number of iterations is ≈ 194.04 and the median is 185, with a range of 63 to 414 iterations;
- When the tailored ordering rule is used, the mean number of iterations is ≈ 214.17 and the median is 187, with a range of 59 to 661 iterations.

We see from this data that more iterations are carried out when the 'smarter' tailored ordering is used than when the ordering is purely random. Looking further into this, we see that the median and minimal values are similar for each group, but looking at the maximum values we have that the large numbers of iterations are achieved by those cases where the tailored ordering is used. This is perhaps a benefit of the diversification introduced by the smarter ordering, and that by encouraging the search to move away from the current optimum towards a less recently explored area we allow the algorithm to find new improvements. This would lead to more iterations (and therefore, it would be expected, more solutions) since once an acceptable solution is found the algorithm moves to the next iteration.

Considering the acceptance criteria used by the algorithm, for the one-minute results we have that

- When requiring improvement on the current solution, the mean number of iterations is ≈ 131.40 and the median is 124, with a range of 40 to 341 iterations;
- When requiring improvement on the best solution, the mean number of iterations is ≈ 96.69 and the median is 91, with a range of 35 to 261 iterations.

While for the numbers of iterations within two minutes, we have that

- When requiring improvement on the current solution, the mean number of iterations is ≈ 236.81 and the median is 218, with a range of 70 to 661 iterations;
- When requiring improvement on the best solution, the mean number of iterations is ≈ 171.40 and the median is 156.5, with a range of 59 to 487 iterations.

It can be seen here that accepting the first non-tabu solution which improves on the current solution allows significantly more iterations to be carried out than waiting for a solution which improves on the best found so far. This is reasonable, since it would be expected that a solution which improves only on the current solution would be easier to find than one which improves on the best so far - although of course this does not apply for those iterations where the current solution is equal to the best found so far. Note also that any solution which improves on the best solution will by definition improve on the current solution as well.

Finally, we break down the number of iterations according to the kick setting used. This was only significant for the two-minute results, for which we have that

- When no kick is used, the mean number of iterations is ≈ 214.54 and the median is 181.5, with a range of 69 to 661 iterations;
- When the kick is activated after 4 or more iterations without improvement, the mean number of iterations is ≈ 197.57 and the median is 187, with a range of 59 to 485 iterations;
- When the kick is activated after 8 or more iterations without improvement, the mean number of iterations is ≈ 200.20 and the median is 185, with a range of 62 to 473 iterations.

It can be seen that there is little difference between the two groups which make use of kicks with the varying activation rules. However, the cases for which no kick was used result in clearly more iterations being carried out on average, and this group has a higher minimum and higher maximum than either of the other groups. Curiously however, the median is lower, and examination of the graphs given in Figure D.35 show that while the distribution for the no-kick group has a longer tail to the right, it also has higher counts in the 101-125 and 126-150 iteration categories. This makes it harder to draw a consistent conclusion, although it can be said that while a higher number of iterations *can* be achieved when no kick is used, this also leads to less consistency in the number of iterations carried out.

6.8.4.7 Time of best solution

In light of the results with respect to cost showing little difference between the solutions found within one minute and after two minutes (for example, as seen in Figure 6.30 or Figure 6.31), it would seem appropriate to look in detail at the point in the algorithm at which the best solution was found. The point at which a solution with the best cost value was first found was recorded for each test run as an iteration number, but not as a time; as shown above, iteration numbers were greatly variable over all the runs, and therefore some manipulation firstly had to be carried out to make this measure comparable across all test runs.

It was discussed for Figure 6.35 above that iteration lengths in the second minute appear to be different than those in the first. However, in the absence of further information, it was assumed that iteration lengths were uniform across each of the two individual minutes of the run. Based on this, and using the iteration and running time information available, it was possible to estimate a time for the finding of the best solution as follows:

1. We say that T_1 is the exact time at which the one-minute results were recorded, and T_2 is the total running time of the two-minute test run;

2. We say that I_1 and I_2 are the total number of iterations carried out within one minute and within two minutes respectively;

3. We say that B_1 and B_2 are the iteration numbers at which the best solution was found as at the one-minute cut-off and after the full two minutes respectively.

4. The time at which the best solution within one minute was found can then be estimated as a proportion of the running time as follows

$$\left(\frac{B_1}{I_1}\right) \times T_1$$

5. If $B_2 = B_1$, then the estimated time at which the best solution within two minutes was found is the same as for the one-minute results; otherwise, it can be estimated as

$$T_1 + \left(\frac{(B_2 - I_1)}{(I_2 - I_1)} \times (T_2 - T_1)\right)$$

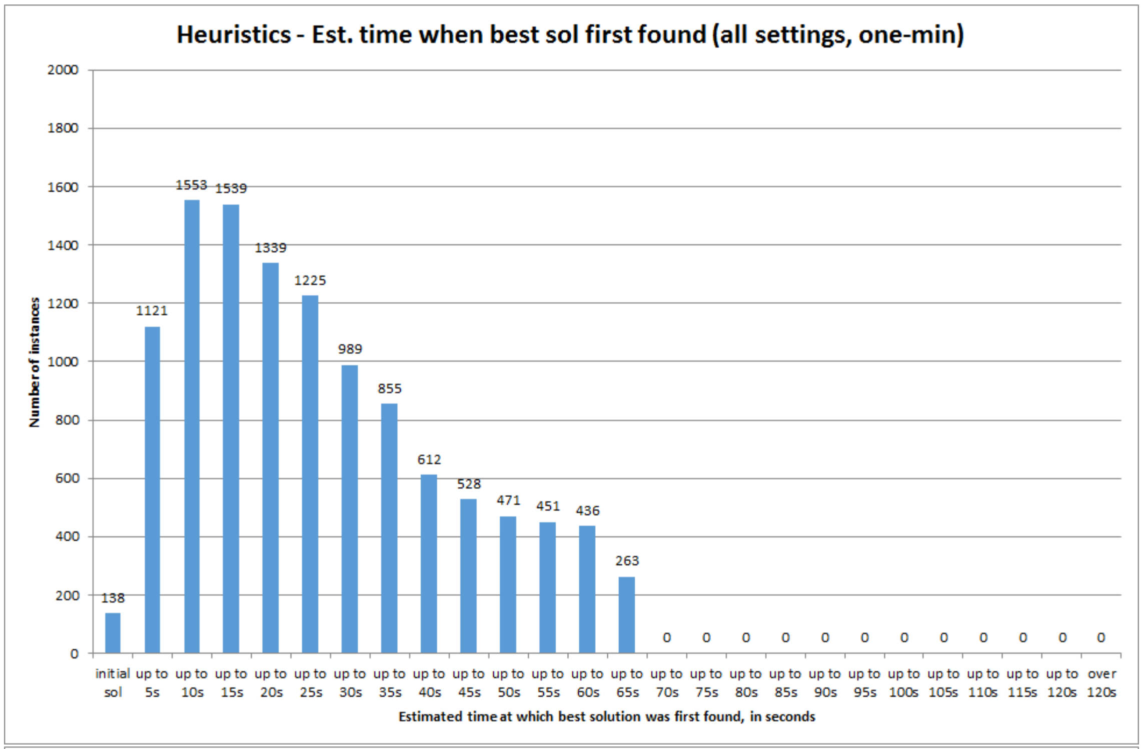
i.e. as a proportion of time over and above the first minute of the test run.

Having calculated these estimated times, they can be summarised in histograms for all instances and all combinations of settings, for both the one-minute and two-minute results. These graphs are shown in Figure 6.36.

Firstly, we note that in both charts there are a small number of cases which appear to have found the best solution after the end of their respective time limits (263 of the one-minute results, $\approx 2.28\%$ of the total; and 40 of the two-minute results, $\approx 0.35\%$ of the total). This can be explained partially by the set-up time of the programme which appears to extend the running time by around one second, and also with the fact that the algorithm terminates at the completion of the first iteration after the time limit has been reached, meaning that the one-minute cut-off and two-minute time limits will actually be timed at approximately 61 seconds and 121 seconds respectively. At the other end of the chart, we see 138 of the one-minute cases ($\approx 1.20\%$ of the total) and 134 of the two-minute cases ($\approx 1.16\%$ of the total) for which the best solution is the initial solution, and therefore the time of the best solution being found is zero seconds. These correspond exactly to the instances in the lowest category of Figure 6.32 which were recorded as making no improvement on the initial solution.

Looking in more detail at the top chart, for the best solution found within one minute, we see that the largest number of solutions are found approximately between 5 and 10 seconds into the algorithm. This category accounts for 1558 cases ($\approx 13.52\%$ of the total); this is closely followed by the neighbouring category of 10 to 15 seconds, which contains 1539 cases ($\approx 13.36\%$ of the total). Nearly half of the test runs (5659, i.e. $\approx 49.44\%$ of the total) find the best solution within approximately the first 20 seconds of the test run, with the frequencies getting progressively smaller as the time increases. The mean time of the best solution within one minute is ≈ 23.61 seconds, while the median is ≈ 20.28 seconds.

It can be seen that the lower graph, showing the approximate time of the best solutions found within two minutes, is actually quite similar in shape. Again, the largest number of solutions (1497 cases, $\approx 12.99\%$ of the total) are found after between approximately 5 and 10 seconds, with the 10 to 15 second category again with the next highest count (1473, $\approx 12.79\%$ of the total). Close to half of the test runs (5450, $\approx 47.31\%$ of the total) still have a best solution which is found within the first 20 seconds of the test run, with only 1629 test runs ($\approx 14.14\%$ of the total) finding the best solution *after* 60 seconds of the run. This fits with the results shown for costs in Figures 6.30, 6.31 and 6.32 which showed very little difference in cost gaps and improvement levels between the one-minute and two-minute results. Overall, the mean time at which the best solution is found is ≈ 30.50 seconds, clearly pulled up by the fact that some solutions are now found between 60 and 125 seconds; the median however is very close to the one-minute median, at ≈ 21.28



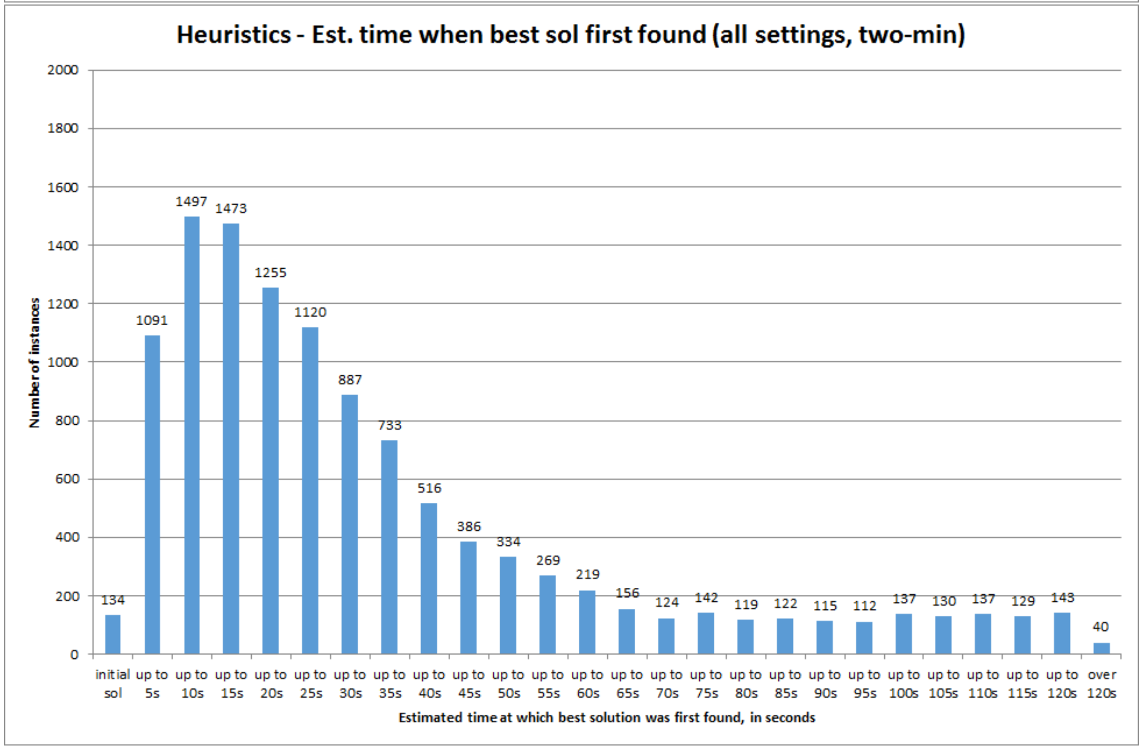


Figure 6.36: Estimated time at which the best solution within one minute (top) and within two minutes (bottom) was first found, across all test instances and all combinations of settings.

seconds for the two-minute results.

We can now look at how these times of the best solution are influenced by the different settings used for the heuristics, as described in section 6.5.4.14. It is clear from Figure 6.36 that the data does not represent a Normal distribution, and therefore we must carry out the Kruskal-Wallis test rather than the F-test. The p-values from these are shown in Table 6.8, with those which are significant at the 5% level marked with an asterisk. It

Table 6.8: Influence of heuristic settings on time that best solution is found (Kruskal-Wallis test).

Setting type	p-value for one-minute result	p-value for two-minute result
Employees to examine	0.611	< 0.001*
Employee order	0.016*	0.072
Acceptance criteria	< 0.001*	< 0.001*
Kick settings	< 0.001*	< 0.001*
Initial solution	0.047*	< 0.001*

can be seen that all five settings have a significant influence on at least one of the two sets of results, with the acceptance criteria, kick settings and initial solution showing a significant effect with respect to both the one-minute and two-minute results. The number of employees examined at each iteration has a bearing on the two-minute results, but not on the time of the best solution found within one minute; conversely, the order in which employees are examined have a significant influence on the one-minute results but not on the two-minute time of the best solution.

Based on these test results, we now break down the time of best solution results according to the settings which show a significant influence. We will only discuss summary statistics here, with more detail given in graphs in the appendix in section D.2.7. We look first at the number of employees to examine at each iteration, which was only significant for the two-minute results. For these two-minute results, we have that

- When all employees are examined, the mean time for the best solution is ≈ 26.85 seconds and the median is ≈ 21.35 seconds;
- When only one third are examined, the mean time for the best solution is ≈ 34.15 seconds and the median is ≈ 21.06 seconds.

This shows that on average, the best solution is found later when only one third of employees are examined, and as can be seen from Figure D.36 this subset of the data has a greater number of test runs falling into each category greater than 55 seconds. However, we also see that the median time is slightly greater when all employees are examined, highlighting that there are also fewer very low values in this subgroup, with the data more closely grouped around the mean. The discovery that best solutions are on average found earlier when all employees are examined can be balanced against the results discussed in section 6.8.4.1 where it was shown that this group also produces a larger gap to the best known bound than when only one third of employees are examined.

We next look at the order in which employees are examined. This was only significant for the one-minute results, for which we have

- When employees are ordered at random, the mean time for the best solution is ≈ 23.73 seconds and the median is ≈ 20.77 seconds;
- When the tailored ordering rule is used, the mean time for the best solution is ≈ 23.50 seconds and the median is ≈ 19.72 seconds.

There is little difference between the means here, but the medians show that those test runs which use the 'smarter' ordering are more likely to reach their best solution earlier. It can be seen in Figure D.37 that the smarter ordering has more cases in the 0 to 5 second and 5 to 10 second categories, as well as accounting for more of the test runs for which no improvement on the initial solution was made. From above, the examination order had no significant impact on the cost gaps or the improvement achieved, but significant influences were found on the number of changes (section 6.8.4.4) and number of equal-best solutions (section 6.8.4.5). These differences were attributed to the intensification and diversification which takes place under the tailored ordering setting; this could also be the case here, with the tailored ordering in some cases allowing the best solution to be found more quickly through the focussing in on certain 'promising' employees. It was also shown in section 6.8.4.6 that test runs which used the smarter ordering generally carried out more iterations than those which were randomized - this leads to a possible explanation that in terms of iterations, both groups find the solutions at around the same time, but because of the shorter iterations this equates to the tailored ordering group finding the best solutions earlier.

We next look at the breakdown according to the acceptance criteria used. For the one-minute results we have that

- When requiring improvement on the current solution, the mean time for the best solution is ≈ 21.49 seconds and the median is ≈ 17.39 seconds;
- When requiring improvement on the best solution, the mean time for the best solution is ≈ 25.74 seconds and the median is ≈ 23.14 seconds.

While for the time of the best solution found within two minutes we have that

- When requiring improvement on the current solution, the mean time for the best solution is ≈ 27.36 seconds and the median is ≈ 18.04 seconds;

- When requiring improvement on the best solution, the mean time for the best solution is ≈ 33.64 seconds and the median is ≈ 24.56 seconds.

It can be seen from these results that requiring improvement on the current solution causes the best solution to be found earlier than waiting for a solution which improves on the best found so far. This can be linked to the findings in section 6.8.4.6 above which discussed that accepting a non-tabu solution which improved on the current solution lead to a significantly higher number of iterations to be carried out. It would seem reasonable to suggest that the main reason that the improve-on-current setting finds solutions more quickly is because it carries out its iterations more quickly. Since the acceptance rule had no significant influence on the quality of the solution found, either in terms of cost or number of changes, we can conclude from this that forcing the algorithm to wait for a solution which improves on the best so far is a waste of time - we can find equally good solutions more quickly with the improve-on-current setting.

Looking next at the breakdown of results according to the kick setting used, we have for the one-minute results that

- When no kick is used, the mean time for the best solution is ≈ 29.07 seconds and the median is ≈ 26.87 seconds;
- When the kick is activated after 4 or more iterations without improvement, the mean time for the best solution is ≈ 19.01 seconds and the median is ≈ 15.44 seconds;
- When the kick is activated after 8 or more iterations without improvement, the mean time for the best solution is ≈ 22.76 seconds and the median is ≈ 19.74 seconds.

While for the solution after two minutes we have that

- When no kick is used, the mean time for the best solution is ≈ 42.47 seconds and the median is ≈ 30.71 seconds;
- When the kick is activated after 4 or more iterations without improvement, the mean time for the best solution is ≈ 21.51 seconds and the median is ≈ 15.69 seconds;
- When the kick is activated after 8 or more iterations without improvement, the mean time for the best solution is ≈ 27.52 seconds and the median is ≈ 20.53 seconds.

The results here show that the best solution is found on average much later when no kick is carried out, while the longer gap between kicks imposed by the 8-or-more-iterations setting appears to lead to a later average time than the 4-or-more-iterations setting. Looking at the graphs in Figures D.39 and D.40 in the appendix it can also be seen that the best solution times when no kick is used are much more uniformly spread across the running time, rather than being clustered towards the lower end as in the graphs for the settings where a kick is used. We can read into this that when no kick is used, the algorithm is more likely to continue to find improvements on the best solution found so far, whereas the kick in general is more likely to move away from the best solution and not subsequently be able to match it. This would be consistent with the findings in, for example, section 6.8.4.1 where it was seen that the test runs which used no kick found solutions with a smaller gap to the best known bound.

Finally, we have the breakdown according to the initial solution type used. Here, for the one-minute results we have that

- When the Task-Based Approximation is used, the mean time for the best solution is ≈ 24.18 seconds and the median is ≈ 20.79 seconds;
- When the Heuristic initial solution is used, the mean time for the best solution is ≈ 23.05 seconds and the median is ≈ 19.81 seconds.

While for the two-minute results we have that

- When the Task-Based Approximation is used, the mean time for the best solution is ≈ 32.21 seconds and the median is ≈ 22.23 seconds;
- When the Heuristic initial solution is used, the mean time for the best solution is ≈ 28.80 seconds and the median is ≈ 20.50 seconds.

Here we see that the best solution tends to be found later when the Task-Based Approximation is used as the initial solution compared to when the Heuristic initial solution is used. It is worth noting however that in the Figure D.41 it can be seen that the Task-Based Approximation shows more cases in the up-to-5-second category than for the Heuristic initial solution; in the averages though this is negated by the lower counts in the other earlier-time categories. This contrast is interesting since it is not attributable to the different iteration lengths (Table 6.7 showed that the initial solution had no significant influence on the number of iterations carried out), nor is it attributable to a poorer quality final solution being found since the Heuristic initial solution actually leads to lower cost solutions (see sections 6.8.4.1 and 6.8.4.2) and a better improvement on the initial solution (as discussed in section 6.8.4.3). We must therefore conclude that the Heuristic initial solution, as well as allowing better improvement, also allows faster improvement towards the best solution which can be found. This may be as a result of the lower number of changes and lower level of disruption to the existing schedule which is created when the Heuristic initial solution is generated, as compared to the Task-Based Approximation solution.

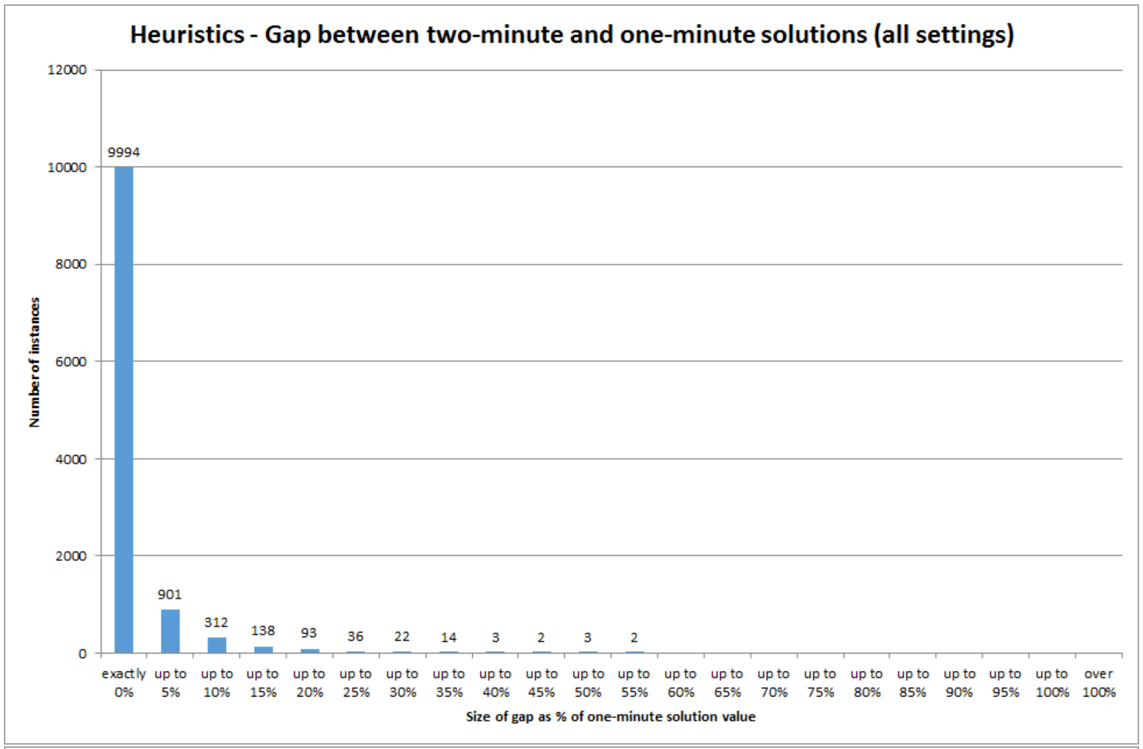
6.8.4.8 Effect of cutting off algorithm early

A theme which has been constant throughout the results presented above is the similarities between the one-minute and two-minute solutions for the majority of test runs. This was particularly evident when discussing the gaps to the best known bound or best known solution (sections 6.8.4.1 and 6.8.4.2), and the improvement on the initial solution (section 6.8.4.3). It could also be seen in section 6.8.4.7 that a large number of the best solutions available after two minutes were actually found within 60 seconds, and in many of these cases even earlier. From these results, it could be suggested that the algorithm could be run for just one minute rather than two, without significant detriment to the quality of the results obtained.

This gives rise to the question of what the effect may be if the algorithm were to be cut short, before even the one-minute cut-off used in the test runs. The solutions available at exact time points (other than the one-minute cut-off) were unfortunately not recorded; however, the best solution value after each iteration was recorded, and therefore the solution at a given time point can be estimated using the same approach used above for estimating the time of the best solution. Using the assumption that all iterations within the first minute are of the same length, we can obtain an iteration number as a proportion of the number of one-minute iterations, according to the number of seconds within the first minute we are interested in. If the number calculated is a fraction, it is rounded *up* to the nearest integer, in recognition of the fact that the algorithm would cut off at the end of the iteration once the time limit had been reached. Once we have the estimated solution value at the given cut-off time, we can calculate the gap between this and the final (i.e. two-minute) solution value, and represent this as a percentage of the early cut-off value. These gaps are summarised in histograms for all instances and all combinations of settings for four different cut-off times. Figure 6.37 shows the gap to the two-minute solution from the one-minute solution and from the estimated 30-second solution; Figure 6.38 summarised the gaps to the two-minute solution from the estimated solution after 20 seconds and after 10 seconds.

Looking first at the top graph of Figure 6.37, showing the gap between the one-minute and two-minute solutions, we see that 9994 test runs ($\approx$ 86.75% of the total) have no difference in best solution cost after two minutes compared to after one minute. Of the small number of test runs which do have a non-zero gap, most of these are at the lower end of the scale with the count tapering off towards the right - we see 901 of the one-minute solutions ($\approx$ 7.82% of the total) are within 5% of the the two-minute solution, while the maximum is $\approx$ 54.73%. Clearly, with more than half the cases having an gap of zero, the median gap between the two solutions is also zero; the mean is $\approx$ 0.80%.

The estimated effect of a 30-second cut-off is shown in the bottom graph of Figure 6.37.



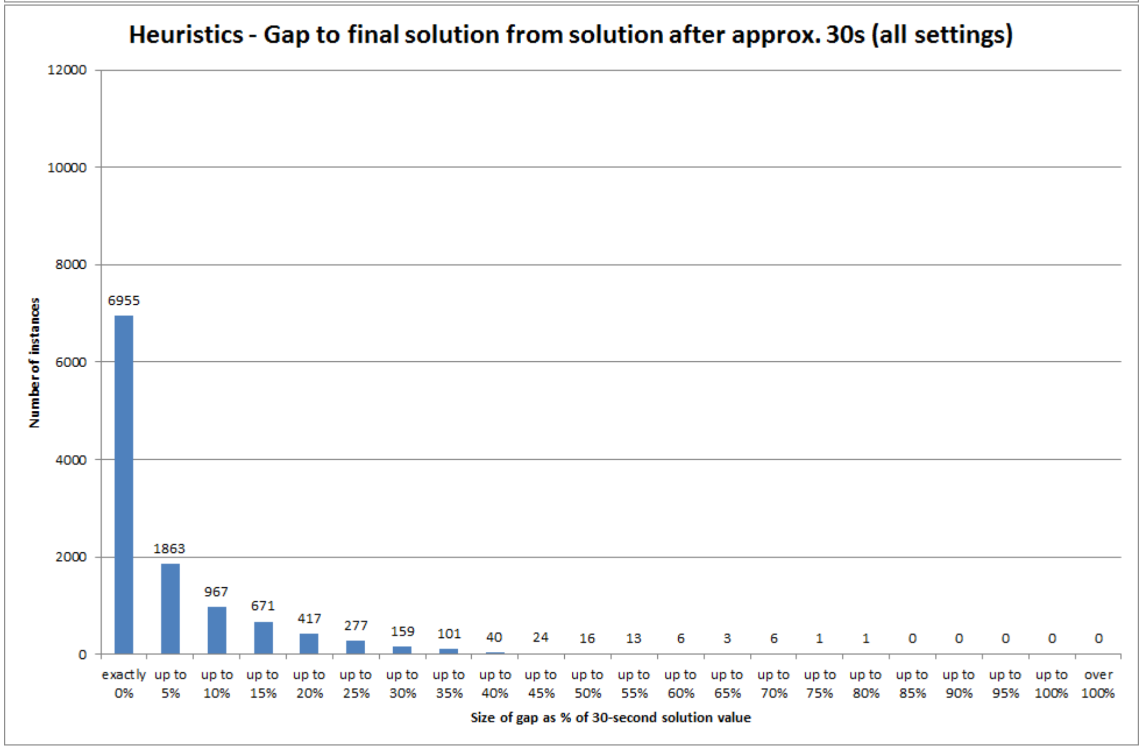
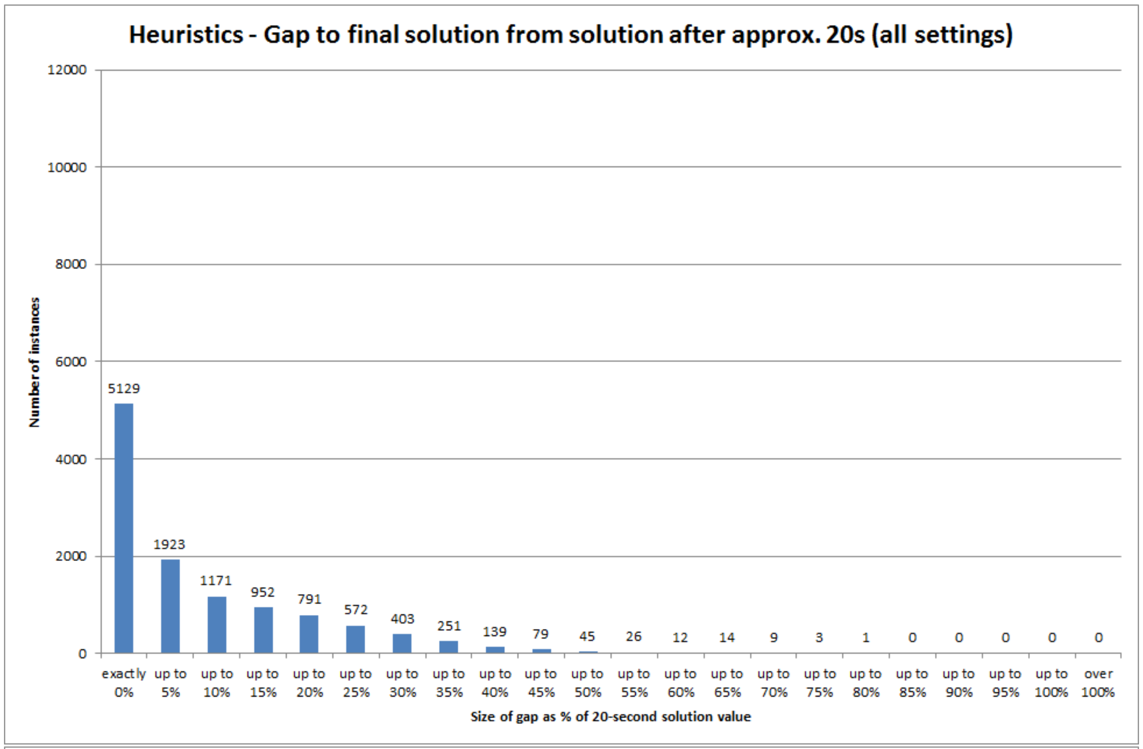


Figure 6.37: Gap to two-minute solution value from one-minute solution (top) and from estimated 30-second solution (bottom), across all test instances and all combinations of settings.



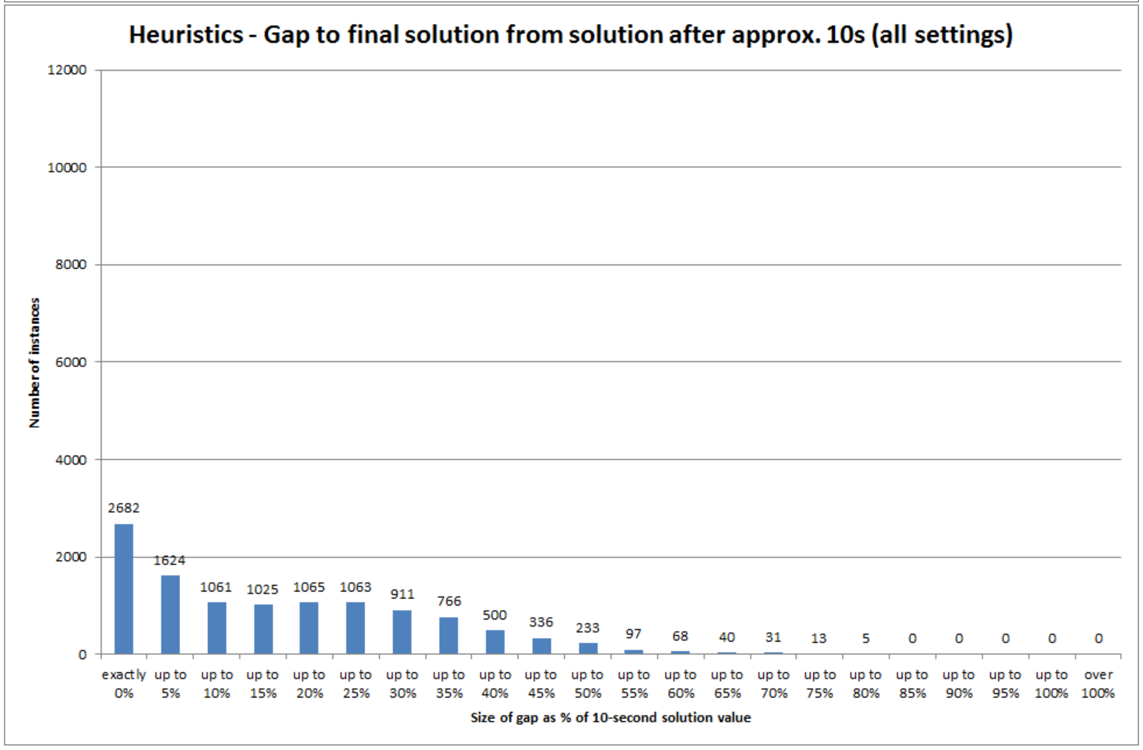


Figure 6.38: Gap to two-minute solution value from estimated 20-second solution (top) and estimated 10-second solution (bottom), across all test instances and all combinations of settings.

As can be seen, more than half of the test runs (6955 cases, ≈ 60.37% of the total) still have no difference in cost compared to the two-minute solution. There is again a tapering of the counts towards the higher percentage gaps, with 1863 cases (≈ 16.17% of the total) having a non-zero gap to the two-minute cost of within 5%, and a further 967 test runs (≈ 8.39% of the total) which have a gap of 10% or lower. The maximum is much larger for the 30-second cut-off than for the one-minute solutions, at ≈ 78.38%, and the mean is a little higher at ≈ 3.88%; the median is of course again zero.

Turning now to the upper graph of Figure 6.38, we see the estimated effect of terminating after 20 second is not all that dissimilar from the 30-second cut-off. Here, 5129 test runs (≈ 44.52% of the total) still have the same solution value as the final (i.e. two-minute) solution, with 1923 cases (≈ 16.69% of the total) having a non-zero gap of up to 5% and a further 1171 (≈ 10.16% of the total) having a gap of up to 10%. The median for the 20-second cut-off is now non-zero, although it is close to zero at ≈ 0.42%; the mean is roughly double that of the 30-second value at ≈ 7.30% while the maximum gap of ≈ 78.38% is the same as above.

Finally, we can look at the possible effect of cutting off the algorithm after 10 seconds, as shown in the lower graph of Figure 6.38. We see the shape of this is a little different from the preceding graphs, with only 2682 test runs (≈ 23.28% of the total) having by this stage found the best solution value they will achieve within two minutes. There are 1624 cases which have not yet reached their two-minute solution value but are within 5% of it, but the spread of gaps is much more even beyond this up to around a 30% gap with around 1000 cases (≈ 8.68% of the total) in each category. The mean and median are much higher for the 10-second cut-off as compared to the longer time limits, at ≈ 15.15% and ≈ 11.92% respectively; the maximum however is very similar, at ≈ 78.51%. We note by way of comparison that if we were to consider a cut-off of zero seconds, we would be looking at the gap between the initial solution and the two-minute solution - this is essentially the improvement achieved by the two-minute time limit, as shown in the lower graph of Figure 6.32. This is shown again, with the y-axis rescaled to match Figure 6.38, in Figure 6.39. This allows us to see that, while the 10-second cut-off produces solutions considerably inferior to even the 20-second cut-off, it still manages to make a considerable improvement as compared to the initial solution. As a reminder, the median improvement on the initial solution (i.e. gap between the initial and two-minute solution) was ≈ 35.90%, with 4558 cases (≈ 39.57% of the total) in the range of a gap greater than 30% and up to 45%. The maximum gap was ≈ 88.08%, compared to ≈ 78.51% for the 10-second cut-off.

As with the other results in this section, we can also investigate whether the different settings used for the heuristics (as described in section 6.5.4.14) have an influence on the gap between the early cut-off solutions and the final (i.e. two-minute) solutions. It is clear that the gaps shown in Figures 6.37 and 6.38 do not come from a Normal distribution,

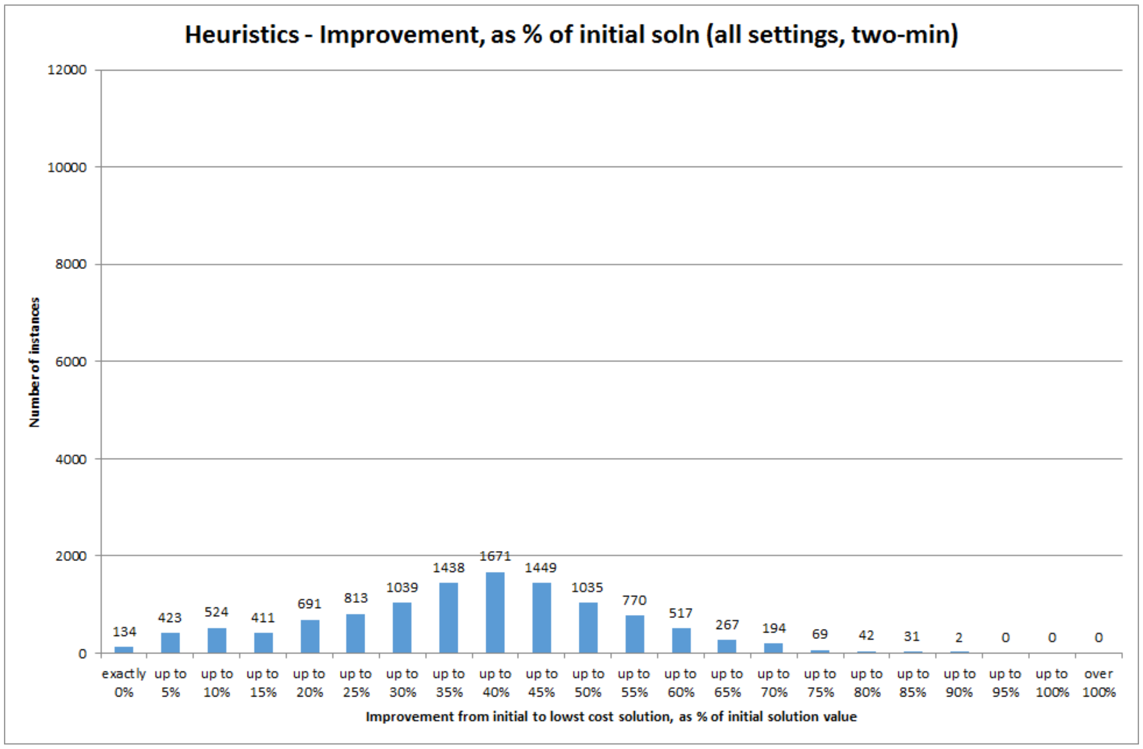


Figure 6.39: Improvement on the initial solution achieved within two minutes given as a percentage of the initial solution value, across all test instances and all combinations of settings (rescaled graph).

and therefore we should use the Kruskal-Wallis test. The p-values from these tests, carried out on the data for all four cut-off times, are shown in Table 6.9, with those which are significant at the 5% level marked with an asterisk. It can be seen that all five settings have a degree of significant influence on the gaps to the final solution. Three of the settings - the number of employees examined at each iteration, the acceptance criteria and the kick settings - have a significant effect on the size of gap for all four of the cut-off times. The order in which employees are examined is significant only for the one-minute cut-off, while the initial solution used has a significant effect on the one-minute, 30-second and 10-second cut-off gaps, but not on the gap with respect to the 20-second cut-off solution. This can be compared to the results in Table 6.4, showing the significance of the settings on the improvement (i.e. gap) between the initial solution and the two-minute solution. There, the F-test indicated that it was the number of employees examined, the kick settings and the initial solution which had a significant effect on the gap.

As before, we will now break down the results in order to investigate further the settings which are highlighted as significant. Summary statistics are given here, while more detail can be seen in the graphs in appendix section D.2.8. As we have four sets of results to

Table 6.9: Influence of heuristic settings on gap to final solution from early cut-off solution (Kruskal-Wallis test).

Setting type	p-value for one-min result	p-value for 30-sec cut-off	p-value for 20-sec cut-off	p-value for 10-sec cut-off
Employees to examine	< 0.001*	0.003*	< 0.001*	< 0.001*
Employee order	< 0.001*	0.102	0.767	0.159
Acceptance criteria	< 0.001*	< 0.001*	< 0.001*	< 0.001*
Kick settings	< 0.001*	< 0.001*	< 0.001*	< 0.001*
Initial solution	< 0.001*	< 0.001*	0.935	< 0.001*

give here (the four different cut-off times), we present our breakdown of results in tabular form. We begin with the number of employees to examine at each iteration, which had a significant influence at all four cut-off points. Table 6.10 shows the mean, median and maximum gap values for each setting at each time limit, as well as the number of cases for which the gap is zero and the proportion of cases in this subgroup this represents. It

Table 6.10: Breakdown of gaps to final solution from early cut-off solutions, according to number of employees to examine at each iteration.

Cut-off time	Statistic	Examine all employees	Examine one third
one minute	mean	$\approx 0.47\%$	$\approx 1.13\%$
	median	0%	0%
	maximum	$\approx 51.48\%$	$\approx 54.73\%$
	count zero gap	5339	4655
	% zero gap	$\approx 92.69\%$	$\approx 80.82\%$
30 seconds	mean	$\approx 3.91\%$	$\approx 3.85\%$
	median	0%	0%
	maximum	$\approx 78.38\%$	$\approx 73.50\%$
	count zero gap	3583	3372
	% zero gap	$\approx 62.20\%$	$\approx 58.54\%$
20 seconds	mean	$\approx 8.22\%$	$\approx 6.37\%$
	median	$\approx 0.83\%$	$\approx 0.20\%$
	maximum	$\approx 78.38\%$	$\approx 73.74\%$
	count zero gap	2489	2640
	% zero gap	$\approx 43.21\%$	$\approx 45.83\%$
10 seconds	mean	$\approx 17.08\%$	$\approx 13.21\%$
	median	$\approx 15.35\%$	$\approx 9.04\%$
	maximum	$\approx 78.38\%$	$\approx 78.51\%$
	count zero gap	1230	1452
	% zero gap	$\approx 21.35\%$	$\approx 25.21\%$

can be seen from this table that on average, the one-minute solutions have a smaller gap to the final solution, and more cases with no gap at all, when all employees are examined compared to when one third of employees are examined at an iteration. This distinction is less clear at the 30-second cut-off point, where examining all employees still gives more cases with zero gap to the final solution, but there is a slightly lower average gap and slightly lower maximum gap where the examine one third setting is used. For the 20-second cut-off, it is estimated that the examination of just one third of the employees at each iteration achieves solutions closer to the final solution, in terms of mean and median, as well as number of cases with a gap of zero. This is also true of the 10-second cut-off, with the difference being more pronounced - in particular, the median gap to the final solution is $\approx 15.35\%$ when all employees are examined, but is just $\approx 9.04\%$ when one third of employees are examined. These results can be linked to the findings for best solution time (section 6.8.4.7), where it was found that examining one third of employees would on average lead to the best solution being found later, but would also have a greater number of cases which found the best solution at a relatively early stage. This would tie in with the 'examine one third' setting having lower gaps to the final solution for earlier cut-offs, but being overtaken by the examine all setting by the one-minute cut-off point.

We next look at the order in which employees are examined at each iteration. This was only found to have a significant influence on the gap between the one-minute and two-minute solutions. Table 6.11 shows the mean, median and maximum gap values for the two settings, as well as the number and proportion of cases for which the gap is exactly zero. We see here that for the more tailored (or 'smarter') ordering, there is a slightly higher

Table 6.11: Breakdown of gaps to final solution from early cut-off solutions, according to order in which employees are examined at each iteration.

Cut-off time	Statistic	Random ordering	More tailored ordering
one minute	mean	$\approx 0.72\%$	$\approx 0.89\%$
	median	0%	0%
	maximum	$\approx 46.15\%$	$\approx 54.73\%$
	count zero gap	5092	4902
	% zero gap	$\approx 88.40\%$	$\approx 85.10\%$

mean gap between the one-minute and two-minute solutions, although it is in either case less than 1%, and also a larger maximum gap between the two solutions. It can also be seen that there are fewer cases for the smarter ordering which have a gap of zero compared to the number of randomly ordered cases. This implies that more of the cases which use smarter ordering would be yet to find their best solution if the algorithm was cut off after one minute. However, from the Kruskal-Wallis test results in Table 6.9 there would appear to be no significant difference between the groups in terms of gaps if the algorithm was to be terminated at any of the earlier cut-off times.

Looking at the breakdown of the acceptance criteria used, we recall that this was found to have a significant effect on the gaps for all four of the early cut-off solutions. The selected statistics for these groups for each cut off are shown in Table 6.12. We see that

Table 6.12: Breakdown of gaps to final solution from early cut-off solutions, according to solution acceptance criteria.

Cut-off time	Statistic	Improvement on current	Improvement on best
one minute	mean	$\approx 0.63\%$	$\approx 0.98\%$
	median	0%	0%
	maximum	$\approx 54.73\%$	$\approx 51.48\%$
	count zero gap	5115	4879
	% zero gap	$\approx 88.80\%$	$\approx 84.70\%$
30 seconds	mean	$\approx 2.93\%$	$\approx 4.82\%$
	median	0%	0%
	maximum	$\approx 65.61\%$	$\approx 78.38\%$
	count zero gap	3853	3102
	% zero gap	$\approx 66.89\%$	$\approx 53.85\%$
20 seconds	mean	$\approx 5.75\%$	$\approx 8.84\%$
	median	0%	$\approx 3.06\%$
	maximum	$\approx 67.56\%$	$\approx 78.38\%$
	count zero gap	2958	2171
	% zero gap	$\approx 51.35\%$	$\approx 37.69\%$
10 seconds	mean	$\approx 13.11\%$	$\approx 17.18\%$
	median	$\approx 8.61\%$	$\approx 15.22\%$
	maximum	$\approx 78.36\%$	$\approx 78.51\%$
	count zero gap	1620	1062
	% zero gap	$\approx 28.13\%$	$\approx 18.44\%$

the general pattern here is for the those test runs which look for an improvement on the current solution will have a smaller mean gap to the final solution than those which require an improvement on the best found so far. This pattern is also followed by the median gaps values for the 20-second and 10-second cut-off; the one-minute and 30-second cut-offs have a median gap of zero for both settings. In keeping with this, we also see that the number of cases with a gap of zero is greater for the group which uses the current solution as the acceptance criterion; the difference between the groups is far less for the one-minute cut-off than it is for the three earlier cut-off times. This is in keeping with the findings earlier which showed that the improve-on-current setting was related to the best solution value being found earlier (see section 6.8.4.7), most likely as a result of iterations being carried out more quickly.

We next look at the effect of the kick settings on the gaps to the best solution at the earlier cut-off times. Again, these were shown to be significant for all four cut-off times, and we give summary information about the effects on each of the cut-off solutions in Table 6.13. It is clear from this table that the mean gaps to the final solution are much larger

Table 6.13: Breakdown of gaps to final solution from early cut-off solutions, according to kick settings used.

Cut-off time	Statistic	No kick used	No improvement after 4+ iters	No improvement after 8+ iters
one minute	mean	≈ 1.43%	≈ 0.35%	≈ 0.62%
	median	0.00%	0.00%	0.00%
	maximum	≈ 54.73%	≈ 40.17%	≈ 51.48%
	count zero gap	2854	3646	3494
	% zero gap	≈ 74.32%	≈ 94.95%	≈ 90.99%
30 seconds	mean	≈ 5.18%	≈ 2.89%	≈ 3.57%
	median	≈ 0.08%	0.00%	0.00%
	maximum	≈ 73.50%	≈ 78.38%	≈ 57.06%
	count zero gap	1770	2798	2387
	% zero gap	≈ 46.09%	≈ 72.86%	≈ 62.16%
20 seconds	mean	≈ 8.94%	≈ 5.86%	≈ 7.08%
	median	≈ 3.79%	0.00%	≈ 3.13%
	maximum	≈ 73.74%	≈ 78.38%	≈ 70.51%
	count zero gap	1194	2177	1758
	% zero gap	≈ 31.09%	≈ 56.69%	≈ 45.78%
10 seconds	mean	≈ 16.88%	≈ 13.44%	≈ 15.11%
	median	≈ 14.97%	≈ 8.60%	≈ 11.95%
	maximum	≈ 78.51%	≈ 78.38%	≈ 75.10%
	count zero gap	605	1151	926
	% zero gap	≈ 15.76%	≈ 29.97%	≈ 24.11%

at the cut-off points when no kick is carried out, while the 4-or-more-iteration kick setting produces smaller gaps on average than the 8-or-more-iteration setting. This same pattern can be seen in the median values for 30-second, 20-second and 10-second cut-offs, and also when looking at the number of cases in each category which have zero gap. In this case, we must bear in mind the results in sections 6.8.4.1 and 6.8.4.2, which showed that using no kick lead to a better solution in terms of cost, and the results in section 6.8.4.7 which showed that the best solution was found much later when no kick was carried out. It can be seen therefore that while the no kick setting has a larger gap to the best solution, this will be largely because it is being measured against a lower cost solution, and so an early cut-off to the algorithm when no kick is used may not be too much of a disadvantage.

Finally we come to the effect of the initial solution which is used. This was significant for three of the four cut-off times: one-minute, 30-second and 10-second, but not for the 20-second cut-off solution. The selected statistics, broken down by initial solution type, for these three cut-off times are shown in Table 6.14. We see from this table that for the one-minute and 30-second cut-off points, the group which uses the Heuristic initial solution has a smaller mean and smaller maximum gap to the final solution, and also has a greater

Table 6.14: Breakdown of gaps to final solution from early cut-off solutions, according to initial solution.

Cut-off time	Statistic	Task-Based Approx.	Heuristic Initial
one minute	mean	≈ 0.98%	≈ 0.62%
	median	0%	0%
	maximum	≈ 54.73%	≈ 43.94%
	count zero gap	4863	5131
	% zero gap	≈ 84.43%	≈ 89.08%
30 seconds	mean	≈ 4.15%	≈ 3.61%
	median	0%	0%
	maximum	≈ 78.38%	≈ 61.92%
	count zero gap	3363	3592
	% zero gap	≈ 58.39%	≈ 62.36%
10 seconds	mean	≈ 13.88%	≈ 16.41%
	median	≈ 9.59%	≈ 14.41%
	maximum	≈ 78.51%	≈ 75.10%
	count zero gap	1398	1284
	% zero gap	≈ 24.27%	≈ 22.29%

percentage of cases which have zero gap to the final solution. This is different however from the 10-second cut off, for which it is the group using the Task-Based Approximation as an initial solution which has the smaller mean and smaller median gap to the final solution, and a slightly greater percentage of cases with a gap of zero. This gives a degree of explanation to the results in Table 6.9 with respect to the 20-second cut-off gaps - as the Heuristic initial solution brings smaller gaps for the one-minute and 30-second cut-offs, but the Task-Based Approximation gives smaller gaps for the 10-second cut-off, we can surmise that the 20-second cut-off falls in the region where the 'cross-over' takes place, and where there is no significant difference between the groups. This ties in with the results for the best solution time discussed in section 6.8.4.2, where it was shown that using the Task-Based Approximation as an initial solution tended to see solutions found later, but also had more cases where the best solution was found within approximately the first 5 seconds, hence the smaller gaps to the final solution for the 10-second cut-off point.

6.8.4.9 Summary of results for Heuristics

Having discussed the performance of the Heuristic algorithm with respect to a number of measures in the preceding sections (6.8.4.1 to 6.8.4.8), and shown the influence of the setting choices on these results, we now summarise these findings.

Firstly, in terms of general measures of the performance of the algorithm, we have seen that it generates solutions with a gap to the best known bound of on average ≈ 71.02% within one minute and ≈ 70.84% in two minutes, while the mean gap to the best known solution across all test runs was $\approx 46.45\%$ in one minute and $\approx 46.04\%$ in two minutes. The cost gaps achieved by the heuristic algorithm were seen to be better than those achieved by the two-minute settings of the direct approach, and far better than the non-cost-constrained solutions found by the change-minimization method; however the comparison with the one-hour direct approach was more even, with it appearing that the heuristics were more consistent in performance although not as likely to find really good solutions. Meanwhile, the final change-minimization results were in general better than the combined heuristic results. The costs were seen to be an average of $\approx 34.18\%$ improvement within one minute and $\approx 34.69\%$ improvement within two minutes on the initial solution, and it was noted that there was very little additional improvement evident in the second minute of most test runs. A little over 1% of the test runs achieved no improvement at all.

Other measures examined included the number of changes, which were shown to range from fewer than 20 up to more than 300 and averaged ≈ 109.22 for the one-minute and ≈ 111.79 for the two-minute results, and the number of solutions found with cost equal to the best, which ranged from single unique solutions to 59 distinct equal solutions and averaged ≈ 6.74 for the one-minute and ≈ 7.96 for the two-minute results. The number of iterations, which can be serve as an indication of the number of solutions which it may be possible to find, took a wide range of values - modal ranges were 51 to 75 iterations in one minute and 101 to 125 iterations in two minutes, while the respective means were ≈ 114.05 and ≈ 204.10 and maximum values were as high as 341 in one minute and 661 iterations in two minutes. Comparing the numbers of iterations within one and two minutes seemed to suggest an increase in iteration length in the second minute of the test runs.

The time of the best solution was also investigated, and it was found that nearly half of the test runs found their best solution within 20 seconds, even when the programme ran for two minutes. Indeed, for the two-minute time limit less than 15% of the test runs found the best solution after the 60-second mark. These findings lead to calculations of expected solution values if the programme was terminated early. This found that the mean gap which the algorithm would have from the final solution was $\approx 0.80\%$ if terminated after one minute, and $\approx 3.88\%$ if terminated after 30 seconds; the median gaps for these cut-off were however zero given the number of cases which found their best solutions earlier than this. These figures were similar for a 20-second cut-off, with a mean gap of $\approx 7.30\%$ and median of just $\approx 0.42\%$; but the estimated gaps for a 10-second cut-off were much higher, with a mean of $\approx 15.15\%$ and median gap of $\approx 11.92\%$. This gap however was shown to still be a lot closer to the final (i.e. two-minute) solution than the initial solutions on average were.

Having looked at these measures in general terms, some analysis was carried out into the effects of the settings used in the heuristic algorithm, which we now summarise. We begin with the number of employees examined at each iteration, which was shown to have a degree of influence on all of the measures discussed. In terms of solution cost, it was found that the test runs which examined only one third of employees at each iteration achieved a better improvement on the initial solution, and found solutions with a smaller gap to the best known bound and the best known solution. This group of test runs also carried out more iterations in the time available, but on average had a greater number of changes in the best solution and a later mean time for finding that solution. The median time of finding the best solution was lower however than when all employees were examined, suggesting there were more best solutions found early when only one third of employees were examined. In terms of earlier cut-offs, the cases where all employees were examined were closer to the final solution value after one minute; however, this group on average gave larger gaps from the final solution for the 30-second, 20-second and 10-second cut-offs. On balance, it would appear more beneficial to use the setting of examining only one third of employees at each iteration rather than examining all employees.

Looking next at the order in which employees were examined, we found that this choice had no significant impact on the solution cost measures of gap to best known bound, gap to best known solution, or the improvement on the initial solution. This setting was also found to have no influence on the number of changes in the one-minute solution; however, for the two-minute solution it was found that the tailored (or 'smarter') ordering actually lead on average to a greater number of changes in the first lowest-cost solution found. The random ordering was also superior in terms of numbers of equal-best solutions found on average, but the tailored ordering achieved a greater average number of iterations, and in terms of the results found within one minute these were more likely to be found earlier than under the random ordering. In terms of the earlier cut-offs, the employee order had no significant influence on the 30-second, 20-second or 10-second gaps to the final solution; for the one-minute results, the smarter ordering demonstrates a slightly higher mean gap and fewer cases which have a gap of exactly zero. Overall, it would appear that the tailored ordering was not particularly successful, and that the random ordering would be the better option to choose for this setting.

Next we consider the influence of the solution acceptance criteria during the test runs. This setting had less impact on the results, with no significant influence on the gaps to best bound or best solution, the improvement on the initial solution, the number of changes or the number of equal-best solutions. It does however have a clear influence on the number of iterations carried out, with the test runs which will accept the first non-tabu solution which improves on the current solution performing roughly a third more iterations on average. This appeared to have an impact on the time that the best solution would be found, with the quicker iterations allowing the group looking for an improvement on the current solution to find their best solution earlier. This also had the effect of meaning the improve-on-current test runs have on average a smaller difference between the earlier cut-off solutions and the final solution. Overall there appears to be no reason why we would choose to improve on the best solution found so far, so we must conclude that the improvement on the current solution should be chosen here.

Summarising the findings relating to the settings used for the random kick procedure, this was found to have a significant influence on every measure investigated, although not in the way originally intended. In terms of the gaps to the best bound and best solution, and the improvement on the initial solution, it was found that those test runs which implemented no kick achieved the best results; the 8-or-more-iteration setting was on average superior to the 4-or-more-iteration setting on these measures. It was also found that having no kick meant the average test run had a solution with a lower number of changes, had a much higher number of equal-best solutions, and had a greater number of iterations carried out. It was hypothesised that the effect of the kick was, in general, to push the search away from the best solution so far and often not allowing the algorithm chance to sufficiently improve the solution. This was seemingly confirmed by the observation that the test runs with no kick found their best solution on average much later than with the other settings. (This also had the effect of creating a larger gap between the early cut-off and final solutions when no kick was used; although it was noted that the final solutions for this group were themselves of lower cost.) It was therefore recommended as future research that the kick implementation rules could be amended to allow a longer gap before a kick was carried out, which would take into account the number of iterations which are possible within two minutes (or even one minute) in the C++ implementation of the algorithm. However, with the algorithm as it currently stands it is clear that it is better not to implement a kick at all.

Finally, in terms the initial solution, this had a substantial influence on a number of measures. Not least, it was found that starting from the Heuristic initial solution allowed the algorithm to find solutions with on average much smaller gaps to the best known bound and the best known solution. While this was in part attributable to the Heuristic initial solutions having a lower cost to begin with, it was also found that these initial solutions actually facilitated a better average improvement over the running time of the programme than was achieved using the Task-Based Approximation solution. In terms of numbers of changes, the test runs which used the Heuristic initial solution had far fewer on average, although this was seen to be a small increase on the number of changes in the initial solution; by contrast, the heuristic algorithm was able to reduce the number of changes on average when starting from the Task-Based Approximation solution. The choice of initial solution had no significant influence on the number of iterations carried out, and had a small (but not insignificant) influence on the number of equal-best solutions found, with those cases starting from the Task-Based Approximation solution finding on average more equal-best solutions. Finally, in terms of times of finding the solutions, it was seen that the Task-Based Approximation initial solution lead to more cases where the best solution was found in the first 5 seconds, but on average saw the best solutions found later in the test runs. As a result, the test runs using the Heuristic initial solution had a larger gap to the final solution at the 10-second cut-off point but, while the 20-second cut-off had no significant different between the groups, the 30-second and one-minute cut-offs recorded smaller average gaps to the final solution when the Heuristic initial solution was used. It would therefore seem clear that the Heuristic initial solution is the better option for selection here.

Drawing together the recommendations here for the settings which should be used, we have the following selection:

1. Examine one third of all employees at each iteration;
2. Examine employees in a random order;
3. Accept the first non-tabu solution which improves on the current solution value;
4. Do not use the random kick; and
5. Use the Heuristic initial solution.

From Table D.2 in the appendix, this combination of settings has been labelled as Combination #26. We can also say that in terms of time limit, it is unlikely that running the algorithm for a full two minutes would be beneficial - terminating the algorithm after one minute is likely to give close to the same results. It may even be possible that the 30 seconds is an adequate time limit; however, this is based on estimated results and should be investigated further before a final decision is made. Histograms for Combination #26 only relating to the measures discussed above are given in section D.2.9 of the appendix; here, we can give an outline of the key points of these results.

Firstly, in terms of the gap to the best known bound (Figure D.53), this combination gives a mean gap of $\approx 67.22\%$ for the one-minute and $\approx 66.81\%$ for the two-minute results, with medians slightly higher at $\approx 70.52\%$ and $\approx 70.09\%$ respectively. There are 2 instances ($\approx 0.83\%$ of the total) for which both the one-minute and two-minute results are within 5% of the best known bound; no instances were solved to known optimality. Comparing the results for Combination #26 with the best known solution (Figure D.54), we have that the mean gap is $\approx 39.67\%$ for the one-minute and $\approx 38.60\%$ for the two-minute results, around seven percentage points less than the overall average for all heuristic settings; the medians are $\approx 37.14\%$ and $\approx 36.25\%$ respectively. It should also be noted that the shape of the graphs are a little different to those shown in Figure 6.31, with a clearer bias towards gaps at the lower (i.e. left hand) end of the x-axis. In terms of improvement (Figure D.55), Combination #26 gives a mean of $\approx 40.36\%$ within one minute and $\approx 41.29\%$ within two minutes, around seven percentage points better than the average across all heuristic settings. There are still 2 instances ($\approx 0.83\%$ of the total) for which no improvement is made, although this is a smaller percentage than across all heuristic settings ($\approx 1.16\%$ of the total for two-minute results).

Looking at other quality measures, we see that the number of changes in the first lowest-cost solution (Figure D.56) has a different shape to the data than for all settings combined, with the greatest proportion of the cases falling into the 21-40 changes category. The mean number of changes was ≈ 76.18 for the cheapest one-minute solutions, and ≈ 77.08 for the two-minute solutions; just over two-thirds of the combined average for all settings. The maxima were also around two-thirds of the overall values, at 230 and 232 for the one-minute and two-minute results respectively. We also see that this combination offers a better selection of results than the average for all heuristics (Figure D.57), with a mean ≈ 17.05 and median 9 equal-best solutions for the one-minute results, and a mean ≈ 21.73 and median 12 equal-best solutions in two minutes. More solutions will also be generated because of the higher number of iterations (Figure D.58), with a mean of ≈ 145.22 being carried out in one minute, and ≈ 279.89 carried out within two minutes. The number of iterations can also be seen to be more closely grouped and more normally distributed round the central mean than for the overall results given in Figure 6.35.

Finally, turning attention to the time the best solution is found (Figure D.59), we have that over half of the best solutions found within one minute are in fact found within 30 seconds - the median is ≈ 29.96 seconds, with the median for the two-minute solutions being ≈ 36.03 seconds; the mean times are a little higher, at ≈ 30.43 seconds and ≈ 45.23 seconds respectively. There are however some instances for which the best solution is found very close to the time limit of the test run (11 instances ($\approx 4.58\%$ of the total) find the best one-minute solution in over 60 seconds; 1 instance has the best two-minute solution found in over 120 seconds). All this means that an early cut-off to the algorithm (Figure D.60) could potentially result in quite small gaps from the final solution. Terminating after one-minute would leave the solution a mean of $\approx 1.54\%$ from the two-minute solution, with 173 instances ($\approx 72.08\%$ of the total) here having the same solution. Similarly, a cut-off after 30-seconds is estimated to give a mean gap of $\approx 5.01\%$ to the two-minute solution, with 94 instances ($\approx 39.17\%$ of the total) having a gap of zero. For the 20-second and 10-second cut-offs, the mean estimated gaps are $\approx 7.22\%$ and $\approx 14.66\%$ respectively. From this, we can suggest that cutting off the algorithm after one minute or even 30 seconds would not be to the serious detriment of the quality of the solution; terminating after 20 seconds however would seem less beneficial, with additional cost reductions being lost for the gain of only a very short amount of time.

6.8.5 Effect of Parameter Values on Results

Having given various computational results in sections 6.3 and 6.8 above, we must look at the potential effects of the parameters which had to be varied when the data was generated as described in section 6.2.2. This investigation takes broadly the same form as in section 5.5.2.3 for the Task-Based problem, where an Analysis of Variance is carried out on measures for which the required Normality assumption holds, and the Kruskal-Wallis test otherwise. As a reminder, there are seven parameter values which were varied and which may have influenced the outputs of interest. These are:

- Availability probabilities p and q;
- Whether or not the time reduction $r(d)$ is used;
- The near- and long-term disruption factors, K_N and K_L respectively;
- An aggregated disruption factor $\hat{K}$, calculated as a weighted average of the near- and long-term factors such that

$$\hat{K} = \frac{(4 \times K_N) + (9 \times K_L)}{13}$$

  and
- The agency penalty factor K_{AG}.

As before, we note that the values of p and q are directly related as described by equation (5.57), and therefore results of tests for the influence of the values of q are identical to those for the parameter p. As a result, in this section we do not explicitly report results relating to the parameter q.

In this section we discuss the effect of the parameter values firstly on the results of the initial computational approaches presented in section 6.3 and the further analysis of these given in section 6.8.1. We then focus on the further computational results, discussing in turn the Task-Based Approximation results given in section 6.8.2, the Heuristic initial solution approach discussed in section 6.8.3, and the results for the main Heuristic algorithm as set on in section 6.8.4.

6.8.5.1 Effect of parameter values on Initial Computational results

We firstly look at the effect of the parameter values on the initial computational results, and the associated further analysis of the gaps to the best known bound. As a reminder, the results given in sections 6.3 and 6.8.1 related both to the ‘direct’ solution approach, run over a two-minute time limit and a one-hour time limit, and a change-minimization algorithm which used a two-minute time limit only. The results reported, and their corresponding histogram, were:

- For the direct solution approach:
  - Gap to best known bound, as percentage of direct solution value - see Figure 6.1 for the two-minute time limit, and Figure 6.2 for the one-hour time limit;
  - Gap to best known solution, as percentage of direct solution value - see Figure 6.13 for both time limits.
- For the change-minimization approach:
  - Number of solutions found - see Figure 6.3;
  - Number of iterations carried out - see Figure 6.4;
  - Gap to best known bound for lowest cost solution, as percentage of lowest cost solution - see Figure 6.5;
  - Gap to best known bound for non-cost-constrained solution, as percentage of non-cost-constrained solution - see Figure 6.6;
  - Gap to best known solution for lowest cost solution, as percentage of lowest cost solution - see Figure 6.14;
  - Gap to best known solution for non-cost-constrained solution, as percentage of non-cost-constrained solution - see Figure 6.15.

From examination of the histograms relating to these outputs, it could be seen that none of them were suitable for the F-test to be carried out; instead, the Kruskal-Wallis test was used to test for influence of these parameters. Note that because the assumption of an approximately normal distribution did not hold, it would also not have been valid to carry out any analysis of correlations between the parameter values and these outputs. The results of the Kruskal-Wallis tests are presented in Table 6.15 in the form of p-values for each output for each parameter. Those which are significant at the 5% level, and which thereby provide evidence that the parameter value influences the distribution of the given output, are marked with an asterisk.

Examining this table, we see that outputs for the 'direct' solution approach are affected much more by the parameter values than the change-minimization approach. Particularly, the disruption factors (K_N, K_L and $\hat{K}$) and the agency penalty factor (K_{AG}) - i.e. the cost-related parameters - seem to have a much larger influence on the direct approach results than for the change-minimization; the main influences on change minimization are the availability parameter p and the agency penalty factor K_{AG}. The Kruskal-Wallis test can only tell us that a parameter has an influence; it cannot give information as to the

Table 6.15: Results of Kruskal-Wallis tests for influence of parameters used in instance generation on Initial computational results.

				Parameter					
Outputs of interest			Test	p	Use $r\,(d)$?	K_N	K_L	$\hat{K}$	K_{AG}
Direct solution approach	Gap to best known bound	two-min	K-W	0.306	0.994	0.011*	$<$ 0.001*	$<$ 0.001*	$<$ 0.001*
		one-hour	K-W	$<$ 0.001*	0.127	0.001*	$<$ 0.001*	$<$ 0.001*	$<$ 0.001*
	Gap to best known solution	two-min	K-W	$<$ 0.001*	0.099	0.004*	$<$ 0.001*	$<$ 0.001*	$<$ 0.001*
		one-hour	K-W	0.128	0.563	$<$ 0.001*	$<$ 0.001*	$<$ 0.001*	0.072
Change-min Approach	No. of solutions		K-W	$<$ 0.001*	0.017*	0.897	0.584	0.958	0.887
	No. of iterations		K-W	$<$ 0.001*	0.786	0.263	0.041*	0.301	0.119
	Gap to best known bound	lowest cost sol	K-W	$<$ 0.001*	0.244	0.704	0.375	0.776	$<$ 0.001*
		non-cost-constr.	K-W	0.987	0.938	0.175	0.031*	0.304	$<$ 0.001*
	Gap to best known solution	lowest cost sol	K-W	0.470	0.058	0.787	0.109	0.331	0.236
		non-cost-constr.	K-W	$<$ 0.001*	0.005*	0.870	0.619	0.929	$<$ 0.001*

nature of this influence. To see this, we must look in greater detail at a breakdown of each output by the relevant significant parameters. Only a summary of these details are given here, with breakdown histograms given in the appendix in section D.3.1.

Looking at these histograms it can be seen that in general the higher values of the disruption factors (K_N, K_L and $\hat{K}$) have a higher incidence of achieving lower gaps to the best bound or best solution in the direct runs. For the agency penalty (K_{AG}) on the other hand, the higher penalty values seem to lead to greater variability - more high valued gaps as well as a greater count of low-value gaps. The higher values of the probability p (which indicate fewer but longer absences for the crew) give rise to much smaller gaps to the best bound for the one-hour settings; however, when compared to the best known solution, the higher values of p appear to give a greater number of very large gaps as well as a higher count of lower-valued gaps.

The graphs for the change-minimization results meanwhile show that more iterations and more solutions can be achieved in cases where a higher value of p was used. More iterations could also be carried out when high long-term disruption penalties (K_L) were applied, and more solutions were found in the cases where the time reduction $r(d)$ was *not* used. In terms of the cost gaps, the lowest cost solution had a smaller gap to the best known bound when there were higher values of absence probability p and when there were lower values of the Agency penalty factor K_{AG}. The non-cost-constrained solution was affected for both the gap to best known bound and the gap to best known solution, with these both being lower when a smaller agency penalty factor was applied. The gap to best bound also was reduced by having larger penalties for long-term disruption, while the gap to the best known solution was smaller in cases where there was a lower value of probability p and where the time reduction factor $r(d)$ was *not* used.

6.8.5.2 Effect of parameter values on Task-Based Approximation results

We next can look at the effect the data generating parameters have on the Task-Based Approximation results discussed in section 6.8.2. As a reminder, this approach produced one single solution for each instance, which could be analysed in a number of ways. The results which were given, listed here with reference to the corresponding histograms, were as follows:

- Gap to best known bound, as percentage of Task-Based Approximation solution - see Figure 6.16;
- Gap to best known solution, as percentage of Task-Based Approximation solution - see Figure 6.17;
- Improvement on two-minute and one-hour direct solution approach, as percentage of

respective direct solution values - see Figure 6.18;

- Improvement on change-minimization best solution, as percentage of change-minimization best solution - see Figure 6.19;
- Running time - see Figure 6.20;
- Number of changes, according to both Task-Based and Time-Windows versions of the solution - see Figure 6.21.

Examining the histograms relating to these outputs it could be seen that the number of changes, both in the Task-Based and Time-Windows representations of the solution, followed an approximately Normal distribution. Therefore it was possible to carry out the Analysis of Variance to test for the effect of the data generation parameters using the F-test for these two outputs; for the other outputs, the Kruskal-Wallis test was used. The p-values from these tests, with those which are significant at the 5% level marked with an asterisk, are shown in Table 6.16. Since the two measures of numbers of changes satisfied the Normality assumption, it was also possible to examine correlations relating to these. Table 6.17 shows the Pearson correlation coefficients (PCC) and their associated p-values for five of the numerical parameters - note that correlation cannot be carried out on the boolean reduction factor $r(d)$, while correlation with probability q is equal but opposite to that for probability p and therefore not given here. As before, all p-values which are significant at the 5% level are marked with an asterisk.

From Table 6.16, we see that the availability parameter p affects all aspects of the Task-Based Approximation results with the exception of the running time, while the use of the time reduction factor $r(d)$ is completely irrelevant. The cost-related parameters have some influence on the gaps and improvements, but not on the number of changes. For the number of changes, which follows an approximately normal distribution, we have the additional correlation information in 6.17. This tells us that the significant influence of probability p on the number of changes has a negative correlation - i.e. for higher values of p, which correspond to longer but fewer employee absences, we would expect to see a lower number of changes in the Task-Based Approximation solution. For those outputs which were tested using Kruskal-Wallis, these tables cannot provide any information about the nature of the significant influences. Instead, we must look in greater detail at a breakdown of each output by the relevant significant parameters. Histograms showing this are given in appendix section D.3.2, while here we will simply give a summary of the findings.

Taking first the cost gaps, we have the instances which implement greater penalties for using agency crew, i.e. larger K_{AG} values, tend towards larger gaps to both the best bound and the best solution. Higher values of the parameter p lead to smaller gaps to the best known bound, but larger gaps to the best known solution, suggesting that perhaps

Table 6.16: Results of F-tests and Kruskal-Wallis tests for influence of parameters used in instance generation on Task-Based Approximation results.

Outputs of interest		Test	Parameter					
			p	Use $r\,(d)$?	K_N	K_L	$\hat{K}$	K_{AG}
Gap to	best known bound	K-W	< 0.001*	0.755	0.576	0.409	0.832	< 0.001*
	best known solution	K-W	0.001*	0.066	0.610	0.680	0.568	< 0.001*
Improvement	on two-min direct	K-W	< 0.001*	0.786	0.009*	< 0.001*	< 0.001*	0.669
	on one-hour direct	K-W	0.008*	0.133	0.064	< 0.001*	< 0.001*	< 0.001*
	on change-minimize	K-W	0.002*	0.625	0.471	0.937	0.492	< 0.001*
Running time		K-W	0.538	0.977	0.032*	0.030*	0.067	0.799
Number of changes	Task-Based repres.	F	< 0.001*	0.931	0.510	0.143	0.373	0.518
	Time-Window repres.	F	< 0.001*	0.196	0.764	0.307	0.653	0.341

Table 6.17: Correlation analysis for influence of parameters used in instance generation on Task-Based Approximation results.

Outputs of interest			Parameter				
			p	K_N	K_L	$\hat{K}$	K_{AG}
Number of changes	Task-Based repres.	PCC	-0.495	-0.066	-0.100	-0.100	0.061
		p-value	< 0.001*	0.305	0.123	0.121	0.345
	Time-Window repres.	PCC	-0.644	-0.062	-0.092	-0.093	0.093
		p-value	< 0.001*	0.337	0.155	0.150	0.150

for larger probability p we will tend to have better 'best known' solutions across all the approaches. In terms of improvement on the direct and change-minimization methods, we have that higher values of the parameter p give a larger improvement on the two-minute direct approach, but are more likely to see highly negative improvements compared to the one-hour direct or change-minimization methods. The disruption factors K_N and K_L do not have a consistent influence on the improvement on the two-minute direct solution, with lower values tending to give a smaller improvement, but the largest values ($K = 10$) giving poorer results. This is also reflected in the fact that instances with the highest values of $\hat{K}$ are most likely to have a negative improvement on the two-minute direct solution.

The effects are more consistent on improvements on the other initial approaches, with higher values of both the long- and near-term disruption factors more likely to see negative improvement on the one-hour direct solution, and higher values of the agency penalty factor K_{AG} more likely to result in negative improvement on both the one-hour direct and the change-minimization solution costs. Disruption factors K_N and K_L also impact the running time, with higher values of the factors more likely to result in a quicker running time.

6.8.5.3 Effect of parameter values on Heuristic initial solution results

We now look into the effect the data generating parameters have on the results for our other initial solution method - the Heuristic initial solution approach, for which the results are given in section 6.8.3. As with the Task-Based Approximation approach, this method produces a single solution for each instance, although multiple runs were carried out to give some indication of variability of the approach. Here we will take the results of the run which generated the initial solutions used in the main heuristic algorithm (the single run also referred to in the results in section 6.8.3) as being representative of the multiple runs which were carried out, and for simplicity only provide analysis of the parameter values with reference to this single run. As a reminder, the results which were presented previously, and the corresponding histograms for them, were as follows:

- Gap to best known bound, as percentage of Heuristic initial solution value - see Figure 6.22;
- Gap to best known solution, as percentage of Heuristic initial solution value - see Figure 6.23;
- Improvement on two-minute and one-hour direct solution approach, as percentage of respective direct solution values - see Figure 6.24;
- Improvement on change-minimization best solution, as percentage of change-minimization best solution - see Figure 6.25;
- Running time - see Figure 6.26;
- Number of iterations carried out - see Figure 6.27;
- Number of changes in the solution - see Figure 6.28;
- Improvement on Task-Based Approximation solution, as percentage of the Task-Based Approximation solution value - see Figure 6.29.

As with the initial computational results, visual inspection of the histograms for these outputs indicated that the assumption of a Normal distribution did not hold. It was therefore not possible to preform the Analysis of Variance, or to carry out any correlation analysis on these outputs. Instead, the Kruskal-Wallis test was used to test for any influence of the data generating parameters on the results. The p-values emerging from these tests are given in Table 6.18, with those which are significant at the 5% level marked with an asterisk.

From the table, we see that of the cost-related parameters only the agency penalty factor K_{AG} has an influence on the cost gaps of the solutions found, with the gap to the best bound also influenced by the value of absence probability p and whether or not these probabilities are multiplied by the time reduction factor. The improvements on the direct solution results (for the two-minute and one-hour settings) are influenced by the three disruption factors K_N, K_L and $\hat{K}$, which ties in with the results in Table 6.15 above which showed these factors to have an influence on the direct solution cost. The improvement on the two-minute direct solution is also influenced by the value of parameter p, while the one-hour direct comparison value has a significant link with the agency penalty factor. The agency penalty factor also has an effect on the improvement on the change-minimization solution, the only factor to do this. In terms of other measures, we have that the absence probability p has an influence on the running time and the numbers of iterations carried out and of changes in the solution; the use of the reduction factor $r(d)$ also influences running time and number of changes, but has no significant link with number of iterations.

Table 6.18: Results of Kruskal-Wallis tests for influence of parameters used in instance generation on Heuristic initial solution results.

Outputs of interest		Test	Parameter					
			p	Use $r\ (d)$?	K_N	K_L	$\hat{K}$	K_{AG}
Gap to best known...	bound	K-W	< 0.001*	0.038*	0.625	0.634	0.903	< 0.001*
	solution	K-W	0.179	0.588	0.740	0.654	0.864	< 0.001*
Improvement on...	two-min direct	K-W	< 0.001*	0.051	0.004*	< 0.001*	< 0.001*	0.106
	one-hour direct	K-W	0.339	0.858	0.001*	< 0.001*	< 0.001*	0.039*
	change-minimize	K-W	0.174	0.091	0.446	0.783	0.909	< 0.001*
Running time		K-W	< 0.001*	0.003*	0.813	0.779	0.835	0.101
Number of iterations		K-W	< 0.001*	0.090	0.897	0.316	0.752	0.746
Number of changes		K-W	< 0.001*	0.003*	0.915	0.794	0.991	0.824
Improvement on Task-Based Approx.		K-W	0.062	0.015*	0.209	0.473	0.203	0.002*

The use of the reduction factor also has a significant effect on the improvement compared to the Task-Based Approximation solution, as does the value of the agency penalty factor.

As above, the Kruskal-Wallis test results only highlight which parameters have a significant influence on the outputs, and can give no further information about the nature of this influence. For more insight, histograms have been created for each significant parameter for each output - a summary of the findings is given here, with the histograms themselves available in appendix section D.3.3. Firstly, with respect to the cost gaps, we have that a smaller gap to the best known bound is observed when there are higher values of parameter p (corresponding to longer absences less often) and when the time reduction factor $r(d)$ is applied. Higher values of the agency penalty factor increases the size of the gaps, both with respect to the best bound and the best known solution.

Comparing the Heuristic initial solution results to the direct approach, we have that higher values of p tend to give a better improvement compared to the two-minute results, although those instances which show negative improvement are more likely to be highly negative. Higher values of the disruption penalty factors mean that the improvement is more likely to be negative (i.e. the direct result is better than the Heuristic initial solution) both with respect to the two-minute and one-hour direct approaches. Similarly, lower values of the agency penalty K_{AG} are more likely to result in a positive improvement on the one-hour direct solution, while high values of K_{AG} are more likely to give a highly negative (i.e. $< -100\%$) 'improvement' on the Change-minimization solution.

For the other measures, it can be seen that the algorithm will likely run faster and require fewer iterations to solve when the value of p is higher. Instances where the time reduction factor is applied also run faster in general, and we have that the final solution will entail fewer changes when p is larger and when the reduction factor $r(d)$ is used. Finally, comparing the results to the Task-Based Approximation, we have that instances where the reduction factor is used or which have a higher value for the agency penalty K_{AG} are more likely to show a positive improvement on the Task-Based Approximation solution.

6.8.5.4 Effect of parameter values on Heuristic algorithm results

Finally, we can examine the effect of the parameter values on the results for the main heuristic algorithm, as discussed in section 6.8.4. In particular, we will examine the parameter value influences on the results for Combination #26, which was identified in section 6.8.4.9 as the setting combination which gave the best results across all the heuristic settings. As a reminder, the results which were reported, listed along with their corresponding histograms (note that these can be found in appendix section D.2.9), were as follows:

- Gap to best known bound (for both the one-minute and two-minute solutions), as percentage of heuristic solution value - see Figure D.53;

- Gap to best known solution (for both the one-minute and two-minute solutions), as percentage of heuristic solution value - see Figure D.54;
- Improvement made on cost of the initial solution (for both the one-minute and two-minute solutions), as percentage of initial solution value - see Figure D.55;
- Number of changes in the first solution found with lowest cost (for both the one-minute and two-minute results) - see Figure D.56;
- Number of distinct solutions found with cost equal to the lowest cost (for both the one-minute and two-minute results) - see Figure D.57;
- Number of iterations carried out (for both one-minute and two-minute time limits) - see Figure D.58;
- Estimated time at which the first solution with the lowest cost value was found (for both one-minute and two-minute time limits) - see Figure D.59;
- Gap to final (i.e. two-minute) solution if early cut-off (one-minute, 30-second, 20-second and 10-second time limits) was implemented - see Figure D.60.

Examining the histograms for setting Combination #26, it could be seen that the majority of outputs do not conform to the Normality assumption and therefore had to be analysed using the Kruskal-Wallis test. However, it could been seen that the improvement on the initial solution for both the one-minute and two-minute results, as well as the number of iterations carried out within two-minutes only, do follow an approximately Normal distribution - these outputs could therefore be analysed using the F-test. Table 6.19 gives the p-values relating to the tests to determine if the data generation parameters have a significant influence on the outputs; those which are significant at the 5% level are marked with an asterisk. For the improvement on the initial solution (for both one-minute and two-minute results) and the number of iterations carried out within two minutes, we also test for correlation with the generating parameter values. This could only be done for the numerical parameter values; it is not a valid analysis for the boolean variable indicating whether or not the reduction factor $r(d)$ is used. The Pearson correlation coefficients (PCC) between these outputs and parameters along with their associated p-values are shown in Table 6.20, with those p-values significant at the 5% level highlighted with an asterisk. Note that the correlations for availability probability q are identical but opposite to those for probability p, and so for simplicity are not included here.

From Table 6.19, we see that the absence probability p has a significant influence on the values of virtually every output examined. The only exceptions to this are the number of equal-best solutions found, for which the results appear to be completely independent

Table 6.19: Results of F-tests and Kruskal-Wallis tests for influence of parameters used in instance generation on Heuristic algorithm results (Combination #26 only).

Outputs of interest		Test	Parameter					
			p	Use r (d)?	K_N	K_L	$\hat{K}$	K_{AG}
Gap to best known bound	one-min	K-W	< 0.001*	0.106	0.869	0.726	0.969	< 0.001*
	two-min	K-W	< 0.001*	0.102	0.856	0.748	0.969	< 0.001*
Gap to best known solution	one-min	K-W	0.006*	0.595	0.979	0.166	0.688	< 0.001*
	two-min	K-W	0.003*	0.518	0.985	0.097	0.508	< 0.001*
Improvement on initial solution	one-min	F	0.015*	0.051	0.066	0.805	0.315	0.004*
	two-min	F	0.005*	0.043*	0.075	0.617	0.240	0.017*
Number of changes	one-min	K-W	< 0.001*	0.064	0.699	0.981	0.996	0.582
	two-min	K-W	< 0.001*	0.030*	0.899	0.974	> 0.999	0.469
Number of equal-best solutions	one-min	K-W	0.222	0.643	0.108	0.979	0.262	0.485
	two-min	K-W	0.403	0.663	0.245	0.342	0.329	0.311
Number of iterations	one-min	K-W	0.132	0.823	0.134	0.825	0.386	0.029*
	two-min	F	0.001*	0.414	0.030*	0.220	0.094	0.068
Est. time of best solution	one-min	K-W	< 0.001*	0.019*	0.187	0.624	0.785	0.080
	two-min	K-W	< 0.001*	0.032*	0.271	0.607	0.670	0.952
Gap to final solution if algorithm is cut off early	one-min	K-W	0.037*	0.662	0.705	0.624	0.870	0.534
	30-sec	K-W	0.002*	0.014*	0.407	0.424	0.768	0.290
	20-sec	K-W	< 0.001*	0.011*	0.475	0.914	0.959	0.415
	10-sec	K-W	< 0.001*	0.007*	0.229	0.580	0.588	0.599

Table 6.20: Correlation analysis for influence of parameters used in instance generation on Heuristic algorithm results (Combination #26 only).

Outputs of interest			Parameter				
			p	K_N	K_L	$\hat{K}$	K_{AG}
Improvement on initial solution	one-min	PCC	-0.186	0.001	-0.058	-0.042	0.153
		p-value	0.004*	0.983	0.374	0.522	0.017*
	two-min	PCC	-0.205	-0.003	-0.076	-0.057	0.153
		p-value	0.001*	0.960	0.239	0.378	0.018*
Number of iterations	two-min	PCC	0.237	-0.036	0.024	0.003	0.028
		p-value	< 0.001*	0.576	0.709	0.968	0.671

of any of the data generating parameters, and the number of iterations in one minute. Other influences which show up as significant include the agency penalty factor K_{AG} on the three cost-related outputs (gaps to best known bound and best known solution, and improvement on the initial solution), and the use of the time reduction factor which is found to be significant on the best solution time and early cut-off results. Disruption penalties K_L and $\hat{K}$ appear to have no significant bearing on any of the outputs, while near-term penalty K_N is only highlighted as significant in relation to the two-minute number of iterations. This is one of the outputs for which correlation was also carried out, although curiously in Table 6.20 we see that no significant correlation for this interaction was found. The number of iterations in two minutes does however have a significant positive correlation with the parameter p, indicating that more iterations will be carried out if the problems have fewer, longer absences. We can also see the significant correlations for the improvement on the initial solution with both the probability p, which are negatively correlated, and the agency penalty K_{AG}, which are positively correlated. From this we can say that problems with shorter but more frequent absences and with higher agency penalty costs can be expected to return better improvements on the initial solution value.

For those interactions for which correlation analysis was not possible, or in the case of the influence of parameter K_N on the two-minute number of iterations where it was not informative, we can look in detail at histograms showing the breakdown of the outputs by the significant parameters. These histograms are given in the appendix in section D.3.4, with their findings summarised here. Looking first at the cost-related outputs, we see that in cases where a higher value of p is used we have a greater spread of gaps to best bound and of improvement on the initial solution, but that the gaps are also more likely to be smaller and the improvements lesser; by contrast, higher values of p seem to imply larger gaps to the best known solution. As might be expected, all three of these cost measures seem to increase with the agency penalty K_{AG}, with the larger values of K_{AG} also causing a greater spread of the improvement as well as a greater number of large improvements.

We also see that, for the two-minute results, the improvement on the initial solution is smaller when the time reduction factor $r(d)$ is used.

Looking at the other outputs, we see that larger values of probability p are associated with a lower number of changes, as is the use of the time reduction factor for the two-minute results. Higher values of p are also associated with more iterations being carried out within two minutes, while the effects of the disruption factor K_N and agency penalty K_{AG} on the number of iterations are less clear - for the two-minute results, $K_N = 1$ appears to give the highest mean number and has much less dispersion of data points, with $K_N = 5$ giving the lowest mean, and $K_N = 2$ and $K_N = 10$ giving similar averages. Meanwhile, for the one-minute results $K_{AG} = 5$ and $K_{AG} = 10$ appear to give similar outputs, while $K_{AG} = 2$ gives a lower median number and smaller spread of data, and $K_{AG} = 1$ results in the largest median number of iterations. Looking at the best solution time, we see that this is earlier in cases where the value of p is higher and where the time reduction factor is used. Similarly, the gaps to the final solution are more likely to be smaller (or indeed zero) when there are longer but fewer absences and when the time reduction factor has been implemented. Note however that this cannot be stated with certainty for the use of $r(d)$ for the one-minute gap to the final solution, as this was not a statistically significant influence.

6.8.5.5 Summary of effect of parameter values

We now summarise the results discuss throughout section 6.8.5 above. To do this, we will take each of the data generating parameters in turn and outline the effects observed when these values are varied.

Taking firstly the absence probabilities, we recall that this was varied because while the daily absence rate was known the patterns of absence was not. Therefore some data sets have greater probability of shorter more frequent absences (lower value of p; higher value of q) and some are more likely to have longer but fewer absences (higher p; lower q). In this way, we can judge how the solution methods may perform depending on the true absence pattern. From the results above, we would expect for higher values of p (and therefore lower values of q) to see all the solution approaches resulting in smaller gaps to the best known bound in terms of cost, except for the two-minute direct approach which is not significantly affected. By contrast, in these conditions we would expect to see a larger gap to the best known solution for the non-cost-constrained change-minimization and Task-Based Approximation solutions, as well as for the solutions found by Combination #26 of the heuristic algorithm; the Heuristic initial solution approach was not affected on this measure. Comparing costs between the approaches, we see that with higher values of p we would expect the Heuristic initial and Task-Based Approximation solutions to make better improvements on the two-minute direct solution, although the Heuristic initial comparison may see a higher number of extreme negative improvements; the Task-Based Approximation would be expected to have more negative improvements in comparison to the one-hour direct and the change-minimization results. The Heuristic algorithm Combination #26 meanwhile would give a greater variety of improvements on the Heuristic initial solution, but in general these improvements would be lessened. Other effects of the true value of p being high would be that more iterations and more solutions would be produced by the change-minimization approach, while more iterations would also be carried out by the main Heuristic algorithm and the best solution possible within the time limit would be found earlier (and an early cut-off would be less likely to increase the cost greatly; conversely, the heuristic initial approach would require fewer iterations and therefore solve in less time. Finally, a greater likelihood of fewer, longer employee absences would be expected to result in a lower number of changes in the solutions found by the Task-Based Approximation, Heuristic initial and Heuristic Combination #26 approaches.

The absence probabilities can be multiplied by a time reduction factor $r(d)$, which takes into account the possible scenario that future absences are not always known at the present time. However, since it was not known if this multiplication was valid, data sets were generated both with and without the reduction factor being used, and we would expect those which use $r(d)$ to contain fewer crew absences and be easier to solve. From the results in section 6.8.5, we see that the use of the time reduction factor has in fact no significant influence on either the direct solution approach (two-minute or one-hour) or the Task-Based Approximation, while the effect of using the time reduction factor appears to be a negative one on the change-minimization results, with those data sets which use the time reduction factor finding fewer solutions and having a larger gap to the best known solution. The time reduction factor has a more positive effect on the cost of the heuristic initial solution, causing a smaller gap to the best known bound, and this is possibly linked to the fact that these instances also appear to have a poorer improvement by the heuristic algorithm (Combination #26) on this initial solution - note that this factor has no significant influence either way on the cost gaps for the main heuristic algorithm. In addition, we would see the heuristic initial solution as more likely to show a positive improvement on the Task-Based Approximation solution. We also see positive implications of the time reduction factor with respect to number of changes, which would be expected to be lower in both the Heuristic initial and main Heuristic solutions. If this factor is applicable, we would also expect the Heuristic initial approach to solve more quickly, and the main Heuristic algorithm to find its best solution more quickly. As a consequence, the 30-second, 20-second and 10-second cut-off solutions will have a smaller gap to the two-minute Heuristic solution in cases where $r(d)$ is used.

The disruption penalty factors were introduced to account for the fact that making changes is in general undesirable, and in particular making changes to the schedule after transport will have been arranged (i.e. within four weeks). It was not clear what values would be appropriate for these near-term and long-term penalty values, K_N and K_L, and so a variety were chosen during generation of the datasets, including setting $K_N = K_L = 1$ which implies no penalty other than the 'true' cost. The value of $\hat{K}$ was later calculated as an aggregated, or weighted average, penalty factor. It may have been anticipated that these penalty values would influence the cost measures of the solution methods, and also the number of changes. However, the effects described through section 6.8.5 do not seem as wide-ranging as expected, with the neither the gaps to best bound and best solution nor the number of changes being significantly influenced by these parameter values in the Task-Based Approximation, the Heuristic initial, or the Heuristic Combination #26 approaches. We do however have that the direct solution approaches achieve lower gaps to the best known solution and to the best known bound when higher values of the disruption factors are applied, while higher values of K_L specifically will reduce the gap to best bound of the non-cost-constrained change-minimization solution. Linked to this, the highest values of the disruption factors give the smallest improvement by the Task-Based Approximation on the two-minute and one-hour direct results, while the improvements made by the Heuristic initial on the direct solution are also more likely to be negative in these conditions. Other effects of higher values of the disruption penalty factors comprise more iterations being carried out by the change-minimization approach, and a faster running time of the Task-Based Approximation algorithm.

The agency penalty factor performs the same role as the disruption penalties, except that it penalises the use of agency crew rather than making changes. Again, data sets were generated using various values of this, including setting $K_{AG} = 1$ which gave no penalty over and above the financial cost of the agency crew. The high values of the agency penalty seemed to have more impact on results than the disruption penalties, with these giving rise to a greater number gaps towards the extremes both to best bound and best solution for the direct approach, and smaller gaps for non-cost-constrained change-minimization solution. The lowest cost change-minimization solutions however have a larger gap to the best known bound when K_{AG} is increased, and larger gaps to best bound and best known solution are also observed for the Task-Based Approximation, Heuristic initial and Heuristic Combination #26 approaches. Comparing costs across the methods, this means that higher values of K_{AG} are more likely to give negative improvements by both the Task-Based Approximation and Heuristic initial approaches on the one-hour direct and the change-minimization solutions. On the other hand, the Heuristic initial approach is more likely to outperform the Task-Based Approximation for cost under these circumstances, while Heuristic Combination #26 would be expected to produce better improvements on the Heuristic initial solution. One of the few non-cost-related influences of K_{AG}, we finally have that the number of Heuristic algorithm iterations is expected to be highest when the agency penalty is equal to 1.

6.9 Practical Implications of Results

This chapter has set out a more realistic model for the Vessel Crew Scheduling Problem than that discussed in Chapter 5. This model is also more complicated and harder to solve, and therefore several different solution methods had to be developed and tested in an attempt to solve the problem to a reasonable degree of quality in a suitable amount of time. In this section, we review the results of these solution methods and evaluate the relative merits of each. This will then allow us to propose how best the methods discussed here can be used to practical benefit by our client company, or others with similar scheduling problems. We then conclude this chapter with an outline of future work which could be carried out to extend the research discussed here.

6.9.1 Comparison of Solution Methods

In total, five solution approaches were developed and tested for the Time-Windows model: a 'direct' cost-minimization approach; a change-minimization algorithm; solving a Task-Based Approximation of the Time-Windows problem; a fast, low-change Heuristic 'initial' solution; and a Heuristic algorithm for improving on a known solution. When developing the Heuristic algorithm a number of different settings were investigated, with one particular combination, referred to as Combination #26, emerging as giving the best results across all combinations of settings.

The 'direct' solution approach was tested using both one-hour and two-minute time limits, implemented using the FICO Xpress software. The one-hour results produced the most best known solutions for instances across all solution methods; however there were still a number of instances which had a large gap to the best solution, and in general the gaps to the best known bound were high. In terms of applicability, the one-hour time limit cannot be used in practice, while the two-minute time limit would be a much more acceptable running time. However, the solutions obtained within this time limit were a long way from the best solutions obtained by other methods, with both the Task-Based Approximation approaches and the Heuristic initial solution method each achieving better solutions for $\approx 90\%$ of instances.

The change minimization approach was also implemented in FICO Xpress, using the in-built solver and applying the process set out in Algorithm 6.1. In terms of costs, this approach seems to produce the best solutions of all the methods studied, with better gaps to the best known solution on average compared to the two-minute Direct, Task-Based

Approximation, Heuristic initial solution and Heuristic Combination #26 approaches (although the comparison with regard to the latter is much closer). Even though the one-hour direct solution approach found more best known solutions, change-minimization produced a lower gap on average, and across all instances appears to be more consistent. This solution method does however have the disadvantage in terms of number of solutions it produces - because of the difficulty of solving the problem at each iteration, far fewer solutions are produced than when this method was applied in the Task-Based problem previously. Consequently, it is unable to produce the variety of solutions available from other methods, especially the Heuristic algorithm. The fact that it requires the FICO Xpress software, or other commercial solver, may also be a barrier to implementation in practice.

The Task-Based Approximation approach was very quick to solve, with all but two instances solving within 8 seconds (compared to two minutes for the change-minimization algorithm) using FICO Xpress, including the time to convert to and from the Task-Based representation. While only one solution was found in the tests carried out, a proposal for future work in light of this quick solution time was to adapt the approach to allow the proposal of multiple solutions. Unfortunately, while solutions were found quickly the quality was not particularly good, with solutions in general being more expensive than those from the change-minimization and one-hour direct approaches (although they were largely an improvement on the two-minute direct solutions). In addition, the artificial divisions of roles into 'tasks' for the approximation was found to lead to a much greater number of changes in these solutions.

The quality of the Heuristic initial solutions tended to be better than the Task-Based Approximation results, with changes in particular being considerably fewer while the cost comparisons were more varied but on average and improvement. This approach however could only improve on the change-minimization solution cost in a handful of instances, although it was found that three quarters of instances resulted in a Heuristic initial solution with cost within $\pm 10\%$ of the one-hour direct cost; this is impressive when we consider that all Heuristic initial solutions were found within 40 seconds. The tests for this solution method were carried out using FICO Xpress, although it makes no use of the in-built solution methods and could therefore be implemented in another language such as C++ without loss of functionality and potentially with a considerable improvement in solution time. A shorter solution time would also open up the possibility for multiple solutions to be proposed from this method, and this is left as a suggestion for future work.

The last solution method developed was the Heuristic algorithm, for which it was found that Combination #26 (which uses the Heuristic initial solution as its starting point) provided the best results of the combinations tested. Some questions were raised however, in particular with regard to the implementation of the random kick procedure which it is believed warrants further investigation in light of the test results - this is however left for future work. Combination #26 of the Heuristic algorithm was found on average to have more expensive solutions than the change-minimization approach; however, these differences were not as large as for the other solution approaches, with Heuristic Combination #26 achieving a median $\approx 70.09\%$ gap to best bound and $\approx 38.60\%$ gap to best known solution, compared to medians of $\approx 64.29\%$ and $\approx 27.15\%$ respectively for the lowest cost change-minimization solution. Comparing to other solution methods, it was found that Combination #26 made on average $\approx 41\%$ improvement on the Heuristic initial solution cost, while there was a small increase in the number of changes although the number was still much lower than the changes entailed in the Task-Based Approximation solution.

The Heuristic algorithm, and Combination #26 in general, also has a number of advantages over the change-minimization approach. The main attribute in its favour is that it has already been implemented in C++ and does not require a commercial solver or any in-built optimization solvers. This means that the cost of implementation should be reduced, and the ease of integration with other company systems increased, making it a more attractive proposal in practice. There is also the advantage of being able to terminate the algorithm early - while the change-minimization procedure requires a full two minutes to obtain its best solutions, we find that Combination #26 would on average be only $\approx 5\%$ from the final solution for that instance if it was terminated after 30 seconds. This means that faster solutions can more readily be found, meaning that there is less restriction on the number of times a Planner can re-run the programme if an iterative approach is required. Finally, we have the number of iterations which are carried out by the Heuristic algorithm. Combination #26 was found to carry out a mean of ≈ 145.22 iterations within the first minute of the run, meaning that a large number of solutions will be found. While the number of high quality solutions is uncertain and should receive further attention, it is clear that more solutions will generally be proposed than from the change-minimization approach (the median number of equal-best solutions from Combination #26 in one minute is 9; the maximum number of solutions from a change-minimization run is 5). The Heuristic algorithm therefore presents the Planner with more choice of solutions. For these reasons it is believed that of the solution methods examined, Heuristic Combination #26 is the most suitable for a practical implementation.

This assessment is unlikely to be affected by concerns about the true values of the parameters used in data generation. Larger values of the parameter p (and therefore lower of q) for example were found to improve the gaps to the best bounds for all approaches, while more iterations would be carried out and more solutions would be found by both the change-minimization and Heuristic Combination #26 methods. If the time reduction factor $r(d)$ is indeed a valid assumption then we would see fewer solutions and a larger gap to the best solution for the change-minimization approach, while Combination #26 would find the best solution more quickly and have fewer changes. If the company decided to have larger penalty values for using agency crew (i.e. increased K_{AG}), we would have larger cost gaps for all approaches, and the increased negative improvements of the Heuristic initial on the change-minimization solutions would be mitigated by the better improvement on the initial solution made by Combination #26. Finally, changing the values of the disruption penalty factors K_N, K_L and $\hat{K}$ would have no significant impact on gaps for either the change-minimization or Heuristic solution approaches.

6.9.2 Possible Implementation

The findings set out in this chapter, and summarised in section 6.9.1 above, allow us to put together a proposal for an implementation which could be of benefit to our client company in practice. Our proposed implementation would take the form of a planning tool, with the methods discussed above underpinning the solution process. We note that this would require work in terms of software development to create a user interface for the tool, particularly with a view to giving the solution output in a clear and user-friendly manner, and to integrate the tool with other company systems to allow the required data to be accessed. This is left as proposed future work for someone with the requisite skill set; here we will simply present an outline of how we envisage the tool being used.

Firstly, we note that it is at present uncertain what additional penalty 'costs' the company may want to apply either for disruptions or for using agency crew. While these have been shown to have little significant effect on the number of changes in the proposed solutions, and the disruption penalties have no significant impact on the Heuristic performance with respect to cost, it is still important that the company can define the problem in the way that is most appropriate for them. Indeed, it may be that the company believes the best option is to leave decisions such as this to the individual Planners, perhaps to be varied in accordance with the situation. We therefore suggest that as an initial step, the planning tool could allow the Planners to indicate on a scale their level of aversion to making near-term and long-term changes, and to using agency crew. Translating these aversions into the penalty factors K_N, K_L and K_{AG}, these could be used to modify the allocation change costs in the problem in a similar way to the generated costs were modified when creating our randomised realistic data sets (see section 6.2.1, Step 5).

After this initial step, it is envisaged that the solution process could take an iterative form, allowing the Planner to view and evaluate the available solutions, and if required make changes to the problem. A flowchart detailing the proposed procedure is given in Figure 6.40. As shown in this diagram, the first step of solving the problem is to apply the Heuristic initial solution method. As discussed, this is unlikely to give the best solution in terms of cost, but should find a low-change solution which does not have excessive cost and it is anticipated, if implemented in a language such as C++, should do so in a very

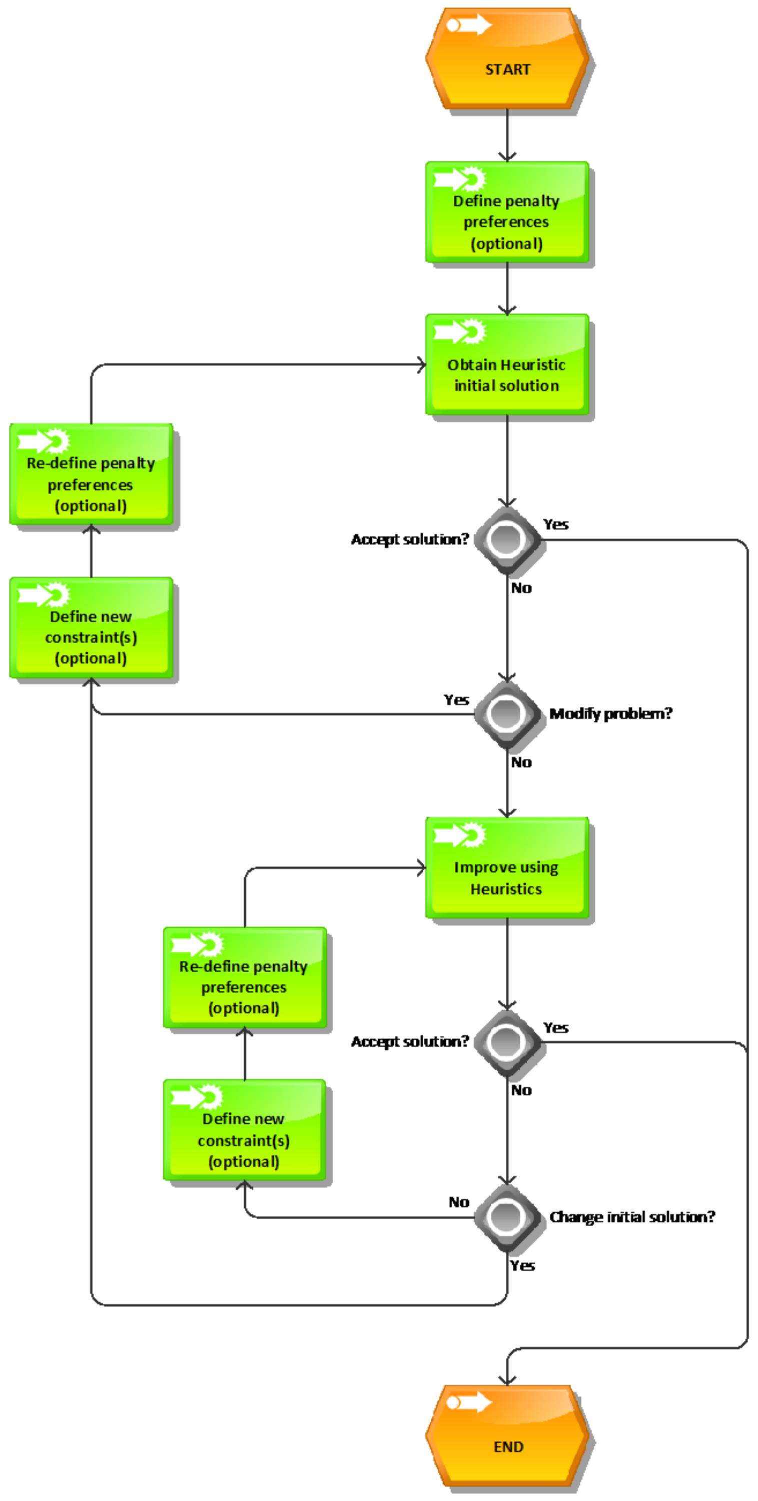

START
Define penalty preferences (optional)
Obtain Heuristic initial solution
Re-define penalty preferences (optional)
Define new constraint(s) (optional)
Accept solution?
Yes
No
Yes
Modify problem?
No
Improve using Heuristics
Re-define penalty preferences (optional)
Define new constraint(s) (optional)
Accept solution?
Yes
No
No
Change initial solution?
Yes
END

Figure 6.40: Flowchart showing the process for using the proposed planning tool for the Time-Windows problem.

short amount of time. As noted in the above discussion, a C++ implementation of this algorithm should be investigated further, and could potentially open up the possibility of multiple solutions being supplied at this stage, allowing the Planner a choice.

It is possible that the Planner may be happy to accept this solution, especially if they wish a solution which requires a small number of changes, in which case the process can terminate now. Alternatively, they may wish to impose an additional constraint (for example if they do not wish a particular employee to be disrupted or to be allocated a certain role at a certain time), or to alter the penalty preferences set earlier. If this is the case, they could make these adjustments and find a new Heuristic initial solution. Otherwise, there is the option to improve on the initial solution using the Heuristic algorithm and specifically, based on the settings tested here, using setting Combination #26. Test results show that within a two minute time limit on average an $\approx 41\%$ improvement on the initial solution cost will be achieved with only a net increase of two changes, and we also know that on average solutions found within 30 seconds have only a $\approx 5\%$ gap to this two-minute solution. Therefore we propose that the Heuristic Combination #26 could be applied for 30 seconds during this implementation.

As described above, the Heuristic algorithm will in most cases generate a large number of solutions, and the planning tool will be able to present the Planner with a selection of these, allowing them to choose between their preferred options. If none of these solutions are acceptable and the Planner has time to carry out additional solution iterations, it would be possible to adjust the penalty preferences or add additional constraints and re-run the heuristics from the same initial solution. Alternatively, the process could be re-started from scratch with new preferences or new constraints, finding a new Heuristic initial solution before applying the Heuristics again.

It is important to note that at this stage we do not know how the quality of schedules which will be implemented as a result of this method would compare to the schedules found by the current process (as described in Chapter 4). Indeed, it is not known how any of the solution methods described compare to the current approach used by the Planners. This was unavoidable during the research process, with data being difficult to access or not stored, and some information being commercially sensitive. Clearly it would be of benefit to have *real*, as opposed to randomly generated *realistic*, data sets on which to test these solution methods, and particularly useful to have access to the solutions found by the Planners in these cases as a benchmark. This information might turn out to be a benefit resulting from a project to implement this decision support tool, perhaps involving carrying out trials of the new support tool against the current scheduling approach.

6.9.3 Future Work

We conclude this chapter with a summary of the proposals for future work and other questions arising from the work discussed. There were a number of these raised throughout the chapter, and these were as follows:

1. Adapting the Task-Based Approximation approach to produce multiple results.
   While it has been found that the Task-Based Approximation does not produce the best results in terms of cost or indeed number of changes, it does have the benefit of producing results quickly. There is still therefore the chance that producing multiple possible schedules will be of benefit to the company. As part of this, some variations to the solution method (note that it is currently solved using the FICO Xpress solver) or the approximation procedure could be tried.

2. Implementing the Heuristic initial solution approach in C++ (or similar).
   This piece of future research could prove to be important in implementing the decision support tool described in section 6.9.2 above. The algorithm itself should not require any significant modification, but implementation in a more efficient language should allow substantial time savings to be made. In conjunction with this, and similar to above, the chance would then arise to generate multiple solutions via this method in a more structured way, thereby presenting the Planner with a greater choice when using the planning tool.
   Note that at time of writing, fellow PhD student Seda Sucu had started work on this avenue of further research.

3. New implementation rules for the 'random kick' procedure in the Heuristic algorithm.
   As discussed in section 6.8.4.9, the best results across the heuristic settings tested were achieved when no kick was implemented. However, indications were that implementing a kick after a longer waiting time was more beneficial than with a shorter waiting time. The suggested here was therefore to investigate new kick implementation rules which would force a longer wait than what has been tested here, thereby allowing the algorithm to better improve the solution before the next kick is carried out. Note that if this proves successful then a new group of Heuristic settings could be used in the proposed planning tool described in section 6.9.2 above.

4. Implementing the planning tool described above.
   Perhaps the key piece of future work arising from this research is to use the methods tested to implement a planning tool which can be of benefit to the company in practice. As noted above, this will require the development of a user interface and integration with the company's current scheduling and data storage systems. We

also recommend a small update to the Heuristic algorithm to allow a number of solutions with cost close to the best found to be stored, as well as those with equal-best cost. A degree of flexibility will have to be considered here, for example to allow for modifications to the Heuristic algorithm if any promising results are found from the new kick implementation settings as proposed above.

5. Obtaining real data instances and real solutions.

   Above, we highlighted the importance of being able to test these solution methods on real data sets and, in particular, being able to compare the solutions we have obtained against the solutions the Planners would find by their current scheduling approach. As noted in section 6.9.2, this may be something which arises from the work to implement the proposed planning tool.

6. The Column Generation approach.

   As well as the methods which were tested ('direct' cost-minimization, minimizing number of changes, Task-Based Approximation, Heuristic initial solution and the main Heuristic algorithm), section 6.7 outlined a way in which a Column Generation solution method could be applied. Unfortunately it was not possible to implement and test all possible solution methods on this problem; however, given that Column Generation is one of the recognised solution methods for scheduling problems it would be interesting to see how this approach compares to the methods tested here.

   Note that at time of writing, fellow PhD student Seda Sucu has started work related to this on a Benders' Decomposition approach for solving the Time-Windows problem.

Chapter 7

Conclusions

This chapter draws to a conclusion the research discussed in this document. The conclusions are, broadly, of two types. Firstly, in section 7.1 we summarise the conclusions given in the previous chapters which focus specifically on the problem studied in this research. We then in section 7.2 go on to discuss the wider implications of the findings, with reference to the literature discussed in chapter 2 and to the generalisability of the work described here. In addition to this, section 7.3 summarises the future work which it is envisaged could arise from this research.

7.1 Conclusions for our Specific Problem

Our specific problem was described in chapter 4. Not only were the constraints of and current approach to the problem explained, but in section 4.2.3 we discussed the ways in which an optimization intervention may be able to make the problem easier to solve. The key aim was to develop a basis for an optimization tool, which would allow the identification of multiple feasible schedules in a short space of time, and which would be able to bring such quality measures as cost and number of proposed changes into consideration when doing this.

The first formulation presented in chapter 5 for the problem made some simplifying assumptions. It was assumed that there was a pre-determined and fixed regular assignment pattern for each vessel, and that consequently all employees' contracts permitted the minimum rest and maximum working periods that this pattern would entail. Consequently, while it was found that the change-minimization approach (described in Algorithm 5.2) was useful in quickly producing a Pareto-optimal set of numerous solutions to this Task-Based formulation, the formulation itself was found not to be a sufficiently accurate representation of the real problem faced in our case study company to be applicable here. However, we can still say that this would provide a possible solution approach for a different company where assumptions of fixed tasks and homogeneous crew group do hold.

A more realistic 'Time-Windows' formulation was presented in chapter 6, which relaxed the simplifying assumptions. The model was much more complex, and as a result the solution approaches used for the Task-Based problem were far less useful for solving the problem. Several solution methods were discussed, including using the Task-Based formulation to approximate the Time-Windows problem, and Heuristic methods both for finding an initial solution and for improving a known solution. A Column Generation approach was also proposed, although there was not the opportunity to test this method and this is therefore left as a suggestion for future work. Of the methods that were tested, the change-minimization approach was found to provide the cheapest solutions within a two-minute time limit, but was seen to have two main disadvantages - firstly that it would produce only a very small number of solutions, and secondly that it relied on the FICO Xpress software for its implementation which may present a barrier financially to its use in a practical setting.

On the other hand, the Heuristic algorithm developed was found to obtain solutions with costs not too much higher than those from the change-minimization algorithm. In addition, the Heuristic approach was found to have a number of advantages over change-minimization - it does not require specialist software such as FICO Xpress, making it a more attractive proposal for a company to use in practice; it can return a far greater number of solutions; and it can be terminated early, e.g. after 30 seconds, without significant loss of solution quality. We can therefore say that it is able to meet the goals of recommending multiple solutions of reasonable quality in a short space of time. We believe this still holds true when we consider that the data used to test the solution methods was randomly generated using realistic parameters, with analysis of the results indicating that the Heuristic method proposed would still be the most suitable of those tested regardless of fluctuations of the true parameter values within the bounds used.

In light of these findings, the chapter concluded in section 6.9.2 with a proposal for a planning tool which could be used by our case study company in practice. It was proposed that the Heuristic initial solution method and improvement algorithm which were developed would underpin an iterative process within the planning tool. The process would see an initial solution found and proposed to the Planners, who would then be able impose additional constraints (e.g. to guarantee or preclude the assignment of an employee to a particular role) or perhaps modify the penalties for disruption or for using agency crew. If the problem was modified it could be re-solved, and otherwise it could be improved upon using the Heuristic algorithm. This process of modification and improvement could be repeated for as long as the Planner required, until one of the multiple solutions provided was found to be acceptable.

We feel that this approach has the potential to greatly improve the crew scheduling process at the company, both with regards to time taken and to the quality of the schedules. However, as was noted earlier it was not possible during this research to evaluate the time taken and the solution quality for the company's current approach. It is hoped that a future research step may be taken to design and implement the proposed scheduling tool in practice, in which case an opportunity should arise to collect data which during this project proved to be difficult to access or locate, if it was stored at all. This would in turn also provide the opportunity to test the solution methods on *real* rather than just randomly generated *realistic* data.

7.2 Wider Implications

When considering the contribution of the work discussed here, we should consider the relevance to the wider scheduling literature as well as to our specific problem. To do this, we will consider some of the other work discussed in chapter 2. As with the literature review, we will firstly consider the other transportation literature and general scheduling literature, before looking at how our findings fit within the more specifically maritime literature and along side other vessel crew scheduling problems. We will conclude with some remarks about the overall generalisability of our results.

Looking firstly at the transportation literature, we note that a majority of the work discussed here do not use heuristic methods. However, we see that some papers have proposed heuristic algorithms for solving crew scheduling problems, albeit different kinds of heuristic methods from that proposed here. For example, Maenhout and Vanhoucke (2010) for the air crew rostering problem and Elizondo et al. (2010) for underground rail crew propose evolutionary algorithms, while Lourenço et al. (2001) have developed a tabu search and genetic algorithm metaheuristic for bus crew scheduling.

Some interesting parallels with our work can be drawn with the work done by Haase et al. (2001), and specifically with the fact that their model was tested on randomly generated data designed to represent a plausible real-world problem. While it can be argued that this means the model has not been proven to be useful and relevant to an actual *real world* problem, it can also be argued that the fact that the model has not been constructed for a specific company or scenario, and does not rely on the structure or layout of a particular set of routes, makes their approach more robust. This can be compared with comments made by Gamache et al. (1999) for example, who point out that the success of their model can be attributed, in part, to some of the peculiarities of the crew rostering system at Air France.

It is also interesting to note comments by Nissen and Haase (2006) on recovery problems in European and North American airlines. According to them, in Europe crew are generally paid fixed salaries, and so the emphasis is on making the fewest changes from the existing schedule and the rescheduling is done based on individual duties. Conversely, in North America crew are paid according to working time, and so the emphasis is on cost-minimization and the recovery problem relies heavily on 'pairings'. This can be linked to our problem, where we have applied both a cost-minimization and change-minimization approach. We recall that for the Task-Based problem these seemed to be very much competing objectives, whereas with the Time-Windows formulation there appeared more of a link between minimizing costs and minimizing changes.

Recalling the discussion of more general scheduling literature, there were a number of papers mentioned here where heuristics were found to be the best solution approach. This covered a variety of methods, with Tsang and Voudouris (1997) proposing local search algorithms for scheduling engineers at BT, Nguyen and Wright (2014) using variable neighbourhood search for the workload balancing problem, Kovacs et al. (2012) using adaptive large neighbourhood search (ALNS) for scheduling service technicians, and Wright (2007a) developing a metaheuristic for cricket umpire scheduling. The problem discussed by Wright (2007a) gave rise to the detailed heuristic experiments discussed by Wright (2007b), and parallels can be drawn between this and the experiments that we have carried out (see description in section 6.5.4 and results in section 6.8.4). While clearly there is a difference in terms of the kinds of setting alterations which were tested, there is a similarity in the sense that both involve multiple runs to determine which settings are most effective. It can be argued that our experiments are not as rigorous, with Wright (2007b) carrying out 100 identical runs on each combinations of settings; however, as noted in that paper the experiments were carried out on a single problem instance. By contrast, ours were tested on 240 different data sets, albeit these all followed the same overall problem structure, and it can perhaps be argued that these results give a better indication of the algorithm's performance on similar problem instances.

Looking now at the other maritime literature, we recall the quote that was drawn from Christiansen et al. (2007):

> *"Crew scheduling for deep-sea vessels is not a major issue. Crew members spend months on the vessel and then get a long shore leave. For short-sea vessels the crew may change frequently, and crew scheduling may be an issue."* (pp.263-264)

Based on our experiences dealing with the case study company, we argue that this *is* an important issue. Certainly our company could be termed a "deep-sea vessels" operating company in the respect that crew members spend many weeks and at times months on board the vessel, with standard shore leave being upwards of four weeks. Despite this, the company's crew scheduling still causes some problems, especially when there is the requirement to reschedule in response to changes being made. From our results, we have also seen that this can be a difficult problem to formulate in a realistic way and yet still be able to solve in an efficient manner, given the time pressures usually placed on the Planners. As discussed in section 2.5, some similarities can be seen between our problem and those studied by Wermus and Pope (1994) (harbour pilot scheduling), Horn et al. (2007) (navy crew scheduling), Ammar et al. (2013) (ferry crew scheduling) and Giachetti et al. (2013) (cruise line crew scheduling); however, there are also some clear distinctions between the problems and also the solution approaches employed.

7.2.1 Contributions to Knowledge

Following from the discussion above, we believe one of the key contributions to knowledge from this work is to demonstrate an application of crew scheduling modelling which has not previously appeared in the literature. As discussed above, crew scheduling in the sector in which our company operates is not generally considered an important problem; however, it has been shown here to be a useful and interesting problem for two main reasons. Firstly, we have demonstrated that the scheduling problem faced by the company is a key part of the business process which must be carried out multiple times, and that it is not always straightforward for planners to find a good solution or indeed a feasible solution. In addition, the difficulties shown in finding good results using optimization methods indicate that this is not a simple problem to solve. Related to this, we can also see there is a potential for substantial financial savings for the company by using an optimization approach. While we unfortunately have no real data instances on which we can test our methods against the solutions implemented by the planners, we can for example compare the lowest-cost and non-cost-constrained solutions found by the change-minimization approach (see Figures 6.5 and 6.6 respectively) to see the range of possible solution costs available for each instance. Therefore, we can say that any method which produces solutions of a cost close to the lowest-cost change-minimization or setting combination #26 of the Heuristics is likely to produce lower cost solutions than the solution found by the planners, as this will be more akin to a minimal-change solution.

In addition, there is a contribution relating to the overall generalisability of the results seen here. Clearly the problem has been formulated, and the solution methods tested, specifically to meet the needs of our case study company; however, it would be expected that the proposals here would also be relevant in another case where the company needs and the problem description were very similar (or, as noted about the Task-Based model in section 7.1 above, at a company where the assumptions made can be seen to hold true). It is more likely however that another company would have slightly different requirements and, particularly, a different problem specification. In a case such as this, we can say that broadly the methods proposed may still be appropriate. In particular, the Heuristic algorithm developed here relies on testing certain rules and criteria for feasibility, and it would not significantly affect the overall algorithm if some of these rules were modified or new ones added. Similarly, if our company were to find itself subject to new contractual rules or new laws, the programme could be adapted to meet these without having to create a new mathematical formulation of the problem. However, by making changes to the algorithm's rule we do not know how this would affect its operation - it is possible that some re-testing of the settings, or experimentation with new settings, may be necessary.

We can also take note of contributions to knowledge relating directly to the heuristic algorithm, as some of the results from the tests carried out may be applicable in other problems where a heuristic algorithm is applied. In particular, it was observed that there was an advantage to reducing the search space (and time) by taking such steps as limiting the number of employees examined at each iteration, and accepting the first improving solution rather than waiting for an improvement on the best solution yet found. This had the effect of increasing the number of iterations, which in turn appeared to allow more opportunity for the algorithm to find better solutions. The effectiveness of this may be attributable to the fact that, while the crew group is not homogeneous, there are a large number of crew with similar if not identical availability and cost, thus rendering an exhaustive search unnecessary. We can therefore say that this approach may be useful in similar situations where there are numerous similar (or effectively identical) employees in the crew group.

An additional observation to from the heuristic tests was the ineffectiveness of the "random kick" applied to move the search away from local optima. Indeed, it appeared that poorer results were found when these were applied. A possible explanation is that, given the structure of the search space with relation to cost, the kick moves the search too far away from a good solution and in some cases to an area with a much higher local optimum. Given then number of potential solutions available, it can be seen why this is likely. Possible remedies for this would be to adjust the activations settings for the kick, so that more time is allowed for the algorithm to bring the solution cost back down to a more acceptable level; or to alter the kick itself to prevent such drastic moves away from the current schedule. It is believed that this lesson too can also be generalised to other problems on which local search is applied, where the structure of the cost function means there are a large number of very expensive solutions available.

7.3 Future Work

Finally, we will summarise the future work which we see potentially arising from the research discussed here. The majority of this relates to the work on the Time-Windows formulation of the problem; however, as discussed in section 5.6.1 the Task-Based model may also give rise for further research opportunities. Ideas relating to this comprised the implementation of the proposed support tool at a suitable company, and looking to make further improvements to the change-minimization algorithm given in Algorithm 5.2, particularly with respect the cost limit as it approaches zero.

Meanwhile, a number of further avenues of research were suggested in section 6.9.3 based on the findings for the Time-Windows version of the problem. This included ideas to improve the existing solution methods, such as adapting the Task-Based Approximation approach to produce multiple results, developing a faster and more efficient implementation of the heuristic initial solution approach in C++ (or a similar language), and investigating new activation rules for the 'random kick' in the Heuristic algorithm. In addition to this, it was also suggested that future work could see the planning tool recommended in section 6.9.2 being implemented in practice, with the additional possibility of taking the opportunity to obtain real data instances and real solutions for further testing and comparison purposes. There is also the interesting possibility of other solution methods being used on the Time-Windows problem. A possible column generation approach was outlined in section 6.7, given that this is another method which is often applied to scheduling problems. It was noted that fellow PhD student Seda Sucu is, related to this, currently exploring a Benders' Decomposition approach to solving the problem.

In addition to this, we recall that it was noted in section 2.2.1 that there were several categories of approach which could be taken to scheduling under uncertainty. Lütjen and Karimi (2012) discussed three categories: proactive, reactive, and predictive-reactive scheduling. We identified our company's current approach to crew scheduling as a predictive-reactive, and sought to develop solution methods which would fit with this approach. We also noted that robust scheduling (i.e. the proactive approach) required further information about the uncertainties and their distributions, which was currently not available at the company. However, it is possible that a robust optimization approach would be beneficial to the company if it allowed robust schedules to be produced, requiring less disruption and rescheduling at a later date. In addition to the work on Benders' Decomposition, Seda Sucu is also currently working on this idea.

Bibliography

Aas, B., O. Halskau, S.W. Wallace. 2009. The role of supply vessels in offshore logistics. *Maritime Economics & Logistics* **11**(3) 302–325.

Ammar, M.H., M. Benaissa, H. Chabchoub. 2013. Grasp for seafaring staff scheduling: Real case. M. Abed, M. Benaissa, eds., *2013 International Conference on Advanced Logistics and Transport*. Ieee, New York., 427–433.

Barnhart, C., L. Hatay, E.L. Johnson. 1995. Deadhead selection for the long-haul crew pairing problem. *Operations Research* **43**(3) 491–499.

Barrett, D. 2008. The offshore supply boat sector. *Marine and Commerce* 36–41.

Blanco, T.A., R.C. Hillery. 1994. A sea story - implementing the navy personnel assignment system. *Operations Research* **42**(5) 814–822.

Butchers, E.R., P.R. Day, A.P. Goldie, S. Miller, J.A. Meyer, D.M. Ryan, A.C. Scott, C.A. Wallace. 2001. Optimized crew scheduling at air new zealand. *Interfaces* **31**(1) 30–56.

Calculator.net. 2016. Inflation calculator. Webpage: http://www.calculator.net/inflation-calculator.html. Accessed on 22nd February 2016.

Cappanera, P., G. Gallo. 2004. A multicommodity flow approach to the crew rostering problem. *Operations Research* **52**(4) 583–596.

Christiansen, M., K. Fagerholt. 2011. Some thoughts on research directions for the future: Introduction to the special issue in maritime transportation. *Infor* **49**(2) 75–77.

Christiansen, M., K. Fagerholt, B. Nygreen, D. Ronen. 2007. Maritime transportation. C. Barnhart, G. Laporte, eds., *Handbook in OR & MS*, vol. 4. Elsevier B.V., 189–284.

Christiansen, M., K. Fagerholt, B. Nygreen, D. Ronen. 2013. Ship routing and scheduling in the new millennium. *European Journal of Operational Research* **228**(3) 467–483.

Christiansen, M., K. Fagerholt, D. Ronen. 2004. Ship routing and scheduling: Status and perspectives. *Transportation Science* **38**(1) 1–18.

Clausen, J., A. Larsen, J. Larsen, N.J. Rezanova. 2010. Disruption management in the airline industry - concepts, models and methods. *Computers & Operations Research* **37**(5) 809–821.

Dodin, B., A.A. Elimam, E. Rolland. 1998. Tabu search in audit scheduling. *European Journal of Operational Research* **106**(2-3) 373–392.

Elizondo, R., V. Parada, L. Pradenas, C. Artigues. 2010. An evolutionary and constructive approach to a crew scheduling problem in underground passenger transport. *Journal of Scheduling* **16** 575–591.

Ernst, A.T., H. Jiang, M. Krishnamoorthy, B. Owens, D. Sier. 2004a. An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research* **127**(1-4) 21–144.

Ernst, A.T., H. Jiang, M. Krishnamoorthy, D. Sier. 2004b. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* **153** 3–27.

European Union. 2003. Directive 2003/88/EC of the European Parliament and of the Council of 4 November 2003 concerning certain aspects of the organisation of working time. *Official Journal of the European Union* **L299**(46) 9–19.

Fagerholt, K., M. Christiansen, L. Magnus Hvattum, T.A.V. Johnsen, T.J. Vabø. 2010. A decision support methodology for strategic planning in maritime transportation. *Omega* **38**(6) 465–474.

Fagerholt, K., H. Lindstad. 2000. Optimal policies for maintaining a supply service in the norwegian sea. *Omega* **28**(3) 269–275.

Gamache, M., A. Hertz, J.O. Ouellet. 2007. A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers & Operations Research* **34**(8) 2384–2395.

Gamache, M., F. Soumis, G. Marquis, J. Desrosiers. 1999. A column generation approach for large-scale aircrew rostering problems. *Operations Research* **47**(2) 247–263.

Garrett, D., D. Dasgupta, J. Vannucci, J. Simien. 2007. Applying hybrid multiobjective evolutionary algorithms to the sailor assignment problem. L.C. Jain, V. Palade, D. Srinivasan, eds., *Advances in Evolutionary Computing for System Design*, vol. 66. Springer-Verlag Berlin, Berlin., 269–301.

Giachetti, R.E., P. Damodaran, S. Mestry, C. Prada. 2013. Optimization-based decision support system for crew scheduling in the cruise industry. *Computers & Industrial Engineering* **64**(1) 500–510.

Gopalakrishnan, B., E. Johnson. 2005. Airline crew scheduling: State-of-the-art. *Annals of Operations Research* **140**(1) 305–337.

Gribkovskaia, I., G. Laporte, A. Shlopak. 2008. A tabu search heuristic for a routing problem arising in servicing of offshore oil and gas platforms. *Journal of the Operational Research Society* **59**(11) 1449–1459.

Haase, K., G. Desaulniers, J. Desrosiers. 2001. Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science* **35**(3) 286–303.

Halvorsen-Weare, E.E., K. Fagerholt, L.M. Nonas, B.E. Asbjornslett. 2012. Optimal fleet composition and periodic routing of offshore supply vessels. *European Journal of Operational Research* **223**(2) 508–517.

Hoffman, K.L., M. Padberg. 1993. Solving airline crew scheduling problems by branch-and-cut. *Management Science* **39**(6) 657–682.

Holder, A. 2005. Navy personnel planning and the optimal partition. *Operations Research* **53**(1) 77–89.

Horn, M., H. Jiang, P. Kilby. 2007. Scheduling patrol boats and crews for the royal australian navy. *Journal of the Operational Research Society* **58**(10) 1284–1293.

Huisman, D., A.P.M. Wagelmans. 2006. A solution approach for dynamic vehicle and crew scheduling. *European Journal of Operational Research* **172**(2) 453–471.

Irnich, S. 2008. Resource extension functions: properties, inversion, and generalization to segments. *OR Spectrum* **30**(1) 113–148.

Jütte, S., M. Albers, U.W. Thonemann, K. Haase. 2011. Optimizing railway crew scheduling at db schenker. *Interfaces* **41**(2) 109–122.

Jütte, S., U.W. Thonemann. 2012. Divide-and-price: A decomposition algorithm for solving large railway crew scheduling problems. *European Journal of Operational Research* **219**(2) 214–223.

Kohl, N., S.E. Karisch. 2004. Airline crew rostering: Problem types, modeling, and optimization. *Annals of Operations Research* **127**(1-4) 223–257.

Kovacs, A.A., S.N. Parragh, K.F. Doerner, R.F. Hartl. 2012. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling* **15**(5) 579–600.

Legato, P., M.F. Monaco. 2004. Human resources management at a marine container terminal. *European Journal of Operational Research* **156**(3) 769–781.

Li, H.T., K. Womer. 2009. A decomposition approach for shipboard manpower scheduling. *Military Operations Research* **14**(3) 67–90.

Li, Z.K., M. Ierapetritou. 2008. Process scheduling under uncertainty: Review and challenges. *Computers & Chemical Engineering* **32**(4-5) 715–727.

Liang, T.T., T.J. Thompson. 1987. A large-scale personnel assignment model for the navy. *Decision Sciences* **18**(2) 234–249.

Lourenço, H.R., J.P. Paix ao, R. Portugal. 2001. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science* **35**(3) 331–343.

Lütjen, M., H.R. Karimi. 2012. Approach of a port inventory control system for the offshore installation of wind turbines. *Proceedings of the Twenty-second International Offshore and Polar Engineering Conference*. 502–508.

Maenhout, B., M. Vanhoucke. 2010. A hybrid scatter search heuristic for personalized crew rostering in the airline industry. *European Journal of Operational Research* **206**(1) 155–167.

Nguyen, T.-H., M. Wright. 2014. Variable neighborhood search for the workload balancing problem in service enterprises. *Computers & Operations Research* **52** 282–290.

Nissen, R., K. Haase. 2006. Duty-period-based network model for crew rescheduling in european airlines. *Journal of Scheduling* **9**(3) 255–278.

Nurmi, K., D. Goossens, J. Kyngäs. 2014. Scheduling a triple round robin tournament with minitournaments for the finnish national youth ice hockey league. *Journal of the Operational Research Society* **65** 1770–1779.

Ouelhadj, D., S. Petrovic. 2009. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling* **12**(4) 417–431.

Pallant, J. 2011. *SPSS Survival Manual: A step by step guide to data analysis using SPSS*. 4th ed. McGraw-Hill, Maidenhead.

Papadakos, N. 2009. Integrated airline scheduling. *Computers & Operations Research* **36**(1) 176–195.

Pidd, M. 2003. *Tools for Thinking: Modelling in Management Science*. 2nd ed. Wiley, Hoboken, N.J.

Potthoff, D., D. Huisman, G. Desaulniers. 2010. Column generation with dynamic duty selection for railway crew rescheduling. *Transportation Science* **44**(4) 493–505.

Psaraftis, H.N. 1999. Foreword to the focused issue on maritime transportation. *Transportation Science* **33**(1) 1–2.

Rezanova, N.J., D.M. Ryan. 2010. The train driver recovery problem - a set partitioning based model and solution method. *Computers & Operations Research* **37** 845–856.

Ribeiro, C.C., S. Urrutia. 2012. Scheduling the brazilian soccer tournament: Solution approach and practice. *Interfaces* **42**(3) 260–272.

Ronen, D. 1983. Cargo ships routing and scheduling: Survey of models and problems. *European Journal of Operational Research* **12**(2) 119–126.

Ronen, D. 1993. Ship scheduling: The last decade. *European Journal of Operational Research* **71**(3) 325–333.

Ropke, S., D. Pisinger. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* **40** 455–472.

Saddoune, M., G. Desaulniers, I. Elhallaoui, F. Soumis. 2011. Integrated airline crew scheduling: A bi-dynamic constraint aggregation method using neighbourhoods. *European Journal of Operational Research* **212** 445–454.

Scholz-Reiter, B., M. Lütjen, J. Heger, A. Schweizer. 2010. Planning and control of logistics for offshore wind farms. *Proceedings of the 12th WSEAS international conference on Mathematical and computational methods in science and engineering*. 242–247.

Stopford, M. 2009. *Maritime Economics*. 3rd ed. Routledge, New York.

Tsang, E., C. Voudouris. 1997. Fast local search and guided local search and their application to british telecom's workforce scheduling problem. *Operations Research Letters* **20**(3) 119–127.

Van den Bergh, J., J. Beliën, P. De Bruecker, E. Demeulemeester, L. De Boeck. 2013. Personnel scheduling: A literature review. *European Journal of Operational Research* **226**(3) 367–385.

Vieira, G.E., J.W. Herrmann, E. Lin. 2003. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling* **6**(1) 39–62.

Wei, G., G. Yu, M. Song. 1997. Optimization model and algorithm for crew management during airline irregular operations. *Journal of Combinatorial Optimization* **1**(3) 305–321.

Wermus, M., J.A. Pope. 1994. Scheduling harbor pilots. *Interfaces* **24**(2) 44–52.

Weske, M. 2007. *Business Process Management: Concepts, Languages, Architectures*. Springer, Berlin.

Wright, M. 2007a. Experiments to shed light on the best way to use iterated local search for a complex combinatorial problem. *Lancaster University Management School Working Paper Series.* **2007/037**.

Wright, M. B. 2007b. Case study: problem formulation and solution for a real-world sports scheduling problem. *Journal of the Operational Research Society* **58**(4) 439–445.

Appendix A

Business Process Maps

The diagrams here (Figures A.1, A.2 and A.3) show an expanded version of the Business Process Map given in Figure 4.1 (in section 4.1). They follow the same principles of Business Process Modelling and Notation (BPMN) 2.0.

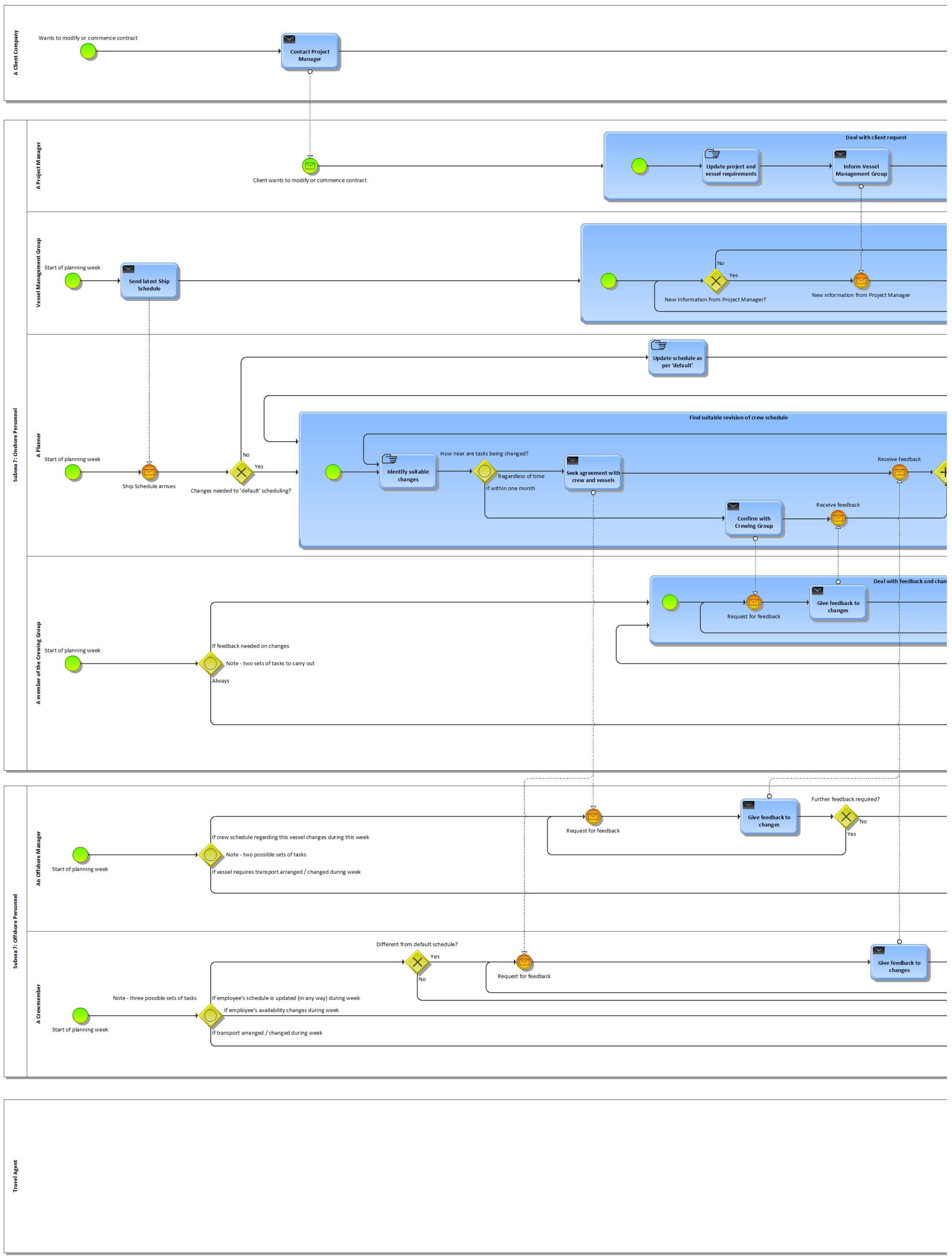

A Client Company
Wants to modify or commence contract
Contact Project Manager
Subsea 7: Onshore Personnel
A Project Manager
Client wants to modify or commence contract
Deal with client request
Update project and vessel requirements
Inform Vessel Management Group
Vessel Management Group
Start of planning week
Send latest Ship Schedule
No
Yes
New information from Project Manager?
New information from Project Manager
A Planner
Update schedule as per 'default'
Find suitable revision of crew schedule
Start of planning week
Ship Schedule arrives
No
Yes
Changes needed to 'default' scheduling?
Identify suitable changes
How near are tasks being changed?
Regardless of time
If within one month
Seek agreement with crew and vessels
Receive feedback
Confirm with Crewing Group
Receive feedback
A member of the Crewing Group
Deal with feedback and chan
Request for feedback
Give feedback to changes
Start of planning week
If feedback needed on changes
Note - two sets of tasks to carry out
Always
Subsea 7: Offshore Personnel
An Offshore Manager
Request for feedback
Give feedback to changes
Further feedback required?
No
Yes
Start of planning week
If crew schedule regarding this vessel changes during this week
Note - two possible sets of tasks
If vessel requires transport arranged / changed during week
A Crewmember
Different from default schedule?
Yes
No
Request for feedback
Give feedback to changes
Note - three possible sets of tasks
Start of planning week
If employee's schedule is updated (in any way) during week
If employee's availability changes during week
If transport arranged / changed during week
Travel Agent

Figure A.1: Full detail of Business Process Map showing crew scheduling process in context - Part 1 of 3

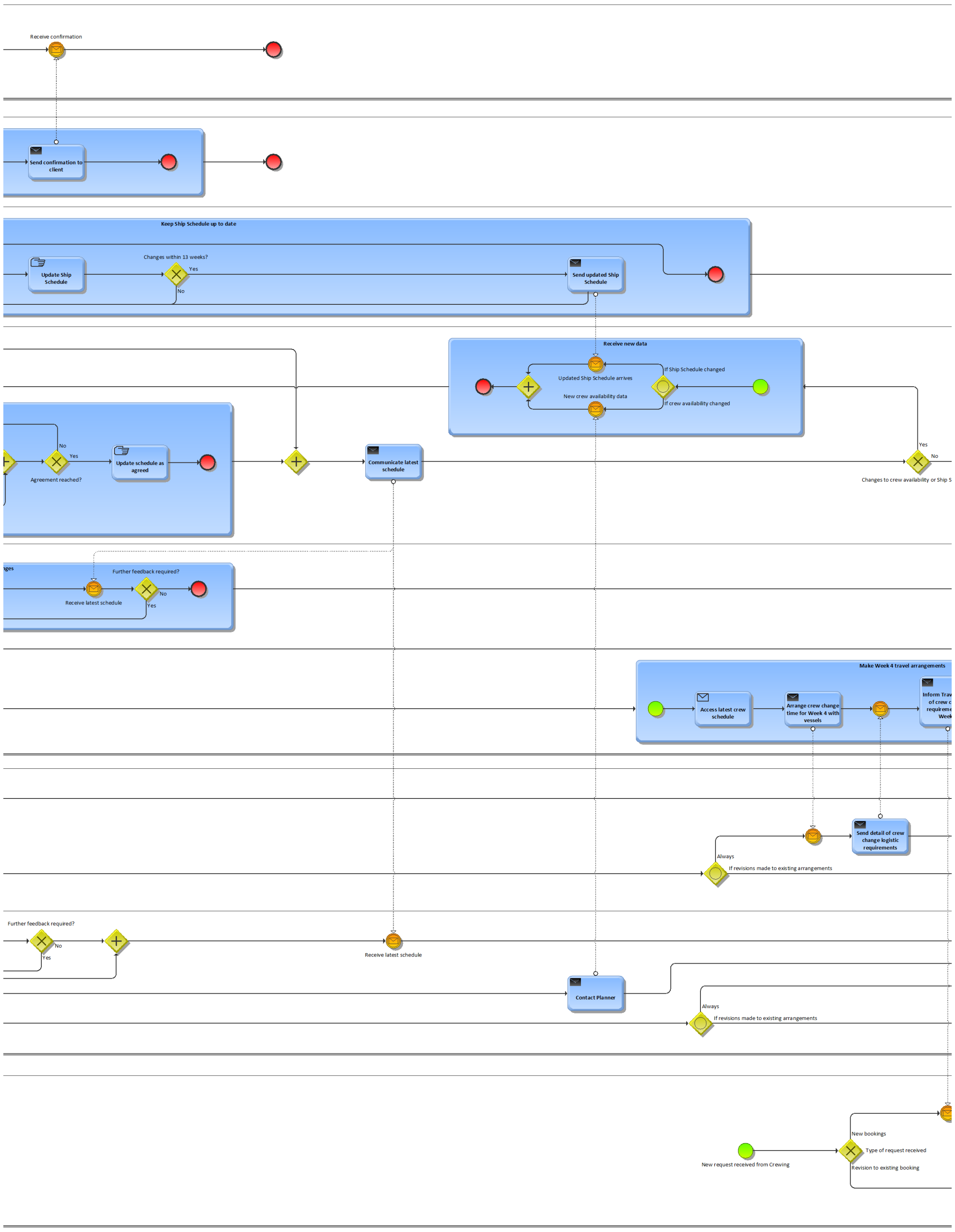

Receive confirmation
Send confirmation to client
Keep Ship Schedule up to date
Update Ship Schedule
Changes within 13 weeks?
Yes
No
Send updated Ship Schedule
Receive new data
If Ship Schedule changed
Updated Ship Schedule arrives
New crew availability data
If crew availability changed
No
Yes
Agreement reached?
Update schedule as agreed
Communicate latest schedule
Yes
No
Changes to crew availability or Ship S
nges
Further feedback required?
Receive latest schedule
No
Yes
Make Week 4 travel arrangements
Access latest crew schedule
Arrange crew change time for Week 4 with vessels
Inform Trav of crew c requireme Week
Send detail of crew change logistic requirements
Always
If revisions made to existing arrangements
Further feedback required?
No
Yes
Receive latest schedule
Contact Planner
Always
If revisions made to existing arrangements
New bookings
Type of request received
Revision to existing booking
New request received from Crewing

Figure A.2: Full detail of Business Process Map showing crew scheduling process in context - Part 2 of 3

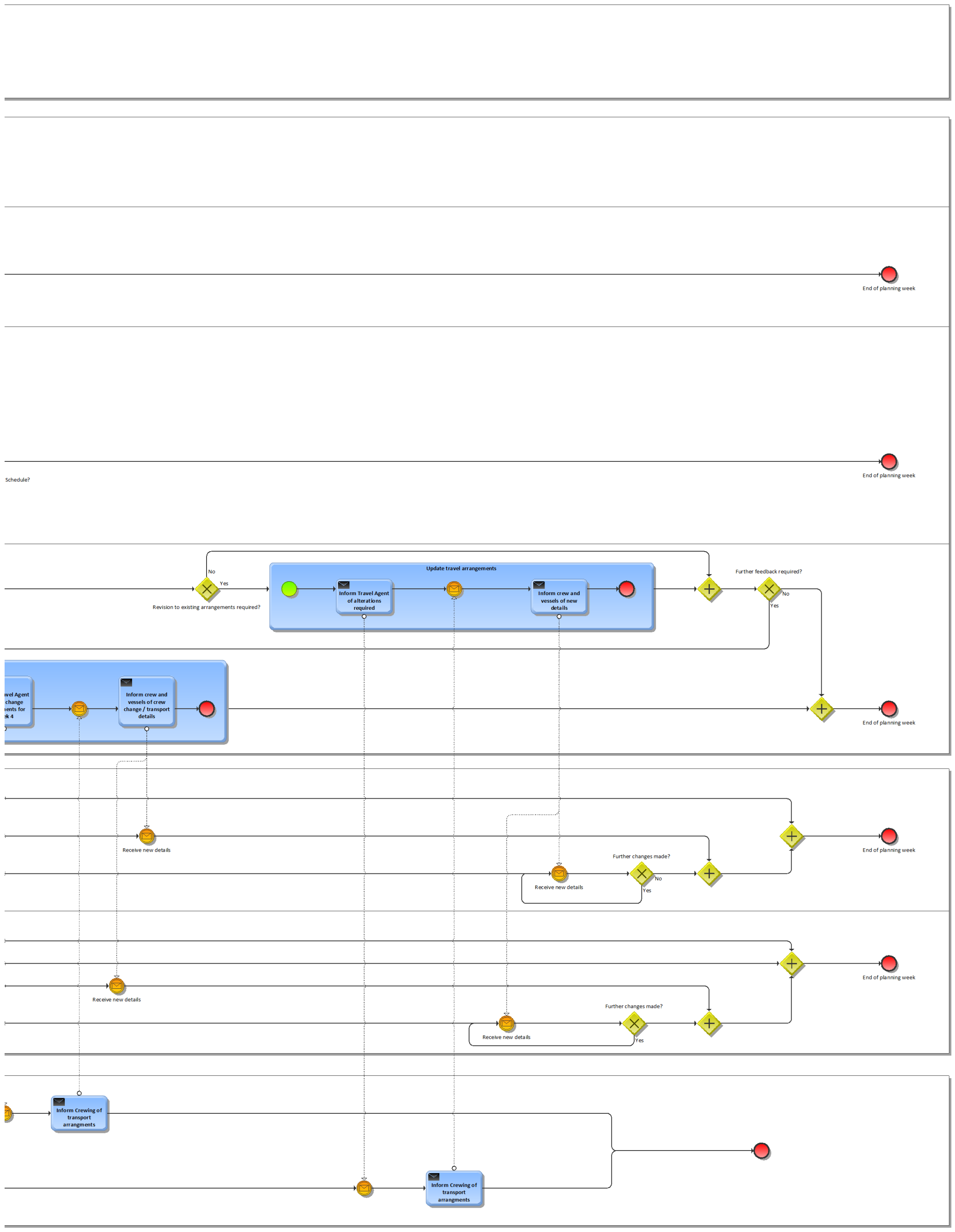

End of planning week
Schedule?
End of planning week
Update travel arrangements
No
Yes
Revision to existing arrangements required?
Inform Travel Agent of alterations required
Inform crew and vessels of new details
Further feedback required?
No
Yes
vel Agent
change
ents for
k 4
Inform crew and vessels of crew change / transport details
End of planning week
Receive new details
End of planning week
Further changes made?
Receive new details
No
Yes
End of planning week
Receive new details
Further changes made?
Receive new details
Yes
Inform Crewing of transport arrangments
Inform Crewing of transport arrangments

Figure A.3: Full detail of Business Process Map showing crew scheduling process in context - Part 3 of 3

Appendix B

Additional histograms for Task-Based results

Here we present histograms used to illustrate the discussion in 5.5.2 regarding which parameters varied during data generation had an impact on the results. This is divided into two sections - section B.1 shows the summary histograms, while section B.2 shows a selection of histograms breaking down the data by parameter values.

B.1 Summary Histograms

Here we present the histograms discussed in section 5.5.2 with reference to determining whether the assumptions of the data following a normal distribution hold. As described in the section 5.5.2.3 on the effect of parameter values, there are 19 outputs plotted, although note that for five of these the values less than or equal to 5% can be aggregated to give an adjusted output to be examined. Plots of these adjusted results are not given here.

B.1.1 Cost-minimization results

The histograms in this section represent the relevant results for the cost-minimization approach. This comprises:

- Running time, for both the two-minute (Figure B.1) and ten-minute (Figure B.2) time limits.
- Number of changes in the solution for the two-minute settings only (Figure B.3).
- The gaps proven in the test runs, for both the two-minute (Figure B.4) and ten-minute (Figure B.5) settings.

- The gap to the best known bound for each instance, for both the two-minute (Figure B.6) and ten-minute (Figure B.7) settings.

Note that of these, only the number of changes in the two-minute cost-minimization setting (Figure B.3) can be considered to follow an approximately normal distribution.

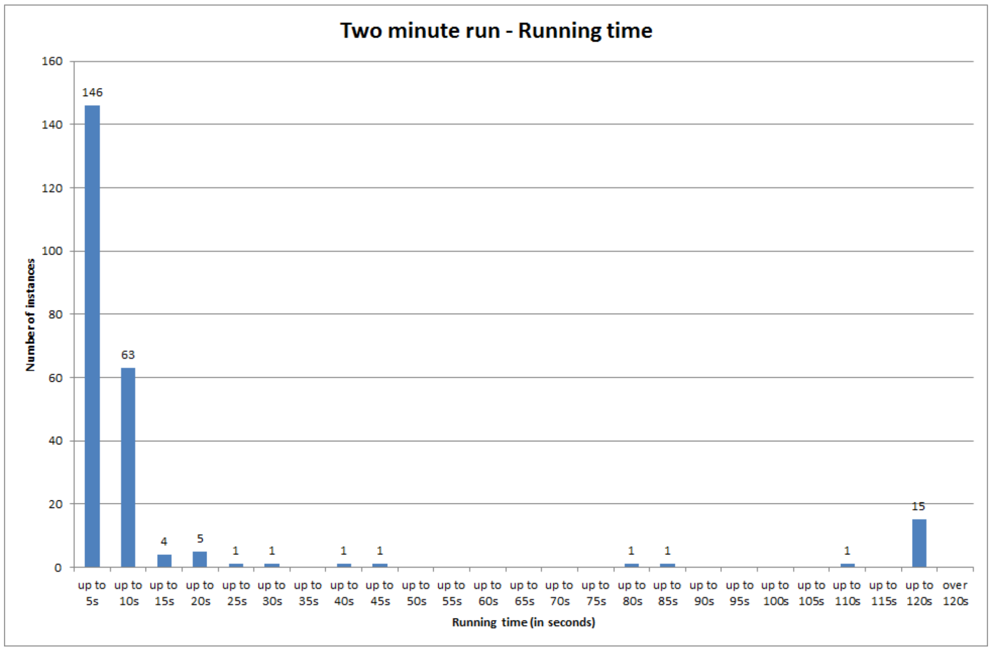


Figure B.1: Running time for the cost-minimization approach with two-minute time limit. Normality assumption not considered to hold here.

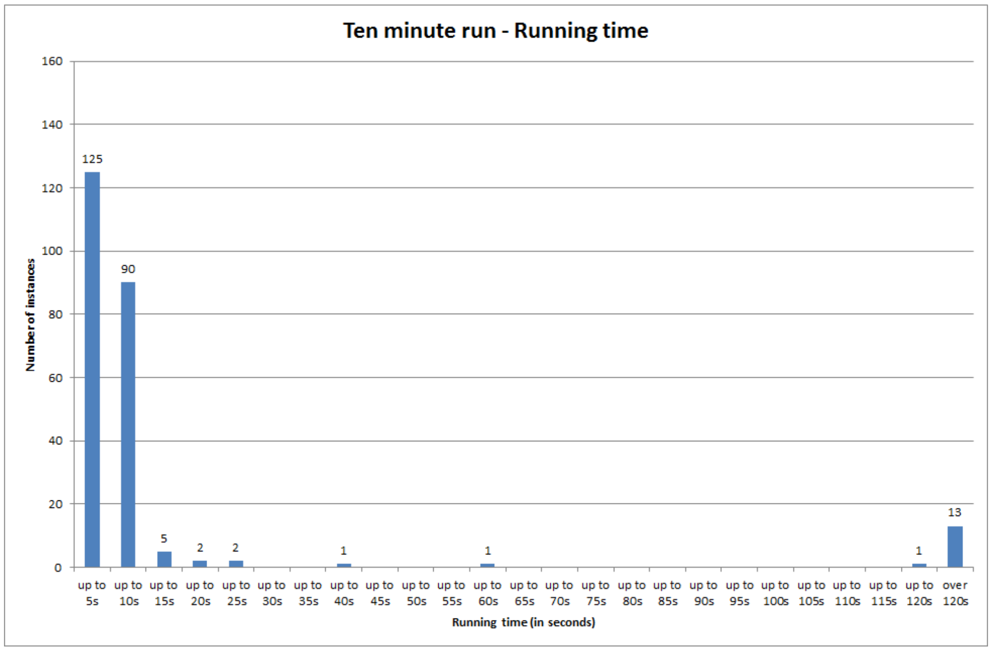


Figure B.2: Running time for the cost-minimization approach with ten-minute time limit. Normality assumption not considered to hold here.

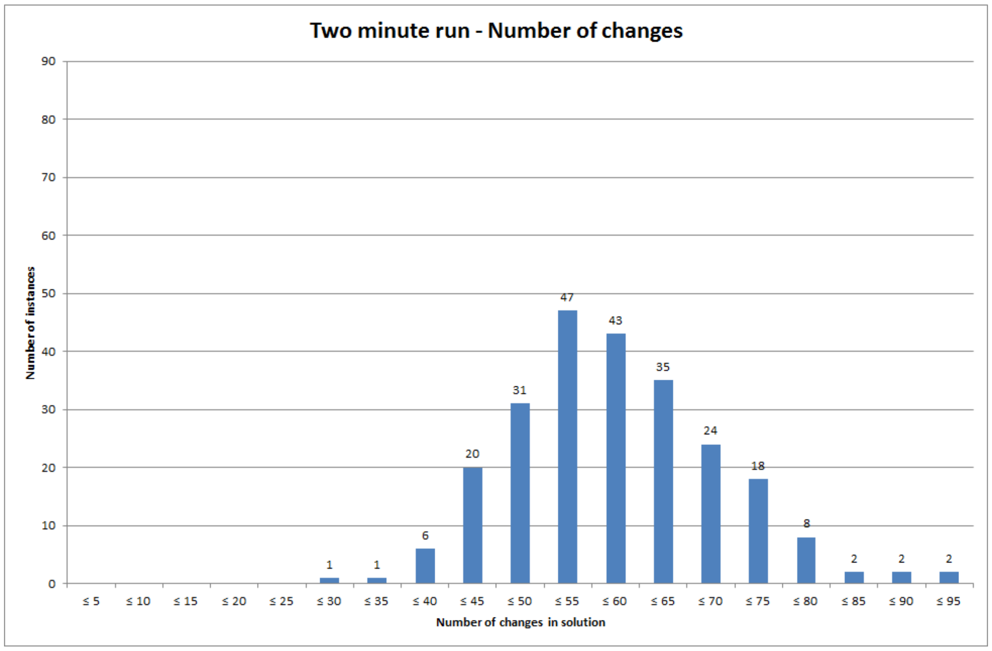


Figure B.3: Number of changes in the solution for the cost-minimization approach with two-minute time limit. Follows an approximately normal distribution.

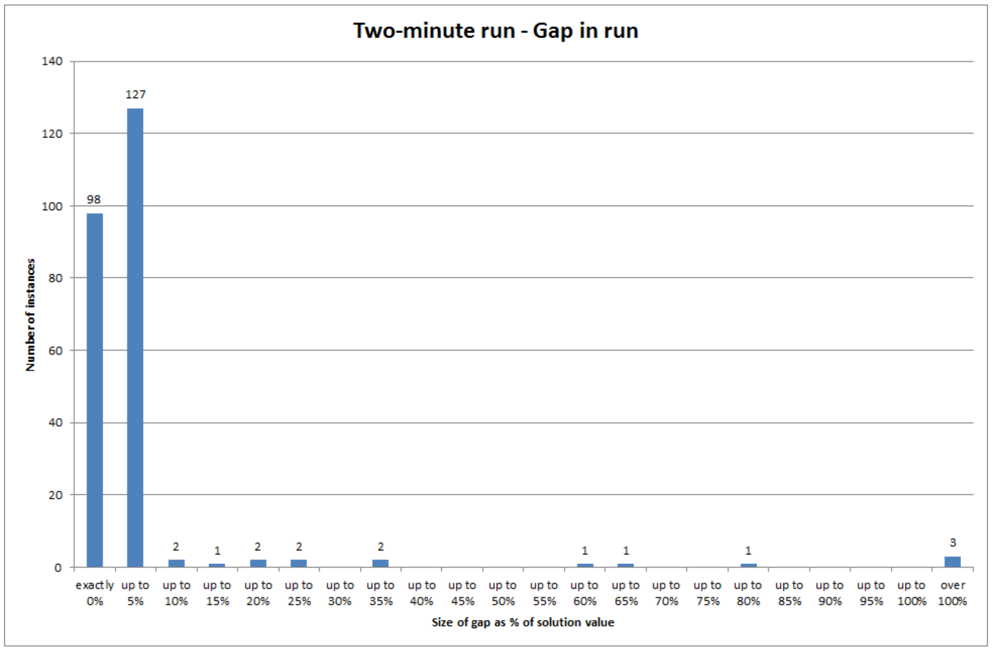


Figure B.4: Proven cost gap (i.e. to bound found during run) for the two-minute cost-minimization run. Normality assumption not considered to hold here.

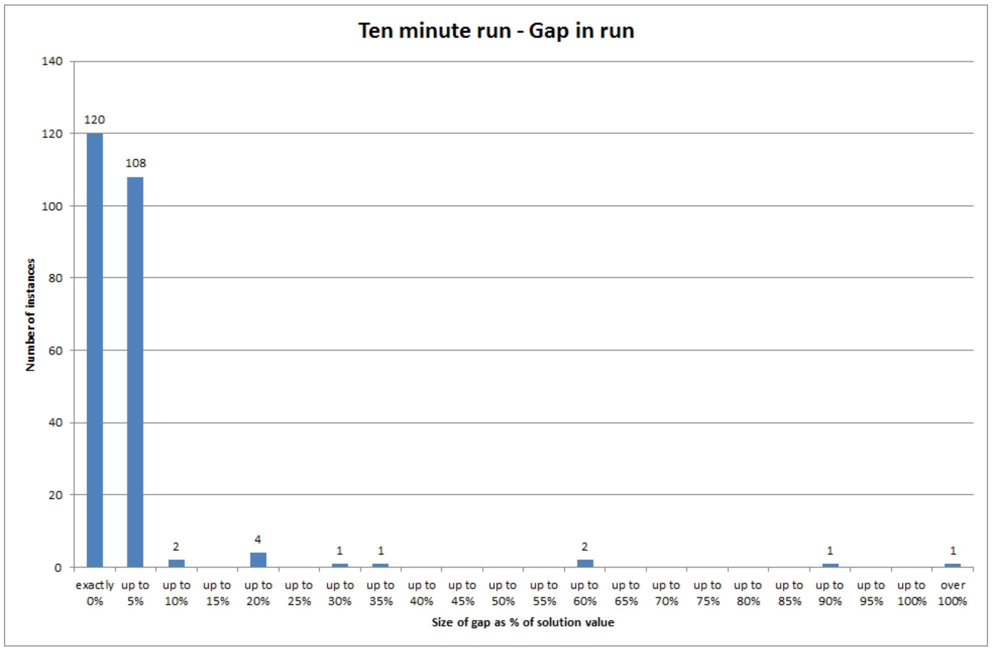


Figure B.5: Proven cost gap (i.e. to bound found during run) for the ten-minute cost-minimization run. Normality assumption not considered to hold here.

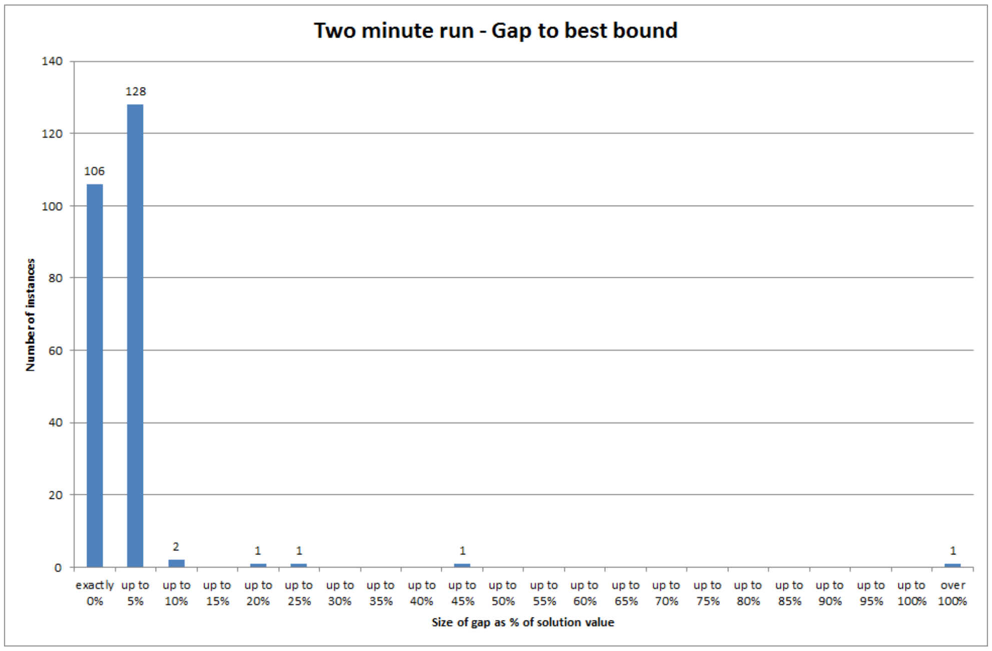


Figure B.6: Gap to best known cost bound for the two-minute cost-minimization run. Normality assumption not considered to hold here.

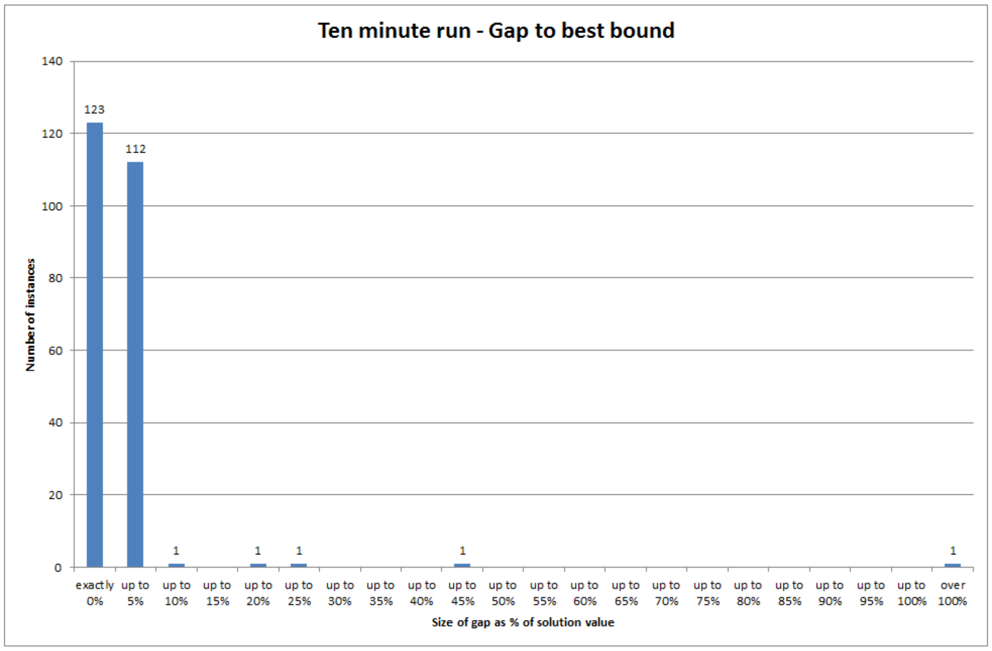


Figure B.7: Gap to best known cost bound for the two-minute cost-minimization run. Normality assumption not considered to hold here.

B.1.2 Change-minimization results

The histograms in this section represent the relevant results for the change-minimization approach. This comprises:

- Running time (Figure B.8).
- Number of iterations (Figure B.9).
- Number of changes in the minimal-change solution (Figure B.10) and in the lowest cost solution (Figure B.11).
- Percentage gap to the best known bound for the costs of the following:
  - The first (i.e. non-cost-constrained) solution (Figure B.12);
  - The cheapest minimal-change solution (Figure B.13);
  - The lowest cost solution (Figure B.14).

Note that the majority of these appear to follow an approximately normal distribution, with the exceptions of running time (Figure B.8) and the cost gap for the lowest cost solution (Figure B.14) for which the normality assumption does not appear to hold.

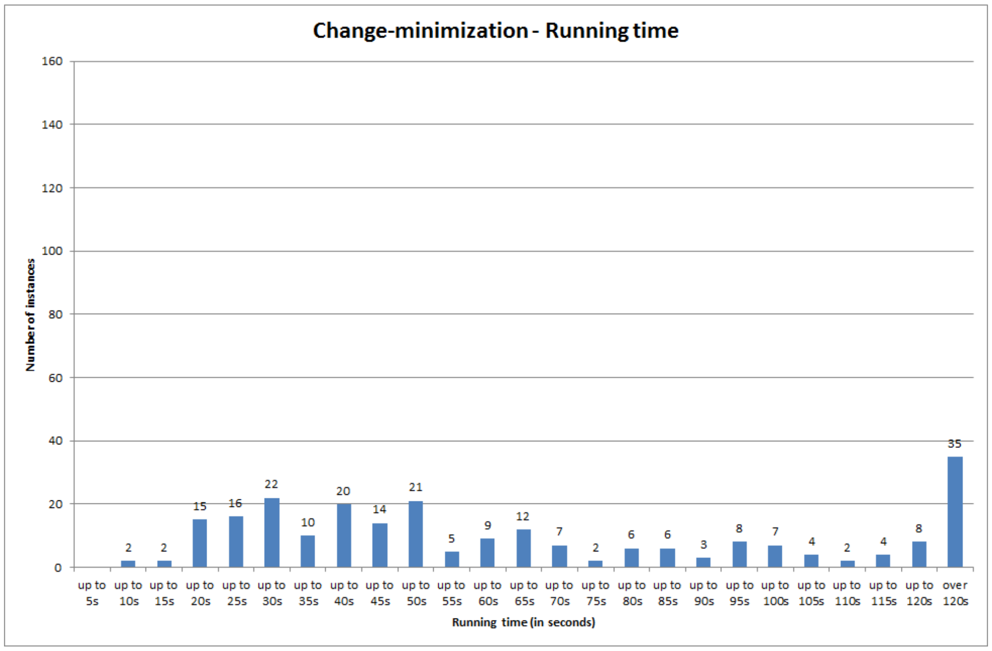


Figure B.8: Running time for the change-minimization algorithm (with two-minute time limit). Normality assumption not considered to hold here.

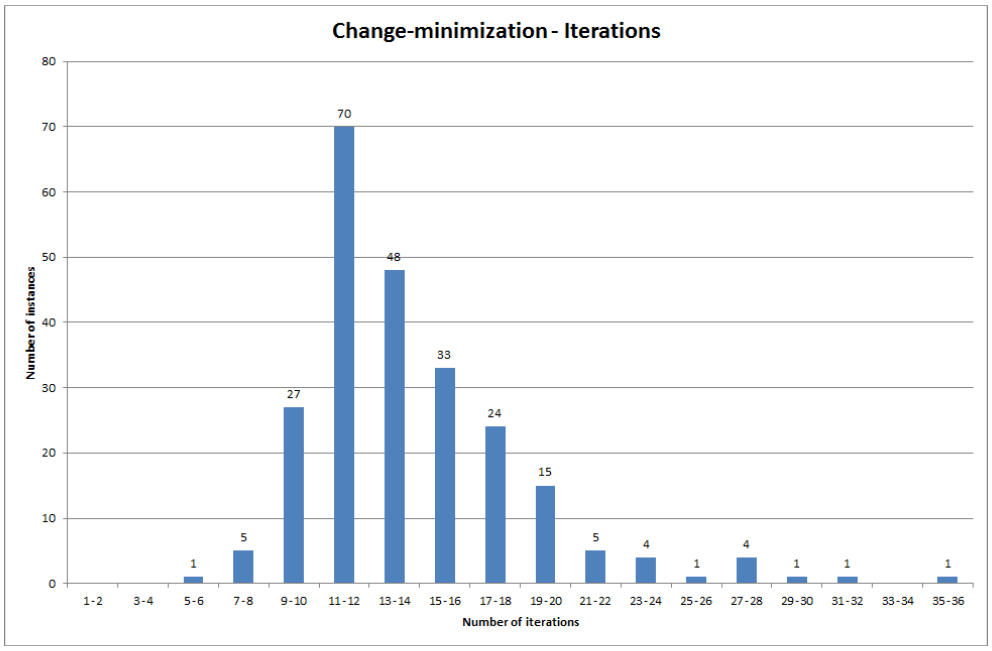


Figure B.9: Number of iterations carried out by change-minimization algorithm. Follows an approximately normal distribution.

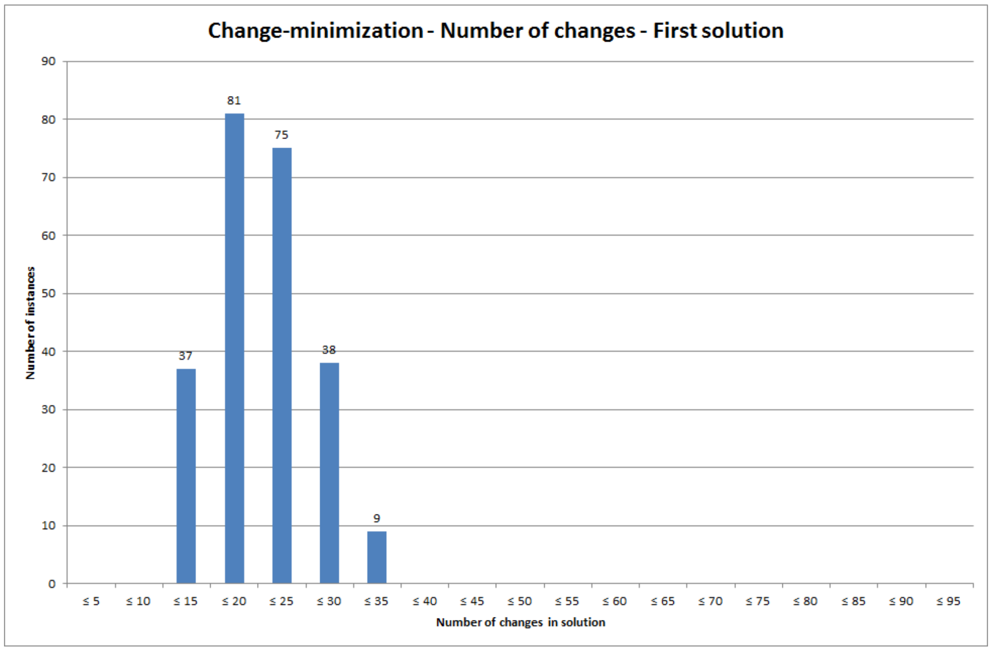


Figure B.10: Minimal number of changes as found in the (first) change-minimization solution. Follows an approximately normal distribution.

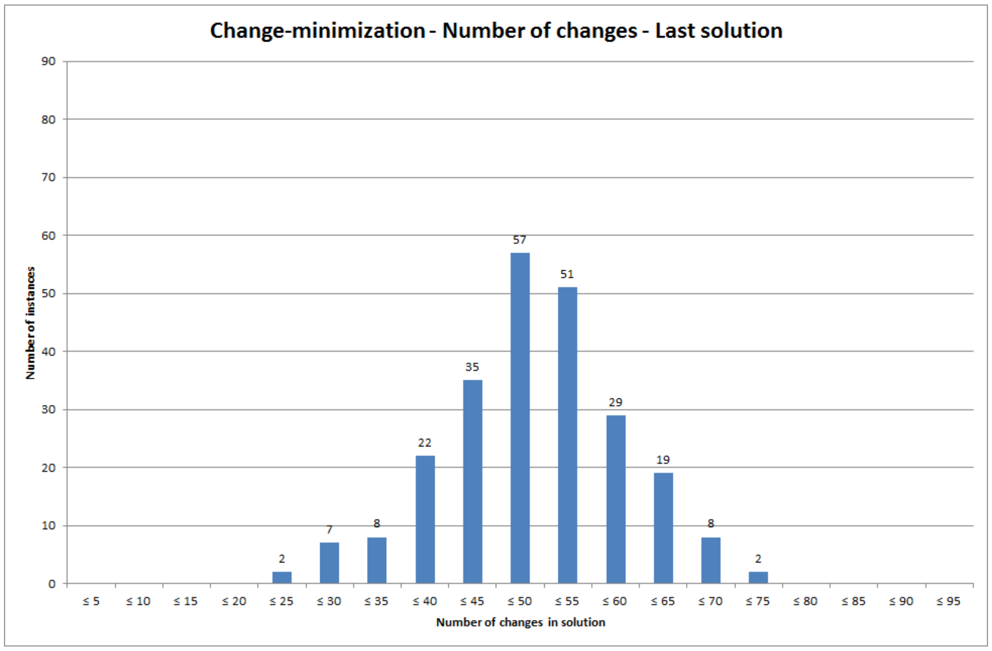


Figure B.11: Number of changes in the lowest cost change-minimization solution. Follows an approximately normal distribution.

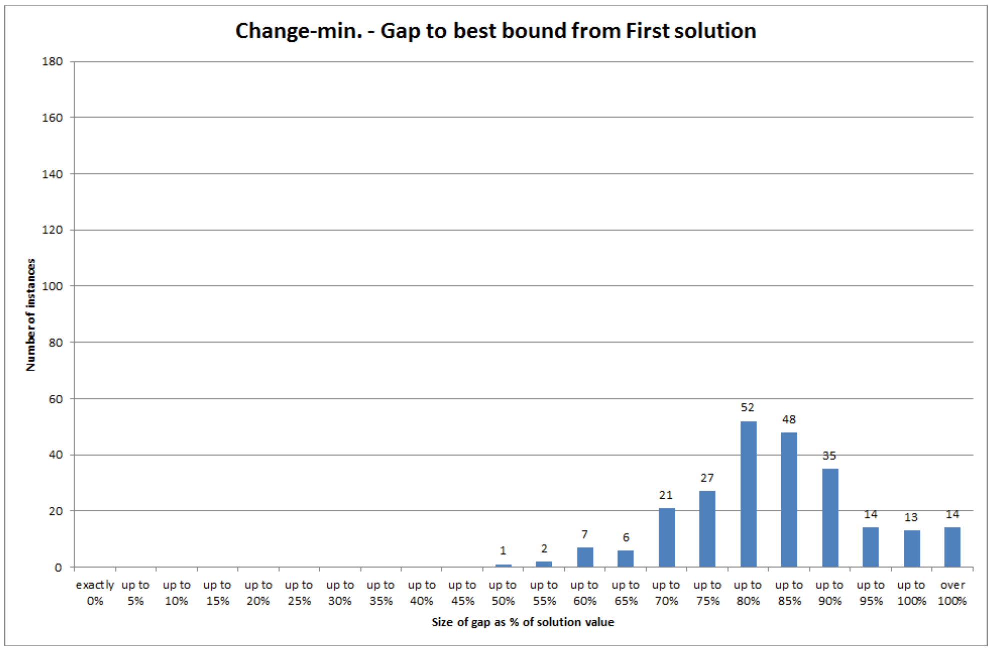


Figure B.12: Gap to best known cost bound for non-cost-constrained change-minimization solution. Follows an approximately normal distribution.

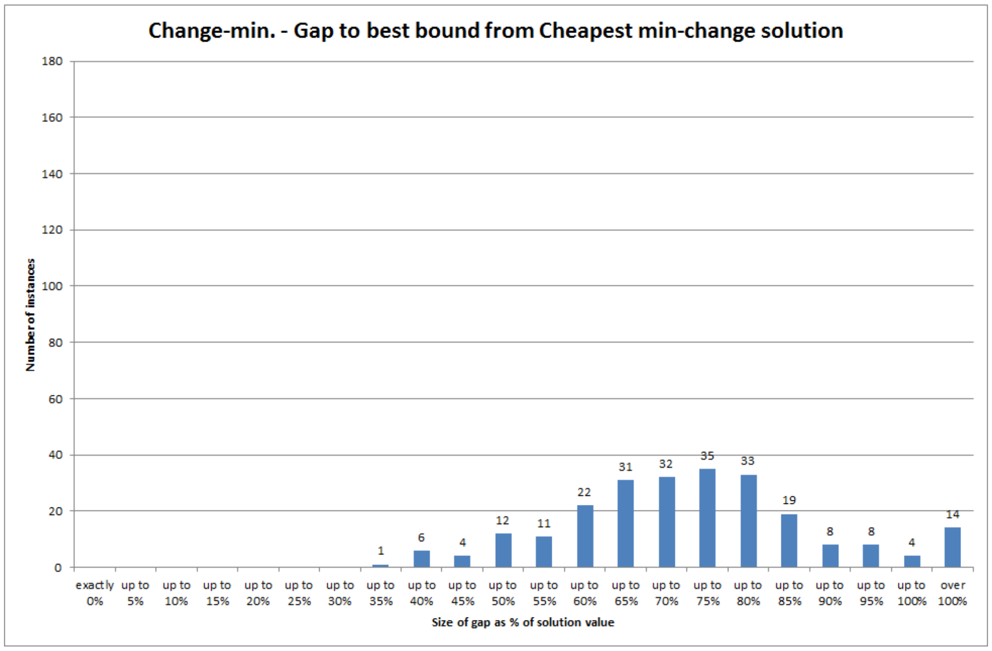


Figure B.13: Gap to best known cost bound for the cheapest minimal-change change-minimization solution. Follows an approximately normal distribution.

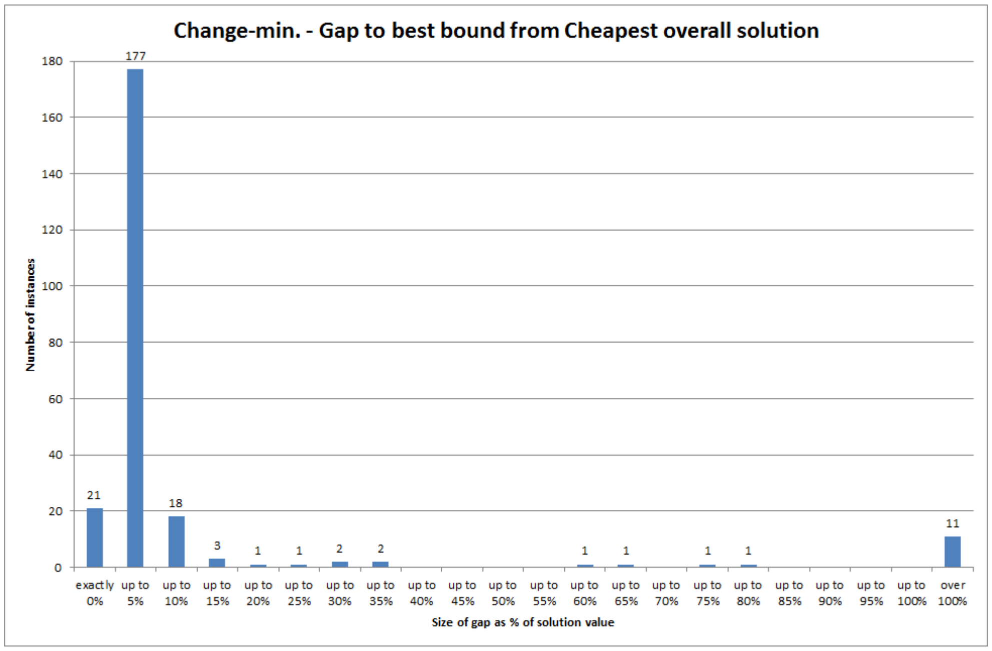


Figure B.14: Gap to best known cost bound for the lowest cost change-minimization solution. Normality assumption not considered to hold here.

B.1.3 Results comparisons

The histograms in this section represent the comparisons between the change-minimization and the two-minute cost-minimization results. This comprises:

- The percentage increase in the number of changes for both the minimal-change (Figure B.15) and lowest cost change-minimization (Figure B.16) solutions.
- The percentage increase in cost for the non-cost-constrained (Figure B.17), the cheapest minimal-change (Figure B.18) and the lowest cost change-minimization (Figure B.19) solutions.

Note that of these, only the comparison of changes with the minimal-change solution (Figure B.15) can be considered to follow an approximately normal distribution.

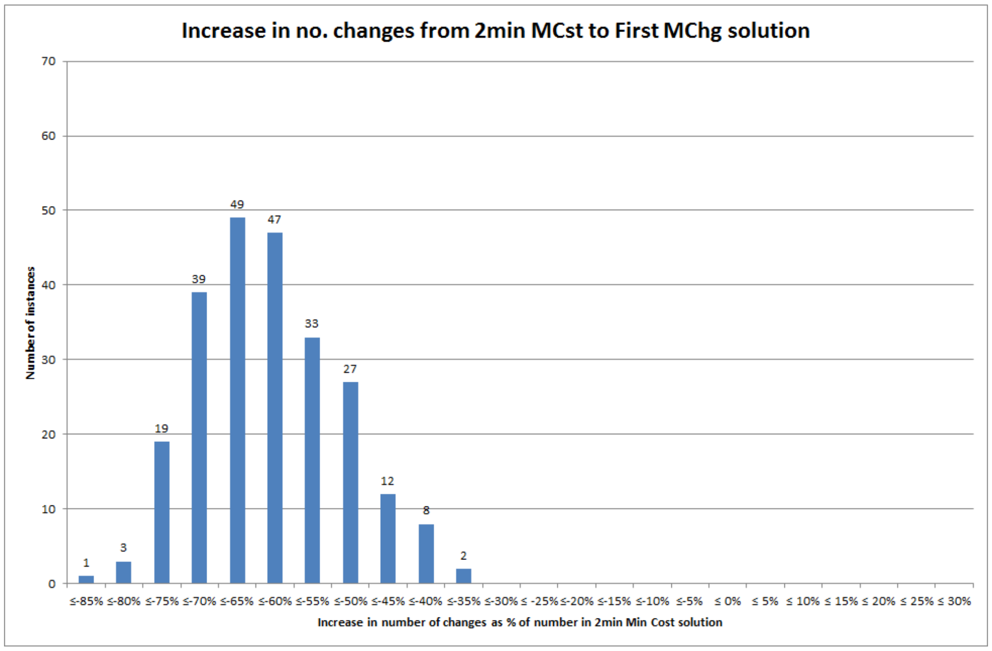


Figure B.15: Increase in number of changes for the minimal-change solution, with the two-minute cost-minimization solution as baseline for comparison. Follows an approximately normal distribution.

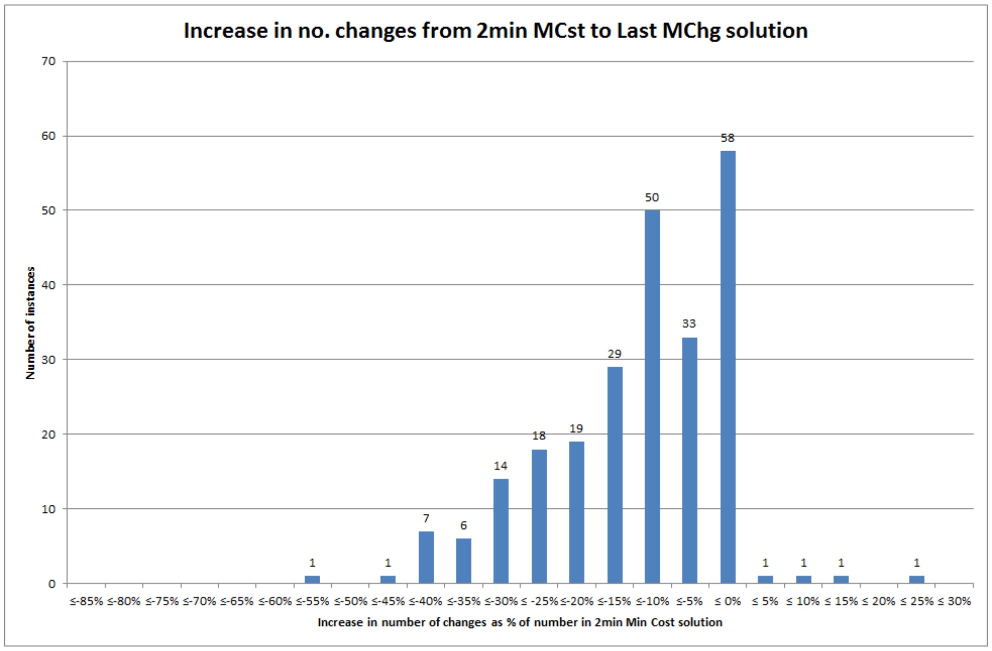


Figure B.16: Increase in number of changes for the lowest cost change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison. Normality assumption not considered to hold here.

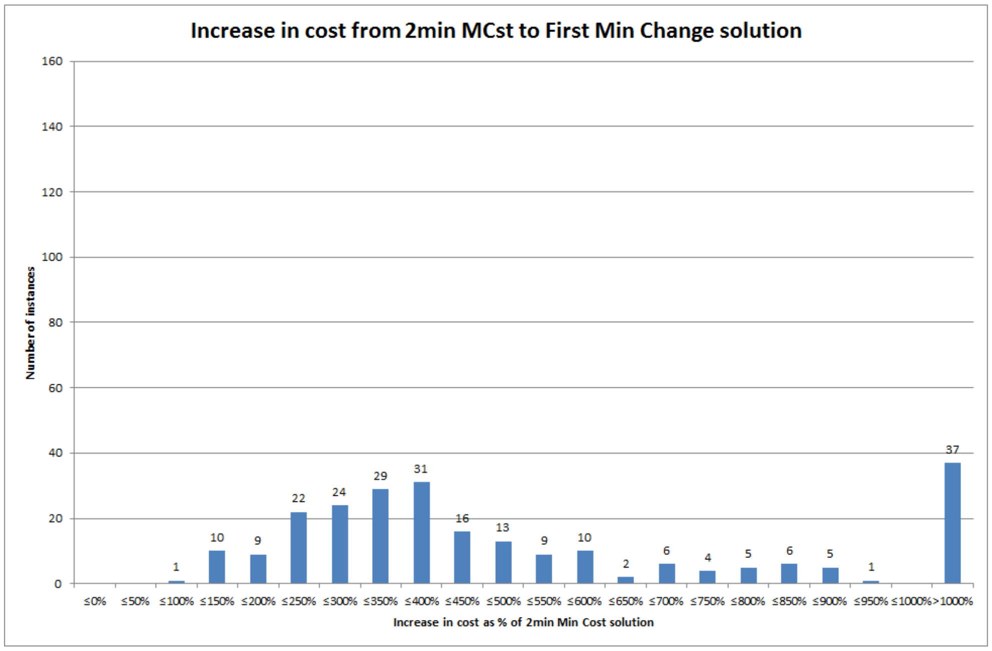


Figure B.17: Increase in cost for the non-cost-constrained change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison. Normality assumption not considered to hold here.

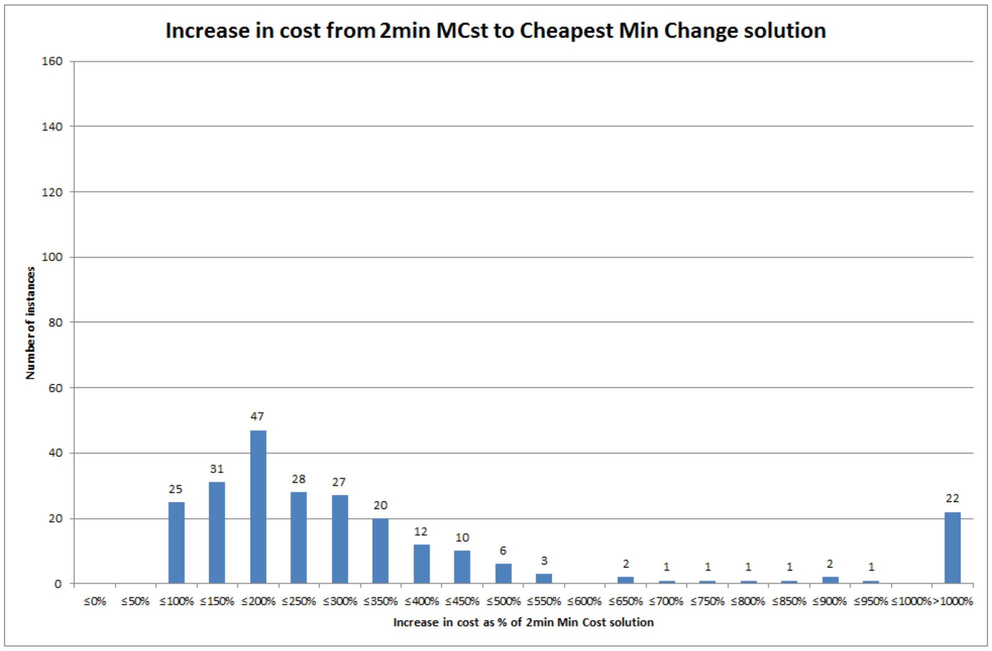


Figure B.18: Increase in cost for the cheapest minimal-change solution, with the two-minute cost-minimization solution as baseline for comparison. Normality assumption not considered to hold here.

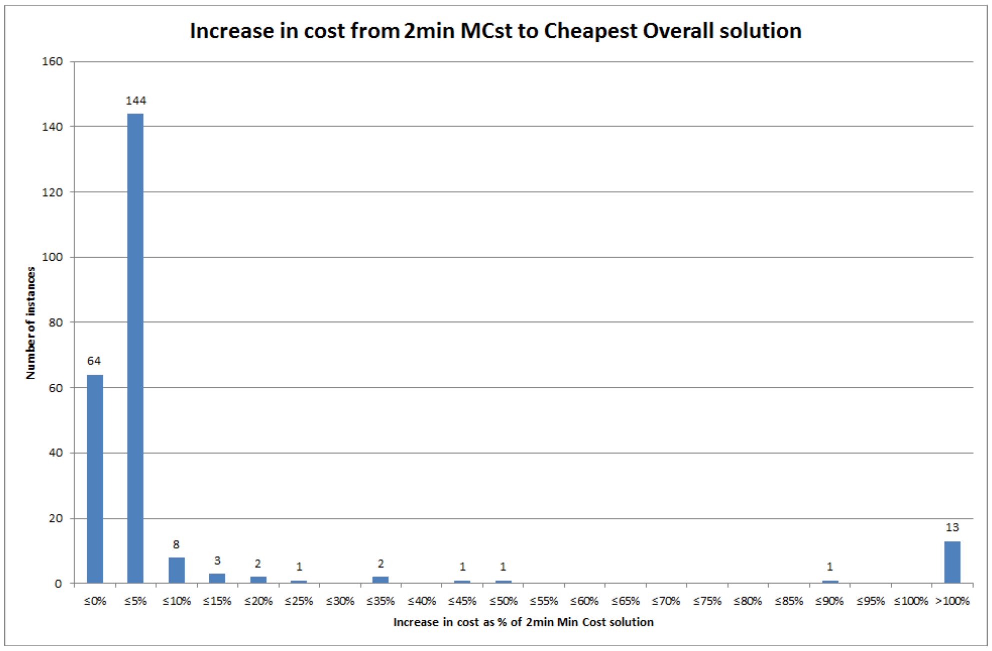


Figure B.19: Increase in cost for the lowest cost change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison. Normality assumption not considered to hold here.

B.2 Breakdown by parameters

Here we present selected histograms which can aid our understanding of the F-test and Kruskall-Wallis test results given in Tables 5.4 and 5.5. These show the same results as in section B.1 above, but broken down according to the values of the relevant parameter. This is further divided up into subsections, according to the graphs which relate to changes, cost gaps, cost comparisons, and to running time respectively.

B.2.1 Relating to changes

The following graphs relate to the breakdown of the comparison of number of changes between the two-minute cost-minimization solution and the lowest cost change-minimization solution.

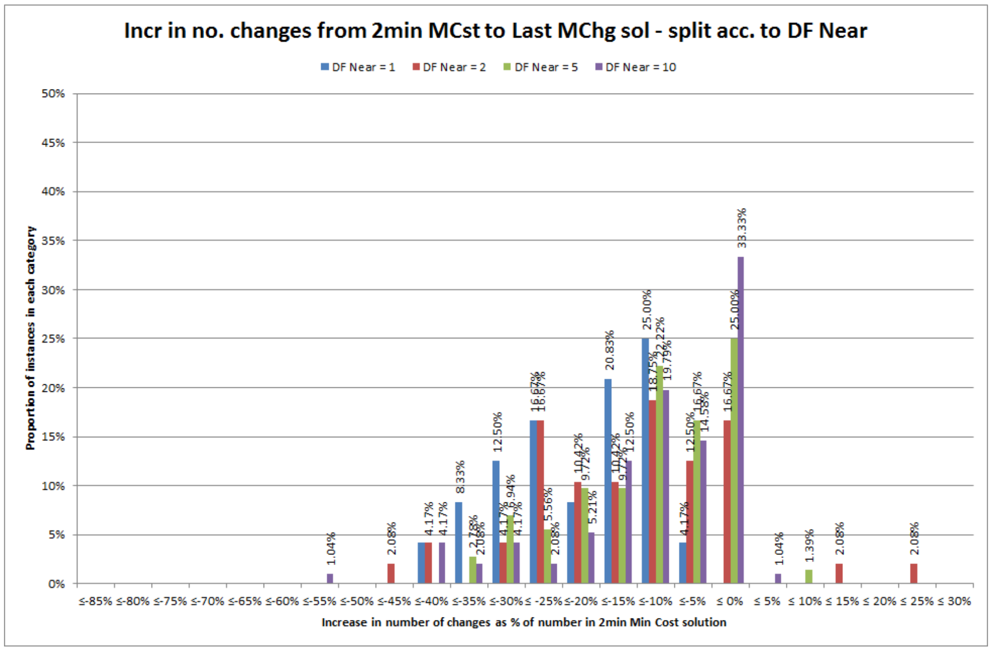


Figure B.20: Increase in number of changes for the lowest cost change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the near-term disruption penalty K_N.

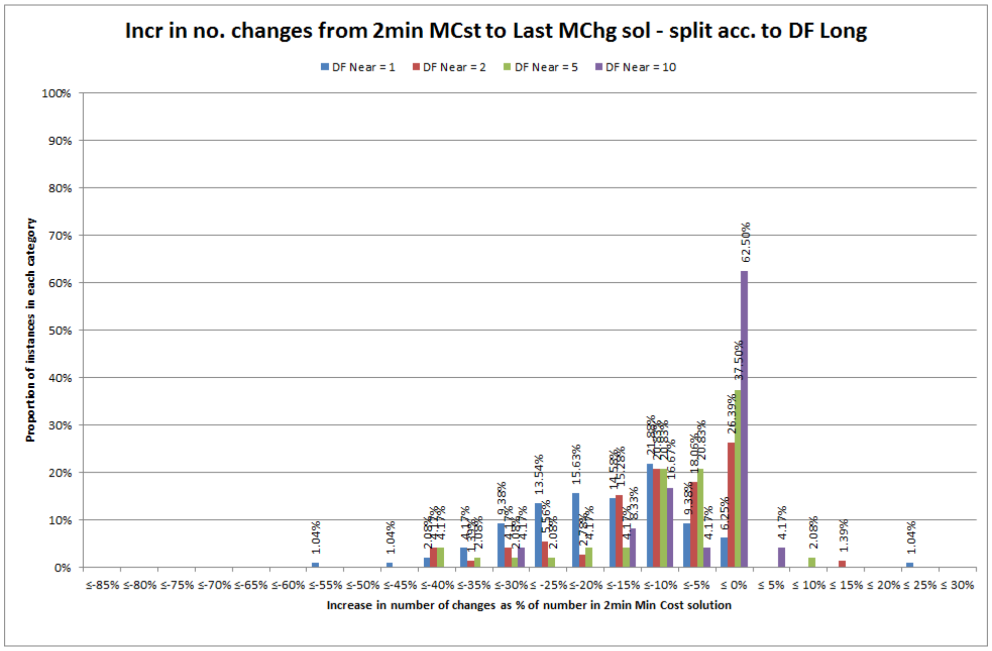


Figure B.21: Increase in number of changes for the lowest cost change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the long-term disruption penalty K_L.

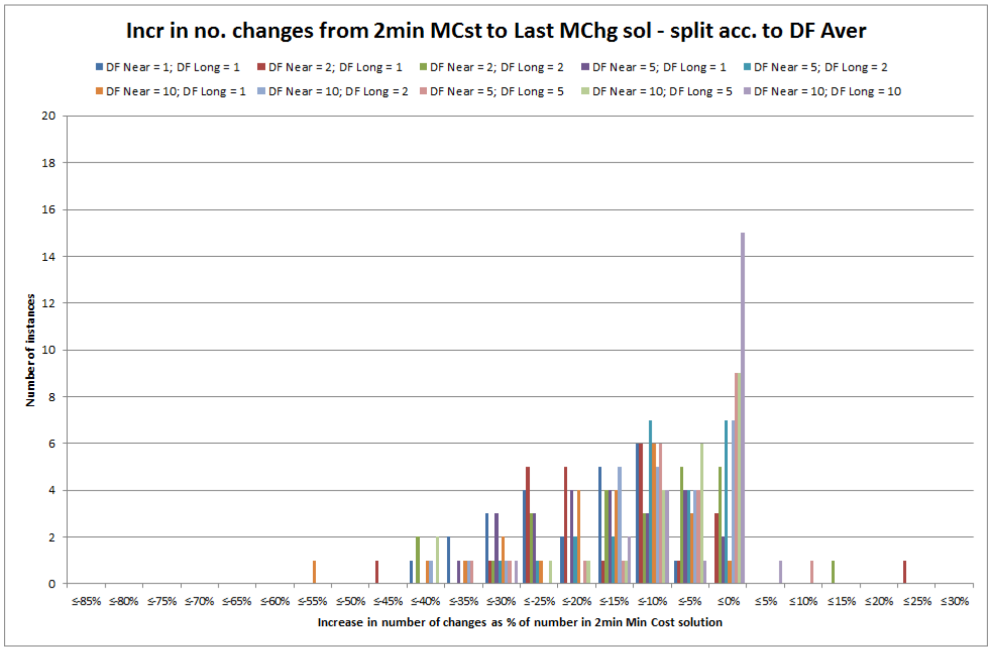


Figure B.22: Increase in number of changes for the lowest cost change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the weighted average disruption penalty $\hat{K}$.

B.2.2 Relating to cost gaps

The following graphs relate to the breakdown of the gaps to the best cost bound for the lowest cost change-minimization solution, and for both gaps (i.e. to best bound and proven during the run) for the two- and ten-minute cost-minimization approaches.

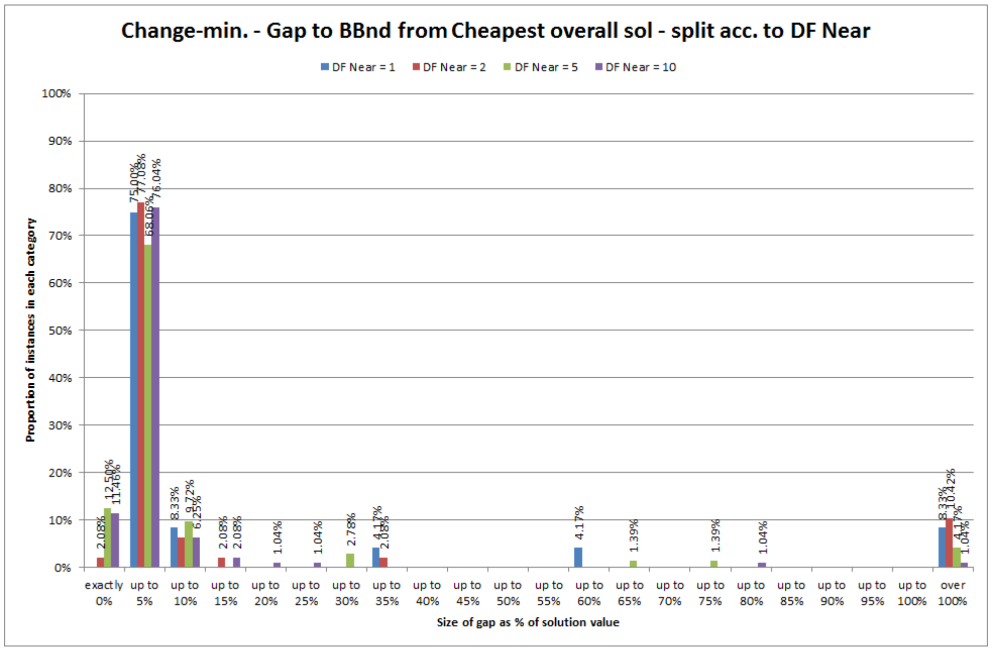


Figure B.23: Gap to best known cost bound for the lowest cost change-minimization solution, broken down by values of the near-term disruption penalty K_N.

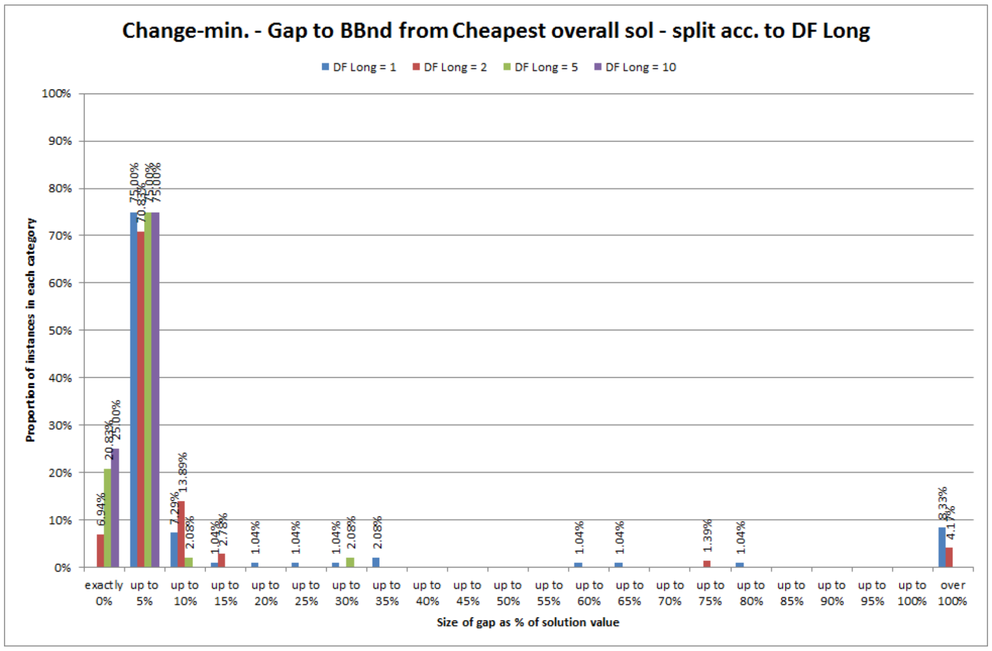


Figure B.24: Gap to best known cost bound for the lowest cost change-minimization solution, broken down by values of the long-term disruption penalty K_L.

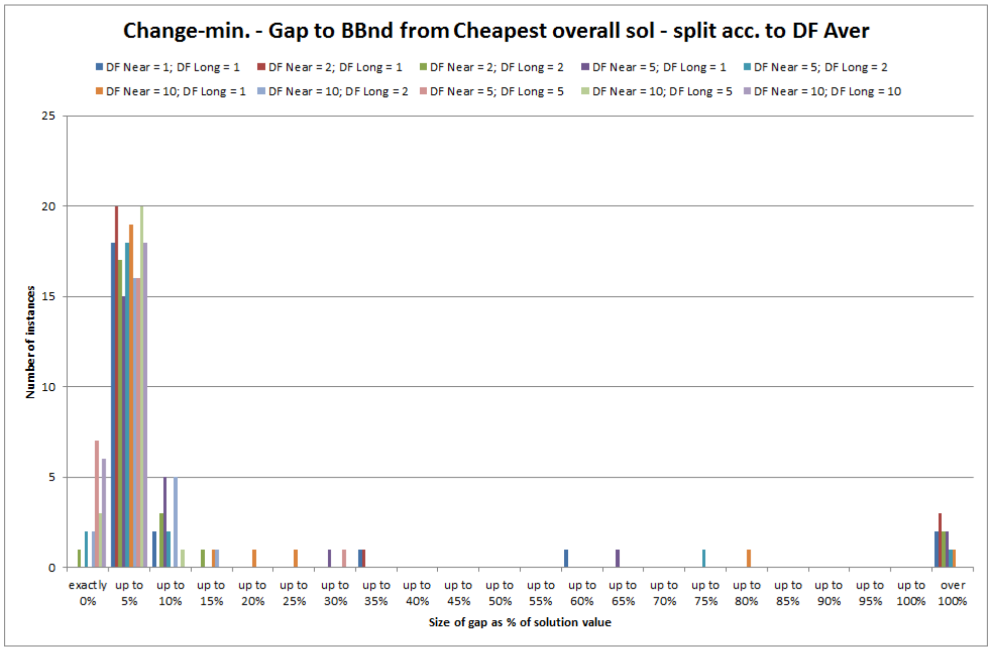


Figure B.25: Gap to best known cost bound for the lowest cost change-minimization solution, broken down by values of the weighted average disruption penalty $\hat{K}$.

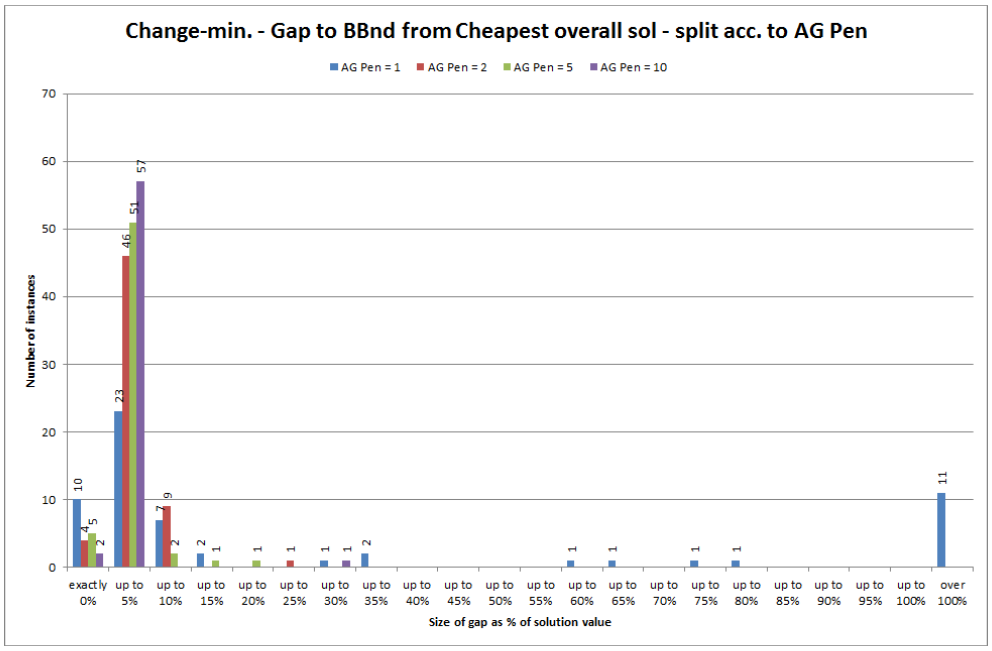


Figure B.26: Gap to best known cost bound for the lowest cost change-minimization solution, broken down by values of the agency penalty factor K_{AG}.

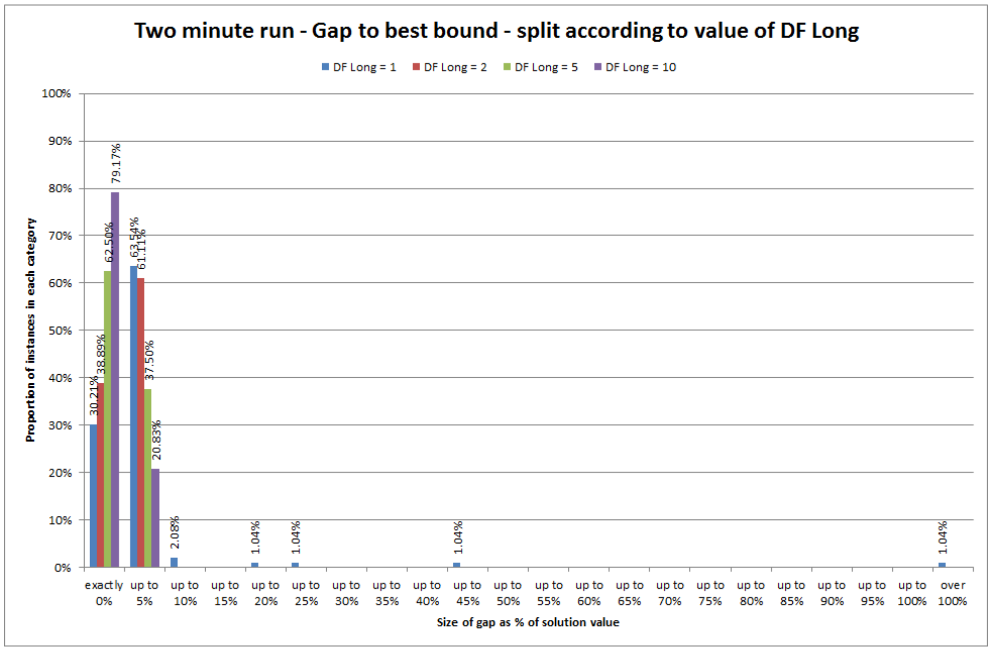


Figure B.27: Gap to best known cost bound for the two-minute cost-minimization run, broken down by values of the long-term disruption penalty K_L.

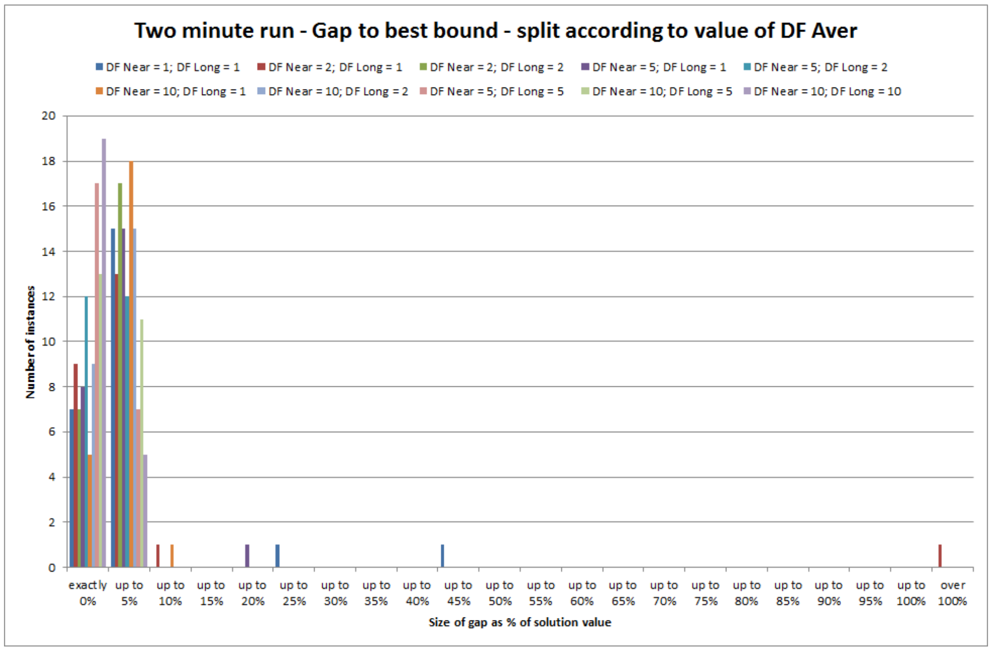


Figure B.28: Gap to best known cost bound for the two-minute cost-minimization run, broken down by values of the weighted average disruption penalty $\hat{K}$.

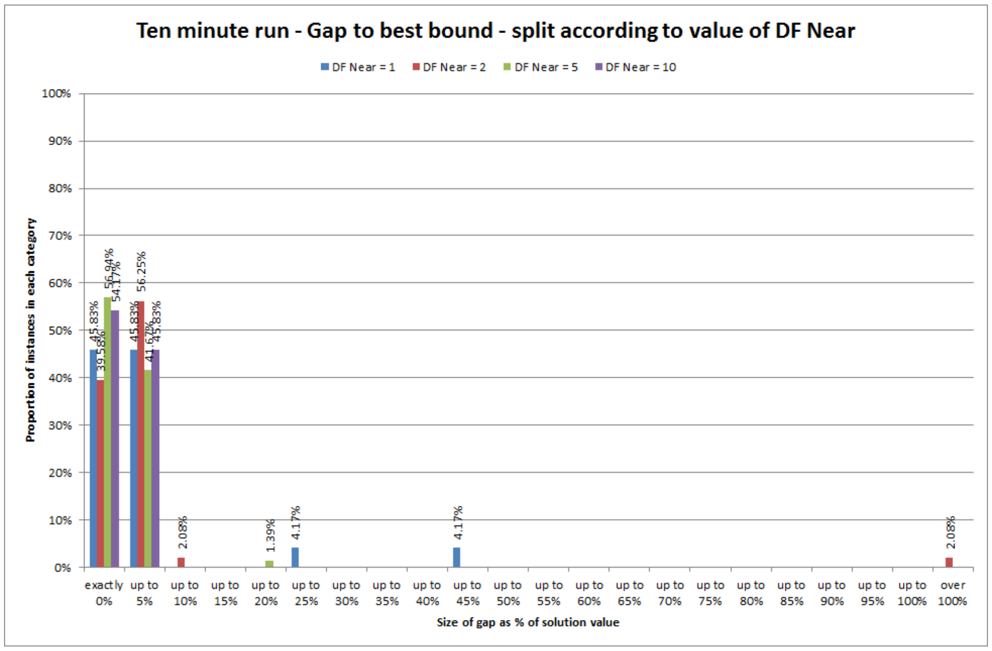


Figure B.29: Gap to best known cost bound for the ten-minute cost-minimization run, broken down by values of the near-term disruption penalty K_N.

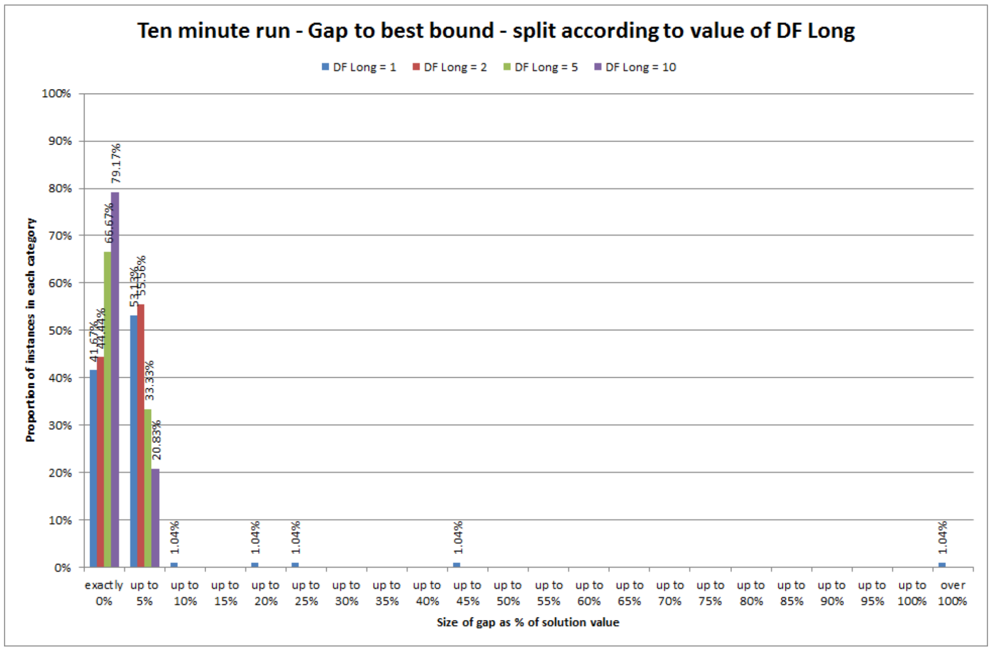


Figure B.30: Gap to best known cost bound for the ten-minute cost-minimization run, broken down by values of the long-term disruption penalty K_L.

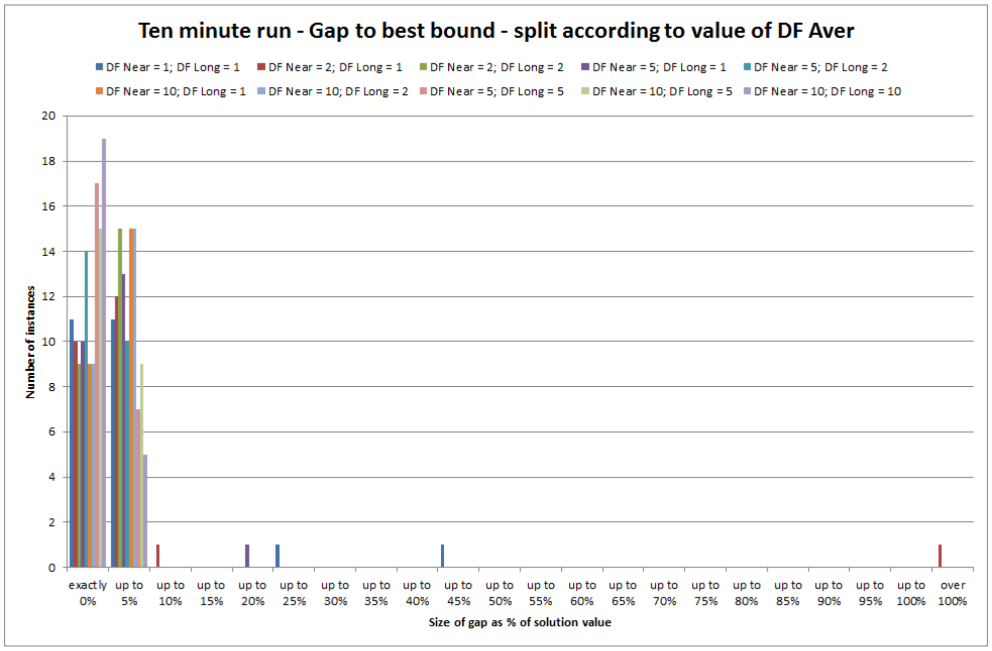


Figure B.31: Gap to best known cost bound for the ten-minute cost-minimization run, broken down by values of the weighted average disruption penalty $\hat{K}$.

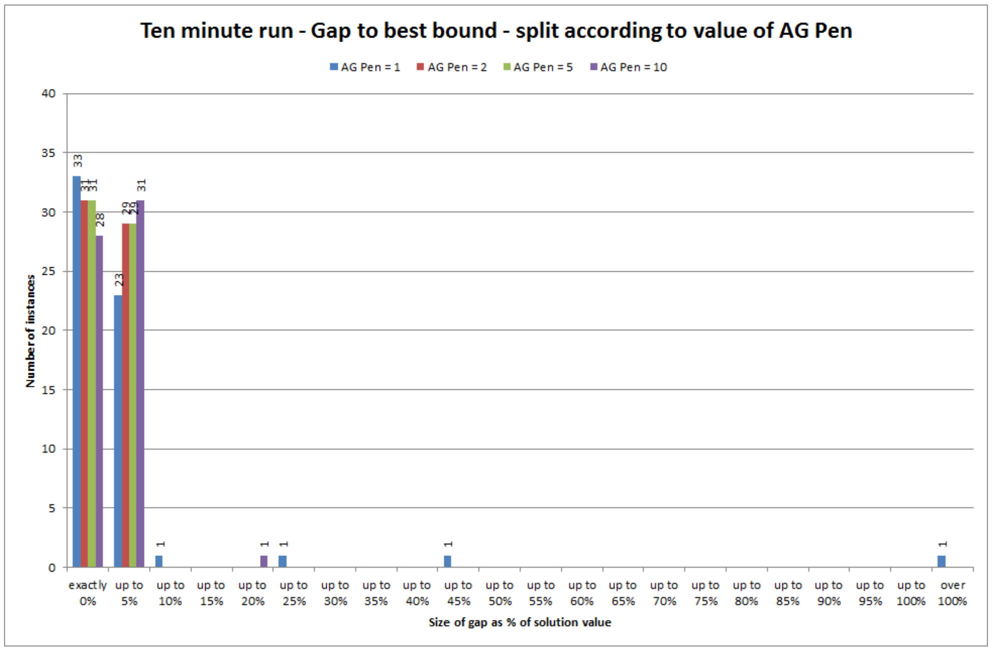


Figure B.32: Gap to best known cost bound for the ten-minute cost-minimization run, broken down by values of the agency penalty factor K_{AG}.

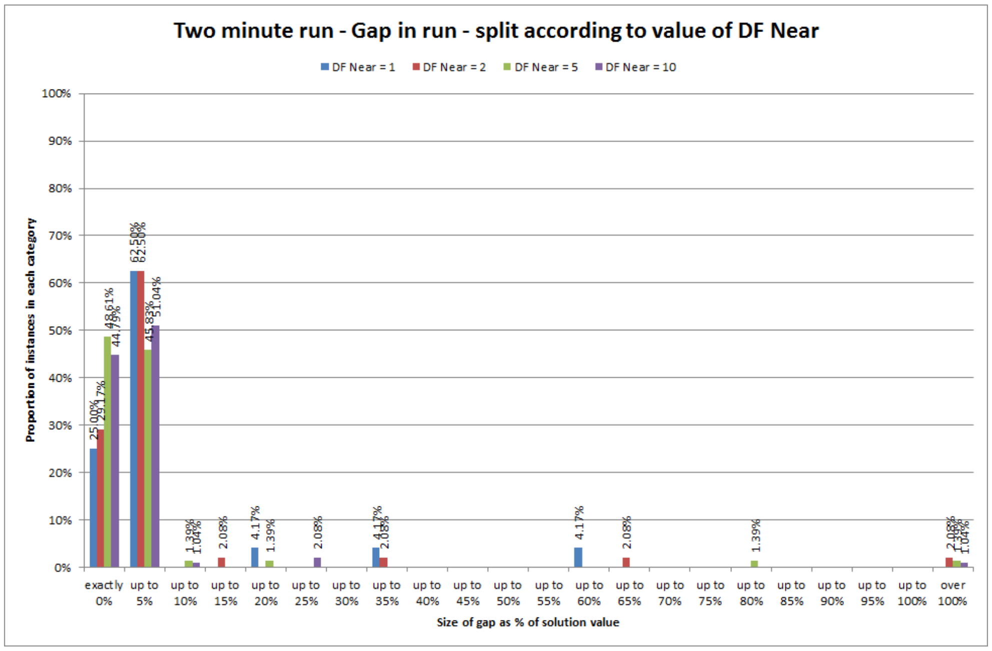


Figure B.33: Proven cost gap (i.e. to bound found during run) for the two-minute cost-minimization run, broken down by values of the near-term disruption penalty K_N.

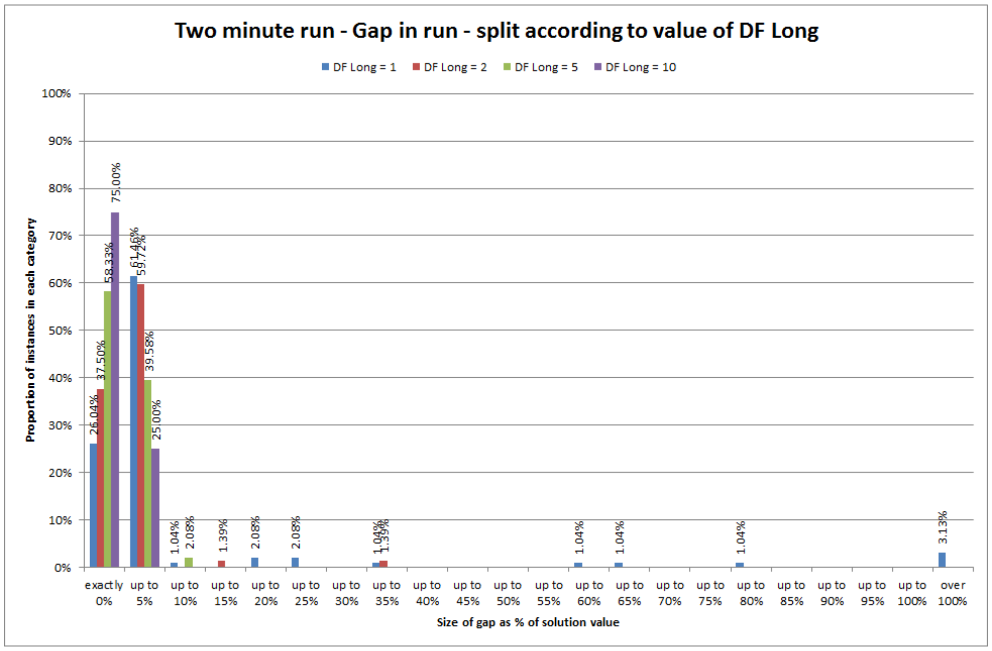


Figure B.34: Proven cost gap (i.e. to bound found during run) for the two-minute cost-minimization run, broken down by values of the long-term disruption penalty K_L.

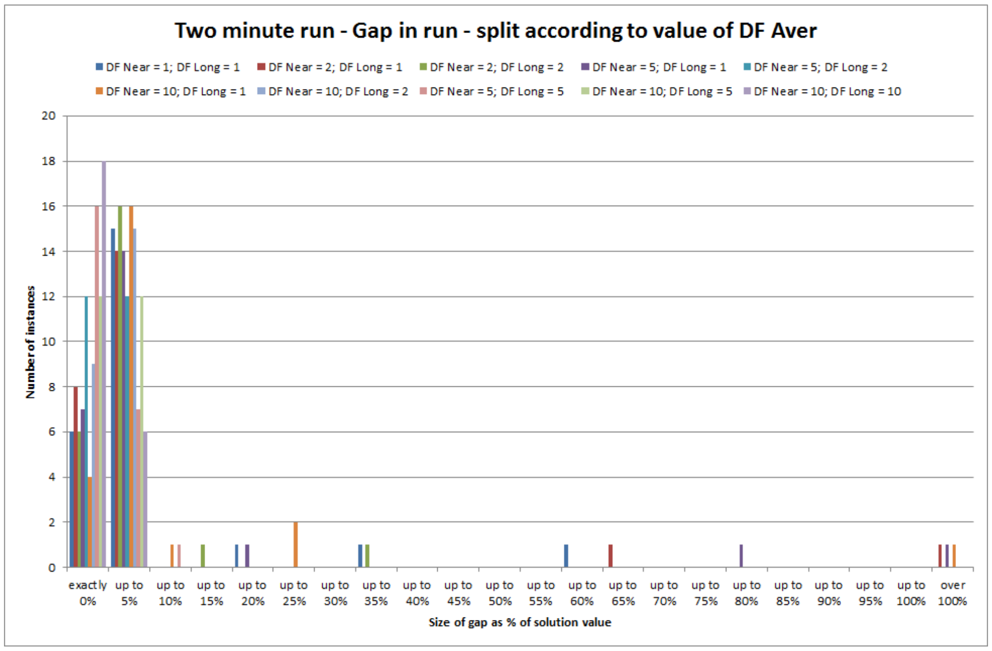


Figure B.35: Proven cost gap (i.e. to bound found during run) for the two-minute cost-minimization run, broken down by values of the weighted average disruption penalty $\hat{K}$.

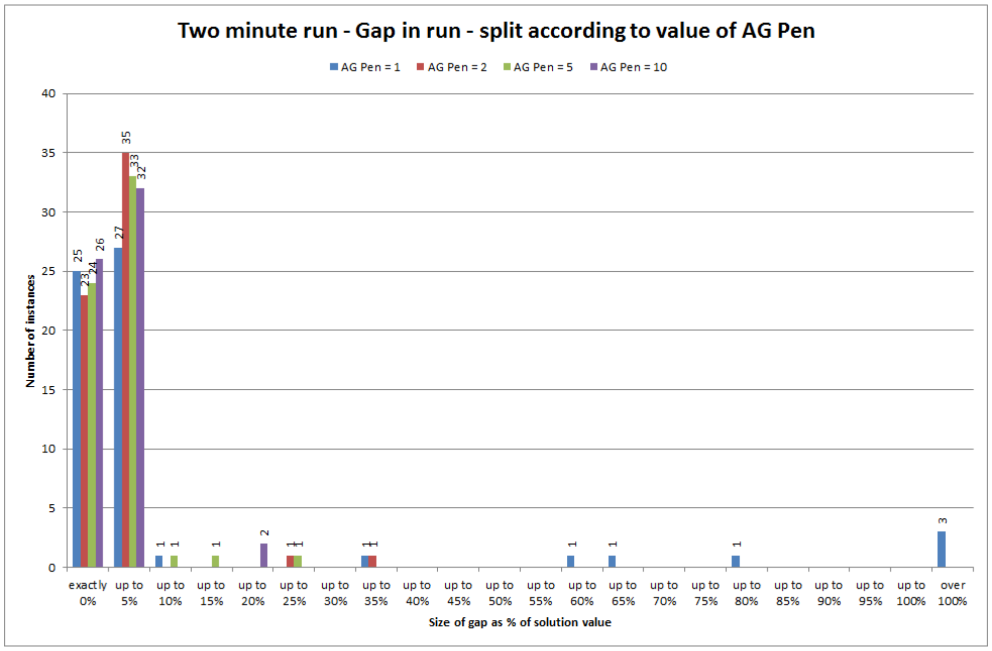


Figure B.36: Proven cost gap (i.e. to bound found during run) for the two-minute cost-minimization run, broken down by values of the agency penalty factor K_{AG}.

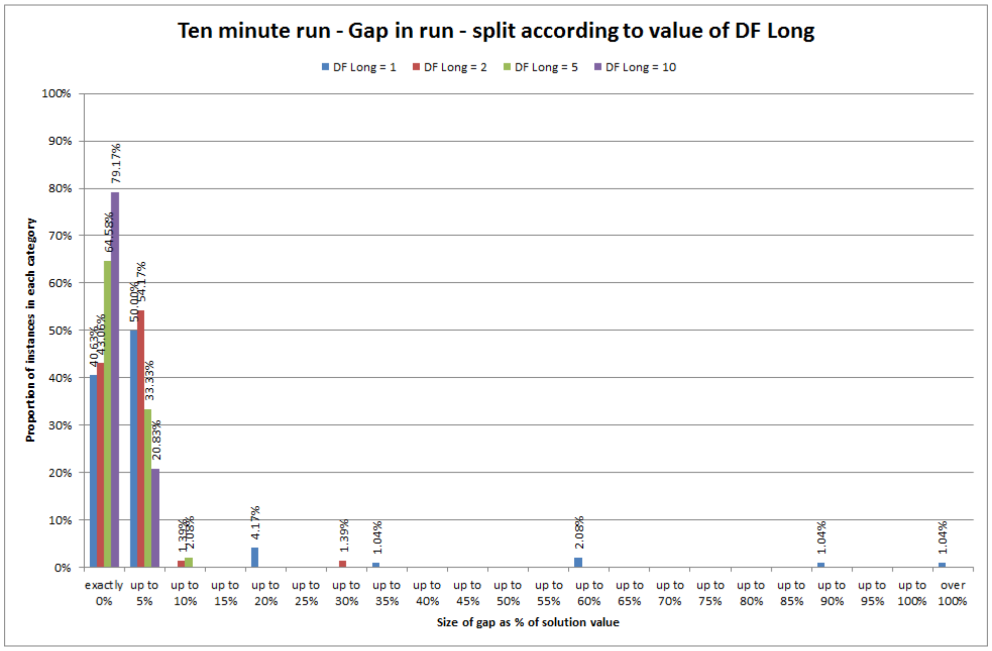


Figure B.37: Proven cost gap (i.e. to bound found during run) for the ten-minute cost-minimization run, broken down by values of the long-term disruption penalty K_L.

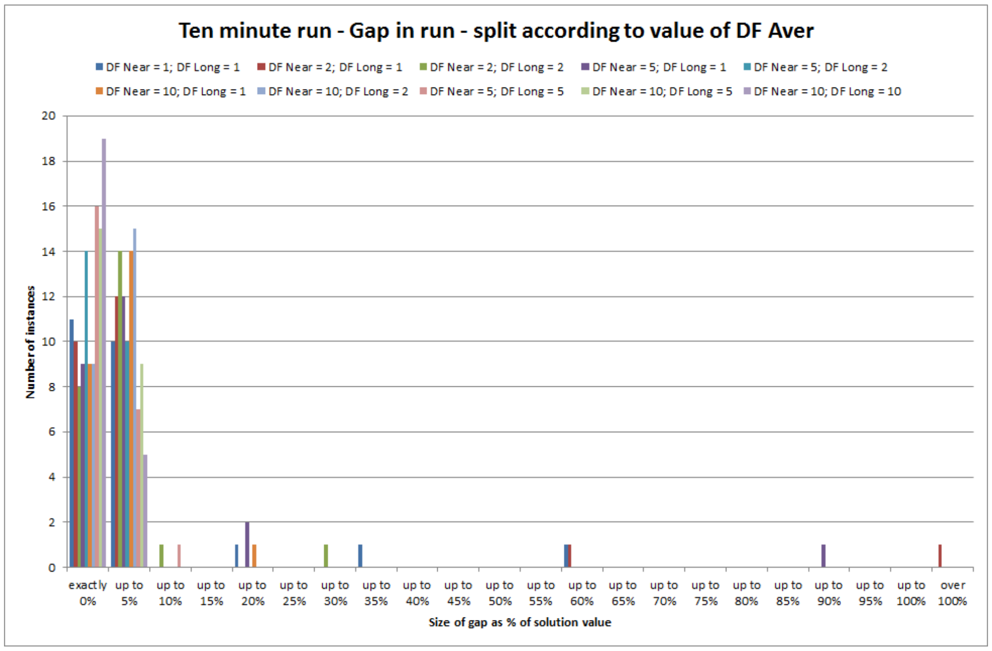


Figure B.38: Proven cost gap (i.e. to bound found during run) for the ten-minute cost-minimization run, broken down by values of the weighted average disruption penalty $\hat{K}$.

B.2.3 Relating to cost comparisons

The following graphs relate to the breakdown of the comparisons between the cost of the two-minute cost-minimization solution and the three cost outputs from the change-minimization algorithm - the non-cost-constrained, the cheapest minimal-change and the lowest cost solution.

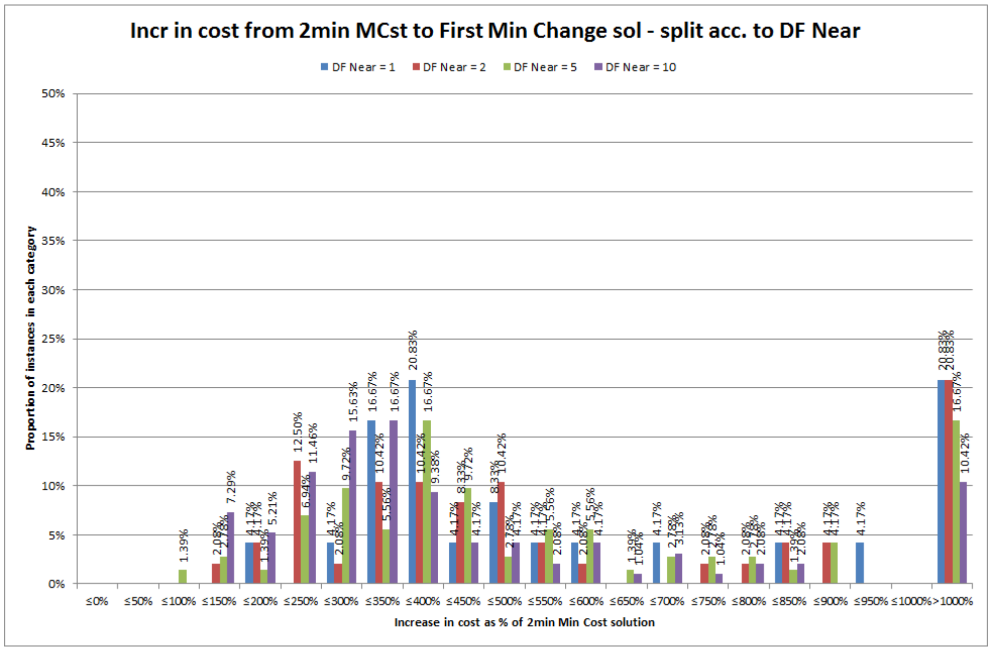


Figure B.39: Increase in cost for the non-cost-constrained change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the near-term disruption penalty K_N.

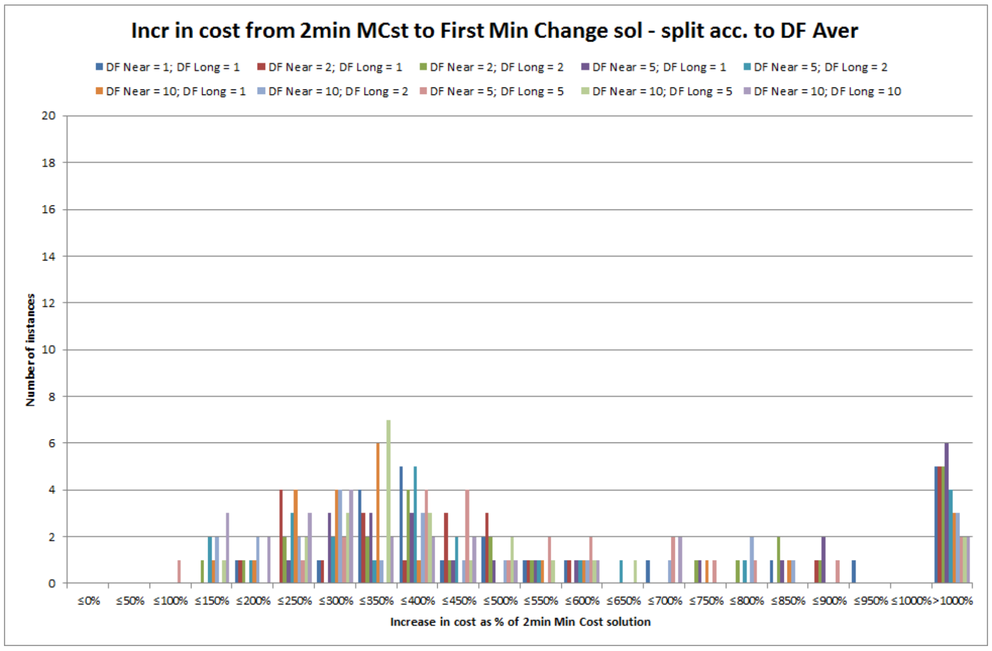


Figure B.40: Increase in cost for the non-cost-constrained change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the weighted average disruption penalty $\hat{K}$.

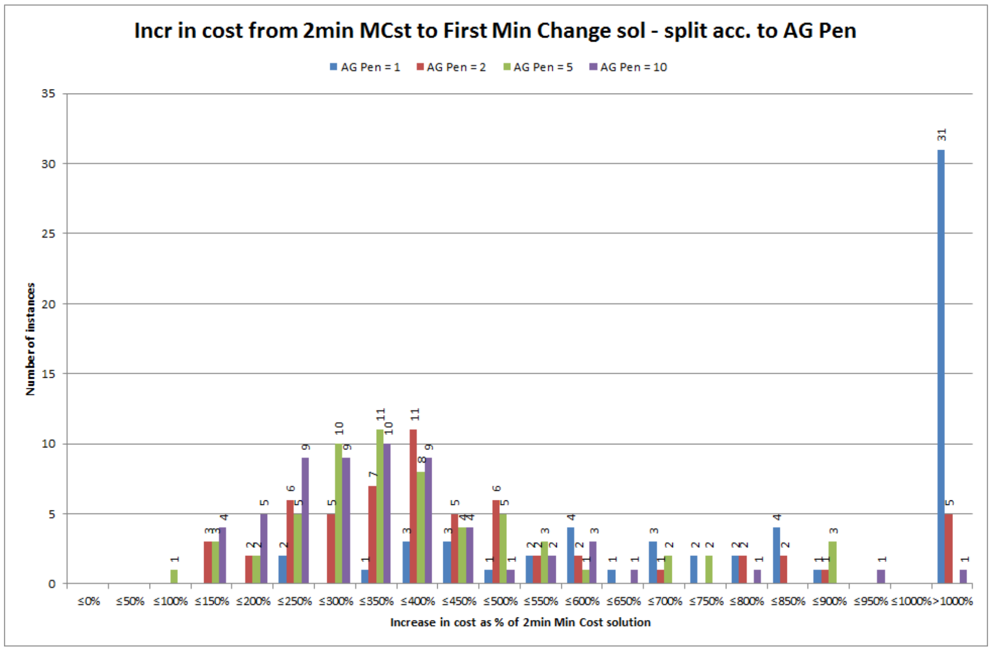


Figure B.41: Increase in cost for the non-cost-constrained change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the agency penalty factor K_{AG}.

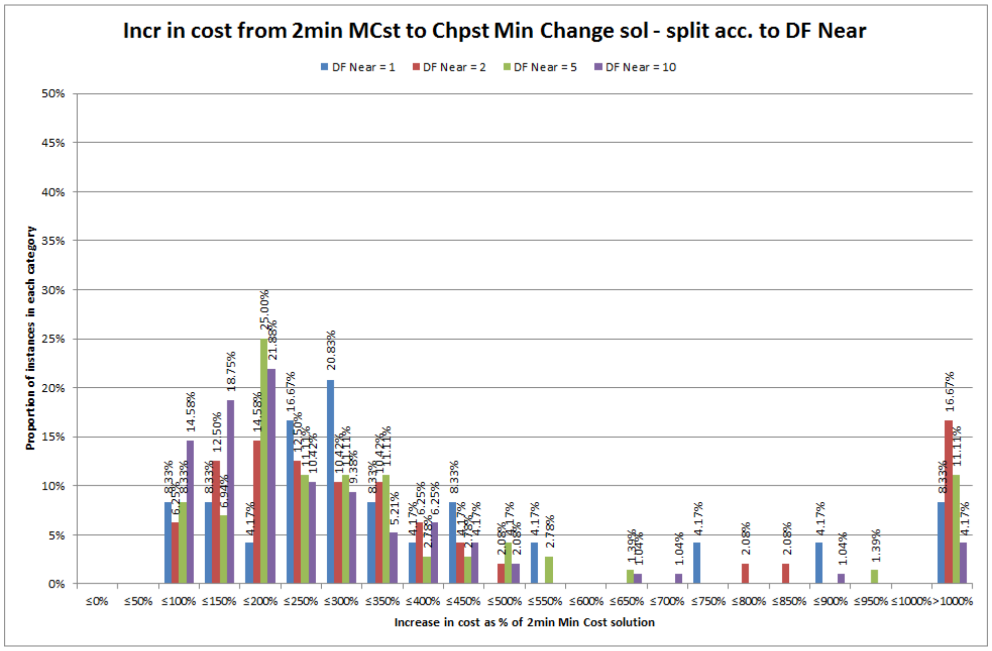


Figure B.42: Increase in cost for the cheapest minimal-change solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the near-term disruption penalty K_N.

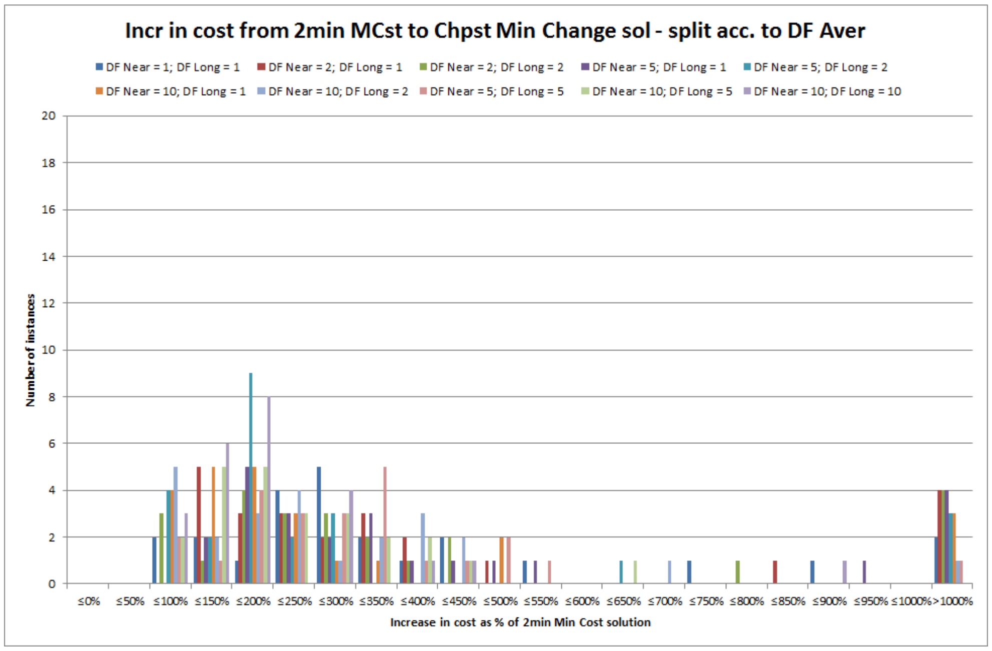


Figure B.43: Increase in cost for the cheapest minimal-change solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the weighted average disruption penalty $\hat{K}$.

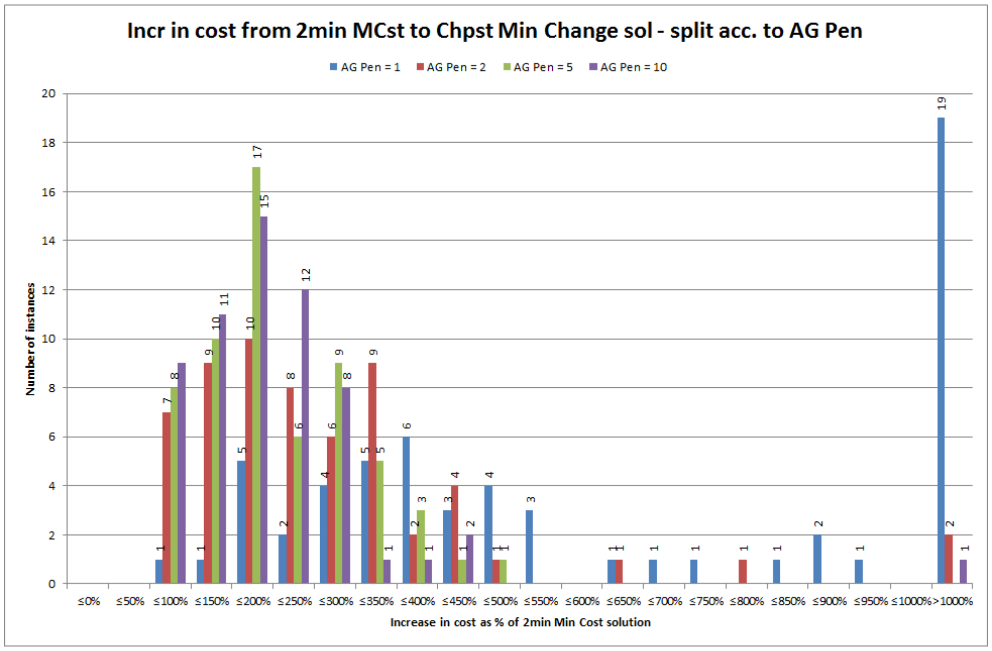


Figure B.44: Increase in cost for the cheapest minimal-change solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the agency penalty factor K_{AG}.

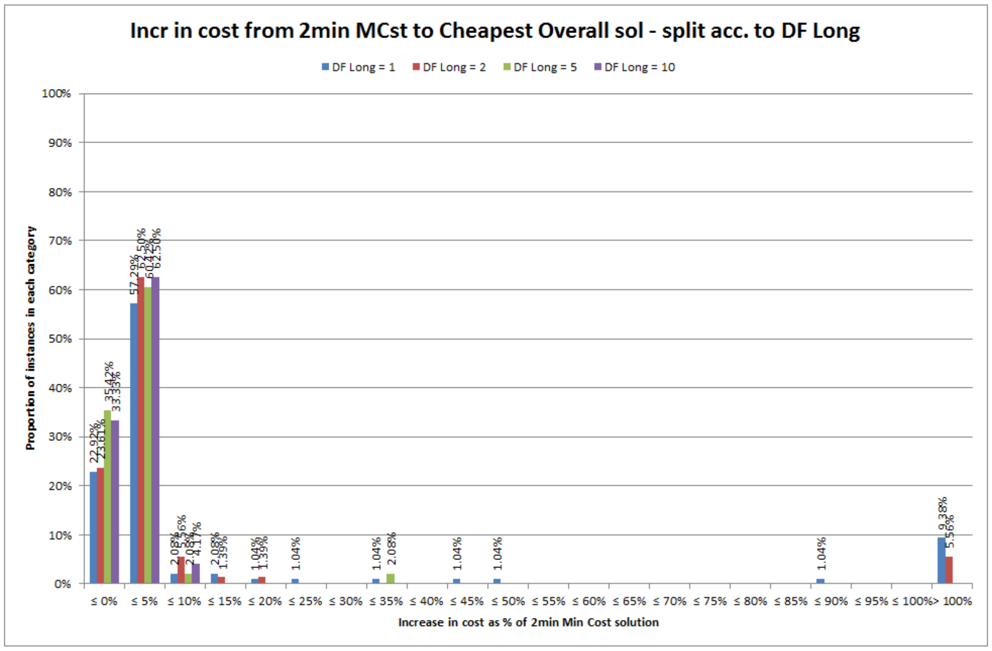


Figure B.45: Increase in cost for the lowest cost change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the long-term disruption penalty K_L.

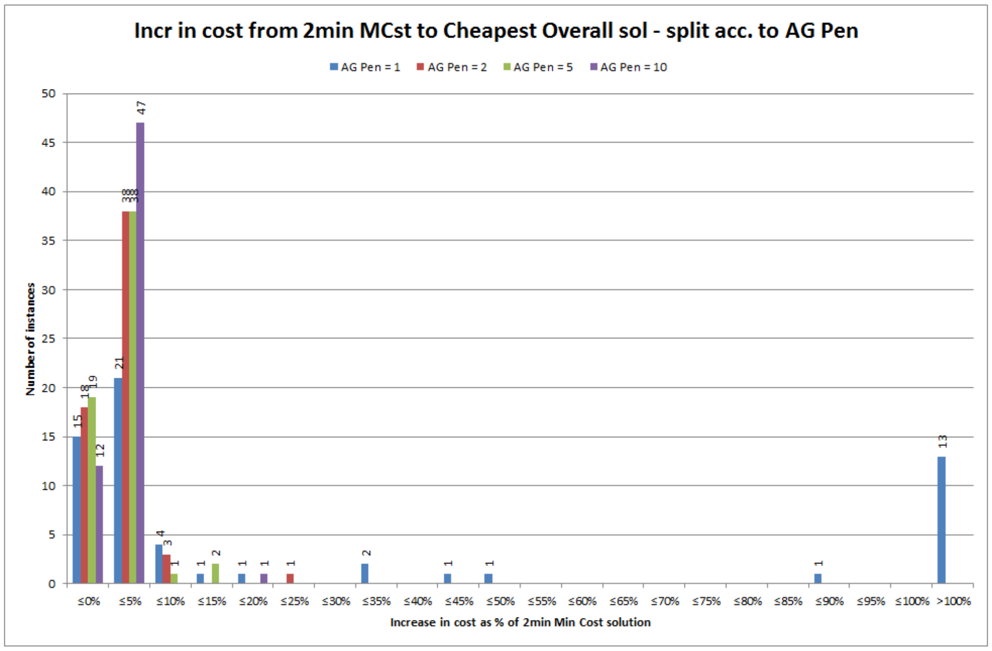


Figure B.46: Increase in cost for the lowest cost change-minimization solution, with the two-minute cost-minimization solution as baseline for comparison, broken down by values of the agency penalty factor K_{AG}.

B.2.4 Relating to running time

The following graphs relate to the breakdown of the comparisons between the cost of the two-minute cost-minimization solution and the three cost outputs from the change-minimization algorithm - the non-cost-constrained, the cheapest minimal-change and the lowest cost solution.

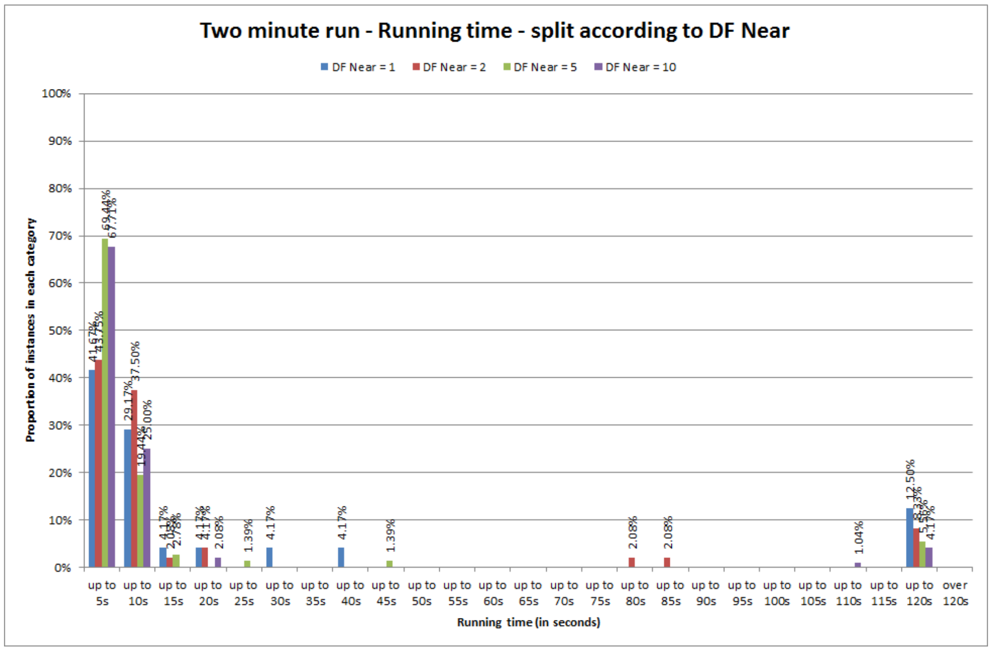


Figure B.47: Running time for the cost-minimization approach with two-minute time limit, broken down by values of the near-term disruption penalty K_N.

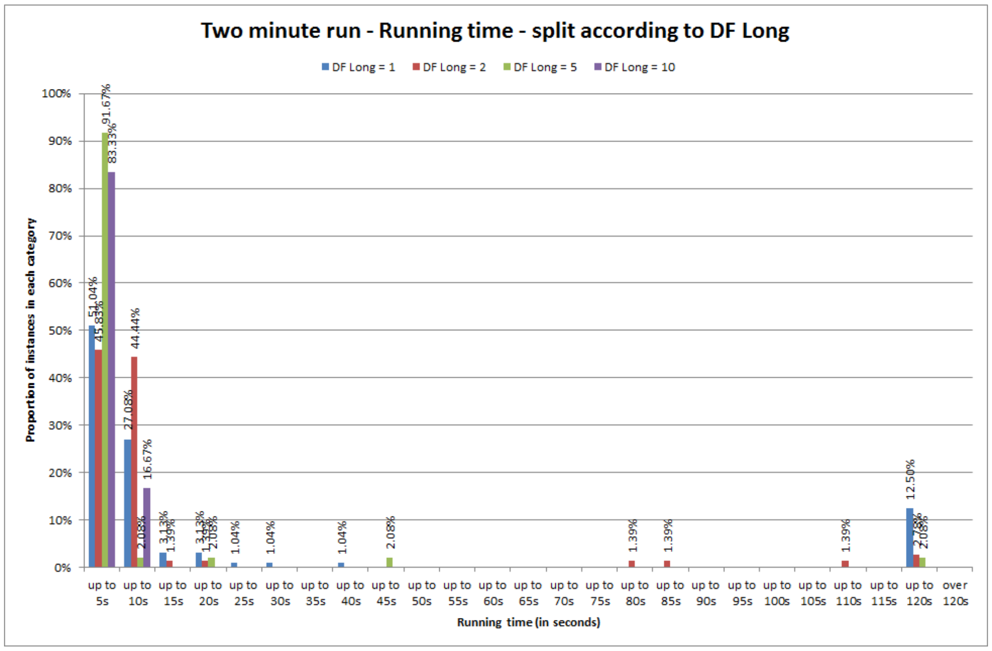


Figure B.48: Running time for the cost-minimization approach with two-minute time limit, broken down by values of the long-term disruption penalty K_L.

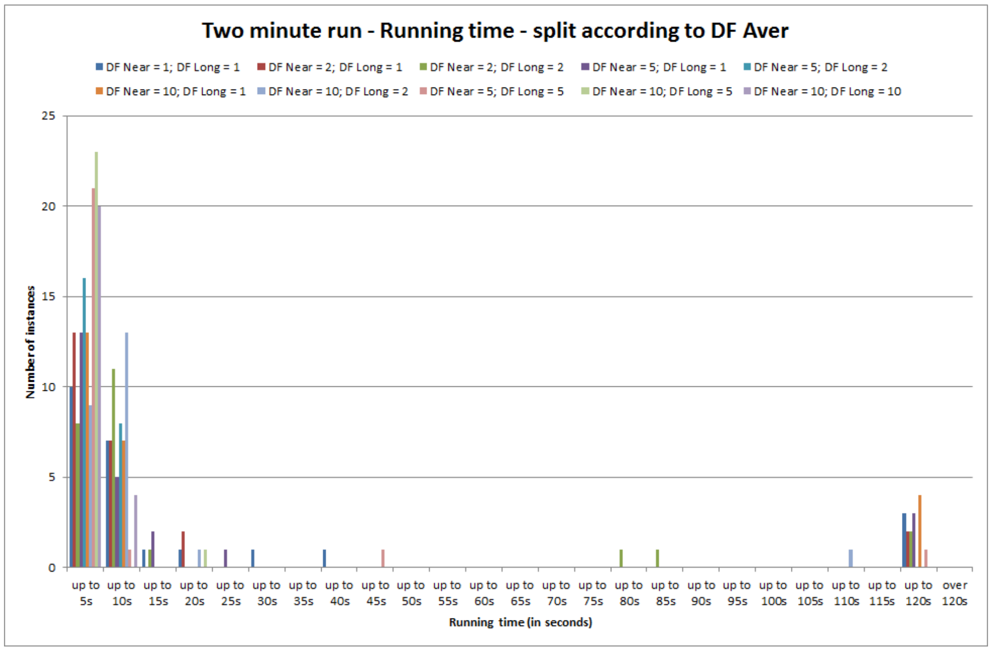


Figure B.49: Running time for the cost-minimization approach with two-minute time limit, broken down by values of the weighted average disruption penalty $\hat{K}$.

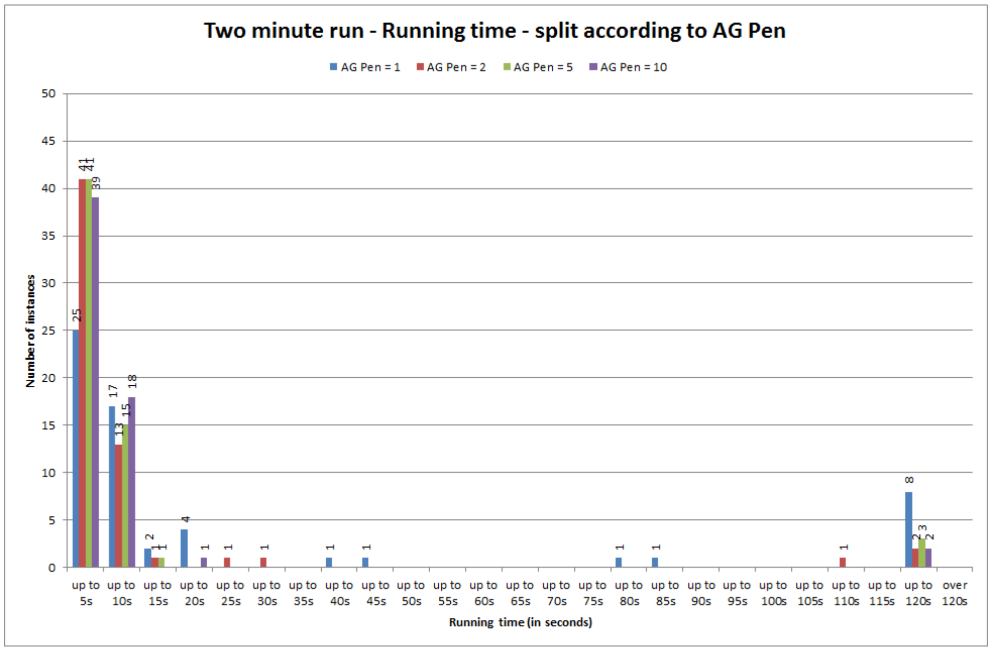


Figure B.50: Running time for the cost-minimization approach with two-minute time limit, broken down by values of the agency penalty factor K_{AG}.

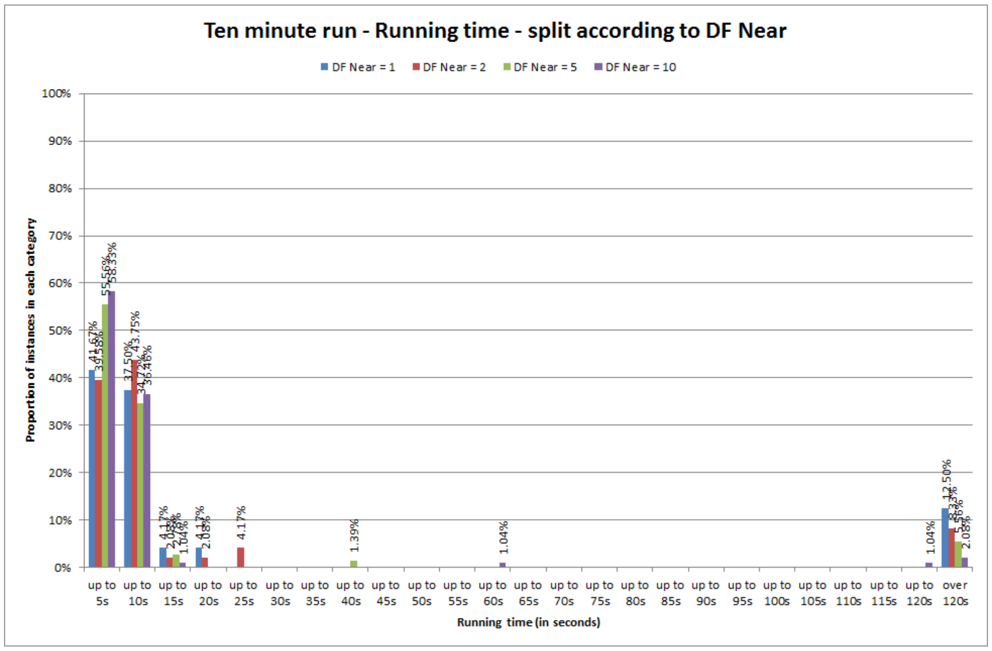


Figure B.51: Running time for the cost-minimization approach with ten-minute time limit, broken down by values of the near-term disruption penalty K_N.

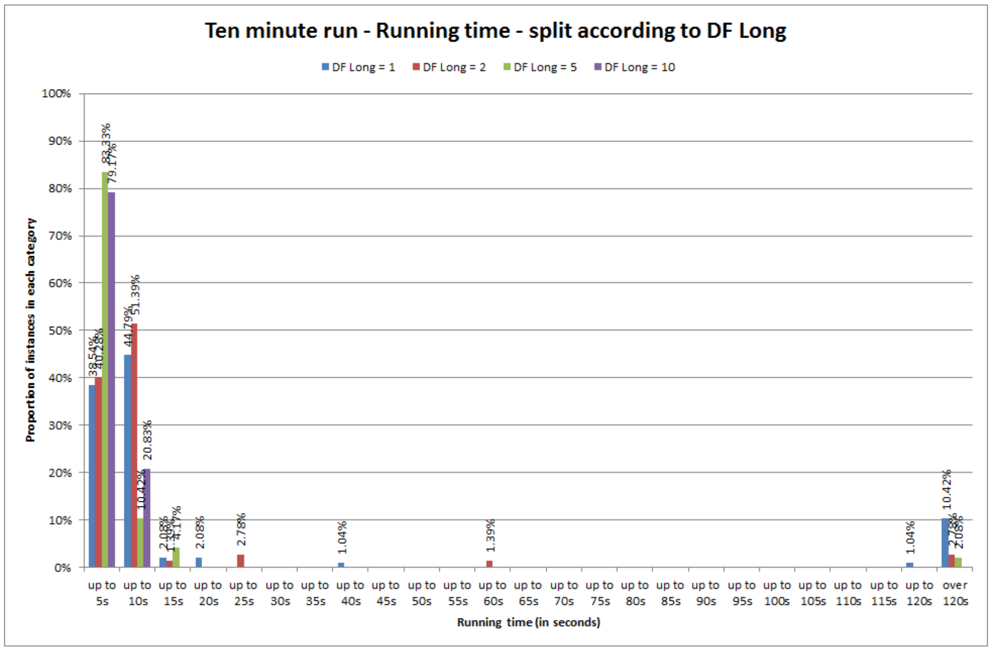


Figure B.52: Running time for the cost-minimization approach with ten-minute time limit, broken down by values of the long-term disruption penalty K_L.

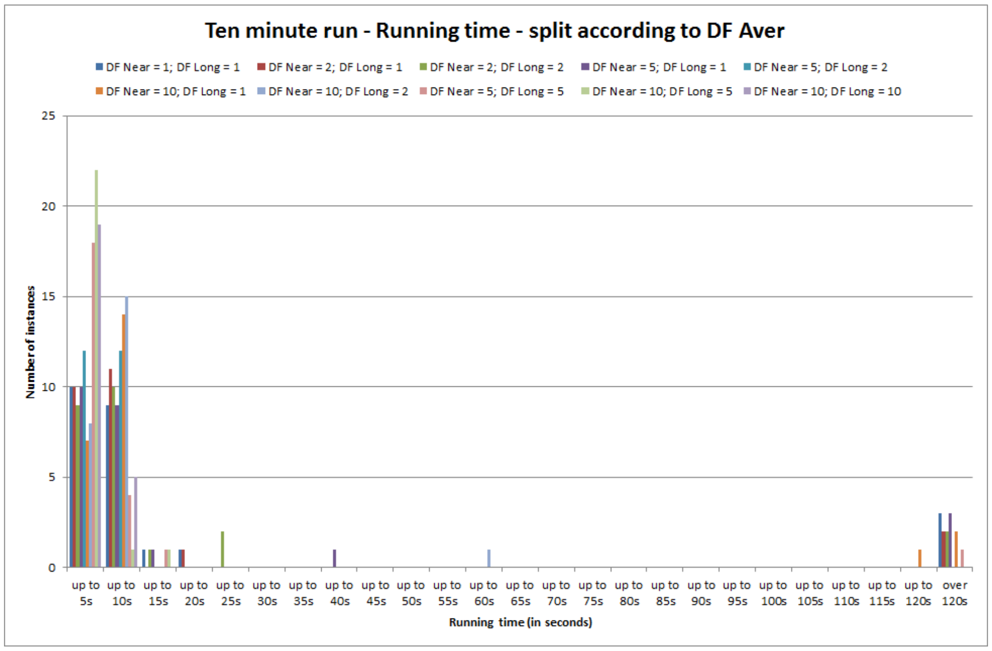


Figure B.53: Running time for the cost-minimization approach with ten-minute time limit, broken down by values of the weighted average disruption penalty $\hat{K}$.

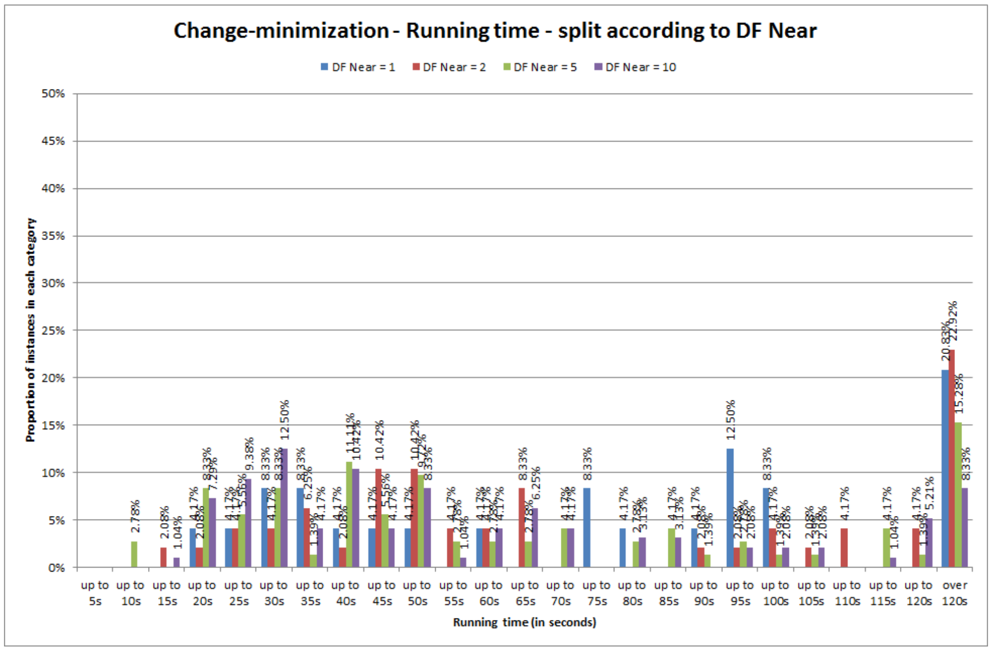


Figure B.54: Running time for the change-minimization algorithm (with two-minute time limit), broken down by values of the near-term disruption penalty K_N.

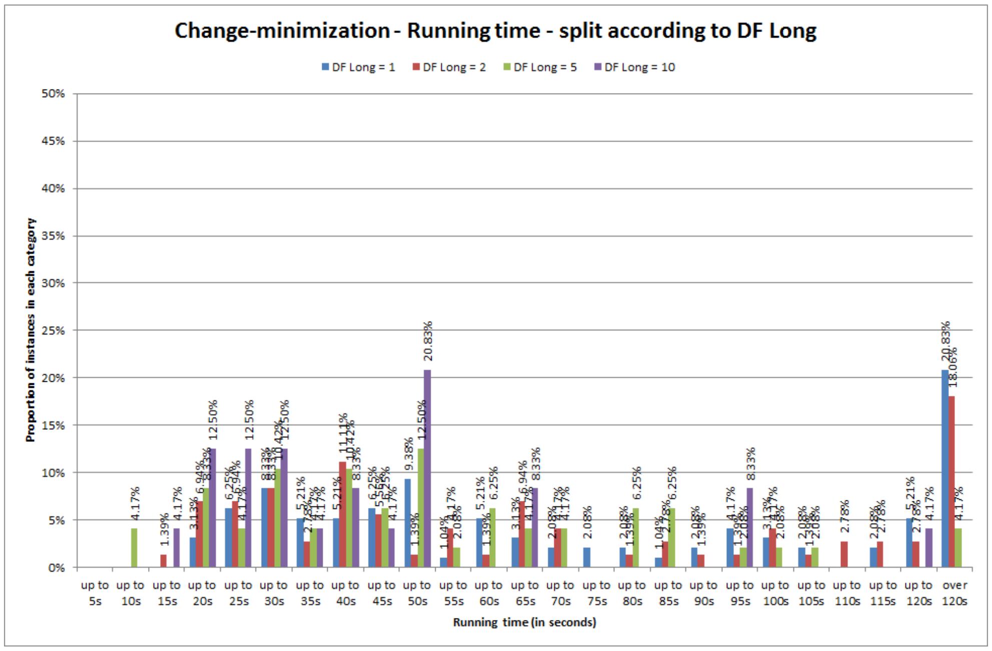


Figure B.55: Running time for the change-minimization algorithm (with two-minute time limit), broken down by values of the long-term disruption penalty K_L.

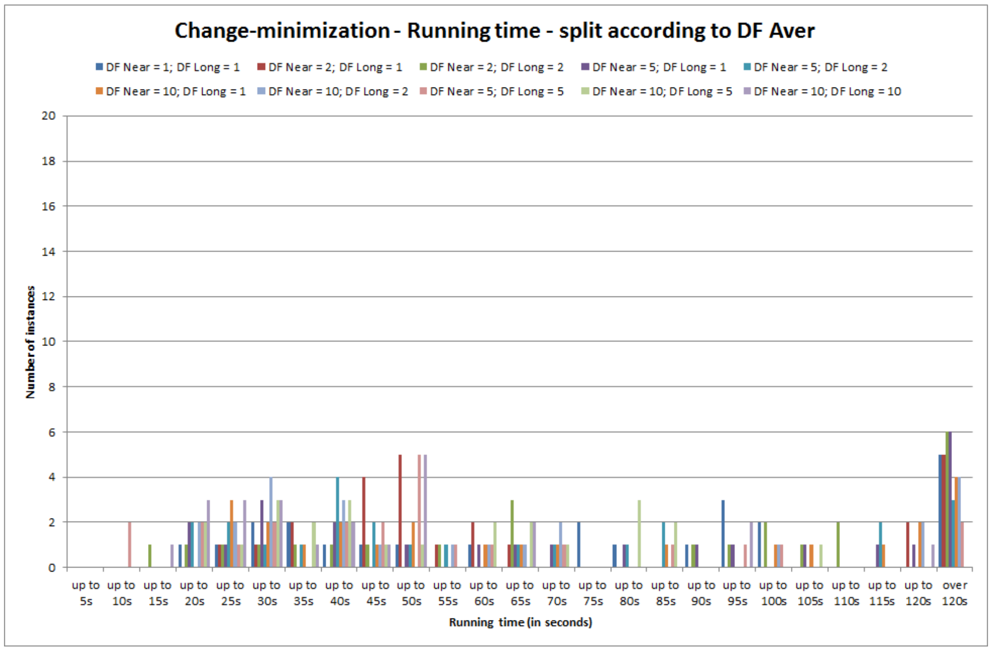


Figure B.56: Running time for the change-minimization algorithm (with two-minute time limit), broken down by values of the weighted average disruption penalty $\hat{K}$.

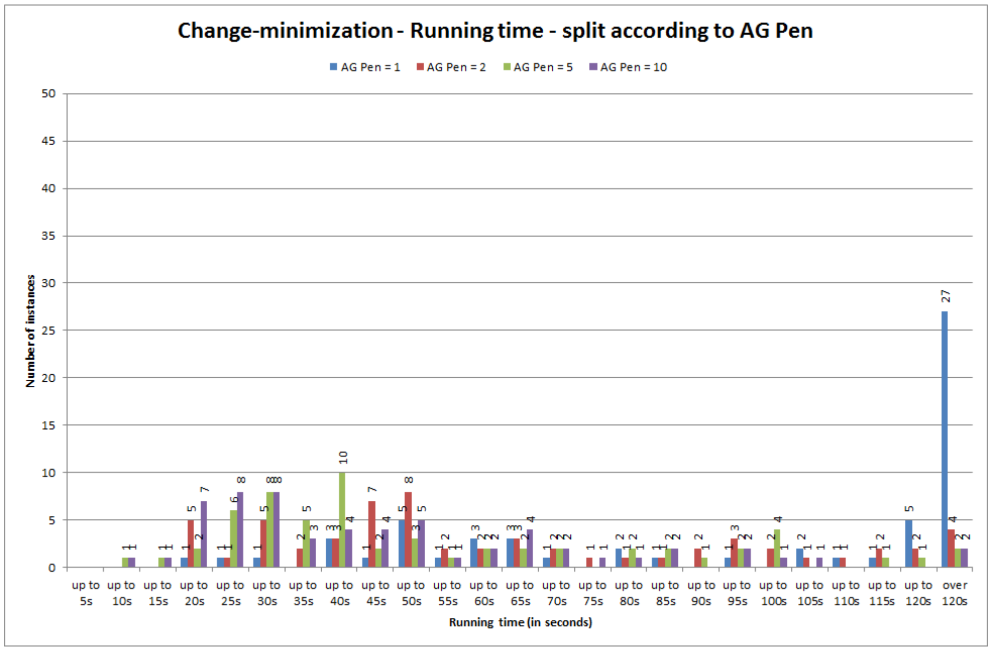


Figure B.57: Running time for the change-minimization algorithm (with two-minute time limit), broken down by values of the agency penalty factor K_{AG}.

Appendix C

Additional details of procedures for the Time Windows problem

Here we give additional details of some of the procedures discussed in chapter 6 relating to the Time-Windows formulation of the problem. These details include:

- In section C.1, a detailed description of the Task-Based Approximation procedure.
- In section C.2, additional details of the flowcharts for the Heuristic algorithm.
- In section C.3, a detailed description of the Heuristic algorithm.
- In section C.4, a detailed description of the Heuristic Initial Solution algorithm.

C.1 Description of Task-Based Approximation procedure

Here we detail the step-by-step procedure for using the Task-Based Approximation approach to solve the Time-Windows problem. As noted in section 6.4.1, this is dived into four parts: converting the data into the Task-Based format, solving the problem, converting the solution to the Time-Windows format, and checking the feasibility. All assumptions needed to ensure the Task-Based solution will be feasible in the Time-Windows formulation are given within the procedure. Note that full details of the algorithm implemented in FICO Xpress are given in appendix section E.2.4.

Part 1 - Convert to the Task-Based data format

1. Read in the Time-Windows data.
2. The following Time-Windows quantities are simple to translate into their Task-Based equivalents:

- The set of employees E is identical, as is the set G of fixed contract employees.
- The planning horizon length T is identical, except that it is defined in days for the Task-Based problem as opposed to weeks.
- The maximum working and minimum resting periods for regular employees (w_i^{max} and ρ_i respectively in the Time-Windows problem; ω_i and ρ_i respectively in the Task-Based formulation) are identical. Note that the agency maximum working period α_j^{max} is not required in the Task-Based formulation.
- The fixed contract details, i.e. under-time and over-time rates, number of guaranteed days and expected working time, are also identical with the proviso that they should be defined in days rather than weeks. For the Time-Windows problem these are defined respectively as c_i^U, c_i^O, g_i and Ω_i; for the Task-based formulation these are labelled as μ_i, ϕ_i, g_i and $\bar{W}_i$ respectively.
- Current excess, denoted as Ω_i for the Task-Based model and which we will denote as Ω_i^{TB} here, can be calculated using the under-time and over-time values, defined in days, in the existing Time-Windows schedule (u_i^* and o_i^* respectively). To do this, we will set
$$\Omega_i^{TB} = (\mu_i u_i^*) + (\phi_i o_i^*) \tag{C.1}$$
- The work resource values at time zero for regular crew, defined as w_{i0} for the Time-Windows problem and W_{i0} for the Task-Based, are identical. Note that the agency work resource values at time zero α_{j0} are not required in the Task-Based formulation.

3. Calculate the number of 'tasks' for the Task-Based version of this problem. Note that this must be done before the further information on these tasks can be defined. This can be done as follows:

   - **Assume that all tasks for a given role will be of the same length.**
   - Deal with each role $j \in V_k$ in turn for each vessel $k \in K$.
   - Using existing schedule data, can identify the employee in this role at time zero (being the employee $i \in E_R$ for which $s_{ik} = 1$ for this vessel, or agency crew if $\sigma_j = 1$ for the role j. Viewing this employee's work resource value at time zero (w_{i0} for $i \in E_R$, or α_{j0} for agency crew) we know the length of time they have been in this role up to time zero.
   - If the employee is a regular employee and has a maximum working time ω_i equal to two or ten weeks, then this implies either a 'Norway' or 'Singapore' contract, and can set this as the standard working length for the role j. Otherwise, should examine the boarding and departing values of the existing schedule (b_{ikt}^*, β_{jt}^*,

d^*_{ikt} and δ^*_{jt}) for this vessel k and role j, with the values of t for which these variables are equal to one indicating a standard length of task for this role.

- We now know the standard length of task for this role, and also the number of weeks into a task that this role is at time zero. From this, we can determine the number of tasks into which the role can be divided during planning period T.
- Once all roles have been considered, the number of tasks for each role can be aggregated to give the number of tasks n_W in total.

4. The start times and durations of the tasks can now be calculated as follows:

   - As with **Step 3** above, deal with each role $j \in \bigcup_{\forall k} V_k$ in turn.
   - From **Step 3** above, we know the initial length of time which has been worked by the employee in the role at time zero, and we also have a standard duration for the tasks relating to this role. We can therefore assign a duration d_j to the first task j in this role, which is simply the length of time remaining of the standard duration. The start time of this task is clearly $s_j = 0$.
   - We can then work chronologically along the timeline, assigning start times s_j and durations d_j for each task j of this role. We will consider the final task for the role to end at the end of the planning horizon, and so therefore the duration of this task may be less than the standard duration for the role.
   - For each task j which has been defined, should record the role to which it corresponds. This information is not required for the Task-Based problem, but will be necessary when translating the Task-Based solution back into the Time-Windows representation (see **Part 3** of this procedure).

5. Once the tasks have been defined, we can evaluate the eligibility matrix and the initial assignments:

   - **Assume that if an employee is ineligible for any part of a task then they are ineligible for the entire task.**
   - For each employee $i \in E$, look at their availability over each of the time periods that each task spans. If the employee is available, i.e. $e_{ijt} = 1$, for the corresponding role j in question during all the relevant time periods t, they are available for the task; otherwise they are not eligible to carry it out. Recall that there were no explicit indicator data for this in the Task-Based problem; instead, the decision variables for allocating an employee (i.e. change variables y_{ij} and new schedule variables z_{ij}) were simply not defined if they were unavailable for a given task.

- For the current schedule, can examine the starting positions of the employees (s_{ik} and σ_j for regular and agency respectively), the period-by-period existing assignments (x^*_{ijt} values), and the currently planned crew movements (indicated by b^*_{ikt} and β^*_{jt} for regular and agency crew boarding, and d^*_{ikt} and δ^*_{jt} for regular and agency crew departures) to determine which employees are currently scheduled to be in each role for the duration of each task. In the Task-Based problem, the current schedule is denoted by the binary data values x^*_{ij}, for each employee i and *task* j.

6. The rest resource values at time zero must be set, and more generally it must be ensured that required rest periods at the start of the planning period are respected. Recall that the rest resource values r_{i0} for the Time-Windows problem are integers, indicating the number of weeks rest still required by the employee at time t; while the values R_{i0} for the Task-Based problem are binary, indicating whether or not a rest is due. Therefore we do the following:

   - If r_{i0} is equal to the minimum rest period ρ_i required by employee i, then we set $R_{i0} = 1$.
   - If $r_{i0} = 0$, then $R_{i0} = 0$.
   - If $0 < r_{i0} < \rho_i$, i.e. employee i is in the middle of a required resting period at time zero, then we set $R_{i0} = 0$ for the Task-Based problem. However, in order to ensure the remaining required resting period is respected, we must alter the eligibility of employee i such that they can be assigned no tasks during the first r_{i0} weeks of the planning period. We therefore denote employee i as ineligible for any task j which starts at a time $s_j < r_{i0}$.
   - If an employee i is in the middle of an assignment at the start of the planning period, then they will have $r_{i0} = 0$ (and so, from above, we will set $R_{i0} = 0$). However, we must account for the possibility that they will be required to vacate this role at time zero (i.e. be de-assigned from the task which represents that role and which starts at time zero). In this case, they would be required to take a rest period prior to being assigned another task. We therefore must alter their eligibility such that *with the exception of the initial task in their current role* (i.e. the continuation of their current assignment), employee i is unavailable for all tasks j which start at time $s_j < \rho_i$.

7. Finally we can calculate the cost of changing each employee's assignment to each task. In general, this can be done by aggregating the relevant boarding, departing, working and consecutive working (also termed 'long working' or 'extension') change costs, but there are some exceptions to this. Also, the assumptions of the Task-Based

formulation may entail some changes being pre-determined during this conversion process - these ‘pre-assignment costs’ are also calculated and recorded in this step. We will discuss the process for regular employees firstly, and for each employee $i \in E_R$ and for each task j we do the following:

- First we note that it is possible, because of the way eligibility is defined (see **Step 5**) that an employee will be assigned to a task according to the current schedule for which they are not eligible. If for the employee i and task j currently under consideration, we have that i is not eligible to carry out task j but $x^*_{ij} = 1$ then we must correct this inconsistency. This can be done as follows:
  - Cancel the employee’s assignment to this task, i.e. set $x^*_{ij} = 0$. This is what we call a ‘pre-assigned’ change to the schedule, and the existing schedule which is fed into the Task-Based solver will already incorporate this change. The cost of it will therefore not influence the Task-Based solution process; however, in order to maintain and be able to check consistency between the Task-Based and Time-Windows versions of this problem, we must calculate the cost of pre-assigning this change.
  - If this task begins at $s_j = 0$ and the employee starts on board the associated vessel k then they must depart the vessel if they are no longer to work the task, and so we must add departing change cost ϕ^D_{ik1} to the cost of this pre-assignment. If $s_j > 0$ or the employee does not start on board vessel k then we must cancel their boarding of the vessel at time $t = 0$, and so we add boarding change cost ϕ^B_{ik1} to the pre-assignment cost.
  - For each week t which task j covers we must add the work change cost $\phi^W_{ij't}$ (where j' is the role associated with task j) to the pre-assignment cost. We should also calculate the consecutive working period entailed by task j (taking into account initial work resource value w_{i0} if task j starts at time $s_j = 0$), and add the consecutive work change costs $\phi^L_{\lambda ij't}$ to the pre-assignment cost for all affected values of consecutive work index λ.
  - If the employee is on a fixed contract (i.e. $i \in G$) then these changes will affect their current over- or under-time excess. In order to ensure this is accounted for properly, the changes must be calculated for each week t that the task covers in turn, with the following being carried out for each t:
    * If the employee currently has overtime in the existing schedule, i.e. if $o^*_i > 0$, then there is a saving of one unit of overtime. Reduce the value of o^*_i by one unit, and reduce the current excess Ω_i by one unit of the weekly overtime rate ϕ_i. The cost of the pre-assignment should also be reduced by the value of ϕ_i.

    * Otherwise, we need to add one unit of under-time to the employee. Increase the value of u_i^* by one unit, increase the current excess Ω_i by one unit of the undertime rate μ_i, and also add a cost of μ_i to the cost of the pre-assignment.

  - Finally we must consider cancelling the employee's departure from the role. Note that in the existing Time-Windows solution, departures ahead of the final week of the planning horizon (i.e. at $t = T$) will not have been set, and therefore if the task ends at or after this point there will be no departure to cancel. Therefore if the task ends at a time $t = s_j + d_j < T - 1$, we should add the departure change cost ϕ_{ikt}^D to the pre-assignment cost; otherwise, no cost is added.

- Alternatively, if the employee i is eligible for task j (irrespective of whether they are assigned to it in the current schedule), we must calculate their change cost c'_{ij} for this task. As above, there may be some changes which must be pre-assigned to remain consistent with the Task-Based formulation of the problem. We begin by initially setting $c'_{ij} = 0$, then proceed as follows:
  - If this task begins at $s_j = 0$ and the employee starts on board the associated vessel k then changing the employee's assignment to the task will involve changing their departure from the vessel, and so we must add departing change cost ϕ_{ik1}^D to the task change cost c'_{ij}. Otherwise (i.e. if $s_j > 0$ or the employee does not start on board vessel k) we must add boarding change cost ϕ_{ik1}^B to c'_{ij}.
  - For each week t for which the task operates, providing t is not the final week of the planning horizon, add the work change cost $\phi_{ij't}^W$ to the change cost c'_{ij}. We must also take into account the consecutive working periods entailed by task j (taking into account initial work resource value w_{i0} if task j starts at time $s_j = 0$), and add the consecutive work change costs $\phi_{\lambda ij't}^L$ to change cost c'_{ij} for all affected values of consecutive work index λ.
  - If the task covers the role in the final week of the planning horizon, but the employee is not assigned to the task in the current schedule (i.e. $x_{ij}^* = 0$), then we can add the work and long work change costs $\phi_{ij't}^W$ and $\phi_{\lambda ij't}^L$ to the task change cost c'_{ij} in the same way as above.
  - If the task is operating in the final week and the employee is initially assigned to the task, then this is slightly more complicated. Assuming the final week assignments have not been decided in the initial Time-Windows solution, we must pre-assign the employee to this role in this week. We therefore add work and extension change costs $\phi_{ij't}^W$ and $\phi_{\lambda ij't}^L$ to the pre-

assignment cost, and *subtract* these costs from the task change cost c'_{ij} (i.e. to cancel out this pre-assignment cost in the event that the employee is removed from this task in the new schedule). If the employee is on a fixed contract (i.e. if $i \in G$), then we must also recalculate the under- and over-time costs as follows:

  * If employee i currently has under-time in the existing schedule, i.e. if $u_i^* > 0$, then we must reduce the value of u_i^* by one unit. The current excess Ω_i should be reduced by one unit of the weekly under-time rate μ_i, and the pre-assignment cost should also be reduced by the value of μ_i.
  * Otherwise, we need to add one unit of over-time to the employee. Increase the value of o_i^* by one unit, increase the current excess Ω_i by one unit of the overtime rate ϕ_i, and also add a cost of ϕ_i to the pre-assignment cost.

- Finally we must consider the cost of the employee departing vessel k after the task. As before, note that in the existing Time-Windows solution departures ahead of the final week of the planning horizon (i.e. at $t = T$) will not have been set. Similarly, any departures *after* the final week of the planning horizon will not be considered here. There are three different scenarios to consider for this calculation:

  * If employee i is not currently assigned to task j (i.e. if $x_{ij}^* = 0$) and the task ends before the final week of the planning period (i.e. at a time $t = s_j + d_j < T$) then simply add departure change cost ϕ_{ikt}^D to change cost c'_{ij}.
  * If $x_{ij}^* = 1$ and the task finishes two or more weeks before the end of the planning period (i.e. at a time $t = s_j + d_j < T - 1$), then we can add the departure change cost ϕ_{ikt}^D to change cost c'_{ij}.
  * If however $x_{ij}^* = 1$ and the task ends at a time $t = s_j + d_j = T - 1$, then the departure must be pre-assigned since it was not considered in the existing Time-Windows schedule. Add departure change cost ϕ_{ikt}^D to the pre-assignment cost, and subtract ϕ_{ikt}^D from the task change cost c'_{ij} (which will cancel out the cost of the pre-assignment in the event that the employee is removed from task j in the new schedule).

8. In a similar manner to the regular employees, we can lastly calculate the change costs for the agency employees. Notice that we have assumed for these datasets that eligibility is never an issue for agency crew, and therefore this calculation is a little simpler. We firstly set $c'_{m+1,j} = 0$ as an initial value, and then for each task j proceed

as follows:

- Because there can be more than one agency employee assigned at a time, or performing back-to-back operations, the calculation at the start of the task is slightly more complex. There are four scenarios to consider:
  - If the task starts after time zero, or if an agency employee does not start in the role, then we can simply add the agency boarding change cost $\phi^{B_A}_{j't}$ to the task change cost $c'_{m+1,j}$.
  - Otherwise, we know from the initial working length calculated in **Step 5** above whether this task is a 'new' task or a 'continuing' task (i.e. whether it actually starts at $s_j = 0$, or if it essentially started before this). If the task is a 'new' task, then we can again simply add the agency boarding change cost $\phi^{B_A}_{j't}$ to the task change cost $c'_{m+1,j}$.
  - If it is a 'continuing' task then if the agency employee is already assigned to the task in the existing schedule, or if they are due to leave ahead of week 1, then we should add the departing change cost $\phi^{D_A}_{j't}$ to the task change cost $c'_{m+1,j}$.
  - Otherwise, we must pre-assign the departure of the agency employee ahead of week 1. To do this, add the departure change cost $\phi^{D_A}_{j't}$ to the pre-assignment cost, and modify the task change cost $c'_{m+1,j}$ by *subtracting* $\phi^{D_A}_{j't}$ from it.
- Next, for each week t in which the task operates, providing t is not the final week of the planning horizon, we must add the work change cost $\phi^{W}_{m+1,j't}$ to the change cost $c'_{m+1,j}$ for this task. We must also take into account the consecutive working periods (bearing in mind the initial work resource value α_{j0} if this task starts at time zero) and add the consecutive work change costs $\phi^{L}_{\lambda,m+1,j't}$ to change cost $c'_{m+1,j}$ for all affected values of consecutive work index λ.
- If the task covers the role in the final week of the planning horizon, but the agency employee is not assigned to the task in the current schedule (i.e. $x^*_{m+1,j} = 0$), then we can add the work and long work change costs $\phi^{W}_{m+1,j't}$ and $\phi^{L}_{\lambda,m+1,j't}$ to the task change cost $c'_{m+1,j}$ in the same way as above.
- If the task is operating in the final week and the agency employee is initially assigned to the task, then we must pre-assign the agency employee to this role in this week. We therefore add work and extension change costs $\phi^{W}_{m+1,j't}$ and $\phi^{L}_{\lambda,m+1,j't}$ to the pre-assignment cost, and *subtract* these costs from the task change cost $c'_{m+1,j}$ (i.e. to cancel out this pre-assignment cost in the event that the agency employee is removed from this task in the new schedule).

- Finally we must consider the cost of the agency employee departing role j' after the task. As with the regular employees, there are three different scenarios to consider for this calculation:
  - If the agency employee is not currently assigned to task j then (i.e. if $x^*_{m+1,j} = 0$) and the task ends before the final week of the planning period (i.e. at a time $t = s_j + d_j < T$) then simply add departure change cost $\phi^{D_A}_{j't}$ to change cost $c'_{m+1,j}$.
  - If $x^*_{m+1,j} = 1$ and the task finishes two weeks or more before the end of the planning period (i.e. at a time $t = s_j + d_j < T - 1$), then we can again add the relevant departing change cost $\phi^{D_A}_{j't}$ to task change cost $c'_{m+1,j}$.
  - If however $x^*_{m+1,j} = 1$ and the task ends at time $t = s_j + d_j = T - 1$, then the departure must be pre-assigned since it was not considered in the existing Time-Windows schedule. Add departure change cost $\phi^{D_A}_{j't}$ to the pre-assignment cost, and subtract $\phi^{D_A}_{j't}$ from the task change cost $c'_{m+1,j}$ (which will cancel out the cost of the pre-assignment in the event that the employee is removed from task j in the new schedule).

Part 2 - Set up and solve the Task-Based formulation

1. We must next carry out the usual procedure to turn the Task-Based data calculated in Part 1 into the quantities necessary to define the actual Task-Based formulation. Firstly, we must define the set N of rest tasks:
   - Define a set of rest task lengths which contains each value of minimum rest period ρ_i across the set of regular employees $i \in E_R$.
   - For each time point for which a working task ends, we must define a rest task corresponding to each of the determined rest task lengths (provided the task would end before the end of the planning period).
2. Define the work resource content w_j and rest resource content r_j for each work 'arc' (i.e. task) $j \in J$ and rest 'arc' $j \in N$:
   - For $j \in J$, $w_j = d_j$, the task duration, and $r_j = 1$.
   - For $j \in N$, $w_j = -T$, i.e. the length of the planning horizon, and $r_j = -1$
3. Define the sets C_γ of overlapping tasks:
   - Use Algorithm 5.1 set out in section 5.1.2.
4. Define the chronologically ordered set B of all the tasks in $J \cup N$:

- This is necessary for defining the work and rest resource tracking constraints.
- As described in section 5.1.1, the ordering is defined such that for all $b \in B$ such that $b \geq 2$ we have that $s_{b-1} \leq s_b$.

5. Ensure correct definition of the 'change' variables y_{ij} and 'new schedule' variables z_{ij} for all employees $i \in E$ for both work and rest tasks:

   - For work tasks $j \in J$, the variables are defined only for tasks for which the employee is eligible to work.
   - For rest tasks $j \in N$, the only permitted assignments are for regular employees $i \in E_R$ to those rest tasks whose lengths d_j correspond to their contractual minimum resting period ρ_i.

6. Define the problem formulation, as described in section 5.2.2 earlier. The objective function (5.14) should be modified to include the pre-assigned costs in addition to the change and over- and under-time costs - this should have no bearing on the solution process, but will be useful when checking equivalence between the Task-Based and Time-Windows versions of the solution. The constraints on the problem (expressions (5.15 - 5.26)) are used in the same way as given in section 5.2.2.

7. The Task-Based version of the problem can now be solved. Note that preliminary tests indicated that a two-minute time limit is sufficient to solve virtually all test instances to optimality, and no additional specifications on the cutting or heuristic strategy were made.

Part 3 - Convert the solution back to the Time-Windows format

1. We can use the new schedule given from the Task-Based formulation to determine the working, boarding, departing and long working values for the Time-Windows version of the new schedule (variables $\hat{x}_{ij't}$, $\hat{b}_{ikt}$ and $\hat{\beta}_{j't}$, $\hat{d}_{ikt}$ and $\hat{\delta}_{j't}$, and $\hat{l}_{\lambda ij't}$ respectively). These variables are all taken to equal zero unless this Step calculates them otherwise. For this, we take each role j' in turn and work through each task j relating to that role. For each of these tasks j, work through each employee $i \in E$ and do the following:

   - If the task j starts at time zero and the employee starts on board the vessel, and if the employee assigned to the task in the new solution (i.e. $z_{ij} = 1$), then:
     - For each period t in the duration of the task, the employee is assigned to that task in the Time-Windows solution, and we set $\hat{x}_{ij't} = 1$.

    - At the start of the task, the employee's working period has been w_{i0} (or $\alpha_{j'0}$ for agency crew). For each period t in the duration of the task, increment this consecutive working period value by one, and set $\hat{l}_{\lambda ij't} = 1$ for all values of λ less than or equal to this cumulative value.
    - If the task ends at some time $t = s_j + d_j < T$ (i.e. it does not end in the final week of the planning horizon), then denote the employee as departing once the task is complete by setting $\hat{d}_{ik,t+1} = 1$ for regular crew, or $\hat{\delta}_{j',t+1} = 1$ for agency crew.
  - If the task starts at time zero and the employee starts on board the vessel, but the employee is *not* assigned to the task in the new solution (i.e. i.e. $z_{ij} = 0$), then the employee must depart the vessel ahead of the first week of the planning period. In this case we set $\hat{d}_{ik1} = 1$ for regular crew, or $\hat{\delta}_{j'1} = 1$ for agency crew.
  - If the task does not start at time zero, or if the employee does not start on board the vessel, and if $z_{ij} = 1$ then:
    - For each period t in the duration of the task, the employee is assigned to that task in the Time-Windows solution, and we set $\hat{x}_{ij't} = 1$.
    - At the start of the task, the employee's working period has been w_{i0} (or $\alpha_{j'0}$ for agency crew). For each period t in the duration of the task, increment this consecutive working period value by one, and set $\hat{l}_{\lambda ij't} = 1$ for all values of λ less than or equal to this cumulative value.
    - If the task ends at some time $t = s_j + d_j < T$ (i.e. it does not end in the final week of the planning horizon), then denote the employee as departing once the task is complete by setting $\hat{d}_{ik,t+1} = 1$ for regular crew, or $\hat{\delta}_{j',t+1} = 1$ for agency crew.

2. From the new schedule calculated in **Step 1** above, we can calculate the new over- and under-time values $\hat{o}_i$ and $\hat{u}_i$ for the fixed contract employees. For each $i \in G$, we simply look at the expecting working time outwith the planning period Ω_i plus the number of working weeks assigned to the employee during the planning period, i.e. $\sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt}$. Examining the difference between this total working time and the employee's number of guaranteed weeks g_i, do the following:

   - If the expected work time is less than g_i, then set $\hat{u}_i$ equal to the (absolute value of) the difference, and set $\hat{o}_i = 0$.
   - If the expected work time is greater than g_i, then set $\hat{o}_i$ equal to the (absolute value of) the difference, and set $\hat{u}_i = 0$
   - If there is no difference, then set $\hat{o}_i = \hat{u}_i = 0$

3. We can now calculate the change variables values, indicating work change ($x^{\pm}_{ij't}$), boarding and departing change ($b^{\pm}_{ikt}$ and $d^{\pm}_{ikt}$ for regular crew; $\beta^{\pm}_{j't}$ and $\delta^{\pm}_{j't}$ for agency crew), consecutive work change ($l^{\pm}_{\lambda ij't}$) and under- and over-time change ($u^{\pm}_{i}$ and $o^{\pm}_{i}$). This can be done using equations (6.30 - 6.37) given in section 6.1.2 earlier.

4. From this, we can calculate the cost of the solution according to the Time-Windows variables. The values calculated above, along with the cost coefficients for the changes, can be used to evaluate the Time-Windows objective function, as given in 6.38.

Part 4 - Check for feasibility, and print the solution

1. If the conversion has been done correctly, all the constraints of the Time-Windows formulation should be satisfied by the Task-Based solution given. In order to check this, we can feed the values calculated in **Part 3** into the Time-Windows constraints and check that they are satisfied. Firstly, three further values must be calculated:

   - The work resource values at each time point for the regular crew ($\hat{w}_{it}$). This can be done for each time point t in order, by evaluating the right hand side of constraint (6.49) of the Time-Windows formulation, and setting $\hat{w}_{it}$ equal to the maximum of this value or zero.
   - The work resource values at each time point for the agency crew ($\hat{\alpha}_{jt}$). This can be done by evaluating the right hand sides of constraints (6.51) and (6.52) of the Time-Windows formulation, and setting $\hat{\alpha}_{jt}$ equal to the maximum of these two values.
   - The rest resource values at each time point for the regular crew ($\hat{r}_{it}$). This can be done by evaluating the right hand sides of constraints (6.54) and (6.55) of the Time-Windows formulation, and setting $\hat{r}_{it}$ equal to the maximum of these two values.

2. We can now check the remaining constraints to see if they are satisfied. All values should now be known and both sides of all expressions should now calculable. We check the following:

   - All roles are covered for all time periods for which cover is required, as per constraints (6.39) in the Time-Windows formulation;
   - No regular employee is assigned to two roles in the same time period, using expression (6.40);
   - The boarding and departing of regular crew is in line with their assignments, as required by constraints (6.41 - 6.44);

- The boarding and departing of agency crew is in line with their assignments, according to equations (6.45) and (6.46);
- Under-time and over-time are calculated correctly, using inequalities (6.47) and (6.48);
- The consecutive working variables are calculated correctly and the maximum working periods are respected for all employees, using expressions (6.50) and (6.53);
- Regular employees are not expected work when they are still due rest, as per constraint (6.56);
- All change variables link correctly between the initial and new schedules, as described by equations (6.57 - 6.64);
- All variables are correctly defined (binary, non-negativity, etc. as required), as described by expressions (6.65 - 6.72).

3. Finally, we can also check the objective values. We compare the Task-Based cost as found when the problem was solved (including the pre-assignment costs), and the Time-Windows objective value as calculated in **Step 4** of **Part 3** above. If the conversions have been done and pre-assignment costs tracked and calculated correctly, then these should be identical.

4. If the solution is feasible, it can be printed out in the required format to be implemented, or improved using heuristics.

C.2 Flowcharts of the Time-Windows Heuristic procedure

Here we expand on the flowcharts given in Figures 6.7 and 6.8 in section 6.5.5 describing the heuristic procedure developed for the Time-Windows problem (section 6.5).

Firstly Figure C.1 shows an expansion of the 'Initialise' step given in Figure 6.7 previ-

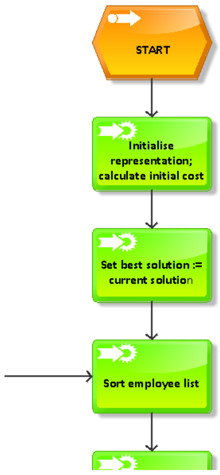

START
Initialise representation; calculate initial cost
Set best solution := current solution
Sort employee list

Figure C.1: Detail of flowchart, showing expansion of 'Initialise' step.

ously. In the expanded form, we now see that it includes evaluating the cost of the initial solution, and ensuring this current solution cost is also recorded as the best found so far.

Next, Figure C.2 shows the detail of the **'Evaluate backwards'** procedure. This takes

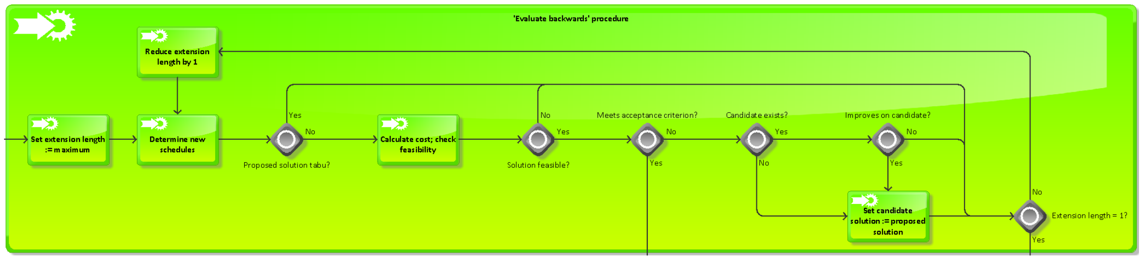

'Evaluate backwards' procedure
Reduce extension length by 1
Set extension length := maximum
Determine new schedules
Yes
No
Proposed solution tabu?
Calculate cost; check feasibility
No
Yes
Solution feasible?
Meets acceptance criterion?
No
Yes
Candidate exists?
Yes
No
Improves on candidate?
No
Yes
Set candidate solution := proposed solution
No
Extension length = 1?
Yes

Figure C.2: Detail of flowchart, showing detail of the 'Evaluate backwards' procedure.

the user through the steps of evaluating the new schedule, performing feasibility and tabu checks and deciding whether to accept the solution outright, store as a new 'candidate' solution, or simply move on. As can be seen, a loop is included to ensure this is done for all possible extension lengths if required. This can be compared to Figure C.3 which shows the expanded detail of the **'Evaluate forwards'** procedure. In essence, the steps of this procedure are identical to those of the **'Evaluate backwards'** routine, with the only difference being the way in which the new schedules are defined.

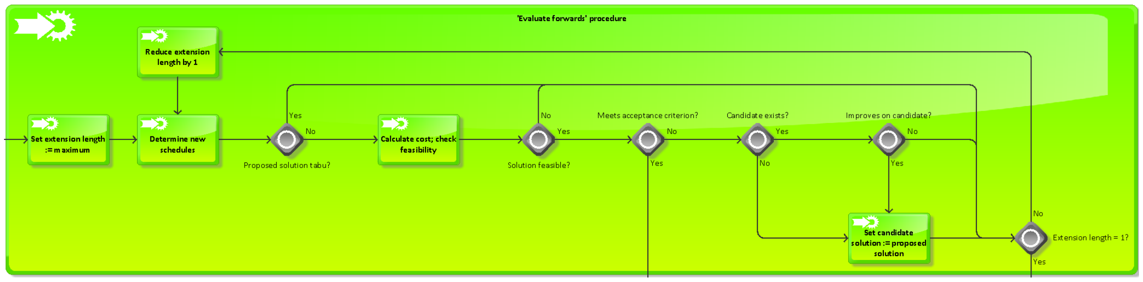

'Evaluate forwards' procedure
Reduce extension length by 1
Set extension length := maximum
Determine new schedules
Yes
No
Proposed solution tabu?
Calculate cost; check feasibility
No
Yes
Solution feasible?
Meets acceptance criterion?
No
Yes
Candidate exists?
Yes
No
Improves on candidate?
No
Yes
Set candidate solution := proposed solution
No
Extension length = 1?
Yes

Figure C.3: Detail of flowchart, showing detail of the 'Evaluate forwards' procedure.

The detail of the **'Evaluate swap'** procedure is shown in Figure C.4. This can be

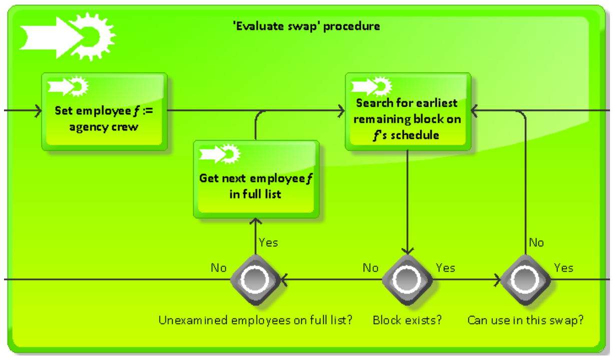

'Evaluate swap' procedure
Set employee *f* := agency crew
Search for earliest remaining block on *f*'s schedule
Get next employee *f* in full list
Yes
No
No
Yes
No
Yes
Unexamined employees on full list?
Block exists?
Can use in this swap?

Figure C.4: Detail of flowchart, showing detail of the 'Evaluate swap' procedure.

seen to perform similarly to the main loops of the programme, in that it searches each employee in turn, and for each employee it examines each block on their schedule. One key difference is that agency crew are now also considered. When a block is found which can be used in a swap, the **'Swap calculation'** procedure is called, which is shown in Figure C.5. This part of the programme performs similarly to the **'Evaluate backwards'** and **'Evaluate forwards'** procedures, except that there is no loop for examining multiple extension lengths. In addition, if the solution evaluated is not accepted outright then rather than going back to the **'Evaluate block'** stage, the programme returns to the **'Evaluate swap'** procedure to search for other potential swaps.

Finally, Figure C.6 gives detail of the **'Random kick'** procedure. Unlike the other solution-generating procedures described, there is no check of acceptance criteria. In this case, feasibility is the only criterion required for a 'kick' to be accepted, and as can be seen the programme will continue to generate random employee-block pairs until a feasible kick is found. Once this has been done, it is simply a case of evaluating the new schedules and the cost of these, plus checking that there is not a new best solution, before the programme

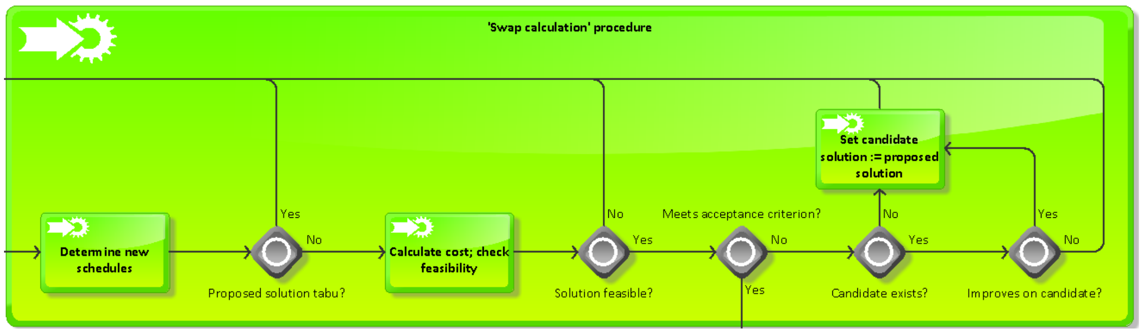

'Swap calculation' procedure
Determine new schedules
Yes
No
Proposed solution tabu?
Calculate cost; check feasibility
No
Yes
Solution feasible?
Meets acceptance criterion?
No
Yes
Set candidate solution := proposed solution
No
Yes
Candidate exists?
Yes
No
Improves on candidate?

Figure C.5: Detail of flowchart, showing detail of the ‘Swap calculation’ procedure.

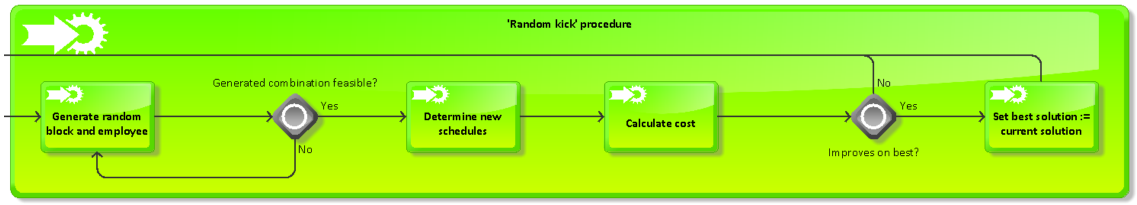

'Random kick' procedure
Generate random block and employee
Generated combination feasible?
Yes
No
Determine new schedules
Calculate cost
No
Yes
Improves on best?
Set best solution := current solution

Figure C.6: Detail of flowchart, showing detail of the ‘Random kick’ procedure.

can move on.

In order to see how these expanded procedures described in Figures C.1 to C.6 fit in to the overall algorithm, a full flowchart containing all the expanded procedures is also given. This can be seen split between Figures C.7 and C.8.

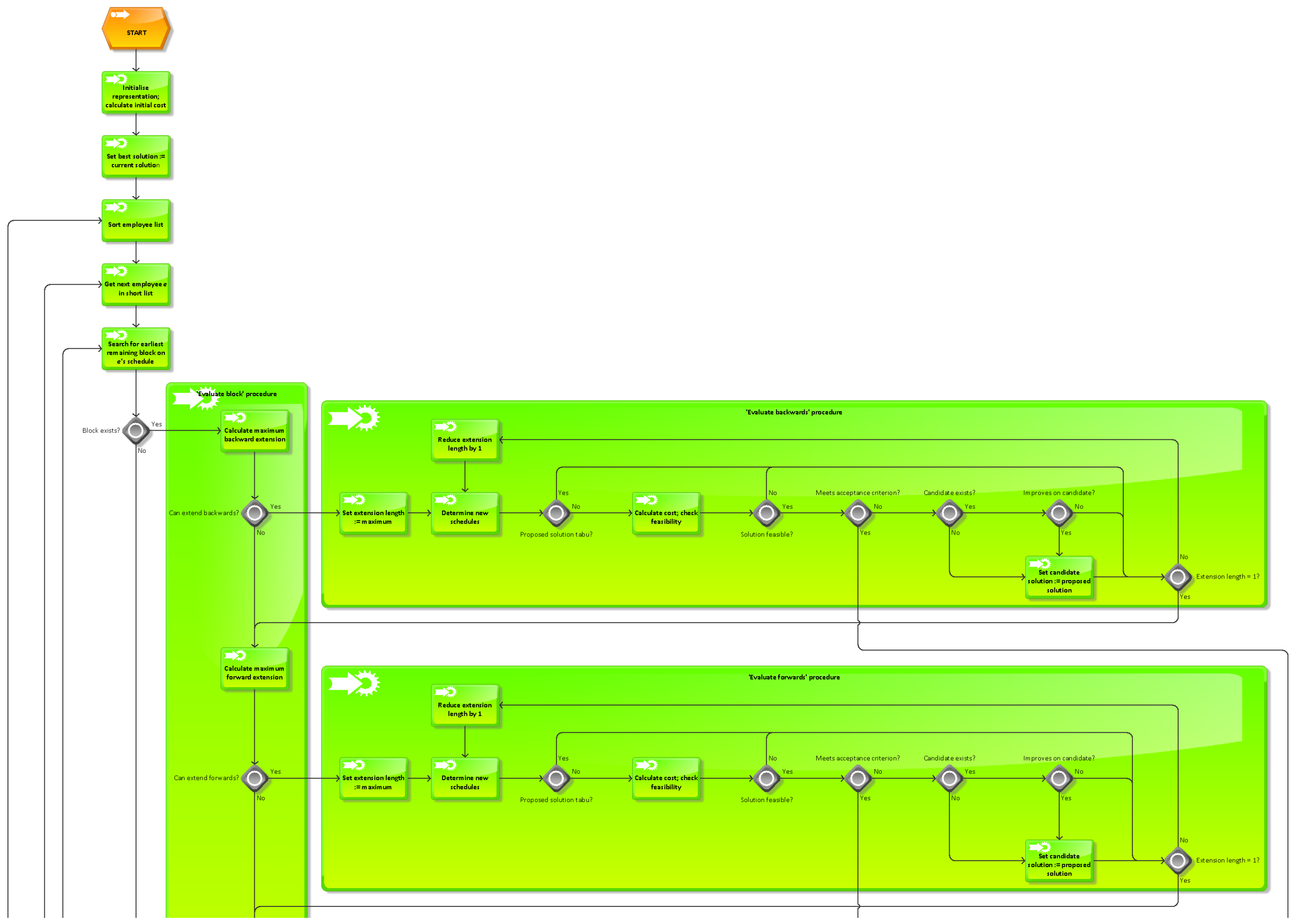

START
Initialise representation; calculate initial cost
Set best solution := current solution
Sort employee list
Get next employee e in short list
Search for earliest remaining block on e's schedule
Block exists?
Yes
No
'Evaluate block' procedure
Calculate maximum backward extension
Can extend backwards?
Yes
No
Calculate maximum forward extension
Can extend forwards?
Yes
No
'Evaluate backwards' procedure
Set extension length := maximum
Determine new schedules
Reduce extension length by 1
Proposed solution tabu?
Yes
No
Calculate cost; check feasibility
Solution feasible?
No
Yes
Meets acceptance criterion?
No
Yes
Candidate exists?
Yes
No
Improves on candidate?
No
Yes
Set candidate solution := proposed solution
Extension length = 1?
No
Yes
'Evaluate forwards' procedure
Set extension length := maximum
Determine new schedules
Reduce extension length by 1
Proposed solution tabu?
Yes
No
Calculate cost; check feasibility
Solution feasible?
No
Yes
Meets acceptance criterion?
No
Yes
Candidate exists?
Yes
No
Improves on candidate?
No
Yes
Set candidate solution := proposed solution
Extension length = 1?
No
Yes

Figure C.7: Expanded flowchart outlining the heuristic algorithm - upper section.

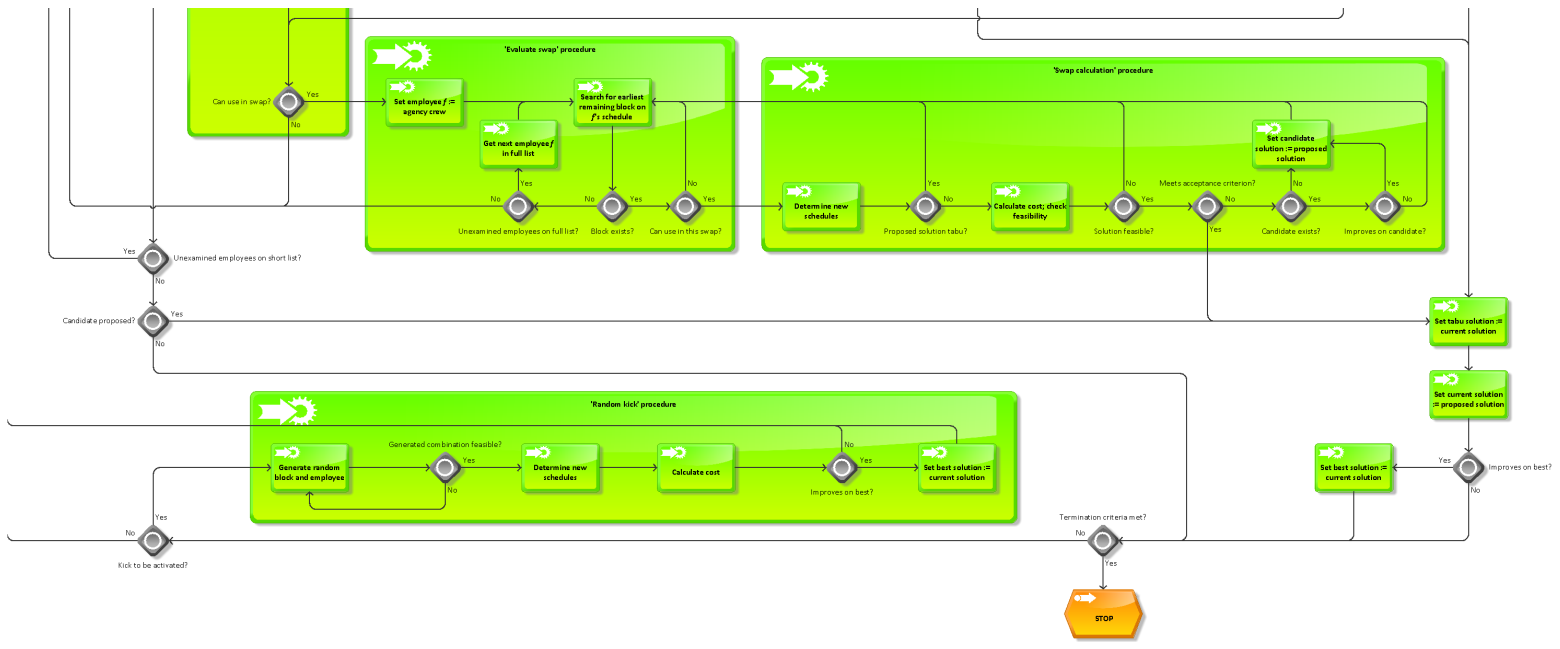

Can use in swap?
Yes
No
'Evaluate swap' procedure
Set employee f := agency crew
Get next employee f in full list
Search for earliest remaining block on f's schedule
Unexamined employees on full list?
Block exists?
Can use in this swap?
'Swap calculation' procedure
Determine new schedules
Proposed solution tabu?
Calculate cost; check feasibility
Solution feasible?
Meets acceptance criterion?
Set candidate solution := proposed solution
Candidate exists?
Improves on candidate?
Unexamined employees on short list?
Candidate proposed?
Set tabu solution := current solution
Set current solution := proposed solution
Improves on best?
Set best solution := current solution
'Random kick' procedure
Generate random block and employee
Generated combination feasible?
Determine new schedules
Calculate cost
Improves on best?
Set best solution := current solution
Kick to be activated?
Termination criteria met?
STOP

Figure C.8: Expanded flowchart outlining the heuristic algorithm - lower section.

C.3 Description of Heuristic algorithm

Here we describe the formal test version of the Heuristic algorithm, as outlined in section 6.5.5, in more detail. Note that the full code of the algorithm as implemented in C++ can be found in the appendix in section E.2.5. Note also that while this formal test version is a product of the development process described above, the actual coding of this algorithm into C++ was carried out by fellow PhD student Seda Sucu.

C.3.1 Main programme

1. Declare all variables required by the algorithm, and read in the data.

2. Call **'Initialisation'** procedure (page 370).

3. While termination criteria are not satisfied, repeat the following:

   (a) Increment iteration count.

   (b) Set employee ordering parameter for this iteration, according to rules defined in section 6.5.4, and call **'Sort employee list'** procedure (page 371).

   (c) Call **'Find usable block'** procedure (page 382).

   (d) If an intermediate time point has been reached, record details of the best solution so far.

   (e) Check termination criteria, according to rules defined in section 6.5.4.

   (f) If termination criteria are not satisfied then check kick implementation rules, as defined in section 6.5.4. If a kick if required then call **'Random kick'** procedure (page 399).

4. Record details of best solution found, and details of the test run overall.

5. **End** of algorithm.

C.3.2 Sub-Programmes

The sub-procedures used for the Heuristics are detailed here, approximately following the order that the algorithm would require them during execution.

C.3.2.1 Initialisation

1. Translate the initial solution into rosters Υ_i for all regular employees $i \in E_R$ and agency sets Θ_j for all roles $j \in J$. Record these as comprising both the 'current' solution, and the 'best' solution found so far.

2. Add all employees $i \in E_R$ and all roles $j \in J$ to the respective sets of employees and roles which have had their schedules changed since the last iteration.

3. Set calculation parameter to ‘current’, and call **‘Calculate cost’** procedure (page 372).

4. Set transfer parameters as ‘from: current; to: best’, and call **‘Transfer solution’** procedure (page 372).

5. Initialise counting parameters:

   - Number of best solutions found = 1;
   - Number of forwards extension, backwards extension, swap and best alternative candidate solutions = 0;
   - Number of kicks carried out and number of non-reducing iterations = 0;
   - Number of tabu and infeasible solutions found = 0;
   - Termination criteria are not satisfied;
   - Current iteration, time last kick carried out, and time of finding best solution = 0; and
   - Last iteration at which employee i’s schedule changed = 0 for all $i \in E_R$.

C.3.2.2 Sort employee list

1. If the ordering parameter value is not a valid one (i.e. ‘latest changed first’, ‘earliest changed first’ or ‘random’), then set the parameter to be ‘random’.

2. Clear ordered list of employees and short ordered list of employees.

3. Generate a random number in the range $[0, 1)$ to associate with each employee.

4. Repeat the following m times:

   (a) Select an employee who has not yet been added to the ordered list.

   (b) For each of the other employees i not yet added to the ordered list, do the following:

      i. If the ordering parameter is not ‘random’, then compare the last recorded time that employee i’s rosters was changed with the last time the selected employee’s roster was changed. If the ordering parameter is ‘latest changed first’, select the employee with the (strictly) latest time changed; if the ordering parameter is ‘earliest changed first’, select the employee with the (strictly) earliest time changed.

ii. If the order parameter is 'random', or the employees are tied for the time last changed, then select the one with the lowest associated random number.

(c) Add the selected employee to the ordered list of employees.

(d) If the length of the short ordered list is still less than the required fraction of m, as defined in section 6.5.4, then add the selected employee to the short ordered list of employees.

C.3.2.3 Transfer solution

1. Check that both the 'from' and 'to' transfer parameters are valid solution types. If not, then **end** this procedure now; otherwise, continue.

2. Copy the values of the solution designated by the 'from' transfer parameter and allocate them to the solution designated by the 'to' transfer parameter. The values to be copied are as follows:

   - The total cost of the solution.
   - Decision variable values for allocation ($\hat{x}_{ijt}$) and consecutive working ($\hat{l}_{\lambda ijt}$) for all employees $i \in E$ including agency.
   - Rosters Υ_i and their associated cost, and decision variable values for boarding the departing vessels ($\hat{b}_{ikt}$ and $\hat{d}_{ikt}$) for regular crew $i \in E_R$.
   - Decision variable values for overtime and undertime ($\hat{o}_i$ and $\hat{u}_i$) for fixed contract employees $i \in G$.
   - Agency allocation sets Θ_j and crew change sets Ξ_j, and their associated costs, for all roles $j \in J$.
   - Decision variable values for agency crew boarding and departing from roles ($\hat{\beta}_{jt}$ and $\hat{\delta}_{jt}$) for all roles $j \in J$.

3. Clear the values of the 'from' and 'to' transfer parameters.

C.3.2.4 Calculate cost

1. Check that the calculation parameter is a valid solution type. If not, then **end** this procedure now; otherwise, continue.

2. Take rosters Υ_i and sets Θ_j associated with the solution type designated by the calculation parameter, and transfer them to the corresponding rosters and sets designated for 'evaluating'.

3. Set total cost of 'evaluating' solution to zero.

4. To calculate the regular employee costs, for each employee $i \in E_R$ do the following:
   (a) If i's schedule is not in the set of those changed since the last iteration then copy relevant solution values designated for 'current' solution to those designated for 'evaluating' solution:
      - Decision variable values for allocation ($\hat{x}_{ijt}$), consecutive working ($\hat{l}_{\lambda ijt}$), boarding and departing the vessels ($\hat{b}_{ikt}$ and $\hat{d}_{ikt}$), and if i is a fixed contract employee then overtime and undertime ($\hat{o}_i$ and $\hat{u}_i$).
      - The change cost associated with roster Υ_i.

      And move onto next employee. Otherwise, carry out remainder of this step.
   (b) Clear the recorded cost for roster Υ_i in the 'evaluating' solution, and set the initial consecutive work count parameter equal to w_{i0}.
   (c) For each time point t from $t = 1$ to $t = T$, do the following:
      i. For all $j \in J$ set the allocation variable $\hat{x}_{ijt} = 0$, and set the consecutive working variables $\hat{l}_{\lambda ijt} = 0$ for all λ.
      ii. If $j_t^i \in J$, i.e. the t^{th} element in the 'evaluating' roster Υ_i is a working task then:
         A. Set $\hat{x}_{i,j_t^i,t} = 1$;
         B. Increment the consecutive work count by one; and
         C. Set $\hat{l}_{\lambda,i,j_t^i,t} = 1$ for all λ less than or equal to the consecutive work count value.
      iii. If $j_t^i = rest$ then reset the consecutive work count parameter to zero.
   (d) For each vessel $k \in K$, do the following:
      i. If for this 'evaluating' solution, $\sum_{j \in V_k} \hat{x}_{ij1} < s_{ik}$ then set $\hat{d}_{ik1} = 1$; otherwise set $\hat{d}_{ik1} = 0$.
      ii. Similarly, if $\sum_{j \in V_k} \hat{x}_{ij1} > s_{ik}$ then set $\hat{b}_{ik1} = 1$; otherwise set $\hat{b}_{ik1} = 0$.
      iii. For each time point t from $t = 2$ to $t = T$, do the following:
         A. If $\sum_{j \in V_k} \hat{x}_{ijt} < \sum_{j \in V_k} \hat{x}_{ij,t-1}$ then set $\hat{d}_{ikt} = 1$; otherwise set $\hat{d}_{ikt} = 0$.
         B. If $\sum_{j \in V_k} \hat{x}_{ijt} > \sum_{j \in V_k} \hat{x}_{ij,t-1}$ then set $\hat{b}_{ikt} = 1$; otherwise set $\hat{b}_{ikt} = 0$.
   (e) If $i \in G$ then calculate under- and over-time costs, as follows:
      i. If $\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt} < g_i$ then set $\hat{u}_i = g_i - \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt} \right)$ and $\hat{o}_i = 0$; else, set $\hat{o}_i = \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt} \right) - g_i$ and $\hat{u}_i = 0$.

ii. Using the input data of the overtime and undertime values for the previous week's solution, i.e. o_i^* and u_i^* respectively, infer the *change* of overtime and undertime for this 'evaluating' solution. This is done by setting $o_i^{\pm} = \hat{o}_i - o_i^*$ and$u_i^{\pm} = \hat{u}_i - u_i^*$ respectively.

iii. Using the undertime and overtime cost rates (c_i^U and c_i^O respectively), add the cost of this change to the cost of employee i's roster.

(f) Infer the change variables under this 'evaluating' solution for allocation ($x_{ijt}^{\pm}$), boarding and departing ($b_{ikt}^{\pm}$ and $d_{ikt}^{\pm}$) and consecutive working $l_{\lambda ijt}^{\pm}$, and add the associated costs to the cost of roster Υ_i in this 'evaluating' solution.

5. To calculate the agency employee costs, for each role $j \in J$ do the following:

(a) If role j is not in the set of those whose agency allocation set Θ_j has changed since the last iteration, then copy relevant solution values designated for 'current' solution to those designated for 'evaluating' solution:

- Decision variable values for allocation ($\hat{x}_{m+1,j,t}$), consecutive working ($\hat{l}_{\lambda,m+1,j,t}$) and boarding and departing the vessels ($\hat{\beta}_{jt}$ and $\hat{\delta}_{jt}$).
- The associated crew change set Ξ_j.
- The change cost of set Θ_j in conjunction with set Ξ_j.

And move onto next role. Otherwise, carry out remainder of this step.

(b) Clear the recorded cost for the combination of sets Θ_j and Ξ_j in the 'evaluating' solution, and record whether an agency employee starts in role j (i.e. the value of σ_j).

(c) As set Θ_j has changed since the last iteration, we need to re-evaluate the contents of set Ξ_j. There are two kinds of time points of interest: those at which a crew change will definitely take place, and those at which a crew change will possibly take place. We will ensure the contents of both the 'definite' set and the 'possible' set are cleared.

(d) For each time point t from $t = 1$ to $t = T$, do the following:

i. If $t \in \Xi_j$ then set $\hat{x}_{m+1,j,t} = 1$ for this 'evaluating' solution; otherwise, set $\hat{x}_{m+1,j,t} = 0$.

ii. If agency crew allocation to the role at time t is different from that at time $t-1$ (i.e. if $\hat{x}_{m+1,j,t} \neq \hat{x}_{m+1,j,t-1}$ for $t \geq 2$, or if $\hat{x}_{m+1,j,t} \neq sigma_j$ for $t = 1$), then add t to the set of 'definite' crew change points.

iii. If agency crew is allocated to role j at both time t and $t-1$ (i.e. if $\hat{x}_{m+1,j,t} = \hat{x}_{m+1,j,t-1} = 1$ for $t \geq 2$, or if $\hat{x}_{m+1,j,t} = sigma_j = 1$ for $t = 1$), then add t to the set of 'possible' crew change points.

(e) If we are not calculating the cost of the initial solution, compare the set Θ_j for this 'evaluating' solution with that for the best solution found so far. If their contents are identical, then set the contents of set Ξ_j for the 'evaluating' solution equal to the crew change set Ξ_j in the best solution.

(f) Otherwise, if the calculation parameter is not set to 'current', and the set Θ_j for this 'evaluating' solution is identical to the set Θ_j for the 'current' solution, then set the contents of set Ξ_j for the 'evaluating' solution equal to the crew change set Ξ_j in the 'current' solution.

(g) Otherwise, if there have been no 'possible' crew change point identified, define the crew change set Ξ_j for the 'evaluating' solution as comprising (only) the 'definite' crew change points identified above.

(h) Otherwise, there are 2^x possibilities to consider, where x is the number of time points identified for a 'possible' crew change. Each number from 1 to 2^x can be represented as an x-digit binary number, with each of the x digits being associated with a different one of the 'possible' crew change points. Using this principle, for each number from 1 to 2^x, do the following:

   i. Set the cost of this permutation to zero, set the consecutive work counter equal to α_{j0}, and assume the current permutation is feasible.

   ii. For each 'possible' crew change point, if the associated binary digit of the current number is 1 then this crew change will take place under this permutation; if the digit $= 0$ then the crew change will *not* take place in this permutation.

   iii. For each time t from $t = 1$ to $t = T$, do the following:

      A. If time t is a 'possible' crew change which is to take place under this permutation then denote this time point as having a possible board and possible departure of an employee and reset the consecutive work counter to zero; otherwise denote no possible boarding or departing at this time point.

      B. If time t is a 'possible' crew change (whether it is to take place or not in this permutation), then use the possible board and possible depart quantity denoted in the previous step to infer the possible changes to the agency board and depart if this permutation were to be accepted, and add the cost of these possible changes to the cost of the permutation.

      C. If $\hat{x}_{m+1,j,t} = 1$ for the 'evaluating' solution then: increment the consecutive work value by 1; if this new value exceeds α_j^{max} then denote this permutation as infeasible; if the consecutive work value is not greater than α_j^{max} then record the possible long work indicator (analogous with

$\hat{l}_{\lambda,m+1,j,t}$) equal to 1 for all work lengths less than or equal to the consecutive work value, and set the possible indicator equal to zero for all other lengths.

   D. If $\hat{x}_{m+1,j,t} = 0$ for the 'evaluating' solution then reset the consecutive work counter to zero, and set the possible long work indicator equal to zero for all lengths.

   E. For all lengths, compare the possible long work indicators described and compare with the input data values $l^*_{\lambda,m+1,j,t}$. If there are any changes, denote these as possible changes and add the cost of these possible changes to the cost of this permutation.

  iv. If this permutation is feasible and we have not yet found a feasible crew change pattern, then define the crew change set Ξ_j for the 'evaluating' solution as comprising exactly the 'possible' crew changes which are to take place under this permutation plus the 'definite' crew change points identified, and record the cost of this permutation.

  v. Otherwise, if the permutation is feasible and we have already allocated values to the set Ξ_j for the 'evaluating' solution, then look at the recorded cost of the set Ξ_j. If that cost is greater than the cost of the current permutation, then replace the contents of Ξ_j with the 'possible' crew changes for this permutation, plus the 'definite' crew change points, and update the recorded cost.

(i) Having now identified the optimal contents of Ξ_j for the 'evaluating' solution, we can recalculate the change cost of set Θ_j in conjunction with set Ξ_j. Firstly, set the consecutive work counter equal to α_{j0}.

(j) For each time t from from $t = 1$ to $t = T$, do the following:

  i. Set the agency boarding and departing variable ($\hat{\beta}_{jt}$ and $\hat{\delta}_{jt}$) for this 'evaluating' solution equal to zero.

  ii. If $t \in \Xi_j$ then reset the consecutive work counter to zero.

  iii. If $t \in \Xi_j$ and agency crew is allocated to the role in the preceding period (i.e. if $\hat{x}_{m+1,j,t-1} = 1$ for $t \geq 2$, or if $sigma_j = 1$ for $t = 1$) then set $\hat{\delta}_{jt} = 1$ for the 'evaluating' solution.

  iv. If $t \in \Xi_j$ and $\hat{x}_{m+1,j,t} = 1$ then set $\hat{\beta}_{jt} = 1$ for the 'evaluating' solution.

  v. If $\hat{x}_{m+1,j,t} = 1$ then increment the consecutive work counter by 1; set $\hat{l}_{\lambda,m+1,j,t} = 1$ for all λ less than or equal to the consecutive work count value, and set it equal to zero for all other values of λ. If $\hat{x}_{m+1,j,t} = 0$ then reset the consecutive work counter to zero.

vi. Infer the change variables under this 'evaluating' solution for allocation ($x^{\pm}_{m+1,j,t}$), boarding and departing ($\beta^{\pm}_{jt}$ and $\delta^{\pm}_{jt}$) and consecutive working $l^{\pm}_{\lambda,m+1,j,t}$, and add the associated costs to the change cost of set Θ_j in conjunction with set Ξ_j in this 'evaluating' solution.

6. We can now define the cost of this 'evaluating' solution to be the sum of the costs associated with each roster Υ_i for all $i \in E_R$, and each combination of sets Θ_j and Ξ_j for all $j \in J$.

7. Set transfer parameters as 'from: evaluating; to: the calculation parameter value', and call **'Transfer solution'** procedure (page 372).

8. If the calculation parameter refers to a 'backward', 'forward', 'swap' or 'kick' solution then call **'Check feasibility'** procedure (page 377).

C.3.2.5 Check feasibility

1. Assume the 'evaluating' solution is feasible.

2. For the values associated with the 'evaluating' solution, for all roles r required at each time t (i.e. all role-time pairs for which $a_{jt} > 0$), check that job cover constraints (6.39) are satisfied. If the left and right hand sides are not equal for any of the equations, then the 'evaluating' solution is infeasible.

3. If the solution is still believed to be feasible, then for all regular employees $i \in E_R$ whose roster has changed from the current solution and for all time points t check the overlapping task constraints (6.40) are satisfied. If, using the values for the 'evaluating' solution, $\sum\limits_{j \in J} \hat{x}_{ijt} > 1$ for any of the equations, then the 'evaluating' solution is infeasible.

4. If the solution is still believed to be feasible, then for all vessels $k \in K$ and all regular employees $i \in E_R$ whose roster has changed from the current solution we will check that the constraints on employees boarding the vessels are satisfied. This involves two checks:

   (a) For $t = 1$, we should check constraints (6.41) for all employees whose roster has been changed. If, using the values for the 'evaluating' solution, the evaluated left and right hand sides do not satisfy an inequality, then the 'evaluating' solution is infeasible.

   (b) For all $t > 1$, we should check constraints (6.42) for all employees whose roster has been changed. If, using the values for the 'evaluating' solution, the evaluated left and right hand sides do not satisfy an inequality, then the 'evaluating' solution is infeasible.

5. If the solution is still believed to be feasible, then for all vessels $k \in K$ and all regular employees $i \in E_R$ whose roster has changed from the current solution we will check that the constraints on employees departing the vessels are satisfied. This also involves two checks:

   (a) For $t = 1$, we should check constraints (6.43) for all employees whose roster has been changed. If, using the values for the 'evaluating' solution, the evaluated left and right hand sides do not satisfy an inequality, then the 'evaluating' solution is infeasible.

   (b) For all $t > 1$, we should check constraints (6.44) for all employees whose roster has been changed. If, using the values for the 'evaluating' solution, the evaluated left and right hand sides do not satisfy an inequality, then the 'evaluating' solution is infeasible.

6. If the solution is still believed to be feasible, then for all roles $j \in J$ which have had their agency allocation set changed from the current solution we will check that the agency boarding and departing constraints are satisfied. Similarly, this involves two checks:

   (a) For $t = 1$, we should check constraints (6.45) for all roles where the agency allocation set has changed. If, using the values for the 'evaluating' solution, the evaluated left and right hand sides are not equal for any of the equations, then the 'evaluating' solution is infeasible.

   (b) For all $t > 1$, we should check constraints (6.46) for all roles where the agency allocation set has changed. If, using the values for the 'evaluating' solution, the evaluated left and right hand sides are not equal for any of the equations, then the 'evaluating' solution is infeasible.

7. If the solution is still believed to be feasible, then for all fixed contract employees $i \in G$ whose roster has changed from the current solution, check that the undertime calculation (6.47) constraints are satisfied. If, using the values for the 'evaluated' solution, the calculated left and right hand sides do not satisfy an inequality, then the 'evaluating' solution is infeasible.

8. If the solution is still believed to be feasible, then for all fixed contract employees $i \in G$ whose roster has changed from the current solution, check that the overtime calculation (6.48) constraints are satisfied. If, using the values for the 'evaluated'

solution, the calculated left and right hand sides do not satisfy an inequality, then the 'evaluating' solution is infeasible.

9. If the solution is still believed to be feasible, then for all regular employees $i \in E_R$ whose roster has changed from the current solution, check that the consecutive work constraints are satisfied. This is done as follows:

   (a) For each time point t, calculate a value for the variable $\hat{w}_{it}$. This can be done by assigning $\hat{w}_{it}$ the lowest non-negative integer value which satisfies inequality (6.49) when calculated using the input data and the values for the 'evaluating' solution.

   (b) For each time point t, all roles $j \in J$ and work length value $\lambda \leq w_i^{max}$, check that the work duration constraints (6.50) are satisfied. If, using the values for the 'evaluated' solution and the values of $\hat{w}_{it}$ from above, the calculated left and right hand sides do not satisfy an inequality, then the 'evaluating' solution is infeasible.

10. If the solution is still believed to be feasible, then for all roles $j \in J$ which have had their agency allocation set changed from the current solution, check that the agency consecutive work constraints are satisfied. This is done as follows:

    (a) For each time point t, calculate a value for the variable $\hat{\alpha}_{jt}$. Firstly, making use of constraints (6.51), and using the relevant values for the 'evaluating' solution, set
    $$\hat{\alpha}_{jt} = \hat{\alpha}_{j,t-1} + \hat{x}_{m+1,jt} - \alpha_j^{max}\hat{\delta}_{jt}$$
    Then, in order to satisfy constraints (6.51), if $\hat{\alpha}_{jt} < \hat{x}_{m+1,jt}$ then we set $\hat{\alpha}_{jt} = \hat{x}_{m+1,jt}$.

    (b) For each time point t and work length value $\lambda \leq \alpha_j^{max}$, check that the work duration constraints (6.53) are satisfied. If, using the values for the 'evaluated' solution and the values of $\hat{\alpha}_{jt}$ from above, the calculated left and right hand sides do not satisfy an inequality, then the 'evaluating' solution is infeasible.

11. If the solution is still believed to be feasible, then for all regular employees $i \in E_R$ whose roster has changed from the current solution, check that the rest constraints are satisfied. This is done as follows:

    (a) For each time point t, calculate a value for the variable $\hat{r}_{it}$. Firstly, making use of constraints (6.54), and using the relevant values for the 'evaluating' solution, set

$$\hat{r}_{it} = \hat{r}_{i,t-1} - \left(1 - \sum_{j \in J} \hat{x}_{ijt}\right)$$

Then, in order to satisfy constraints (6.55), if this calculated value of $\hat{r}_{it}$ is less than $(\rho_i - 1) \sum\limits_{k \in K} \hat{d}_{ikt}$ then we set

$$\hat{r}_{it} = (\rho_i - 1) \sum_{k \in K} \hat{d}_{ikt}$$

(b) For each time point t, check that the constraints linking work and rest duration (6.56) are satisfied. If, using the values for the 'evaluated' solution and the values of $\hat{r}_{it}$ from above, the calculated left and right hand sides do not satisfy an inequality, then the 'evaluating' solution is infeasible.

12. If the solution is still believed to be feasible, then check that the constraints linking the new schedule and the changes required to create it under the 'evaluating' solution with the data of the previous weeks' schedule are satisfied. This is done as follows:

(a) For each employee $i \in E_R$ whose roster has changed from the current solution and all roles $j \in J$ and time t, and for agency employees for all roles $j \in J$ which have had their agency allocation set changed from the current solution and all time points t, check the allocation variable constraints (6.57). If the values from the 'evaluating' solution do not satisfy an equation, then the solution is infeasible.

(b) For each employee $i \in E_R$ whose roster has changed from the current solution and all vessels $k \in K$ and all time points t, check the boarding variable constraints (6.58) and departing variable constraints (6.59). If the values from the 'evaluating' solution do not satisfy an equation, then the solution is infeasible.

(c) For each role $j \in J$ which has had its agency allocation set changed from the current solution and all time points t, check the agency boarding variable constraints (6.60) and agency departing variable constraints (6.61). If the values from the 'evaluating' solution do not satisfy an equation, then the solution is infeasible.

(d) For each employee $i \in E_R$ whose roster has changed from the current solution and all roles $j \in J$, times t and consecutive work values λ; and for agency employees for all roles $j \in J$ which have had their agency allocation set changed from the current solution and all time points t and consecutive work values λ, check the consecutive work variable constraints (6.57). If the values from the 'evaluating' solution do not satisfy an equation, then the solution is infeasible.

(e) For each fixed contract employee $i \in G$ whose roster has changed from the current solution, check the undertime variable constraints (6.63) and overtime variable constraints (6.64). If the values from the 'evaluating' solution do not satisfy an equation, then the solution is infeasible.

13. If the solution is still believed to be feasible, then check that the variable values conform to the definitions given in the formulation. This is done as follows:

    (a) For each employee $i \in E_R$ whose roster has changed from the current solution and all roles $j \in J$ and time t, and for agency employees for all roles $j \in J$ which have had their agency allocation set changed from the current solution and all time points t, check that the allocation variables $\hat{x}_{ijt}$ and $x^{\pm}_{ijt}$ satisfy condition (6.65). If any of these variables under the 'evaluating' solution is not $\in \{0,1\}$, then the solution is infeasible.

    (b) For each employee $i \in E_R$ whose roster has changed from the current solution and all roles $j \in J$, times t and consecutive work values λ; and for agency employees for all roles $j \in J$ which have had their agency allocation set changed from the current solution and all time points t and consecutive work values λ, check that the consecutive work variables $\hat{l}_{\lambda ijt}$ and $l^{\pm}_{\lambda ijt}$ satisfy condition (6.66). If any of these variables under the 'evaluating' solution is not $\in \{0,1\}$, then the solution is infeasible.

    (c) For each employee $i \in E_R$ whose roster has changed from the current solution and all vessels $k \in K$ and all time points t, check that the boarding variables $\hat{b}_{ikt}$ and $b^{\pm}_{ikt}$ and departing variables $\hat{d}_{ikt}$ and $d^{\pm}_{ikt}$ satisfy condition (6.67). If any of these variables under the 'evaluating' solution is not $\in \{0,1\}$, then the solution is infeasible.

    (d) For each role $j \in J$ which has had its agency allocation set changed from the current solution and all time points t, check that the agency boarding variables $\hat{\beta}_{jt}$ and $\beta^{\pm}_{jt}$ and agency departing variables $\hat{\delta}_{jt}$ and $\delta^{\pm}_{jt}$ satisfy condition (6.68). If any of these variables under the 'evaluating' solution is not $\in \{0,1\}$, then the solution is infeasible.

    (e) For each fixed contract employee $i \in G$ whose roster has changed from the current solution, check that the undertime variables $\hat{u}_i$ and overtime variables $\hat{o}_i$ satisfy condition (6.69). If any of these variables under the 'evaluating' solution are < 0, then the solution is infeasible.

    (f) For each employee $i \in E_R$ whose roster has changed from the current solution and all time points t, check that the calculated work resource values $hatw_{it}$ and rest resource values $\hat{r}_{it}$ satisfy condition (6.70). If any of these quantities have been calculated to be < 0, then the solution is infeasible.

14. If the 'evaluating' solution is now considered to be infeasible, then increment the number of infeasible solutions found by one.

C.3.2.6 Find usable block

1. Record that no update has yet been done and no 'candidate' solution has yet been found, and clear the set of employees changed for the 'candidate' solution.
2. For each regular employee i on the short ordered list of employees, do the following:
   (a) If no update has yet been done, then clear the record of selected employee and details of block (i.e. role, start time, end time and length), and record that no block has yet been found.
   (b) If no update has yet been done then for each vessel $k \in K$, look for whether employee i starts on board the vessel. If so, then record a block as having been found. This is a *current* block, as defined in section 6.5.2, for selected employee i with start time $t^S = (1 - w_{i0})$ and an as yet unknown duration. It takes place on vessel k, although the role is not yet known. Also, denote the earliest time at which a swapping block can start as time = 1.
   (c) While no update has yet been done, for each time t from $t = 1$ to $t = T$, do the following:
      i. Record that no *new* block has been found.
      ii. If no block has yet been found and in roster Υ_i in the current solution element $j_t^i \neq rest$ then record a *new* block as having been found. This is a *planned* block, as defined in section 6.5.2, for selected employee i in role j_t^i (and hence the vessel on board which it takes place is also known), with start time $t^S = t$ and an as yet unknown duration. The earliest start time for a block to be swapped with this one is t if $t = 1$ or if $(t-1) \leq r_{i0}$ (as per condition (6.153)); and $(t-1)$ otherwise.
      iii. Otherwise, if a block has already been found and in roster Υ_i in the current solution element $j_t^i \neq rest$ then must check this is part of the same block. If j_t^i is the same role as that of the block already found, then no action need be taken; however, if j_t^i is not equal to the role of the block found, then should check whether these roles take place on the same vessel k, and proceed as follows:

A. If the roles do take place on the same vessel, then consider these to be the same block, and denote j_t^i as the role of the block.

B. If they do not come from the same vessel, then identify $(t-1)$ as the end time of the block, calculate block length, and denote $t-1$ also as the latest time that a swapping block can end. If the role for this block is known, then call **'Evaluate block'** procedure (page 384); if only the vessel is known then if the block length is less than w_i^{max} then for all roles $j \in V_k$ set j as the role for the block and call **'Evaluate block'** procedure (page 384). A new *planned* block is also starting, so record the details of this block as being for selected employee i in role j_t^i (and hence the vessel on board which it takes place is also known), with start time $t^S = t$ and earliest start time for a block to be swapped also time t. End time and length are as yet unknown, so clear the record of these.

iv. Otherwise, if a block has already been found and in roster Υ_i in the current solution element $j_t^i = rest$, then record $(t-1)$ as the end time of the block, calculate block length, and denote time t as the latest time that a swapping block can end. If the role for this block is known, then call **'Evaluate block'** procedure (page 384); if only the vessel is known then if the block length is less than w_i^{max} then for all roles $j \in V_k$ set j as the role for the block and call **'Evaluate block'** procedure (page 384). Finally, reset record of block start time, end time and duration, selected employee and role and vessel; and record that no block has been found at this stage.

v. If a *new* block is recorded as having been found, then update record to reflect that a block has been found and that no *new* block has been found.

vi. If $t = T$ and a block has been found, then record this block as ending at time T, calculate block length, and also denote time T as the latest time that a swapping block can end. If the role for this block is known, then call **'Evaluate block'** procedure (page 384); if only the vessel is known then if the block length is less than w_i^{max} then for all roles $j \in V_k$ set j as the role for the block and call **'Evaluate block'** procedure (page 384).

(d) If no update has been done, then for all regular employees $i' \in E_R$ such that $i' \neq i$, add employee i' to the set of those who have been examined for swapping blocks with current employee i.

3. If no update has been done, and a 'candidate' solution exists, then do the following:

(a) Record the rosters Υ_i for all $i \in E_R$ and agency sets Θ_j for all $j \in J$ for the 'current' solution as now being the 'tabu' solution.

(b) If the cost of this 'candidate' solution is less than that of the 'current' solution, then reset the number of non-reducing iterations to zero; otherwise increase the number of non-reducing iterations by one.

(c) Set transfer parameters as 'from: candidate; to: current', and call **'Transfer solution'** procedure (page 372). Record than an update has now been done, and increase by one the count of candidate solutions used.

(d) Call **'Compare to best'** procedure (page 402).

(e) Denote the set of employees to be updated as the set of employees changed in this candidate solution, and call **'Update swaps and changes'** procedure (page 403)

C.3.2.7 Evaluate block

1. Reset costs of backward extension, forward extension and swap to zero; and note that at this stage no backward extension, forward extension or swap is to be carried out.

2. Find the maximum extension length Λ^E for a backwards extension, i.e. the largest non-negative integer value which satisfies conditions (6.164), (6.165), (6.167) and (6.169), relating respectively to block length, requirement, eligibility and initial rest periods. If $\Lambda^E > 0$ for the backwards extension then call **'Evaluate backwards'** procedure (page 385).

3. If a backward extension is not to be carried out, then find the maximum extension length Λ^E for a forwards extension, i.e. the largest non-negative integer value which satisfies conditions (6.164), (6.166), (6.168) and (6.169), relating respectively to block length, requirement, eligibility and initial rest periods, and also conditions (6.171) and (6.172) relating to rest periods and the employee's location at time zero. If $\Lambda^E > 0$ for the backwards extension then call **'Evaluate forwards'** procedure (page 388).

4. If neither a backward nor forward extension is to be carried out, and if the selected block has start time $t^S > 0$, then call **'Evaluate swap'** procedure (page 392).

5. If a backward extension is to be carried out, set the transfer parameter as 'from: backward' and increase by one the count of backward extensions implemented; otherwise, if a forward extension is to be carried out, set the transfer parameter as 'from: forward' and increase by one the count of forward extensions implemented; otherwise, if a swap is to be carried out, set the transfer parameter as 'from: swap' and increase by one the count of swaps implemented.

6. If one of a backward extension, forward extension or swap is to be carried out, then do the following:

   (a) Record the rosters Υ_i for all $i \in E_R$ and agency sets Θ_j for all $j \in J$ for the 'current' solution as now being the 'tabu' solution.

   (b) Set transfer parameters as 'to: current' and call **'Transfer solution'** procedure (page 372). Record than an update has now been done.

   (c) Call **'Compare to best'** procedure (page 402).

C.3.2.8 Evaluate backwards

1. Set extension length equal to calculated maximum, i.e. set $\lambda^E = \Lambda^E$.

2. Clear the list of employees whose rosters have changed and the list of roles which have had their agency assignments changed from the current solution.

3. Copy all rosters $\Upsilon_{i'}$ for all $i' \in E_R$ such that $i' \neq i$ and agency sets Θ_j for all $j \in J$ from the 'current' solution into those designated for the 'backward' solution.

4. For selected employee i, clear the contents of roster Υ_i for the 'backward' solution, and add i to the set of employees whose roster has changed from the current solution.

5. Set the initial count of rest periods to zero, and the initial working length count equal to w_{i0}, and clear the list of role-time pairs which are being held 'in reserve'.

6. For each time t from $t = 1$ to $t = T$, do the following:

   (a) If t is not greater than the block end time, i.e. $t \leq (t^S + \lambda^B - 1)$, then j_t^i is assigned the same value for this 'backward' solution as it takes for the 'current' solution. If $j_t^i = rest$ then reset the length count to zero; otherwise, increase its value by one.

   (b) Otherwise, if t is within the period of extension of the block, i.e. $t \leq (t^S + \lambda^B + \lambda^E - 1)$, then j_t^i for this 'backward' solution is assigned the role of the extending block and the length count value is increased by one. If j_t^i for the 'current' solution is not a *rest* task and is not the extending role, then add the 'current' role of j_t^i at time t to the reserve list.

   (c) Otherwise $t > (t^S + \lambda^B - 1)$, and if the $j_t^i \neq rest$ in the 'current' solution and either the rest count is less than ρ_i or the length count is greater than w_i^{max}, then set $j_t^i = rest$ for this 'backward' solution. Add the 'current' role of j_t^i at time t to the reserve list, increase the rest count value by one, and reset the length count to zero.

(d) Otherwise, j_t^i is assigned the same value for this 'backward' solution as it takes in the 'current' solution. If $j_t^i = rest$ then increase the rest count by one and reset the length count to zero; otherwise increase the length count by one.

7. Set the count of conflicting tasks found to zero.

8. For each time t from $t = (t^S + \lambda^B)$ to $t = (t^S + \lambda^B + \lambda^E - 1)$, look for whether $t \in \Theta_j$, where j is the role of the selected block. If t is in this set, then remove it for this 'backward' solution, record j as a role that has had its agency assignment changed from the current solution, and increase the number of conflicting tasks found by one.

9. For each role $j \in J$ which has time periods on the reserve list, do the following:

   (a) For each time point $t > 1$ on the reserve list for role j, look for whether time $(t - 1) \in \Theta_j$ in this 'backward' solution. If so, then add t to set Θ_j for the 'backward' solution, record j as a role that has had its agency assignment changed from the current solution, and remove time t for role j from the reserve list.

10. For each employee $i' \in E_R$ such that i' is not selected employee i, do the following:

    (a) If the number of conflicting tasks found is equal to λ^E and there are no remaining role-time pairs on the reserve list, then proceed to the next employee; otherwise, continue with this loop.

    (b) Clear the contents of roster $\Upsilon_{i'}$ for this 'backward' solution and set the initial length count equal to $w_{i'0}$, and clear the record of the next role to be added. If $w_{i'0}$ is non-zero, then record that there is a previous working period and set the initial rest count to zero; otherwise, record that there is no previous working period, and set the initial rest count equal to $(\rho_{i'} - r_{i'0})$.

    (c) For each time t from $t = 1$ to $t = T$, do the following:

       i. If t is not greater than the block end time, i.e. $t \leq (t^S + \lambda^B - 1)$, then $j_t^{i'}$ is assigned the same value for this 'backward' solution as it takes for the 'current' solution. If $j_t^{i'} = rest$ then reset the length count to zero, increase the rest count by one, and record that there is no previous working period; otherwise, reset the rest count to zero, increase the length count by one, and record that there is a previous working period.

       ii. Otherwise, if $j_t^{i'} = rest$ in the 'current' solution then do the following:

          A. If there is a record of a next role j' to be added, then set $j_{i'}^t = j'$ for the 'backward' solution, remove j' at time t from the reserve list, and add employee i' to the list of employees whose roster has changed from the current solution. Increase the value of the length count by one, record that this a previous working task, and reset the rest count to zero. If $t = T$ or the length count is $\geq w_{i'}^{max}$ or employee i' is not eligible to perform role j' in period $(t+1)$ or role j' at time $(t+1)$ is not on the reserve list, then clear the record of the next role to be added.

B. If there is not a record of a next role to be added, then set $j_{i'}^{t} = rest$, reset the length count to zero, increase the rest count value by one, and record that there is no previous working task.

iii. Otherwise (i.e. if $j_t^{i'} \neq rest$ in the 'current' solution) then do the following:

A. If $j_t^{i'}$ is the same role as the extended block and t is within the period of extension of the block (i.e. $t \leq (t^S + \lambda^B + \lambda^E - 1)$), then set $j_t^{i'} = rest$ for this 'backward' solution, increase the number of conflicting tasks found by one, and add i' to the set of employees whose roster has changed from the current solution. Reset length count to zero, increase the value of the rest count by one, and record that there is not a previous working task.

B. Otherwise, if there is not a previous working task and the rest count value is less than $\rho_{i'}$ then set $j_t^{i'} = rest$ for this 'backward' solution, and add time period t to the relevant agency employee set instead. record this role (the value of $j_t^{i'}$ in the 'current' solution) as having had its agency assignments changed from the current solution, and increase the rest count value by one.

C. Otherwise, $j_t^{i'}$ is assigned the same value for this 'backward' solution as it takes in the 'current' solution. Increase the value of the length count by one, reset the rest count to zero, and record that there is a previous working task. If $t = T$ or the length count is $\geq w_{i'}^{max}$ or employee i' is not eligible to perform role $j_t^{i'}$ in period $(t+1)$ or role $j_t^{i'}$ at time $(t+1)$ is not on the reserve list, then clear the record of the next role to be added; otherwise, record role $j_t^{i'}$ as the next role to be added.

11. For each role $j \in J$ which still has time periods on the reserve list, add these time periods to agency set Θ_j, and record role j as having had its agency assignments changed from the current solution.

12. Set tabu check parameter to 'backward' and call **'Check tabu'** procedure (page 401).

13. If this 'backward' solution is found to be tabu, increase the count of tabu solutions by one; otherwise, do the following:

(a) Set calculation parameter to ‘backward’, and call **‘Calculate cost’** procedure (page 372).

(b) If the solution is not feasible, skip to the final step of this procedure; otherwise, continue.

(c) Calculate the cost difference between this solution and the current solution.

(d) If the total cost of this solution is less than that of the solution type defined in section 6.5.4, then note that a backward extension should be carried out, and reset the number of non-reducing solutions to zero. Denote the set of employees to be updated as the set of employees changed from the current solution, and call **‘Update swaps and changes’** procedure (page 403).

(e) Otherwise, if no ‘candidate’ solution exists or the cost of the ‘backward’ solution is less than that of the existing candidate then record that a candidate solution now does exist. Set transfer parameters as ‘from: backward; to: candidate’, and call **‘Transfer solution’** procedure (page 372). Store the cost of this new candidate solution, and add each emloyee changed from the current solution in this ‘backward’ solution to the set of employees changed in the ‘candidate’ solution.

14. If no backward extension is to be carried out and $\lambda^E > 1$, reduce value of λ^E by one and return to **Step 2** of this procedure; otherwise **end** this procedure.

C.3.2.9 Evaluate forwards

1. Set extension length equal to calculated maximum, i.e. set $\lambda^E = \Lambda^E$.

2. Clear the list of employees whose rosters have changed and the list of roles which have had their agency assignments changed from the current solution.

3. Copy all rosters $\Upsilon_{i'}$ for all $i' \in E_R$ such that $i' \neq i$ and agency sets Θ_j for all $j \in J$ from the ‘current’ solution into those designated for the ‘forward’ solution.

4. For selected employee i, clear the contents of roster Υ_i for the ‘forward’ solution, and add i to the set of employees whose roster has changed from the current solution.

5. Set the initial count of rest periods to zero and clear the list of role-time pairs which are being held ‘in reserve’.

6. For each time t in reverse order from $t = T$ to $t = 1$, do the following:

   (a) If t is not less than the block start time, i.e. $t \geq t^S$, then j^i_t is assigned the same value for this ‘forward’ solution as it takes for the ‘current’ solution.

(b) Otherwise, if t is within the period of extension of the block, i.e. $t \geq (t^S - \lambda^E$, then j^i_t for this 'forward' solution is assigned the role of the extending block. If j^i_t for the 'current' solution is not a *rest* task and is not the extending role, then add the 'current' role of j^i_t at time t to the reserve list.

(c) Otherwise $t < (t^S - \lambda^E)$, and if the $j^i_t \neq rest$ in the 'current' solution and the rest count is less than ρ_i, then set $j^i_t = rest$ for this 'forward' solution. Add the 'current' role of j^i_t at time t to the reserve list, increase the rest count value by one, and reset the length count to zero.

(d) Otherwise, j^i_t is assigned the same value for this 'forward' solution as it takes in the 'current' solution. If $j^i_t = rest$ then increase the rest count by one.

7. Set the count of conflicting tasks found to zero.

8. For each time t (in reverse chronological order) from $t = (t^S - 1)$ to $t = (t^S - \lambda^E)$, look for whether $t \in \Theta_j$, where j is the role of the selected block. If t is in this set, then remove it for this 'forward' solution, record j as a role that has had its agency assignment changed from the current solution, and increase the number of conflicting tasks found by one.

9. For each role $j \in J$ which has time periods on the reserve list, do the following:

   (a) For each time point $t < T$ on the reserve list for role j, look for whether time $(t+1) \in \Theta_j$ in this 'forward' solution. If so, then add t to set Θ_j for the 'forward' solution, record j as a role that has had its agency assignment changed from the current solution, and remove time t for role j from the reserve list.

10. For each employee $i' \in E_R$ such that i' is not selected employee i, do the following:

    (a) If the number of conflicting tasks found is equal to λ^E and there are no remaining role-time pairs on the reserve list, then proceed to the next employee; otherwise, continue with this loop.

    (b) Clear the contents of roster $\Upsilon_{i'}$ for this 'forward' solution and set the initial length count to zero, and clear the record of the next role to be added. Record that there is no previous working period and set the initial rest count equal to $\rho_{i'}$).

    (c) For each time t in reverse order from $t = T$ to $t = 1$, do the following:

       i. If t is not earlier than the block start time, i.e. $t \geq (t^S + \lambda^B - 1)$, then $j^{i'}_t$ is assigned the same value for this 'forward' solution as it takes for the 'current' solution. If $j^{i'}_t = rest$ then reset the length count to zero and record that there is no previous working period; otherwise, increase the length count by one and record that there is a previous working period.

ii. Otherwise, if $j_t^{i'} = rest$ in the 'current' solution then do the following:

A. If there is a record of a next role j' to be added, then set $j_{i'}^t = j'$ for the 'forward' solution, remove j' at time t from the reserve list, and add employee i' to the list of employees whose roster has changed from the current solution. Increase the value of the length count by one, record that this a previous working task, and reset the rest count to zero. Under the following criteria, clear the record of the next role to be added:

- If $t = 1$ or the length count is $\geq w_{i'}^{max}$; or
- If employee i' is not eligible to perform role j' in period $(t-1)$, or if role j' at time $(t-1)$ is not on the reserve list, or if $(t-1) \leq r_{i'0}$; or
- If $(t-1) \leq \rho_{i'}$ *and* $\sum_{k \in K : j' \notin V_k} s_{i'k} = 1$; or
- If $(t-1) \geq 2$ *and* $(t-1) \leq \rho_{i'}$ *and* $\sum_{k \in K : j' \in V_k} s_{i'k} = 1$.

B. If there is not a record of a next role to be added, then set $j_{i'}^t = rest$, reset the length count to zero, increase the rest count value by one, and record that there is no previous working task.

iii. Otherwise (i.e. if $j_t^{i'} \neq rest$ in the 'current' solution) then do the following:

A. If $j_t^{i'}$ is the same role as the extended block and t is within the period of extension of the block (i.e. $t \geq (t^S - \lambda^E)$, then set $j_t^{i'} = rest$ for this 'forward' solution, increase the number of conflicting tasks found by one, and add i' to the set of employees whose roster has changed from the current solution. Reset length count to zero, increase the value of the rest count by one, and record that there is not a previous working task.

B. Otherwise, if there is not a previous working task and the rest count value is less than $\rho_{i'}$ then set $j_t^{i'} = rest$ for this 'forward' solution, and add time period t to the relevant agency employee set instead. Record this role (the value of $j_t^{i'}$ in the 'current' solution) as having had its agency assignments changed from the current solution, and increase the rest count value by one.

C. Otherwise, $j_t^{i'}$ is assigned the same value for this 'forward' solution as it takes in the 'current' solution. Increase the value of the length count by one, reset the rest count to zero, and record that there is a previous working task. Under the following criteria, clear the record of the next role to be added:

- If $t = 1$ or the length count is $\geq w_{i'}^{max}$; or
- If employee i' is not eligible to perform role j' in period $(t-1)$, or if role j' at time $(t-1)$ is not on the reserve list, or if $(t-1) \leq r_{i'0}$; or
- If $(t-1) \leq \rho_{i'}$ *and* $\sum_{k \in K: j' \notin V_k} s_{i'k} = 1$; or
- If $(t-1) \geq 2$ *and* $(t-1) \leq \rho_{i'}$ *and* $\sum_{k \in K: j' \in V_k} s_{i'k} = 1$.

Otherwise, record role $j_t^{i'}$ as the next role to be added.

11. For each role $j \in J$ which still has time periods on the reserve list, add these time periods to agency set Θ_j, and record role j as having had its agency assignments changed from the current solution.

12. Set tabu check parameter to 'forward' and call **'Check tabu'** procedure (page 401).

13. If this 'forward' solution is found to be tabu, increase the count of tabu solutions by one; otherwise, do the following:

    (a) Set calculation parameter to 'forward', and call **'Calculate cost'** procedure (page 372).

    (b) If the solution is not feasible, skip to the final step of this procedure; otherwise, continue.

    (c) Calculate the cost difference between this solution and the current solution.

    (d) If the total cost of this solution is less than that of the solution type defined in section 6.5.4, then note that a forward extension should be carried out, and reset the number of non-reducing solutions to zero. Denote the set of employees to be updated as the set of employees changed from the current solution, and call **'Update swaps and changes'** procedure (page 403).

    (e) Otherwise, if no 'candidate' solution exists or the cost of the 'forward' solution is less than that of the existing candidate then record that a candidate solution now does exist. Set transfer parameters as 'from: forward; to: candidate', and call **'Transfer solution'** procedure (page 372). Store the cost of this new candidate solution, and add each employee changed from the current solution in this 'forward' solution to the set of employees changed in the 'candidate' solution.

14. If no forward extension is to be carried out and $\lambda^E > 1$, reduce value of λ^E by one and return to **Step 2** of this procedure; otherwise **end** this procedure.

C.3.2.10 Evaluate swap

1. Assume that an agency swap is permitted.
2. As per condition (6.150), check that agency crew are eligible to be assigned the block of work. If not, then an agency swap is not permitted.
3. If an agency swap is permitted, then for all roles $j' \in J$ such that $j' \neq j$ (the role of the selected block), do the following:
   (a) If a swap is not yet to be carried out, clear the record of the swapping block details (i.e. employee, role, start and end time, and duration), and record that no block has been found, a block has not been found that starts too early, and that a swap is currently allowed.
   (b) For each time t from $t = 1$ to $t = T$ while no swap is to be carried out, do the following:
      i. Record that no *new* block has yet been found.
      ii. If no block has yet been found and $t \in \Theta_{j'}$ in the 'current' solution and t is not greater than the latest time that a swapping block can end, then do the following:
         A. If t is earlier than the noted earliest time that the swapping block can start, record that the block is too early.
         B. Check conditions (6.159) and (6.160) relating to the selected employee's location at time zero. If these are not satisfied, then record than the block is too early.
         C. As per eligibility condition (6.151), if selected employee i cannot perform role j' at time t then record that a swap is not allowed.
         D. Record that a *new* block has been found, and that this is an agency block starting at time t and in role j'.
      iii. Otherwise, if a block has been found and $t \in \Theta_{j'}$ in the 'current' solution, then do the following:
         A. If $t \in \Xi_{j'}$ in the 'current' solution, then record the block as ending at time $t - 1$ and calculate the length λ'^{B}. If the block is not too early and the swap is allowed then, in accordance with condition (6.152) if $\lambda'^{B} \leq w_i^{max}$ then call **'Swap calculation'** procedure (page 396).
         B. Also if $t \in \Xi_{j'}$ in the 'current' solution, then if t is not greater than the latest time that a swapping block can end and a swap is not yet to be carried out, update the found block details to be role j' and starting at time t and clear the record of block end time and length. If t is

earlier than the noted earliest time that the swapping block can start, or conditions (6.159) and (6.160) relating to the selected employee's location at time zero are not satisfied, then record the block as too early. If selected employee i cannot perform role j' at time t (as per eligibility condition (6.151)), then record that the swap is not allowed.

C. If $t \in \Xi_{j'}$ in the 'current' solution but t is later than the latest time a swapping block can end or if a swap is already to be carried out, then record that no block has been found.

D. If $t \notin \Xi_{j'}$ in the 'current' solution, then if t is later than the latest time a swapping block can end record that no block has been found. If t is earlier (or equal to) this time, then if selected employee i cannot perform role j' at time t (as per eligibility condition (6.151)), record that the swap is not allowed.

iv. Otherwise, if a block has been found and $t \notin \Theta_{j'}$ in the 'current' solution, then do the following:

A. Record the block as ending at time $t-1$ and calculate the length λ'^B. If the block is not too early and the swap is allowed then, in accordance with condition (6.152) if $\lambda'^B \leq w_i^{max}$ then call **'Swap calculation'** procedure (page 396).

B. Record again that no block has been found.

C. If no swap is yet to be carried out, then reset the record of the swapping block details (i.e. employee, role, start and end time, and duration), and record that a block is not too early and that the swap is allowed.

(c) If a *new* block has been found, then record that a block has been found, and reset to say that another *new* block has not yet been found.

(d) If $t = T$ and a block has been found, then do the following:

i. As per eligibility condition (6.151), if selected employee i cannot perform role j' at time t then record that a swap is not allowed.

ii. Record the block as ending at time t and calculate the length λ'^B. If the block is not too early and the swap is allowed then, in accordance with condition (6.152) if $\lambda'^B \leq w_i^{max}$ then call **'Swap calculation'** procedure (page 396).

4. If the agency swap is permitted, and no swap is yet to be carried out, then record details of a *rest* block: the employee is agency, the role is $j' = rest$, and the start and end times are identical to those of the selected block. Call **'Swap calculation'** procedure (page 396).

5. For each regular employee $i' \in E_R$, do the following:

   (a) Assume that employee i' can be the swapping employee in this case.

   (b) If $i' = i$, then i' is the selected employee so cannot also be the swapping employee.

   (c) Check conditions (6.154), (6.155) and (6.156), relating respectively to eligibility, block length and required resting period at time zero, and also conditions (6.157), (6.162) and (6.163) relating to the starting location of employee i'. If any of these are not satisfied, then employee i' cannot be the swapping employee.

6. For each regular employee $i' \in E_R$ such that employee i' is permitted to be the swapping employee and i' is not in the set of employees who have already been examined for swapping with selected employee i, do the following:

   (a) If a swap is not yet to be carried out, clear the record of the swapping block details (i.e. employee, role and vessel, start and end time, and duration), and record that no block has been found, a block has not been found that starts too early, that a swap is currently allowed, and that the block found is so far a *rest* block.

   (b) For each time t from $t = 1$ to $t = T$ while no swap is to be carried out, do the following:

      i. If $j_t^{i'} \neq rest$ in the 'current' solution and t is not earlier than the earliest time the swapping block can start and t is not later than the latest time a swapping block can end, then record that the block is not a *rest* block.

      ii. Record that no *new* block has yet been found.

      iii. If no block has yet been found and $j_t^{i'} \neq rest$ in the 'current' solution and t is not greater than the latest time that a swapping block can end, then do the following:

         A. If t is earlier than the noted earliest time that the swapping block can start, record that the block is too early.

         B. Check conditions (6.159) and (6.160) relating to the selected employee's location at time zero. If these are not satisfied, then record than the block is too early.

         C. As per eligibility condition (6.151), if selected employee i cannot perform role j' at time t then record that a swap is not allowed.

         D. Record that a *new* block has been found, currently worked by employee i', starting at time t and in role $j' = j_t^{i'}$ (and the therefore the vessel is also known).

iv. Otherwise, if a block has been found and $j_t^{i'} \neq rest$ and $j_t^{i'} \neq j'$ in the 'current' solution, then should check whether roles $j_t^{i'}$ and j' take place on the same vessel k, and proceed as follows:

   A. If the roles do take place on the same vessel, then can consider this to be the same block. If t is later than the latest time a block can end, denote that no block has been found; otherwise, if employee i is not eligible to carry out role $j_t^{i'}$ at time t then record that this swap is not allowed; otherwise, denote $j_t^{i'}$ as the role of the block.

   B. If they do not come from the same vessel, then identify $(t-1)$ as the end time of the block and calculate block length. If the block length is not greater than w_i^{max} and the block found does not start too early and the swap is allowed, then call **'Swap calculation'** procedure (page 396).

   C. If they do not come from the same vessel, and t is not later than the latest time a block can end, and we are not yet to carry out a swap, then should record details of the new block starting. Record the new role as $j' = j_t^{i'}$ (and herefore the vessel is also known) and start time t, and clear the record of end time and block length. If t is earlier than the noted earliest time that the swapping block can start, record that the block is too early; otherwise, check conditions (6.159) and (6.160) relating to the selected employee's location at time zero and if these are not satisfied, then record than the block is too early; otherwise the block is not too early. As per eligibility condition (6.151), if selected employee i cannot perform role j' at time t then record that a swap is not allowed; otherwise, the block is allowed.

   D. If they do not come from the same vessel, but either t *is* later than the latest end time or we now have a swap to carry out, then record than no block has been found.

v. Otherwise, if a block has been found and $j_t^{i'} = j'$ in the 'current' solution, then do the following:

   A. If t is later than the latest time the swapping block can end, then record that no block has been found.

   B. Otherwise, if the swap is currently allowed and selected employee i cannot perform role j' at time t (as per eligibility condition (6.151)), record that the swap is not allowed.

vi. Otherwise, if a block has been found and $j_t^{i'} = rest$ in the 'current' solution, then do the following:

A. Record the block as ending at time $t-1$ and calculate the length λ'^B. If the block is not too early and the swap is allowed then, in accordance with condition (6.152) if $\lambda'^B \leq w_i^{max}$ then call **'Swap calculation'** procedure (page 396).

B. Record again that no block has been found.

C. If no swap is yet to be carried out, then reset the record of the swapping block details (i.e. employee, role and vessel, start and end time, and duration), and record that a block is not too early and that the swap is allowed.

vii. If a *new* block has been found, then record that a block has been found, and reset to say that another *new* block has not yet been found.

viii. If $t = T$ and a block has been found, then do the following:

A. As per eligibility condition (6.151), if selected employee i cannot perform role j' at time t then record that a swap is not allowed.

B. Record the block as ending at time t and calculate the length λ'^B. If the block is not too early and the swap is allowed then, in accordance with condition (6.152) if $\lambda'^B \leq w_i^{max}$ then call **'Swap calculation'** procedure (page 396).

(c) If no swap is yet to be carried out and there has been a *rest* block found, then record block details as a *rest* block on the roster of employee i', starting and ending at the same time as the selected block, and call **'Swap calculation'** procedure (page 396).

C.3.2.11 Swap calculation

1. Clear the list of employees whose rosters have changed and the list of roles which have had their agency assignments changed from the current solution.

2. Copy all rosters $\Upsilon_{i''}$ for all $i'' \in E_R$ and agency sets $\Theta_{j''}$ for all $j'' \in J$ from the 'current' solution into those designated for the 'swap' solution.

3. For selected employee i, clear the contents of roster Υ_i for the 'swap' solution, and add i to the set of employees whose roster has changed from the current solution.

4. For each time t from $t = 1$ to $t = T$, do the following:

   (a) If t is earlier than the start of both the selected block and swapping block, i.e. $t < t^S$ and $t < t'^S$, then do the following:

i. If $t'^S < t^S$ and $t \geq (t'^S - \rho_i)$ and $j_t^i \neq rest$ in the 'current' solution, then set $j_t^i = rest$ in the 'swap' solution, add time t to agency set $\Theta_{j_t^i}$, and record role j_t^i as having had its agency assignments changed from the current solution.

ii. Otherwise, j_t^i takes the same value in the 'swap' solution as for the 'current' solution.

(b) Otherwise, if t is within the time of the swapping block, i.e. if $t'^S \leq t \leq (t'^S + \lambda'^B - 1)$, then set $j_t^i = j'$ in the 'swap' solution. If $j_t^i \neq rest$ in the 'current' solution and either $t < t^S$ or $t > (t^S + \lambda^B - 1)$, i.e. it is outwith the limits of the originally selected block, then add time t to agency set $\Theta_{j_t^i}$, and record role j_t^i as having had its agency assignments changed from the current solution.

(c) Otherwise, if t is within the time of the selected block, i.e. if $t^S \leq t \leq (t^S + \lambda^B - 1)$, then set $j_t^i = rest$ in the 'swap' solution.

(d) Otherwise, if $(t'^S + \lambda'^B - 1) > (t^S + \lambda^B - 1)$ and $t \leq (t'^S + \lambda'^B - 1) + \rho_i$ and $j_t^i \neq rest$ in the 'current' solution, then set $j_t^i = rest$ in the 'swap' solution, add time t to agency set $\Theta_{j_t^i}$, and record role j_t^i as having had its agency assignments changed from the current solution.

(e) Otherwise, j_t^i takes the same value in the 'swap' solution as for the 'current' solution.

5. If the swapping employee i' is not an agency employee, then clear the contents of roster $\Upsilon_{i'}$ for the 'swap' solution, and add i' to the set of employees whose roster has changed from the current solution. Then,

6. If the swapping employee i' is not an agency employee, then for each time t from $t = 1$ to $t = T$, do the following:

(a) If t is earlier than the start of both the selected block and swapping block, i.e. $t < t^S$ and $t < t'^S$, then do the following:

i. If $t \geq (t^S - \rho_{i'})$ and $j_t^{i'} \neq rest$ in the 'current' solution, and either $t^S < t'^S$ or $j' = rest$, then set $j_t^{i'} = rest$ in the 'swap' solution, add time t to agency set $\Theta_{j_t^{i'}}$, and record role $j_t^{i'}$ as having had its agency assignments changed from the current solution.

ii. Otherwise, $j_t^{i'}$ takes the same value in the 'swap' solution as for the 'current' solution.

(b) Otherwise, if t is within the time of the selected block, i.e. if $t^S \leq t \leq (t^S + \lambda^B - 1)$, then set $j_t^{i'} = j$ in the 'swap' solution. If $j_t^{i'} \neq rest$ in the 'current' solution and either $t < t'^S$ or $t > (t'^S + \lambda'^B - 1)$, i.e. it is outwith the limits of the swapping block, then add time t to agency set $\Theta_{j_t^{i'}}$, and record role $j_t^{i'}$ as having had its agency assignments changed from the current solution.

(c) Otherwise, if t is within the time of the swapping block, i.e. if $t'^S \leq t \leq (t'^S + \lambda'^B - 1)$, then set $j_t^{i'} = rest$ in the 'swap' solution.

(d) Otherwise, if $t \leq (t^S + \lambda^B - 1) + \rho_{i'}$ and $j_t^{i'} \neq rest$ in the 'current' solution and either $(t^S + \lambda^B - 1) > (t'^S + \lambda'^B - 1)$ or $j' = rest$, then set $j_t^{i'} = rest$ in the 'swap' solution, add time t to agency set $\Theta_{j_t^{i'}}$, and record role $j_t^{i'}$ as having had its agency assignments changed from the current solution.

(e) Otherwise, $j_t^{i'}$ takes the same value in the 'swap' solution as for the 'current' solution.

7. If swapping employee i' is an agency employee, then do the following:

   (a) For all time points in the selected block, i.e. for all t from $t = t^S$ to $t = (t^S + \lambda^B - 1)$, add time point t to the agency set Θ_j.

   (b) Record role j as having had its agency assignments changed from the current solution.

   (c) If $j' \neq rest$, then for all time points in the swapping block, i.e. for all t from $t = t'^S$ to $t = (t'^S + \lambda'^B - 1)$, remove time point t from the agency set $\Theta_{j'}$, and record role j' as having had its agency assignments changed from the current solution.

8. Set tabu check parameter to 'swap' and call **'Check tabu'** procedure (page 401).

9. If this 'swap' solution is found to be tabu, increase the count of tabu solutions by one; otherwise, do the following:

   (a) Set calculation parameter to 'swap', and call **'Calculate cost'** procedure (page 372).

   (b) If the solution is not feasible, **end** this procedure; otherwise, continue.

   (c) Calculate the cost difference between this solution and the current solution.

   (d) If the total cost of this solution is less than that of the solution type defined in section 6.5.4, then note that this swap should be carried out, and reset the number of non-reducing solutions to zero. Denote the set of employees to be updated as the set of employees changed from the current solution, and call **'Update swaps and changes'** procedure (page 403).

   (e) Otherwise, if no 'candidate' solution exists or the cost of the 'swap' solution is less than that of the existing candidate then record that a candidate solution now does exist. Set transfer parameters as 'from: swap; to: candidate', and call **'Transfer solution'** procedure (page 372). Store the cost of this new candidate solution, and record the employees changed in the 'candidate' solution as the selected employee i along with the swapping employee i' if this is not an agency employee.

C.3.2.12 Random kick

1. Record that no update has yet been done.

2. Clear the list of employees whose rosters have changed and the list of roles which have had their agency assignments changed from the current solution.

3. Select at random a regular employee $i' \in E_R$ and a role $j \in J$, with each employee and role having equal probability of being selected.

4. Select a block length λ^B from the range $(1, w_i^{max})$, with an equal probability of selecting any integer value in that range.

5. Select a block start time t^S from the range $(1, (T-\lambda^B+1))$, with an equal probability of selecting any integer value in that range.

6. The block end time can therefore be calculated as being $(t^S + \lambda^B - 1)$.

7. Check that kick feasibility conditions (6.154) and (6.156), relating respectively to eligibility and initial rest periods, and also conditions (6.157), (6.162) and (6.163) relating to rest periods and the employee's starting location, are satisfied. If any of the conditions are not satisfied (i.e. the proposed kick is infeasible), skip to the final step of this procedure; otherwise, continue.

8. Copy all agency sets $\Theta_{j'}$ for all $j' \in J$ from the 'current' solution into those designated for the 'kick' solution.

9. For each time t from from $t = t^S$ to $t = (t^S + \lambda^B - 1)$, if t is in the set Θ_j (where j is the role selected for the kick) for the 'kick' solution then remove it from the set and record role j as having the agency assignment changed from the current solution.

10. For each employee $i \in E_R$ such that $i \neq i'$, do the following:

    (a) Clear the contents of roster Υ_i for the 'kick' solution.

    (b) If employee i starts on board the vessel on which the selected role j takes place, then set the initial rest count equal to ρ_i; otherwise set the initial rest count to zero.

(c) For each time t from from $t = 1$ to $t = T$, do the following:

   i. If $t = t^S - 1$ and in the 'current' solution $j_i^t = j$, then set j_i^t for the 'kick' solution also equal to j and set the rest count equal to ρ_i.

   ii. Otherwise, if $t = t^S$ then, if in the 'current' solution $j_i^t = j$ then set $j_i^t = rest$ for the 'kick solution', record employee i as having had their roster changed from the current solution, and reduce the rest count value by one; otherwise (i.e. if $j_i^t \neq j$ in the 'current' solution), j_i^t takes the same value in the 'kick' solution as the 'current' solution, and the rest count should be reset to zero.

   iii. Otherwise, if t is within the generated block, i.e. $t^S < t \leq (t^S + \lambda^B - 1)$, then if in the current solution $j_i^t = j$ or $j_i^t = rest$ then set $j_i^t = rest$ for the 'kick' solution and reduce the rest count value by one, and if $j_i^t = j$ in the current solution then also record employee i as having been changed from the current solution. If j_i^t has a different value in the current solution then if the rest count value is non-zero, then set $j_i^t = rest$, add time t to agency set $\Theta_{j_i^t}$ and record this role as having changed from the current solution, and reduce the rest count value by one; otherwise, j_i^t takes the same value for the 'kick' solution as for the 'current' solution.

   iv. Otherwise, if $t > (t^S + \lambda^B - 1)$ and $j_i^t \neq rest$ in the 'current' solution and the rest count value is non-zero, then set $j_i^t = rest$, add time t to agency set $\Theta_{j_i^t}$ and record this role as having changed from the current solution, and reduce the rest count value by one.

   v. Otherwise, j_i^t takes the same value for the 'kick' solution as for the 'current' solution.

11. For the employee selected for the kick, i', do the following:

   (a) Clear the contents of roster $\Upsilon_{i'}$ for the 'kick' solution.

   (b) For each time t from from $t = 1$ to $t = T$, do the following:

      i. If t is more than the minimum rest period earlier than the start of the block, i.e. if $t < t^S - \rho_{i'}$, then $j_{i'}^t$ takes the same value for the 'kick' solution as for the 'current' solution.

      ii. Otherwise, if $t < t^S$ then $j_{i'}^t = rest$ for the 'kick' solution. If $j_{i'}^t \neq rest$ in the 'current' solution, then add t to the agency set $\Theta_{j_{i'}^t}$ for the 'kick' solution and record role $j_{i'}^t$ as having its agency allocation changed from the current solution.

      iii. Otherwise, if $t \leq (t^S + \lambda^B - 1)$ then $j_{i'}^t$ takes the value of the generated role j for the 'kick' solution. If $j_{i'}^t \neq rest$ and $j_{i'}^t \neq j$ in the 'current' solution, then add t to the agency set $\Theta_{j^t_{i'}}$ for the 'kick' solution and record role $j^t_{i'}$ as having its agency allocation changed from the current solution.

iv. Otheriwse, if $t \leq (t^S + \lambda^B - 1) + \rho_{i'}$ then set $j^t_{i'} = rest$ for the 'kick' solution. If $j^t_{i'} \neq rest$ in the 'current' solution, then add t to the agency set $\Theta_{j^t_{i'}}$ for the 'kick' solution and record role $j^t_{i'}$ as having its agency allocation changed from the current solution.

v. Otherwise, $j^t_{i'}$ takes the same value for the 'kick' solution as for the 'current' solution.

12. Set tabu check parameter to 'kick' and call **'Check tabu'** procedure (page 401).

13. If this 'kick' solution is found to be tabu, increase the count of tabu solutions by one; otherwise, do the following:

    (a) Set calculation parameter to 'kick', and call **'Calculate cost'** procedure (page 372).

    (b) If the solution is not feasible, skip to the final step of this procedure; otherwise, continue.

    (c) Record the rosters Υ_i for all $i \in E_R$ and agency sets Θ_j for all $j \in J$ for the 'current' solution as now being the 'tabu' solution.

    (d) Set transfer parameters as 'from: kick; to: current', and call **'Transfer solution'** procedure (page 372).

    (e) Record the last time a kick was carried out as being the current iteration, record than an update has been done, and reset the number of non-reducing solutions to zero.

    (f) Call **'Compare to best'** procedure (page 402).

    (g) Denote the set of employees to be updated as the set of employees changed from the current solution, and call **'Update swaps and changes'** procedure (page 403).

14. If no update has been done, return to **Step 2** of this procedure; otherwise **end** this procedure.

C.3.2.13 Check tabu

1. Assume that the solution to be checked *is* tabu.

2. If the tabu check parameter does not refer to a valid solution type, then **end** this procedure now; otherwise, continue.

3. For each regular employee $i \in E_R$ while the solution is still considered tabu, do the following:
   (a) Compare roster Υ_i for the solution type referred to by the tabu check parameter against the equivalent roster for the 'tabu' solution. If there is any difference between these rosters, then the solution is *not* tabu.
4. For each role $J \in J$ while the solution is still considered tabu, do the following:
   (a) Compare agency allocation set Θ_j for the solution type referred to by the tabu check parameter against the equivalent set for the 'tabu' solution. If there is any difference between these sets, then the solution is *not* tabu.

C.3.2.14 Compare to best

1. If the cost of the 'current' solution is equal to that of the 'best' solution found so far, then do the following:
   (a) Assume that this is a new best solution.
   (b) While this is still considered to be a new best solution, for each of the equal best solutions found so far, do the following:
      i. Assume that the 'current' solution is the same as the equal best solution under consideration.
      ii. For each regular employee $i \in E_R$ while these solutions are still considered to be the same, do the following:
         A. Compare roster Υ_i for the 'current' solution against the equivalent roster for the equal best solution under consideration. If there is any difference between these rosters, then these solutions are not the same.
      iii. For each role $J \in J$ while these solutions are still considered to be the same, do the following:
         A. Compare agency allocation set Θ_j for the 'current' solution against the equivalent set for the equal best solution under consideration. If there is any difference between these sets, then these solutions are not the same.
      iv. If the 'current' solution is the same as the equal best solution under consideration, then it is not a new best solution.
   (c) If the 'current' solution is a new best solution then increase by one the count of equal best solutions found, and store all rosters Υ_i and sets Θ_j from the 'current' solution as describing an additional equal best solution.

2. Otherwise, if the cost of the 'current' solution is less than the cost of the best recorded solution found so far, do the following:

    (a) Set transfer parameters as 'from: current; to: best', and call **'Transfer solution'** procedure (page 372).

    (b) Reset the number of equal best solutions found to 1, and store all rosters Υ_i and sets Θ_j from the 'current' solution as describing this equal best solution.

    (c) Record the iteration number as the time at which the best solution was found.

C.3.2.15 Update swaps and changes

1. For all regular employees $i \in E_R$ which are in the set of employees to be updated, do the following:

    (a) Clear the list of employees who have been examined for swapping blocks with employee i.

    (b) For all regular employees $i' \in E_R$ who are *not* in the set of employees to be updated, if employee i is in the set of employees who have been examined for a swap with employee i' then remove employee i from this set.

2. If the time of the last kick is the current iteration time (i.e. a kick has just been carried out), then for employees $i \in E_R$ denote the last time employee i's schedule changed as the current iteration; otherwise, for all employees i in the set of employees to be updated denote the last time their schedule has been changed as the current iteration.

C.4 Description of Heuristic Initial Solution algorithm

Here we give a step-by-step procedure for the Heuristic Initial Solution algorithm, as introduced in 6.6.1. Note that the full code of the algorithm as implemented in FICO Xpress can be found in appendix section E.2.6.

C.4.1 Main programme

1. Declare all required variables, and read in the data (including the existing schedule).

2. Call **'Initialisation'** procedure (page 404).

3. Set calculation parameter to 'initial', and call **'Calculate cost'** procedure (page 406). Note that this cost is for reference purposes only.

4. Set transfer parameter as ‘from: initial; to: current’, and call **‘Transfer solution’** procedure (page 406).

5. Call **‘Initial feasibility check’** procedure (page 405).

6. Call **‘Sort employee list’** procedure (page 405).

7. Call **‘Sort role list’** procedure (page 406).

8. Set iteration count equal to zero. While iteration count ≤ 20 and number of vacancies < 0, do the following:

   (a) Call **‘Find vacancies’** procedure (page 407)

   (b) Increment iteration count.

9. Set checking parameter as ‘current’, and call **‘Check feasibility’** procedure (page 414).

10. If the solution is infeasible, the user may wish to re-run the programme from **Step 6** above (i.e. with re-randomized lists); or if the infeasibility arises because of unfilled vacancies, the user may wish to resolve these by allocating agency crew to all vacant positions (possibly resulting in an expensive solution).

11. Output the details of the solution found, and for testing purposes details of the test run.

12. **End** of algorithm.

C.4.2 Sub-Programmes

The sub-procedures used are detailed here, approximately following the order that the algorithm would require them during execution.

C.4.2.1 Initialisation

1. Initially, assume all roles are vacant - set vacancy indicator = 1 for all roles at all times; and record that no assignments of the solution are yet ‘fixed’.

2. For each employee $i \in E$, role $j \in J$ and time $t \in \{1, \ldots, T\}$ do the following:

   (a) If i is assigned to j at time t in the existing schedule and is still eligible to carry this out, then record this assignment as part of the ‘initial’ and ‘current’ solutions for this algorithm, and record that role j at time t is not vacant; otherwise, record that i is not assigned to role j at time t in either the ‘initial’ or ‘current’ solutions.

(b) If i is not eligible to perform role j at time t, then record that their assignment (or 'non-assignment') to this role at this time is 'fixed'.

3. Count the number of role-time pairs which are vacant in the current solution.

4. Record all employees $i \in E_R$ and all roles $j \in J$ as having had their schedules changed since the last iteration.

C.4.2.2 Initial feasibility check

1. Clear the list of employees whose rosters have changed and the list of roles which have had their agency assignments changed since the last cost calculation.

2. Examine the starting indicator s_{ik} for all employees $i \in E_R$ and vessels $k \in K$. If i starts on board vessel k, must ensure that rest periods which may relate to this starting position are respected. Therefore for each i and k such that $s_{ik} = 1$, do the following:

   (a) Repeat for all $j \in J$.

   (b) If $j \notin V_k$, or if $j \in V_k$ and i is not eligible to perform role j at time $t = 1$ (i.e. $e_{ij1} = 0$), then must ensure the minimum rest period length for employee i is kept clear. Therefore, for all times t such that $1 \leq t \leq \rho_i$, do the following:

      i. If employee i is assigned to role j at time t, then: remove this assignment from the current solution; record role j at time t as vacant; increase the count of vacancies by one; and record employee i as having their schedule changed.

      ii. Fix the assignment (or 'non-assignment') of employee i to role j at time t.

3. If any employee's schedule has been changed, then set calculation parameter to 'current', call **'Calculate cost'** procedure (page 406), and then call **'Show updates'** procedure (page 414).

C.4.2.3 Sort employee list

This procedure is essentially the same as that used for the main heuristic algorithm (see page 371), with the order parameter set to 'random'. The only exception is that the *short* ordered list is not required by this algorithm, therefore we can disregard mention of it (in **Step 2** and **Step 4(d)** of the procedure).

C.4.2.4 Sort role list

This procedure performs the same function for roles as the **'Sort employee list'** procedure (above) for employees. We therefore do not give details here, and simply observe that this procedure can be carried out by substituting 'roles' for 'employees', and index j for index i, in the **'Sort employee list'** procedure above. Note that full details of the FICO Xpress code used can be found in the appendix in section E.2.6.

C.4.2.5 Transfer solution

This procedure is virtually identical to the that used for the main heuristic algorithm (see page 372), except that in addition to the quantities listed in **Step 2**, we record the number of changes associated with each roster Υ_i and agency allocation set Θ_j which must be copied along with the associated cost.

C.4.2.6 Calculate cost

This procedure is largely the same as that used in the main heuristic algorithm (see page 372), however the following modifications must be made in this case:

- At **Step 4(a)** we must also copy the number of changes associated with roster Υ_i, as well as the cost.
- At **Step 4(b)** we must also clear the number of changes associated with roster Υ_i, as well as the cost.
- We must add a **Step 4(g)**, where we should sum the values of the change variables for allocation ($x^{\pm}_{ijt}$), and record this as the number of changes associated with roster Υ_i in this 'evaluating' solution.
- At **Step 5(a)** we must also copy the number of changes associated with allocation set Θ_j, as well as the cost.
- At **Step 5(b)** we must also clear the number of changes associated with set Θ_j, as well as the cost.
- We should disregard **Step 5(e)** and **Step 5(f)** for this algorithm.
- We must add a **Step 5(j) vii.**, where we should add the value of the change variable for allocation ($x^{\pm}_{m+1,jt}$) to the number of changes associated with allocation set Θ_j in this 'evaluating' solution.

- At **Step 6** in addition to defining the total cost, we should also define the total number of changes in this 'evaluating' solution to be the sum of the number of changes associated with each roster Υ_i for all $i \in E_R$ and each agency allocation set Θ_j for all $j \in J$.

- Note that **Step 8** is now redundant, since there are no 'backward', 'forward', 'swap' or 'kick' solutions for this algorithm, and so this step can be disregarded.

Note that full details of the new procedure as implemented in FICO Xpress can be found in the appendix in section E.2.6.

C.4.2.7 Find vacancies

1. Select the next role j on the randomized list of roles.

2. Clear the record of any previously identified block (i.e. role, vessel, start time, end time and length), record that no block has yet been found, and record that no modification has yet been made.

3. For each time t from $t = 1$ to $t = T$, do the following:

   (a) Record that no *new* block has been found.

   (b) If no modification has yet been made, no block has yet been found, and role j is vacant at time t, then record a *new* block as having been found. Record the task j, the associated vessel k, and the start time $t^S = t$ of this block.

   (c) Otherwise, if no modification has yet been made, a block has already been found and role j is not vacant at time t, then we have identified the end of a block. Record $(t-1)$ as the end time of the block, calculate block length $\lambda^B = t - t^S$ and call **'Repair vacancies'** procedure (page 408). Record that a modification has now been made, reset the records of the block (i.e. start time, end time, length, role and vessel), and record that no block has been found at this stage.

   (d) If a *new* block is recorded as having been found, then update record to reflect that a block has been found and that no *new* block has been found.

   (e) If $t = T$, no modification has yet been made, and a block has been found, then record this block as ending at time T and calculate block length $\lambda^B = T - t^S + 1$. Call **'Repair vacancies'** procedure (page 408), and record that a modification has now been made.

4. If all roles have been examined, **end** this procedure; otherwise, return to **Step 1**.

Note that this procedure is similar to the **'Find usable block'** procedure in the main heuristic algorithm (see page 382), except that it looks for blocks which are not currently assigned to any employee rather than blocks on an employee's roster. Since it deals with only one role at a time, it is a little less complex than the **'Find usable block'** procedure.

C.4.2.8 Repair vacancies

1. Record that no update has yet been done.

2. We must identify the employees who are assigned to the role j immediately before and after the vacant block. Firstly clear any previous record of these, and then for all employees $i \in E$ do the following:

   (a) If $t^S > 1$, then if $\hat{x}_{ij,(t^S-1)} = 1$ then record employee i as employee i^b who works in role j before the vacant block.

   (b) If $t^S + \lambda^B - 1 < T$, then if $\hat{x}_{ij,(t^S+\lambda^B)} = 1$ then record employee i as employee i^a who works in role j after the vacant block.

3. Clear the list of employees whose rosters have changed and the list of roles which have had their agency assignments changed since the last cost calculation, and record that we have not decided to allocate the block to the employees assigned to the role immediately before or after the block.

4. If $\lambda^B = T$ (i.e. the block is the length of the entire planning horizon), then should assign the block to agency crew. For all $t \in \{1, \ldots, T\}$, assign agency crew to role j in the current solution and denote this assignment as fixed. Record that role j is no longer vacant at any time t, and for each t reduce the count of vacancies by one. Finally, add role j to the list of roles whose agency assignments have changed, record that an update has been done.

5. Otherwise, if the block ends at the end of the planning horizon (i.e. $t^S + \lambda^B - 1 = T$) and an employee i^b has been identified as working before the block, then look at feasibility of assigning this employee to the vacant block. If the employee before is agency, then record that we should assign this employee; otherwise, must check that the following are satisfied:

   - The block length is less than the maximum working period of employee i^b, i.e. $\lambda^B < w_{i^b}^{max}$.
   - Employee i^b can perform the vacant block without working a longer consecutive period that their permitted maximum, which can be checked by ensuring $\hat{l}_{(w_{i^b}^{max}-\lambda^B+1),i^b,j,(t^S-1)} = 0$ in the current solution.

- The employee is not assigned to any other role for the duration of the block, i.e. $\hat{x}_{i^b,j',t)} = 0$ in the current solution for all $j' \in J$ and t such that $t^S \leq t \leq (t^S + \lambda^B - 1)$.
- Employee i^b is eligible to carry out role j for every period t during the block, and their assignment (or 'non-assignment') to this role at these times has not previously been fixed.

If these conditions are all satisfied, then record that we should assign this employee.

6. Otherwise, if $t^S = 1$ and an employee i^a has been identified as working immediately after the vacant block, then look at the feasibility of assigning this employee to the vacant block. If this is an agency employee, then record that we should assign this employee; otherwise, must check that the following are satisfied:

   - The rest period required by i^a at the start of the planning period is $r_{i^a,0} = 0$.
   - The employee is not assigned to any other role for the duration of the block, i.e. $\hat{x}_{i^a,j',t)} = 0$ in the current solution for all $j' \in J$ and t such that $t^S \leq t \leq (t^S + \lambda^B - 1)$.
   - Employee i^a is eligible to carry out role j for every period t during the block, and their assignment (or 'non-assignment') to this role at these times has not previously been fixed.

   If these conditions are all satisfied, then record that we should assign this employee.

7. Otherwise, the vacant block is at neither the very start nor very end of the planning horizon, potentially with employees both before and after who could be assigned to it. In this case, we must proceed as follows:

   (a) If an employee i^b has been identified as working before the block, and this is not an agency employee, then should carry out the same checks as described in **Step 5**. If these are all satisfied, then record that we should assign this employee.

   (b) If we have not recorded that we should assign employee i^b and we have identified an agency employee $i^a = m + 1$ as working after the block then record that we should assign this employee to the vacant block. If we have not recorded that we should assign employee i^b and we have identified an employee $i^a \in E_R$ as working after the block then (similarly to **Step 6**) we should check the following:

   - Either the block start time is $t^S \leq \rho_{i^a}$ and $t^S > r_{i^a,0}$, and $\hat{x}_{i^a,j',t)} = 0$ in the current solution for all $j' \in J$ and t such that $1 \leq t \leq (t^S + \lambda^B - 1)$; or block start time is $t^S > \rho_{i^a}$ and $\hat{x}_{i^a,j',t)} = 0$ in the current solution for all $j' \in J$ and t such that $(t^S - \rho_{i^a}) \leq t \leq (t^S + \lambda^B - 1)$.

- Employee i^a is eligible to carry out role j for every period t during the block, and their assignment (or 'non-assignment') to this role at these times has not previously been fixed.

If these conditions are all satisfied, then record that we should assign this employee.

(c) If we have not recorded that we should assign employee i_a, and the employee working before the block is an agency employee (i.e. $i^b = m + 1$), then record that we should assign employee i_b.

8. If the vacant block is at neither the very start nor very end of the planning horizon, and we have not during **Step 7** recorded that either the employee before or the employee after should be assigned to the block, then may wish to remove employee i_b or i_a from their assignment. Proceed as follows:

   (a) Starting immediately after the vacant block, i.e. at time $t = (t^S + \lambda^B)$, count the number of consecutive periods λ_a^B for which employee i_a is assigned to role j and that this assignment is *not* fixed. Once a time $t \geq (t^S + \lambda^B)$ is found such that $\hat{x}_{i^a,j,t)} = 0$ in the current solution, or that the assignment of i_a to role j at time t is fixed, the count should be stopped.

   (b) Starting immediately before the vacant block, i.e. at time $t = (t^S - 1)$, count the number of consecutive periods λ_b^B for which employee i_b is assigned to role j and that this assignment is *not* fixed. Once a time $t \leq (t^S - 1)$ is found such that $\hat{x}_{i^b,j,t)} = 0$ in the current solution, or that the assignment of i_b to role j at time t is fixed, the count should be stopped.

   (c) If $\lambda^B \leq 2$ or $\lambda_b^B \leq 2$ or $\lambda_a^B \leq 2$ or $(\lambda^B + \lambda_b^B) \leq 4$ or $(\lambda^B + \lambda_a^B) \leq 4$, then continue with this step; otherwise, move on to **Step 9**.

   (d) If $\lambda_b^B < \lambda_a^B$, and $\lambda_b^B > 0$, then we will remove the assignment of employee i_b before the vacant block (this creates a larger vacant block which will be more cost-effective to assign to another employee). To do this, do the following:

      i. For all t such that $(t^S - \lambda_b^B) \leq t \leq (t^S - 1)$, set $\hat{x}_{i^b,j,t)} = 0$ for the current solution and record this assignment (or 'non-assignment') as fixed. Record role j as vacant during each of these time periods, and for each t increase the count of vacancies by one.

      ii. For all t such that $t \leq r_{i_b,0}$ and if $t^S - \lambda_b^B$ all $t \leq \rho_{i_b}$, look at the assignment of employee i_b to any role. If $\hat{x}_{i^b,j',t)} = 1$ in the current solution for any $j' \in J$ for any of these t, then set $\hat{x}_{i^b,j',t)} = 0$ and record this as fixed, record role j' as vacant at time t and increment the count of vacancies by one.

iii. Add employee i_b to the set of employees whose schedules have been changed, and record that at update has now been done.

(e) Otherwise, if $\lambda_a^B > 0$, then we will remove the assignment of employee i_a after the vacant block as follows:

i. For all t such that $(t^S + \lambda^B) \leq t \leq (t^S + \lambda^B + \lambda_a^B - 1)$, set $\hat{x}_{i^a,j,t)} = 0$ for the current solution and record this assignment (or 'non-assignment') as fixed. Record role j as vacant during each of these time periods, and for each t increase the count of vacancies by one.

ii. Add employee i_a to the set of employees whose schedules have been changed, and record that at update has now been done.

9. If we have indicated that employee i_b should be assigned the vacant block, then we should implement this change as follows:

(a) For all time points t during the vacant block, i.e. in $t^S \leq t \leq (t^S + \lambda^B - 1)$, set $\hat{x}_{i^b,j,t)} = 1$ for the current solution and record this assignment as fixed. Record that role j is no longer vacant at time t, and for each of these times t reduce the count of vacancies by one.

(b) If $i_b = m + 1$ (i.e. it is an agency employee), then add role j to the set of roles for which the agency assignments have changed.

(c) Otherwise, add employee i_b to the set of employees whose schedules have been changed, and ensure that the employee's minimum rest period after the block is respected. This means that for all weeks t in the range $(t^S + \lambda^B) \leq t \leq (t^S + \lambda^B + \rho_{i_b,0})$ such that $t \leq T$, we must clear any assignments of employee i_b. If $\hat{x}_{i^b,j',t)} = 1$ in the current solution for any $j' \in J$ for any of these t, then set $\hat{x}_{i^b,j',t)} = 0$ and record this as fixed, record role j' as vacant at time t and increment the count of vacancies by one.

10. Otherwise, if we have indicated that employee i_a should be assigned the vacant block, then we should implement this change as follows:

(a) For all time points t during the vacant block, i.e. in $t^S \leq t \leq (t^S + \lambda^B - 1)$, set $\hat{x}_{i^a,j,t)} = 1$ for the current solution and record this assignment as fixed. Record that role j is no longer vacant at time t, and for each of these times t reduce the count of vacancies by one.

(b) If $i_a = m + 1$ (i.e. it is an agency employee), then add role j to the set of roles for which the agency assignments have changed.

(c) Otherwise, add employee i_a to the set of employees whose schedules have been changed, and ensure that the employee's maximum working period is respected.

To do this, set the consecutive working count equal to λ^B, and then for each time t in the range $(t^S + \lambda^B) \leq t \leq T$ do the following:

i. If $\hat{x}_{i^a,j,t)} = 1$ in the current solution and the consecutive work count is less than $w_{i_a}^{max}$, then increase the consecutive working count by one.

ii. Otherwise, if $\hat{x}_{i^a,j,t)} = 1$ in the current solution and the consecutive work count is equal to $w_{i_a}^{max}$, then must set $\hat{x}_{i^b,j,t)} = 0$, record this as fixed, record role j as vacant at time t, and increment the count of vacancies by one.

iii. Otherwise, we should reset the consecutive work count to zero.

11. Otherwise, if no update has yet been done we must look at the possibility of assigning some other employee. This should be done as follows:

(a) Record that no candidate yet exists and clear the cost and record of employee i_c associated with this candidate solution.

(b) For all employees i in the ordered list, such that $i \neq i_b$ and $i \neq i_a$, do the following:

i. Clear the list of employees whose rosters have changed and the list of roles which have had their agency assignments changed since the last cost calculation.

ii. Set quantity 'earliest' equal to the maximum of $(t^S - \rho_i)$ and 1.

iii. Set quantity 'latest' equal to the minimum of $(t^S + \lambda^B + \rho_i - 1)$ and T.

iv. Check whether the following are true:

- Employee i has no assignments from the 'earliest' period until the end of the vacant block in the current solution.
- Employee i is eligible to carry out the vacant block in all periods t from $t^S \leq t \leq (t^S + \lambda^B - 1)$, and their current assignment (or 'non-assignment') to this role in these periods has not been fixed.
- Block length $\lambda^B \leq w_i^{max}$, and block start $t^S > r_{i0}$.
- Either:
  - $t^S > 1$, and either $s_{ik} = 0$ for all vessels $k \in K$ or $t^S > \rho_i$; or
  - $w_{i0} \leq (w_i^{max} - \lambda^B)$, and $s_{ik} = 0$ for all vessels $k \in K$ such that the vacant role $j \notin V_k$.
- Either 'latest' $= T$, or the employee i has no assignments from the end of the vacant block until the week 'latest' to which their assignment is fixed.

If these statements all hold, then continue with this step for employee i; otherwise, this is the **end** of **Step 11(b)** for this employee. If these statements all hold, then evaluate the cost of assigning the vacant block employee i as follows:

A. Set transfer parameter as 'from: current; to: trial', and call **'Transfer solution'** procedure (page 406).

B. For all time points t during the block, i.e. in $t^S \leq t \leq (t^S + \lambda^B - 1)$, assign employee i to role j by setting $\hat{x}_{ijt)} = 1$ in this trial solution.

C. For all time points t within the minimum rest period after the block, i.e. in $(t^S + \lambda^B) \leq t \leq (t^S + \lambda^B + \rho_i - 1)$, set $\hat{x}_{ij't)} = 0$ in this trial solution for all roles $j' \in J$.

D. Add employee i to the set of employees whose schedules have been changed, set calculation parameter to 'trial' and call **'Calculate cost'** procedure (page 406).

E. If a candidate does not currently exist, record that a candidate now does exist, store employee i as associated candidate employee i_c and store the cost of this trial solution as the candidate cost.

F. Otherwise, if this trial solution has a lower cost than the current candidate cost, then store employee i as the candidate employee i_c and replace the existing candidate cost with the cost of this trial solution.

(c) If no candidate yet exists, record that a candidate now does exist and set $i_c = m + 1$, i.e. agency crew.

(d) Otherwise, must evaluate the cost of assigning agency crew as follows:

i. Set transfer parameter as 'from: current; to: trial', and call **'Transfer solution'** procedure (page 406).

ii. Clear the list of employees whose rosters have changed and the list of roles which have had their agency assignments changed since the last cost calculation.

iii. For all time points t during the block, i.e. in $t^S \leq t \leq (t^S + \lambda^B - 1)$, assign agency crew to role j by setting $\hat{x}_{m+1,j,t)} = 1$ in this trial solution.

iv. Add role j to the set of roles for which the agency assignments have changed, set calculation parameter to 'trial' and call **'Calculate cost'** procedure (page 406).

v. If the trial solution has a lower cost than the current candidate cost, then set candidate employee $i_c = m + 1$ and replace the existing candidate cost with the cost of this trial solution.

(e) Clear the list of employees whose rosters have changed and the list of roles which have had their agency assignments changed since the last cost calculation.

(f) For all time points t during the block, i.e. in $t^S \leq t \leq (t^S + \lambda^B - 1)$, assign the candidate employee i_c to role j by setting $\hat{x}_{ijt)} = 1$ in the current solution and denote this assignment as fixed. Record that that role j is no longer vacant at time t, and reduce the count of vacancies by one for each week of the block.

(g) If $i_c = m + 1$, add role j to the set of roles for which the agency assignments have changed.

(h) Otherwise, add employee i_c to the set of employees whose schedules have been changed, and clear any assignments employee i_c may have within the minimum rest period after the block has finished. To do this, search for any time t in $(t^S + \lambda^B) \leq t \leq (t^S + \lambda^B + \rho_{i_c} - 1)$ and any role $j' \in J$ such that $\hat{x}_{i_c,j',t)} = 1$ in the current solution. If any are found, set $\hat{x}_{i_c,j',t)} = 0$ in the current solution, record role j' as vacant in period t, and increase the count of vacancies by one.

12. Set calculation parameter to 'current', call **'Calculate cost'** procedure (page 406), and then call **'Show updates'** procedure (page 414).

C.4.2.9 Check feasibility

This procedure is almost identical to that used in the main heuristic algorithm (see page 377). Two minor modifications must be made for this algorithm:

- Because the **'Check feasibility'** procedure can be called independently of the **'Calculate cost'** procedure in the heuristic initial solution algorithm, we must add a **Step 1A** between **Step 1** and **Step 2** to determine whether the checking parameter is a valid solution type. If it is not, then should **end** this procedure now; otherwise, continue.

- We can disregard **Step 14**, as this is no longer required.

C.4.2.10 Show updates

This procedure gives output showing which assignments have been updated at the most recent step. No quantities are calculated here and no schedule modifications are made, therefore no details are given here. The code implemented in FICO Xpress can be found in the appendix in section E.2.6.

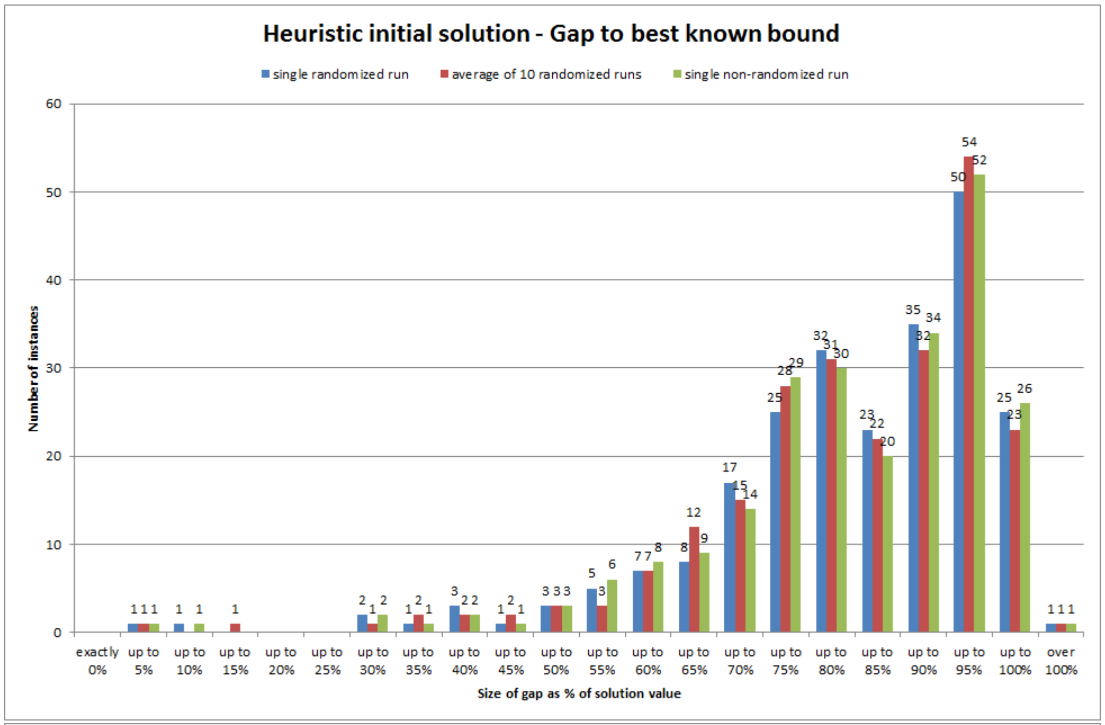
Appendix D

Additional histograms for Time-Windows results

Here we give additional histograms relating to the results of tests on the solution methods for the Time-Windows formulation of the problem, as discussed in section 6.8. This comprises section D.1 with additional histograms relating to the Heuristic Initial solution approach, and section D.2 which gives a more detailed breakdown of the results for the main Heuristic algorithm.

D.1 Relating to the Heuristic Initial solution approach

As discussed in section 6.8.3, in addition to the results reported for the single randomized Heuristic initial solution test run there were two further test carried out. One of these comprised ten further randomized runs, while the other was a non-randomized solution which was obtained by examining all employees and roles in the order they appeared in the datasets. For each graph shown for the single randomized run (Figures 6.22 to 6.29), here we present two corresponding graphs. In each of Figures D.1 to D.9, the upper chart presents the single randomized run measure along side the given metric averaged across the ten additional randomized runs and for the singe non-randomized run; in the lower chart, the minimum and maximum values across the ten randomized runs are given.



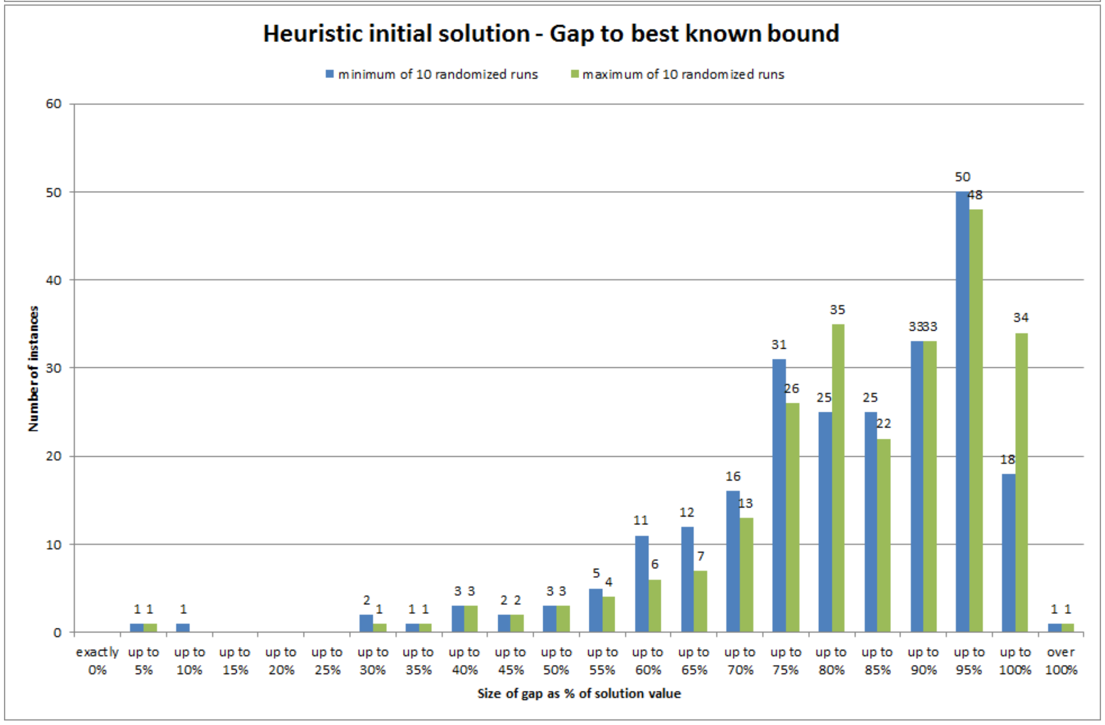
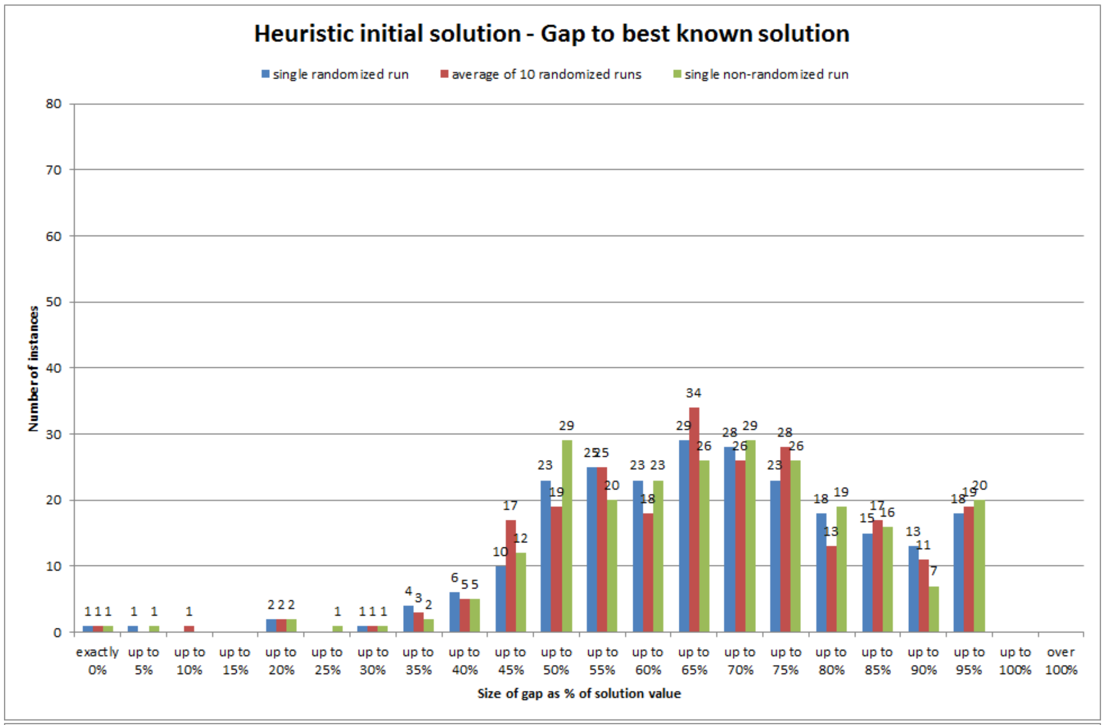


Figure D.1: Histograms of percentage gaps to best known bounds for solutions found using the Heuristic initial solution approach. Results are shown for the single randomized run, the average across the ten randomized runs, and the non-randomized run (top); and for the minimum and maximum across the ten randomized runs (bottom).



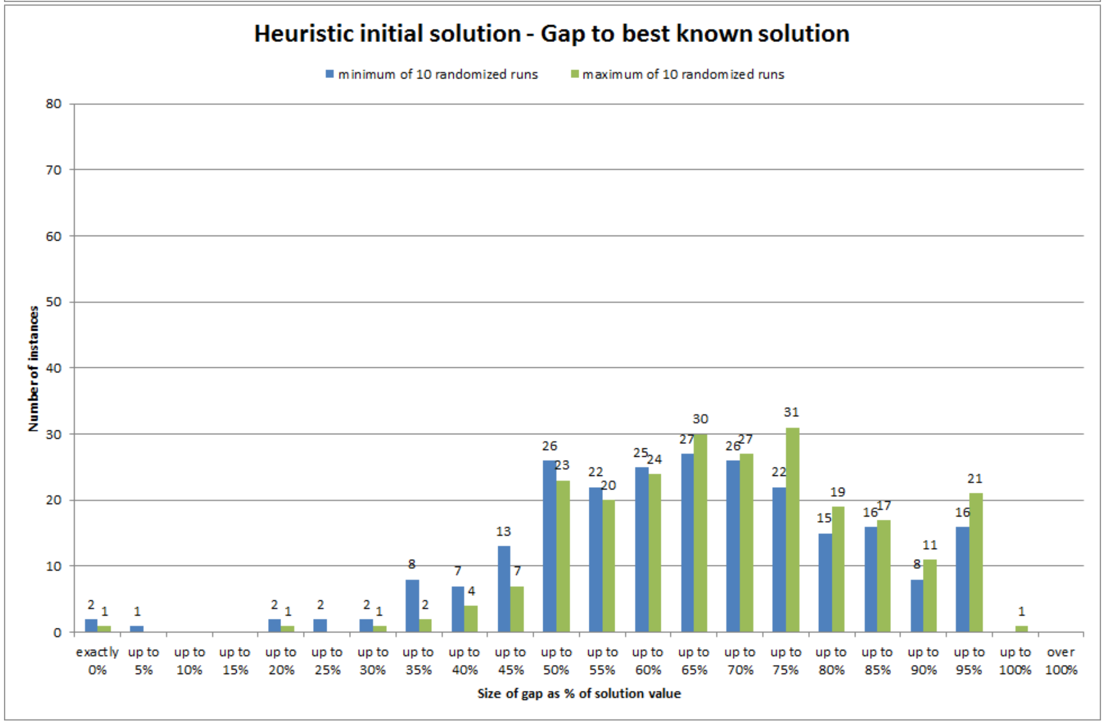
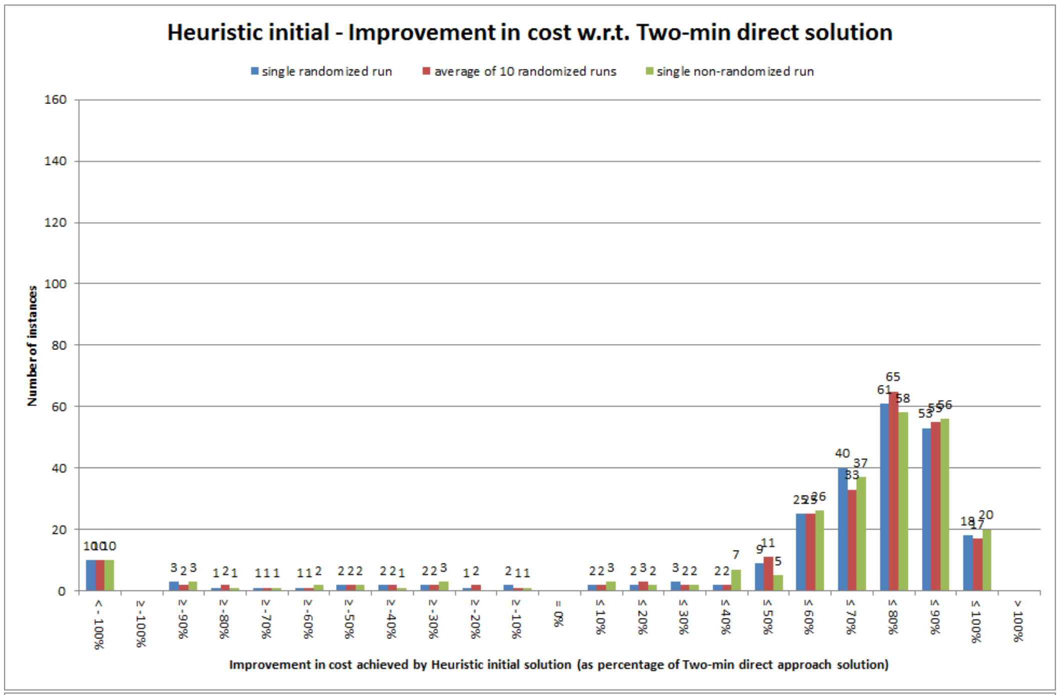


Figure D.2: Histograms of percentage gaps to best known bounds for solutions found using the Heuristic initial solution approach. Results are shown for the single randomized run, the average across the ten randomized runs, and the non-randomized run (top); and for the minimum and maximum across the ten randomized runs (bottom).



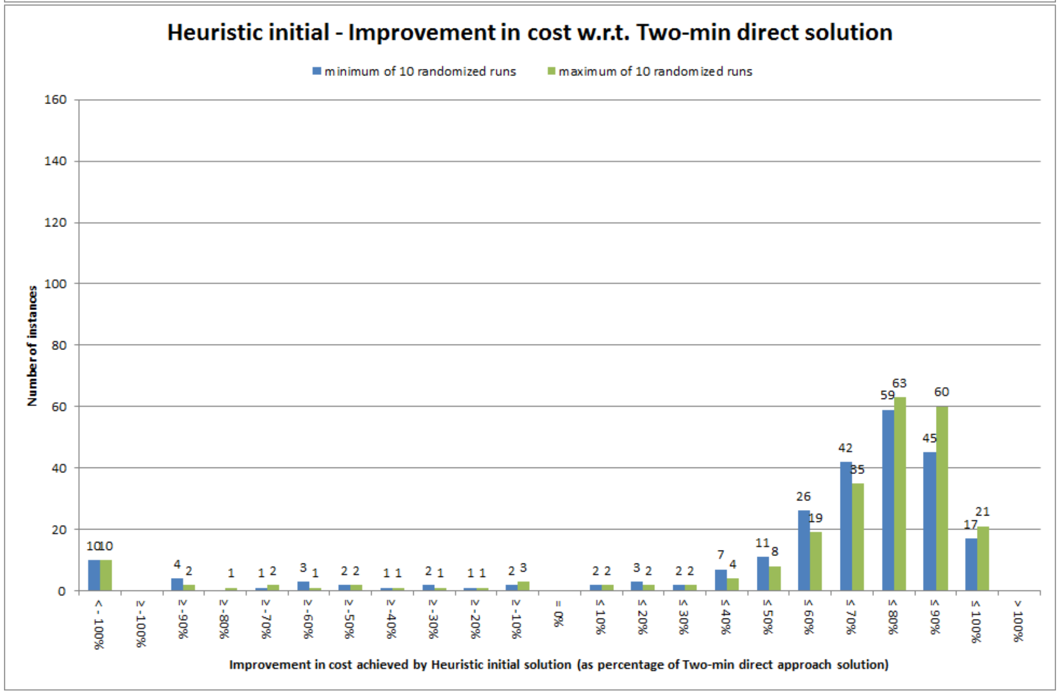
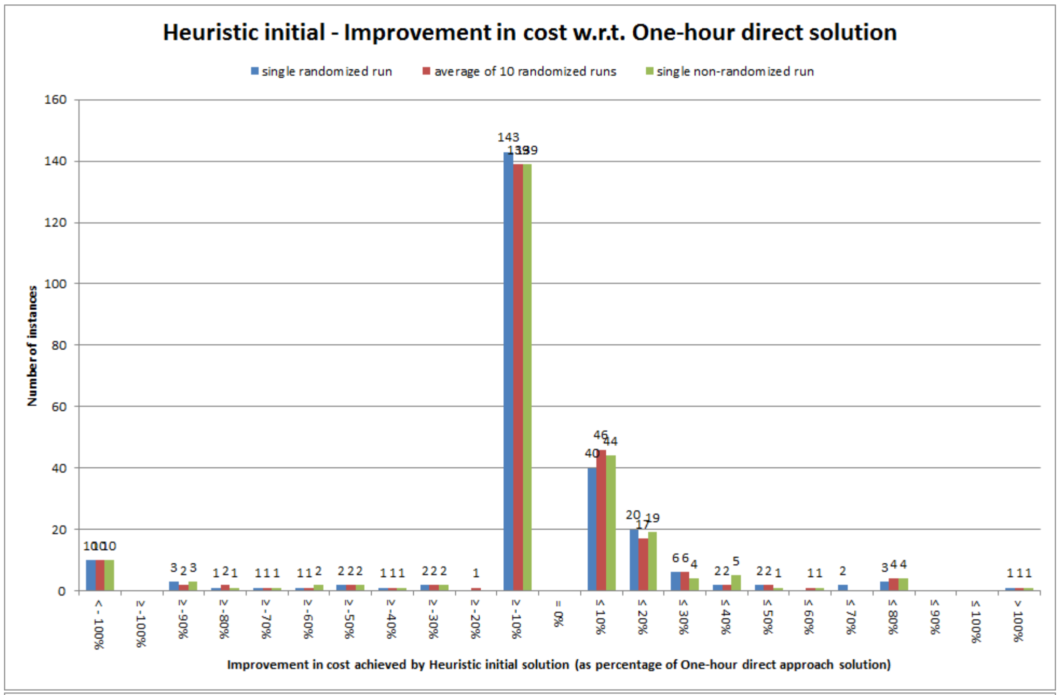


Figure D.3: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the two-minute ‘direct’ approach, shown as a percentage of the direct approach solution value. Results are shown for the single randomized run, the average across the ten randomized runs, and the non-randomized run (top); and for the minimum and maximum across the ten randomized runs (bottom).



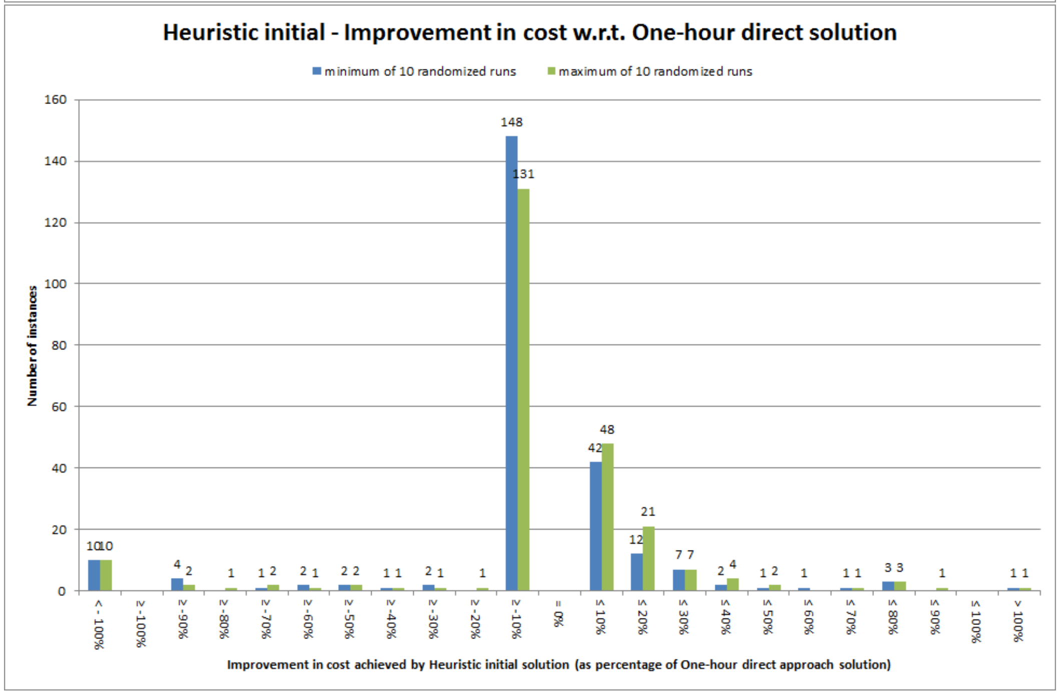
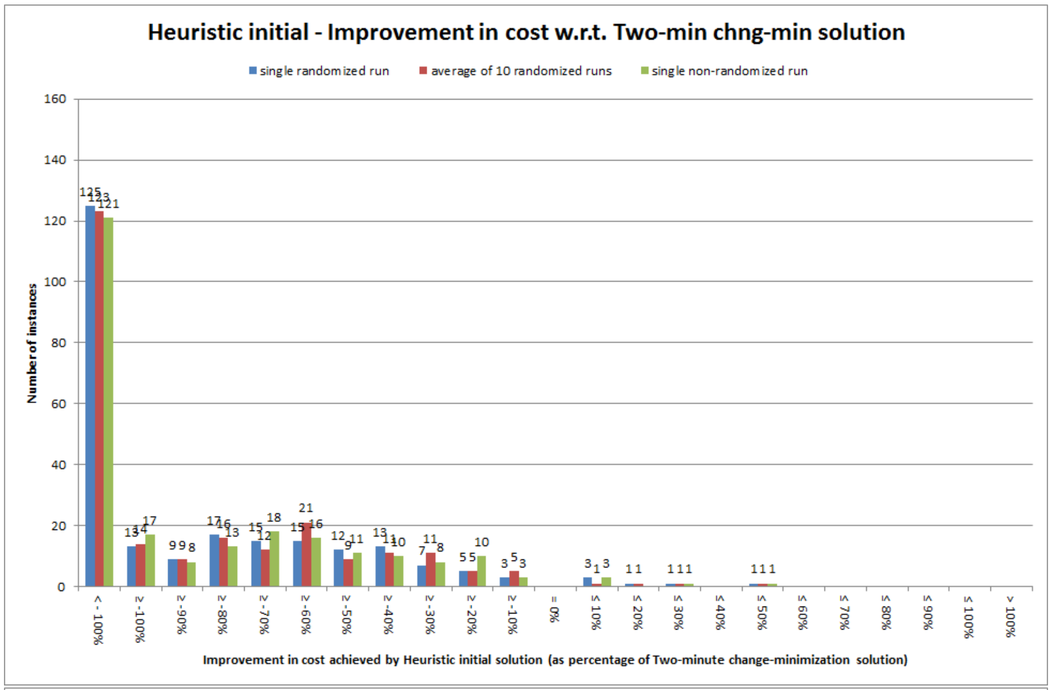


Figure D.4: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the one-hour ‘direct’ approach, shown as a percentage of the direct approach solution value. Results are shown for the single randomized run, the average across the ten randomized runs, and the non-randomized run (top); and for the minimum and maximum across the ten randomized runs (bottom).



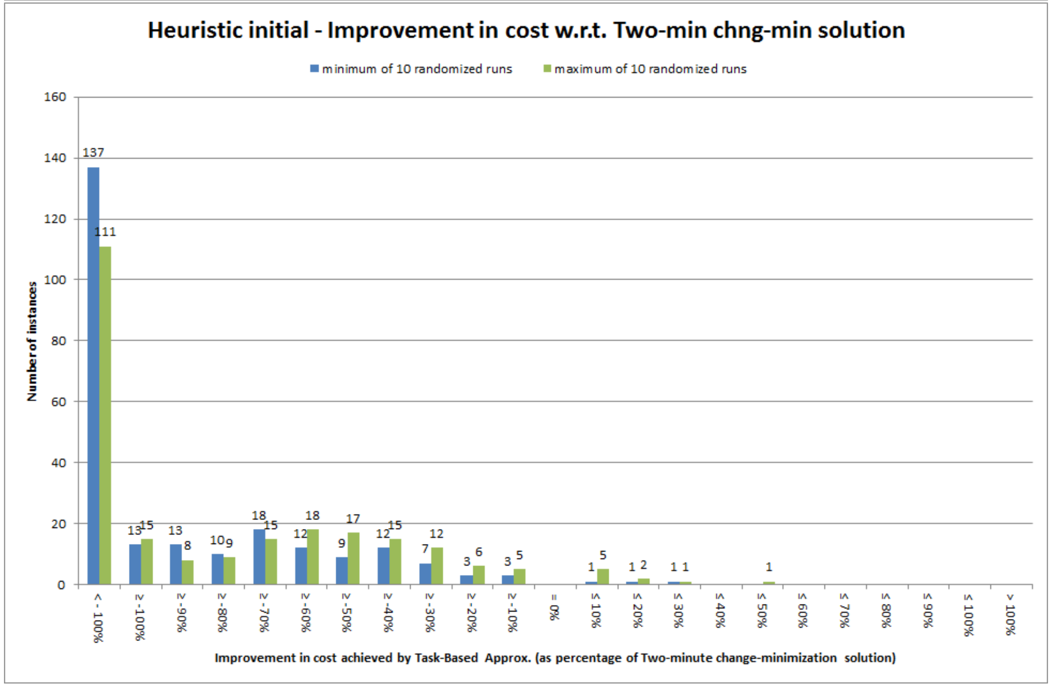
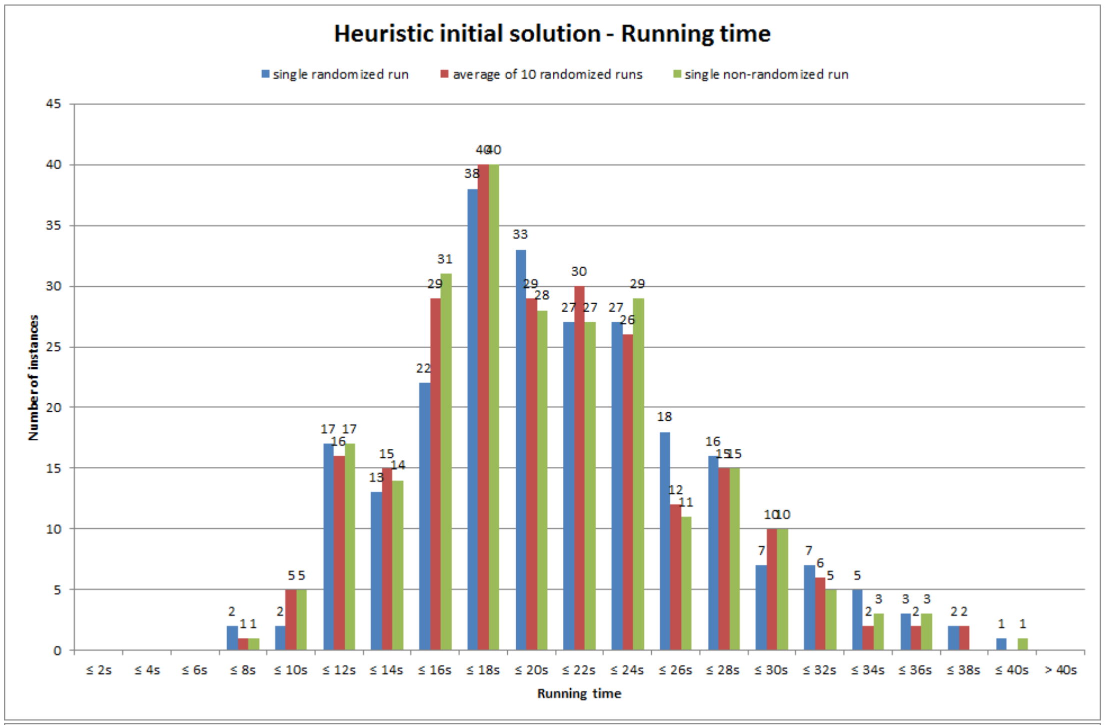


Figure D.5: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the two-minute change-minimization approach, shown as a percentage of the change-minimization solution value. Results are shown for the single randomized run, the average across the ten randomized runs, and the non-randomized run (top); and for the minimum and maximum across the ten randomized runs (bottom).



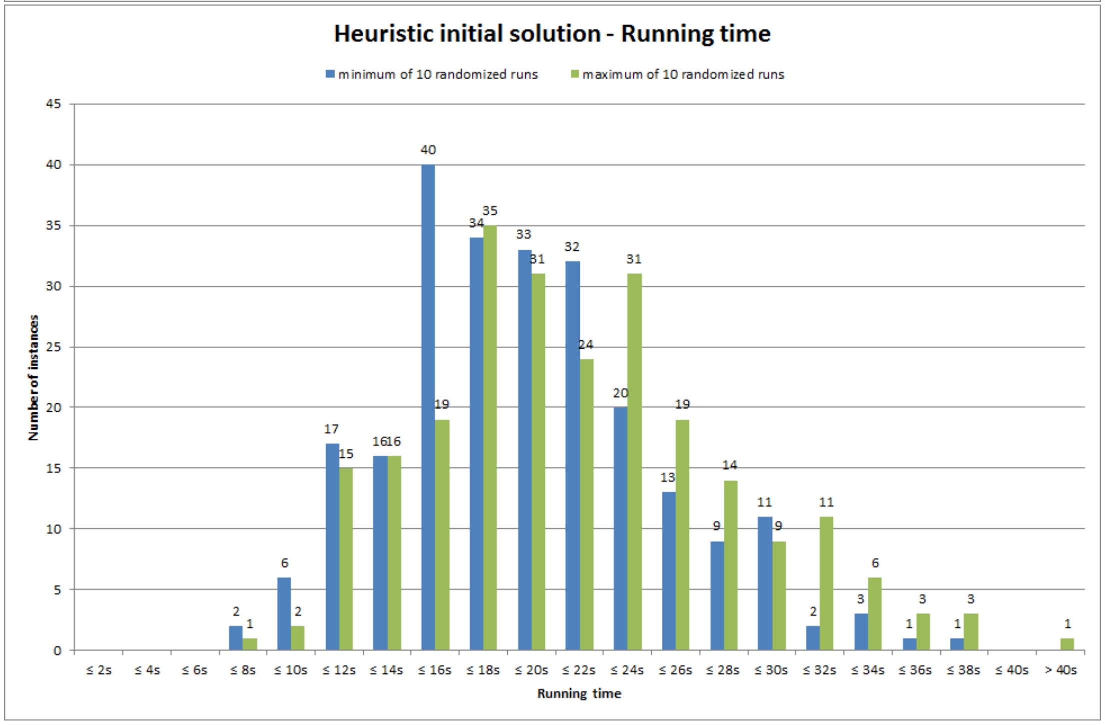
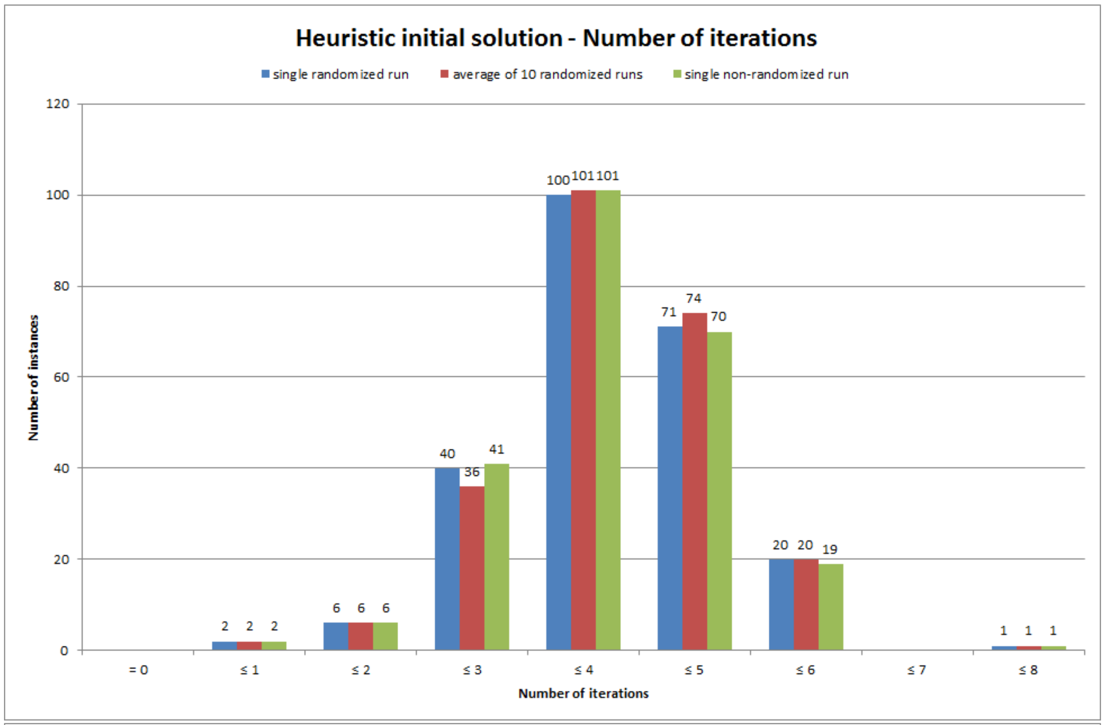


Figure D.6: Histograms summarising total running time of the Heuristic initial solution algorithm for each instance. Results are shown for the single randomized run, the average across the ten randomized runs, and the non-randomized run (top); and for the minimum and maximum across the ten randomized runs (bottom).



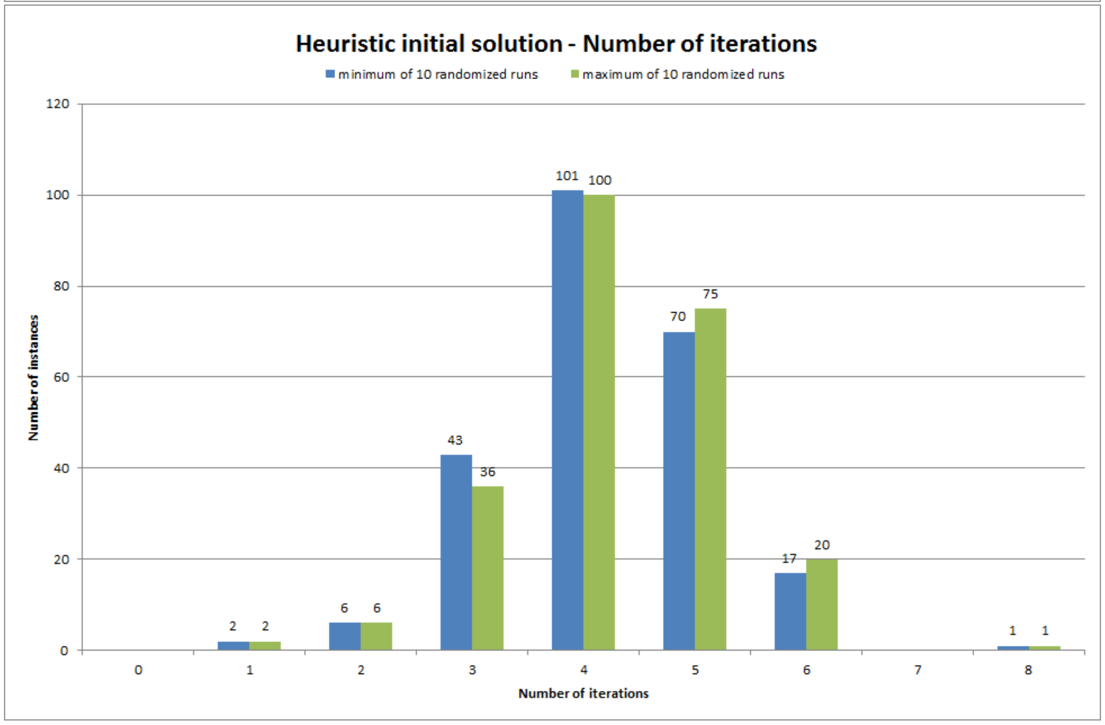
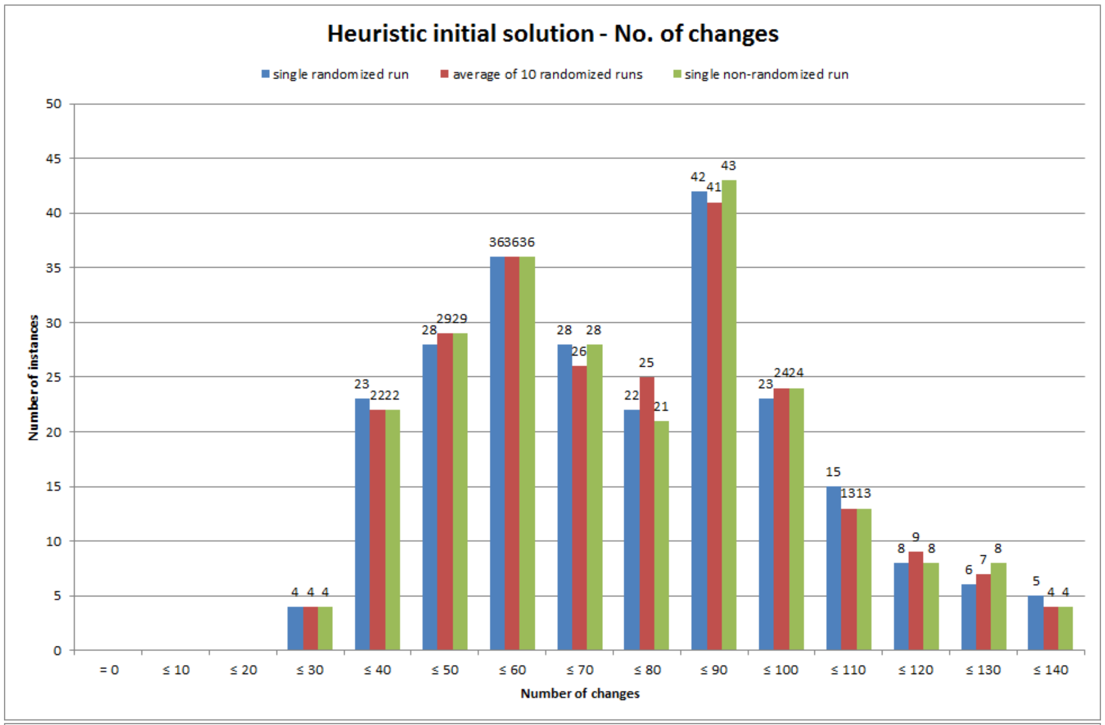


Figure D.7: Histograms summarising the number of iterations carried out by the Heuristic initial solution algorithm for each instance. Results are shown for the single randomized run, the average across the ten randomized runs, and the non-randomized run (top); and for the minimum and maximum across the ten randomized runs (bottom).



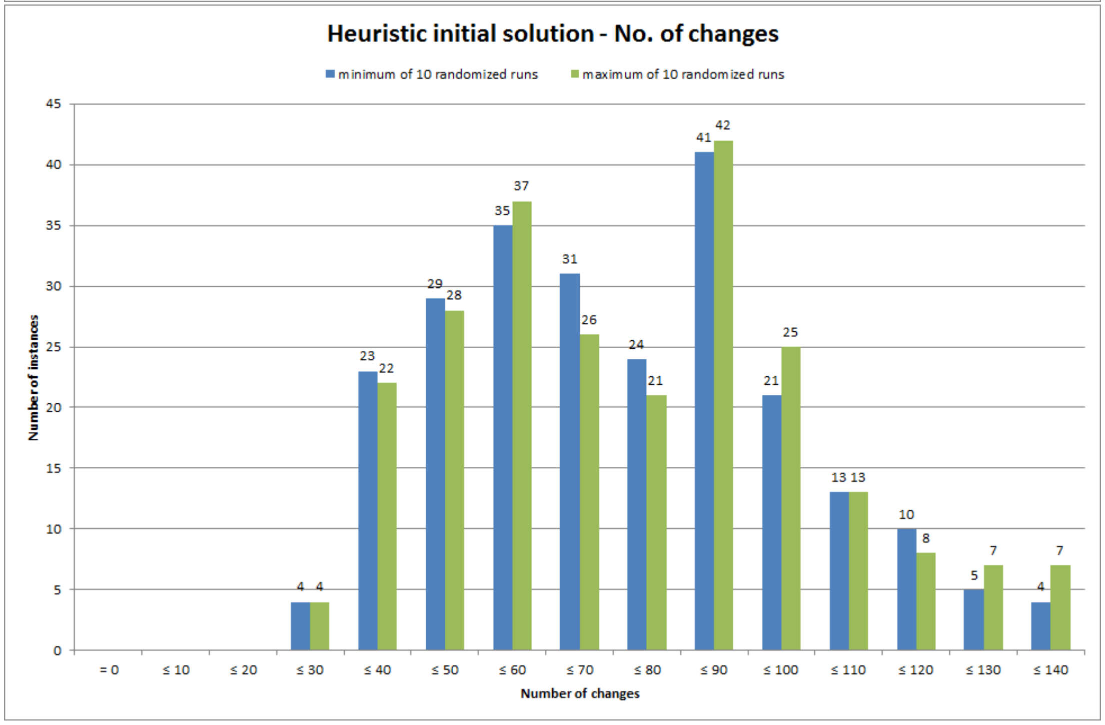
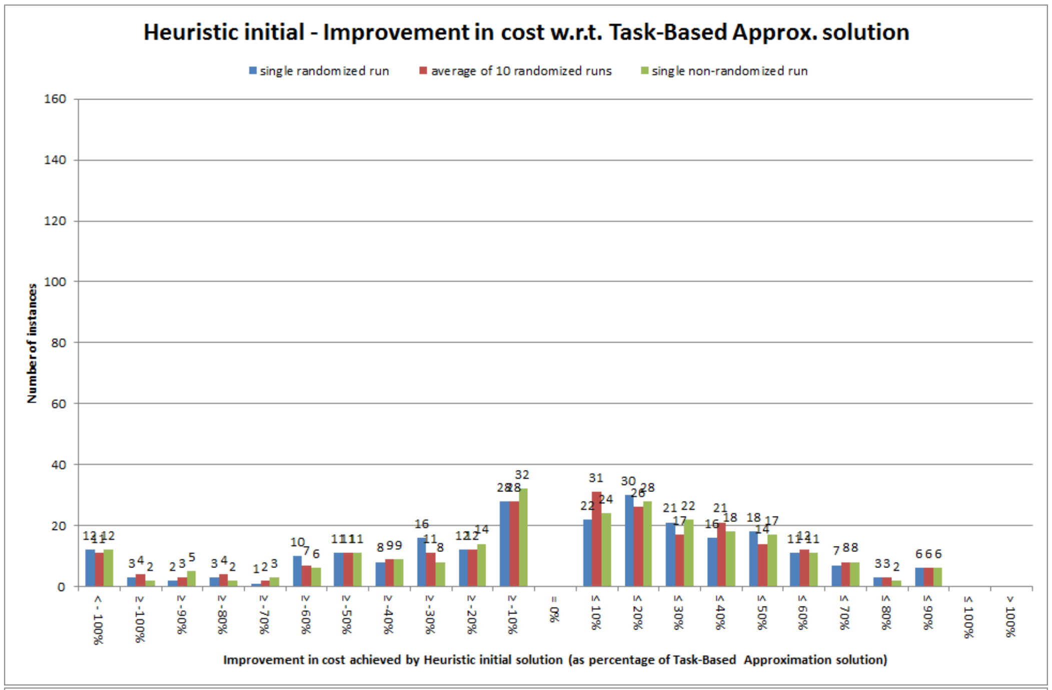


Figure D.8: Histograms summarising the number of changes in the Heuristic initial solution for each instance. Results are shown for the single randomized run, the average across the ten randomized runs, and the non-randomized run (top); and for the minimum and maximum across the ten randomized runs (bottom).



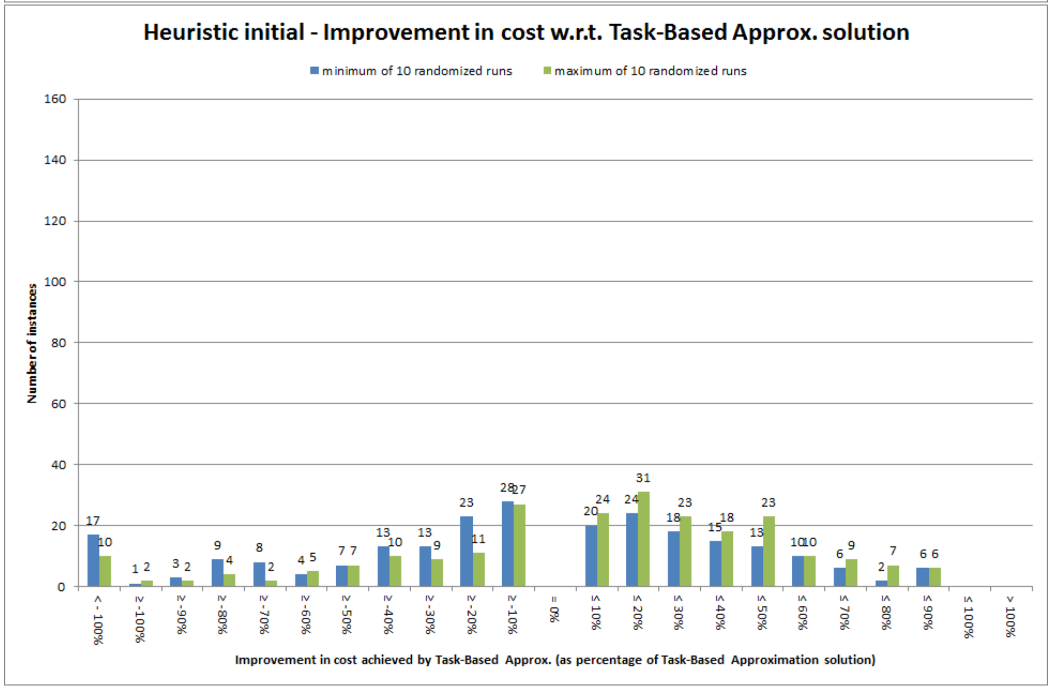


Figure D.9: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the Task-Based Approximation approach, shown as a percentage of the Task-Based Approximation solution value. Results are shown for the single randomized run, the average across the ten randomized runs, and the non-randomized run (top); and for the minimum and maximum across the ten randomized runs (bottom).

D.2 Relating to the main Heuristic algorithm

Here we show graphs to further illustrate the breakdown of results given in section 6.8.4. This is done for each of the metrics discussed in that results section, for each of the factors which are shown statistically to have a significant influence on that metric. Firstly, for reference purposes, Tables D.1 and D.2 shows settings used and the corresponding index for each combination of settings for the heuristic algorithm.

Table D.1: Settings used under each combination for the heuristic algorithm - Part 1.

Combination number	Employees to examine	Employee order	Accept if improve on	Kick if no improv. in	Initial solution
#1	1/3	smarter	current	(no kick)	Task-Based
#2	1/3	random	current	(no kick)	Task-Based
#3	1/3	smarter	current	4+ iters	Task-Based
#4	1/3	random	current	4+ iters	Task-Based
#5	1/3	smarter	current	8+ iters	Task-Based
#6	1/3	random	current	8+ iters	Task-Based
#7	1/3	smarter	best	(no kick)	Task-Based
#8	1/3	random	best	(no kick)	Task-Based
#9	1/3	smarter	best	4+ iters	Task-Based
#10	1/3	random	best	4+ iters	Task-Based
#11	1/3	smarter	best	8+ iters	Task-Based
#12	1/3	random	best	8+ iters	Task-Based
#13	all	smarter	current	(no kick)	Task-Based
#14	all	random	current	(no kick)	Task-Based
#15	all	smarter	current	4+ iters	Task-Based
#16	all	random	current	4+ iters	Task-Based
#17	all	smarter	current	8+ iters	Task-Based
#18	all	random	current	8+ iters	Task-Based
#19	all	smarter	best	(no kick)	Task-Based
#20	all	random	best	(no kick)	Task-Based
#21	all	smarter	best	4+ iters	Task-Based
#22	all	random	best	4+ iters	Task-Based
#23	all	smarter	best	8+ iters	Task-Based
#24	all	random	best	8+ iters	Task-Based

Table D.2: Settings used under each combination for the heuristic algorithm - Part 2.

Combination number	Employees to examine	Employee order	Accept if improve on	Kick if no improv. in	Initial solution
#25	1/3	smarter	current	(no kick)	Heuristic
#26	1/3	random	current	(no kick)	Heuristic
#27	1/3	smarter	current	4+ iters	Heuristic
#28	1/3	random	current	4+ iters	Heuristic
#29	1/3	smarter	current	8+ iters	Heuristic
#30	1/3	random	current	8+ iters	Heuristic
#31	1/3	smarter	best	(no kick)	Heuristic
#32	1/3	random	best	(no kick)	Heuristic
#33	1/3	smarter	best	4+ iters	Heuristic
#34	1/3	random	best	4+ iters	Heuristic
#35	1/3	smarter	best	8+ iters	Heuristic
#36	1/3	random	best	8+ iters	Heuristic
#37	all	smarter	current	(no kick)	Heuristic
#38	all	random	current	(no kick)	Heuristic
#39	all	smarter	current	4+ iters	Heuristic
#40	all	random	current	4+ iters	Heuristic
#41	all	smarter	current	8+ iters	Heuristic
#42	all	random	current	8+ iters	Heuristic
#43	all	smarter	best	(no kick)	Heuristic
#44	all	random	best	(no kick)	Heuristic
#45	all	smarter	best	4+ iters	Heuristic
#46	all	random	best	4+ iters	Heuristic
#47	all	smarter	best	8+ iters	Heuristic
#48	all	random	best	8+ iters	Heuristic

D.2.1 Gap to the best known bound

As shown in Table 6.2 in section 6.8.4.1, three of the heuristic settings had a significant influence on the gap to the best known bound from the solutions found. Here we show the following graphs in relation to this:

- A breakdown of the two-minute results according to the number of employees to examine at each iteration (Figure D.10);
- A breakdown according to the kick activation rules used, for both the one-minute results (Figure D.11) and the two-minute results (Figure D.12); and
- A breakdown according to the type of initial solution used, for both the one-minute and two-minute results (shown together in Figure D.13).

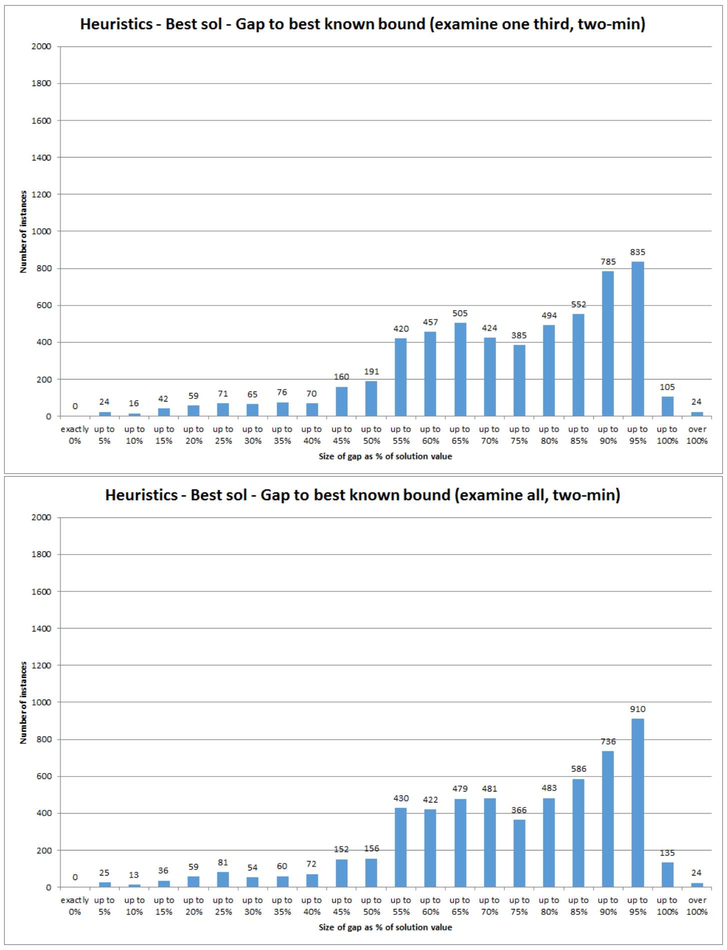


Figure D.10: Gap to best known bound from solutions found after two minutes, broken down by whether one third of employees are examined at each iteration (top) or all employees are examined (bottom).

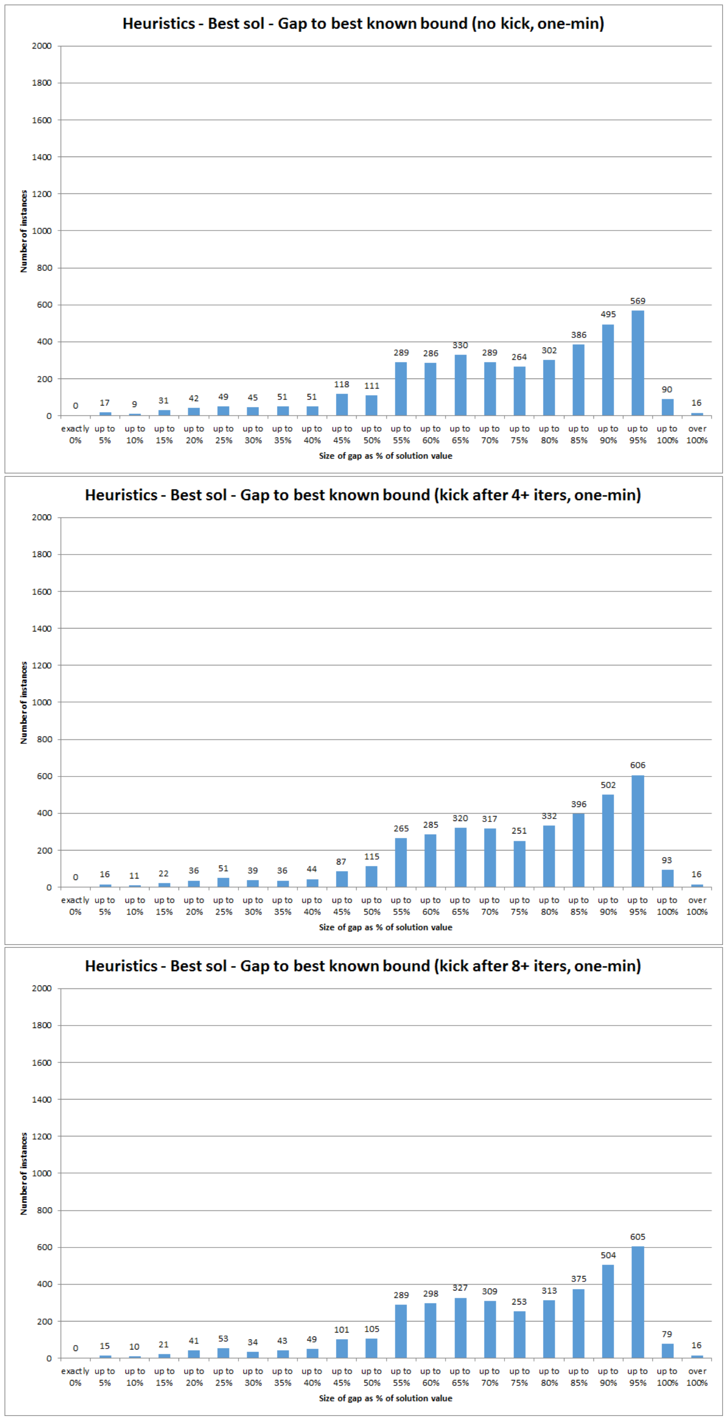


Figure D.11: Gap to best known bound from solutions found after one minute, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

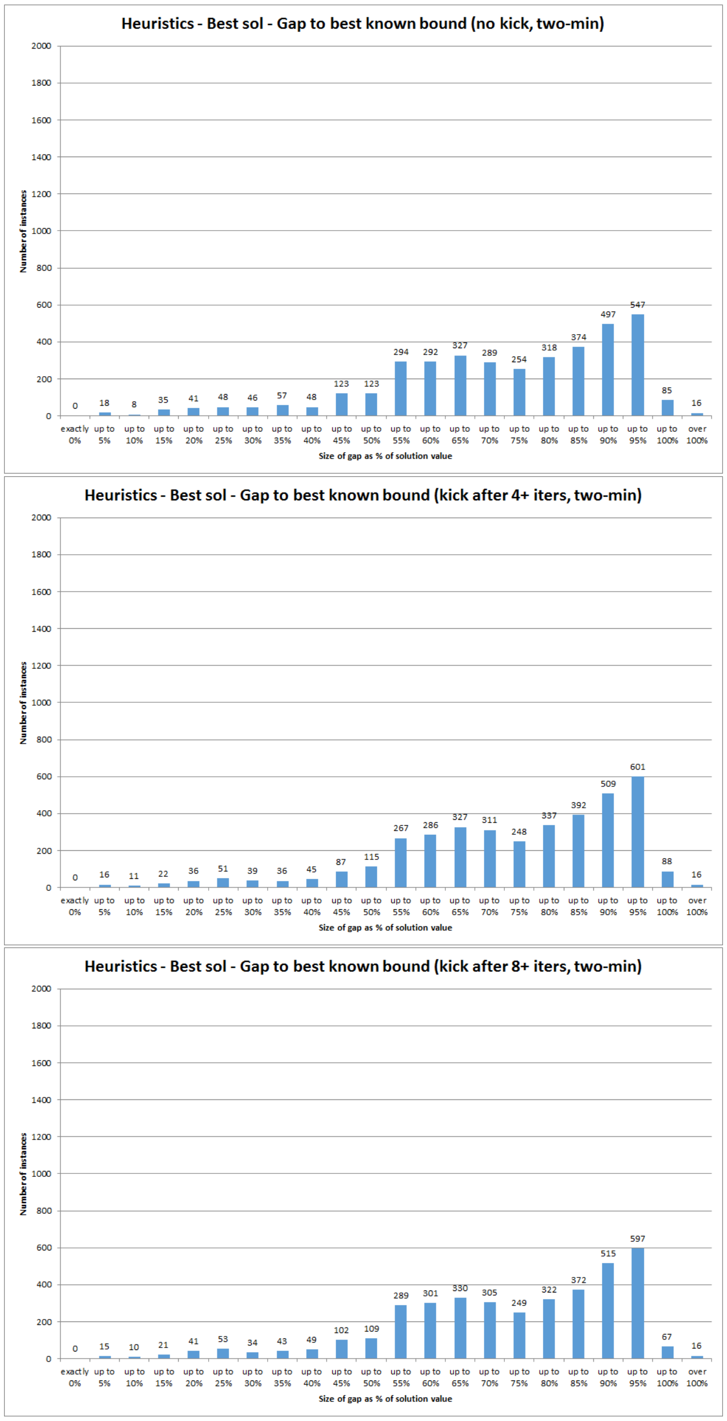


Figure D.12: Gap to best known bound from solutions found after two minutes, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

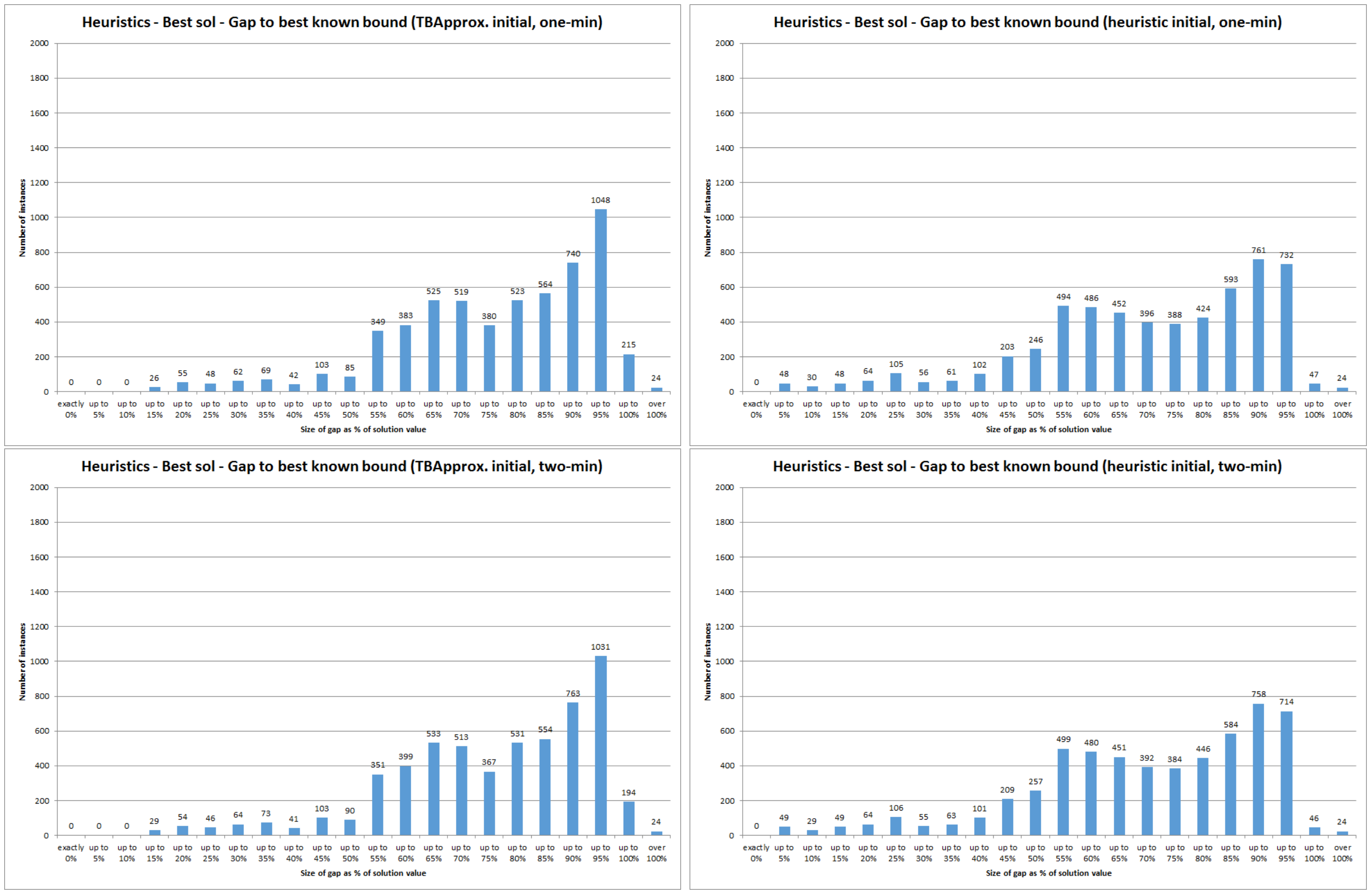


Figure D.13: Gap to best known bound from solutions found after one minute (top) and two minutes (bottom), broken down by whether the Task-Based Approximation (left hand side) or Heuristic (right hand side) initial solution was used.

D.2.2 Gap to the best known solution

As shown in Table 6.3 in section 6.8.4.2, three of the heuristic settings had a significant influence on the gap to the best known solution from the solutions found. Here we show the following graphs in relation to this:

- A breakdown according to the number of employees to examine at each iteration, for both the one-minute and two-minute results (shown together in Figure D.14);
- A breakdown according to the kick activation rules used, for both the one-minute results (Figure D.15) and the two-minute results (Figure D.16); and
- A breakdown according to the type of initial solution used, for both the one-minute and two-minute results (shown together in Figure D.17).

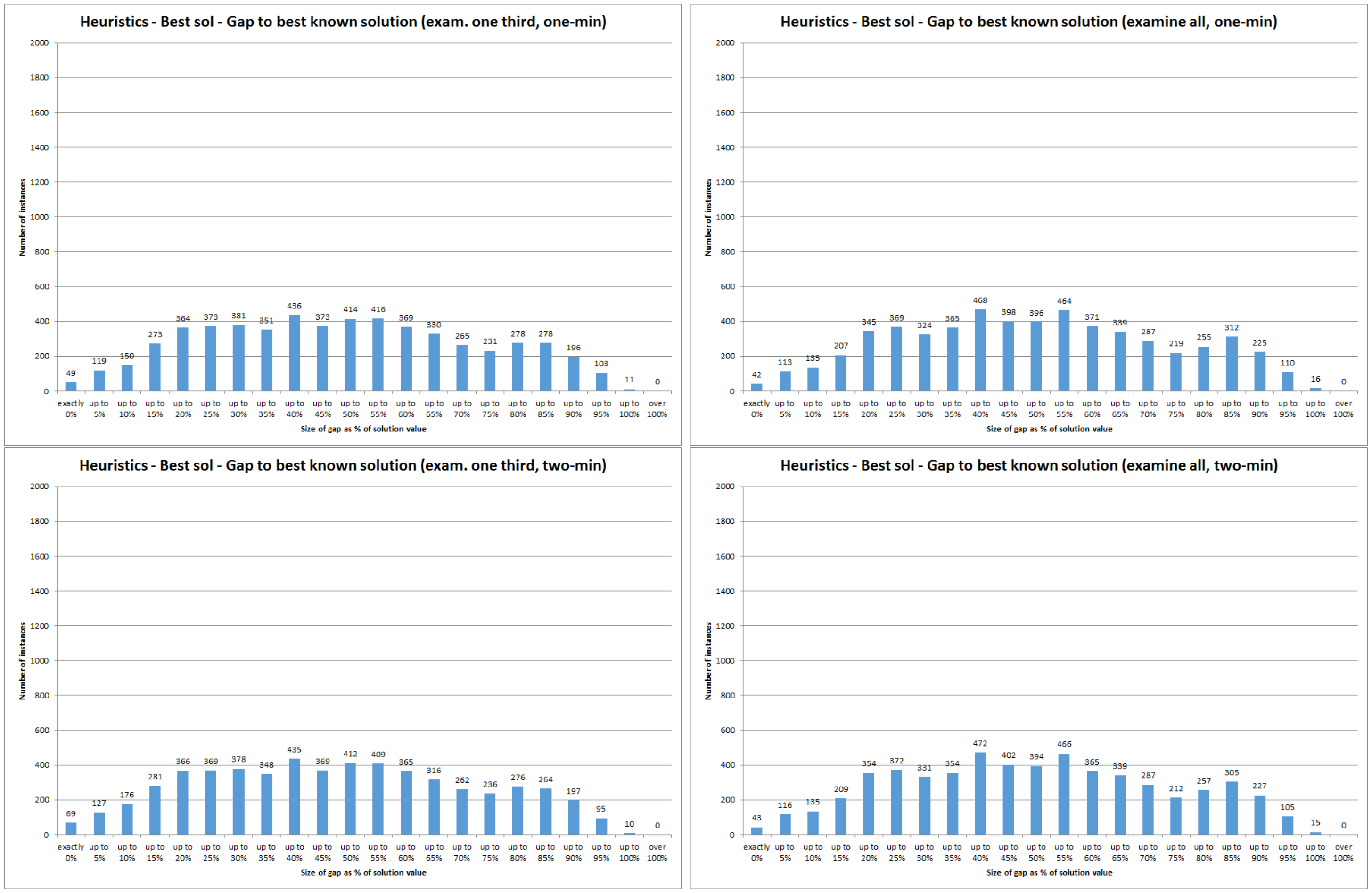


Figure D.14: Gap to best known solution overall from solutions found after one minute (top) and two minutes (bottom), broken down by whether one third of employees are examined at each iteration (left hand side) or all employees are examined (right hand side).

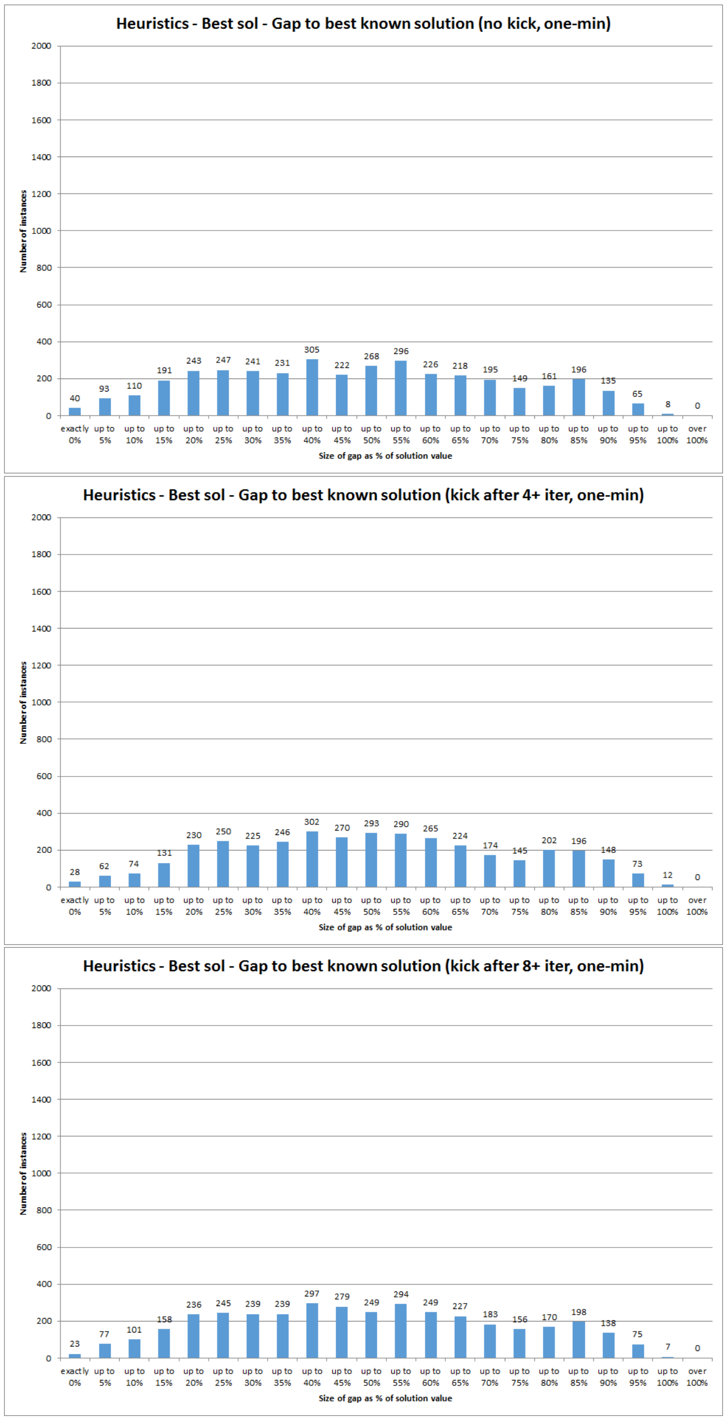


Figure D.15: Gap to best known solution overall from solutions found after one minute, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

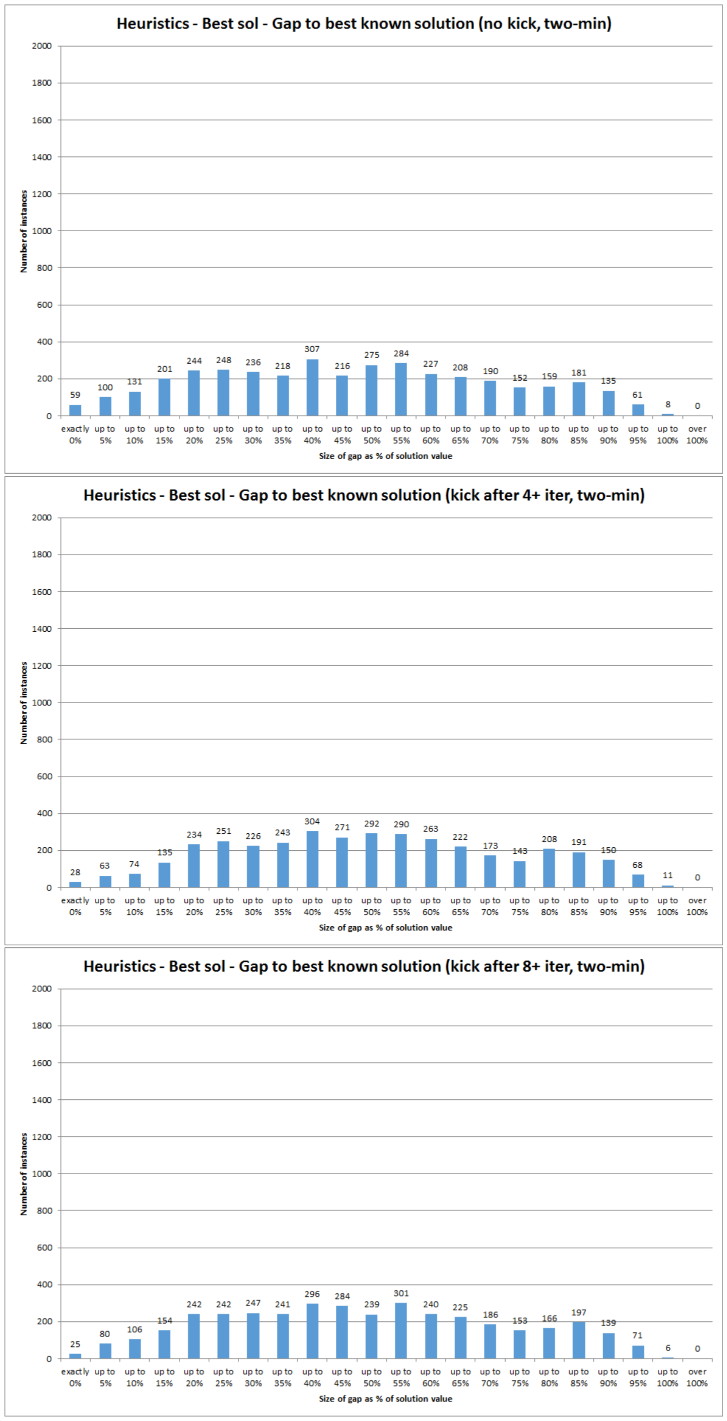


Figure D.16: Gap to best known solution overall from solutions found after two minutes, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

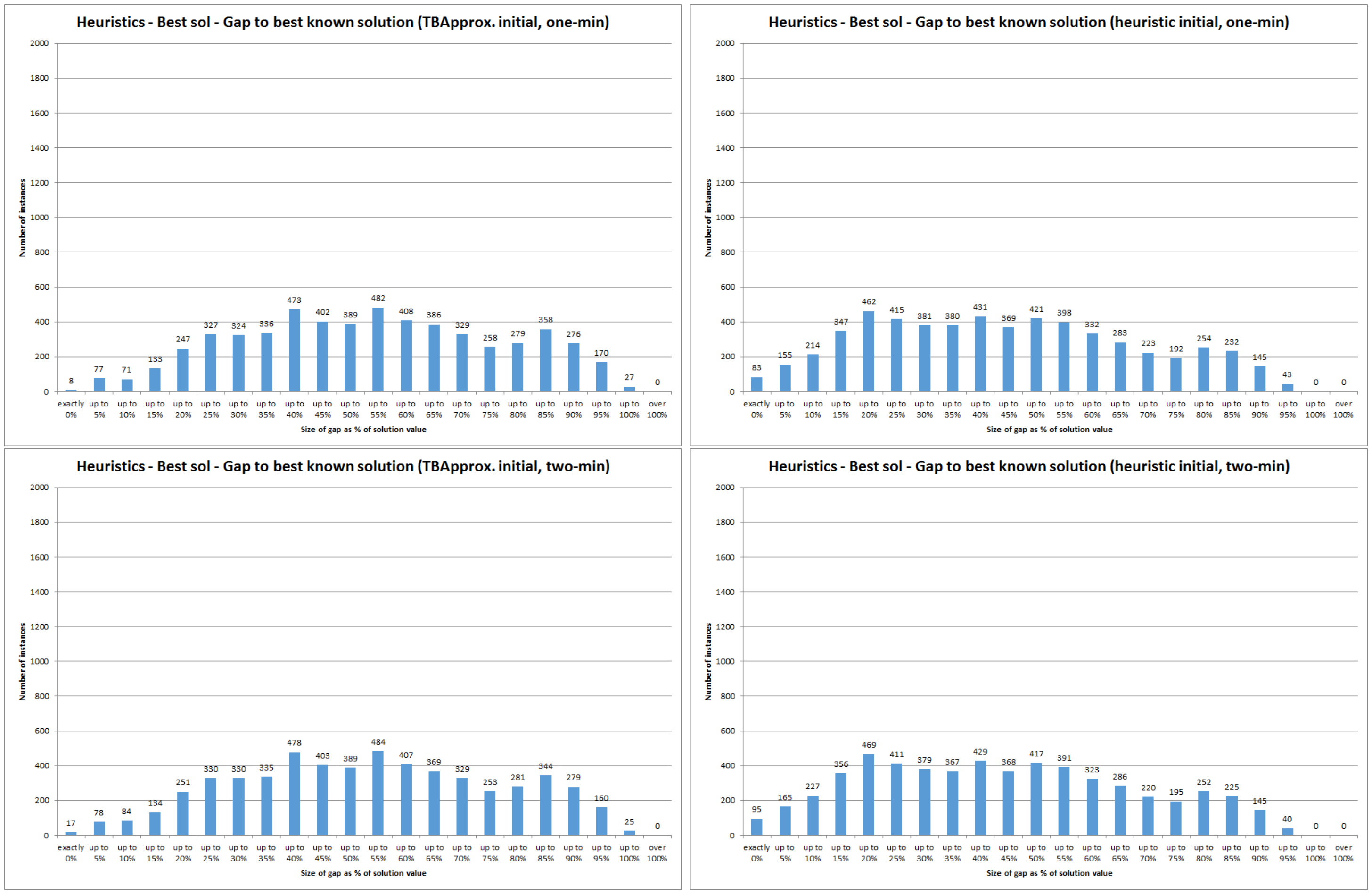


Figure D.17: Gap to best known solution overall from solutions found after one minute (top) and two minutes (bottom), broken down by whether the Task-Based Approximation (left hand side) or Heuristic (right hand side) initial solution was used.

D.2.3 Improvement on initial solution

As shown in Table 6.4 in section 6.8.4.3, three of the heuristic settings had a significant influence on the improvement on the initial solution achieved by the algorithm. Here we show the following graphs in relation to this:

- A breakdown according to the number of employees to examine at each iteration, for both the one-minute and two-minute results (shown together in Figure D.18);
- A breakdown according to the kick activation rules used, for both the one-minute results (Figure D.19) and the two-minute results (Figure D.20); and
- A breakdown according to the type of initial solution used, for both the one-minute and two-minute results (shown together in Figure D.21).

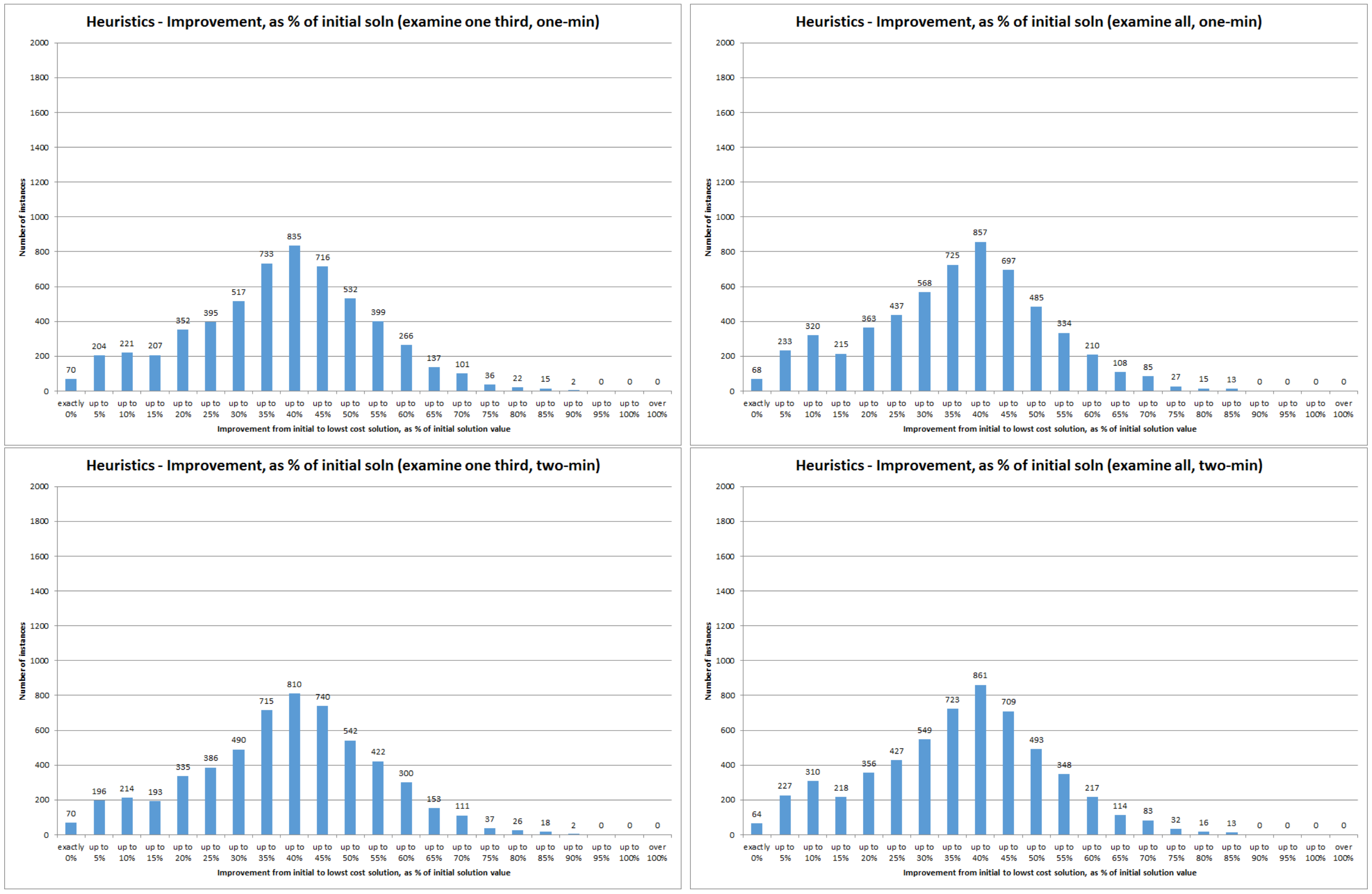


Figure D.18: Improvement on the initial solution achieved within one minute (top) and two minutes (bottom) given as a percentage of the initial solution value, broken down by whether one third of employees are examined at each iteration (left hand side) or all employees are examined (right hand side).

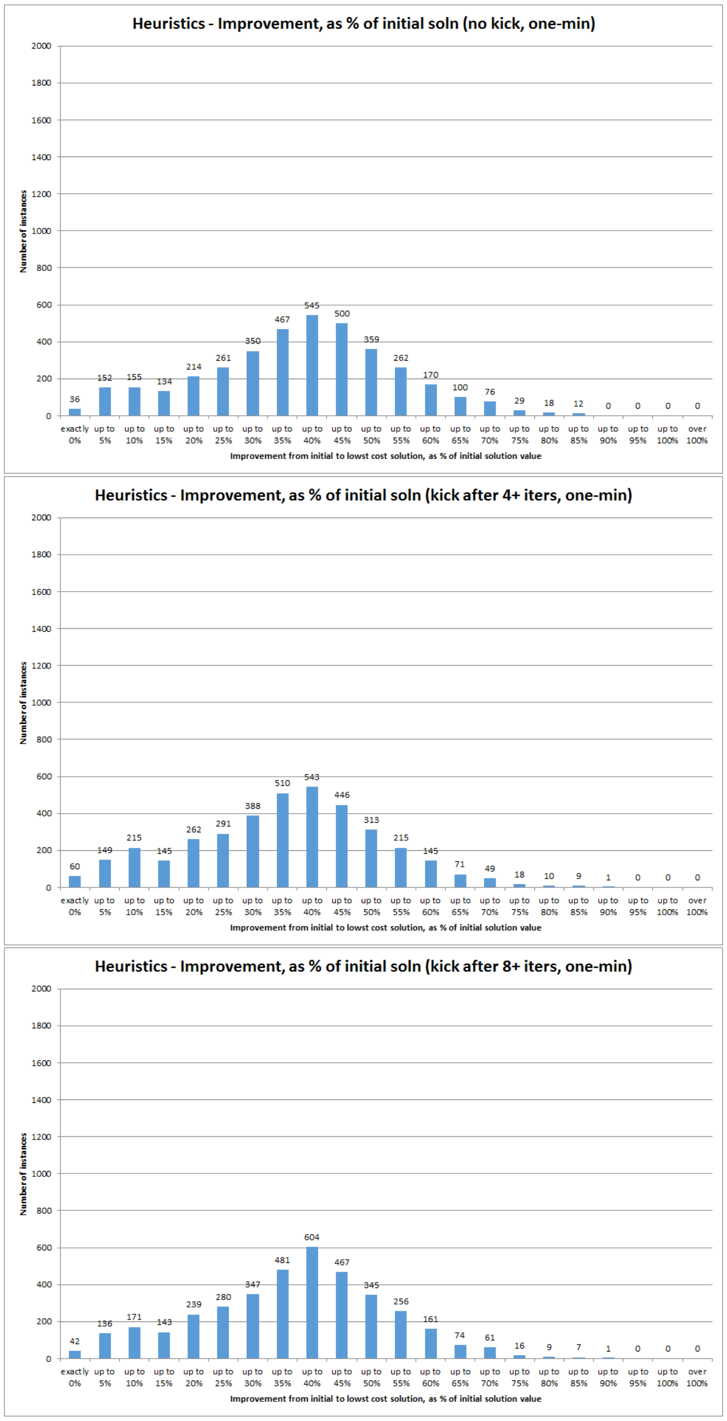


Figure D.19: Improvement on the initial solution achieved within one minute given as a percentage of the initial solution value, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

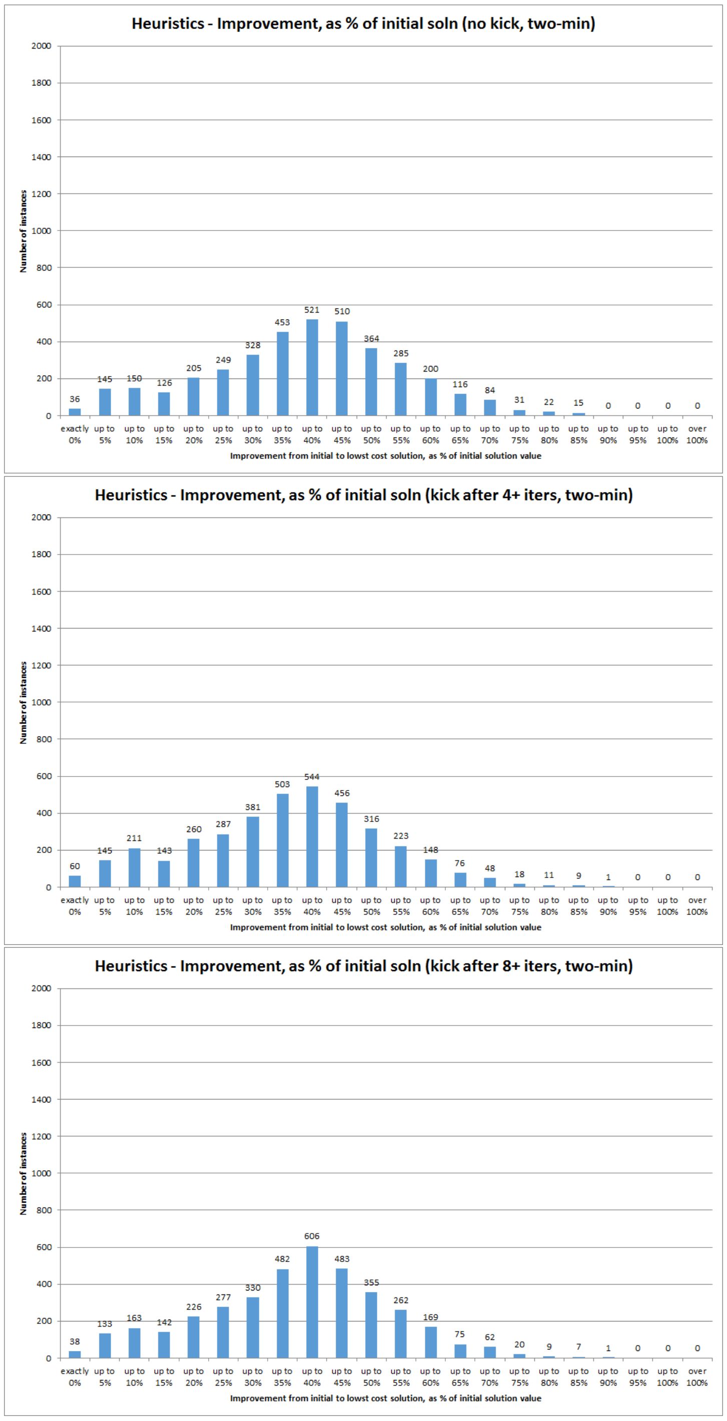


Figure D.20: Improvement on the initial solution achieved within two minutes given as a percentage of the initial solution value, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

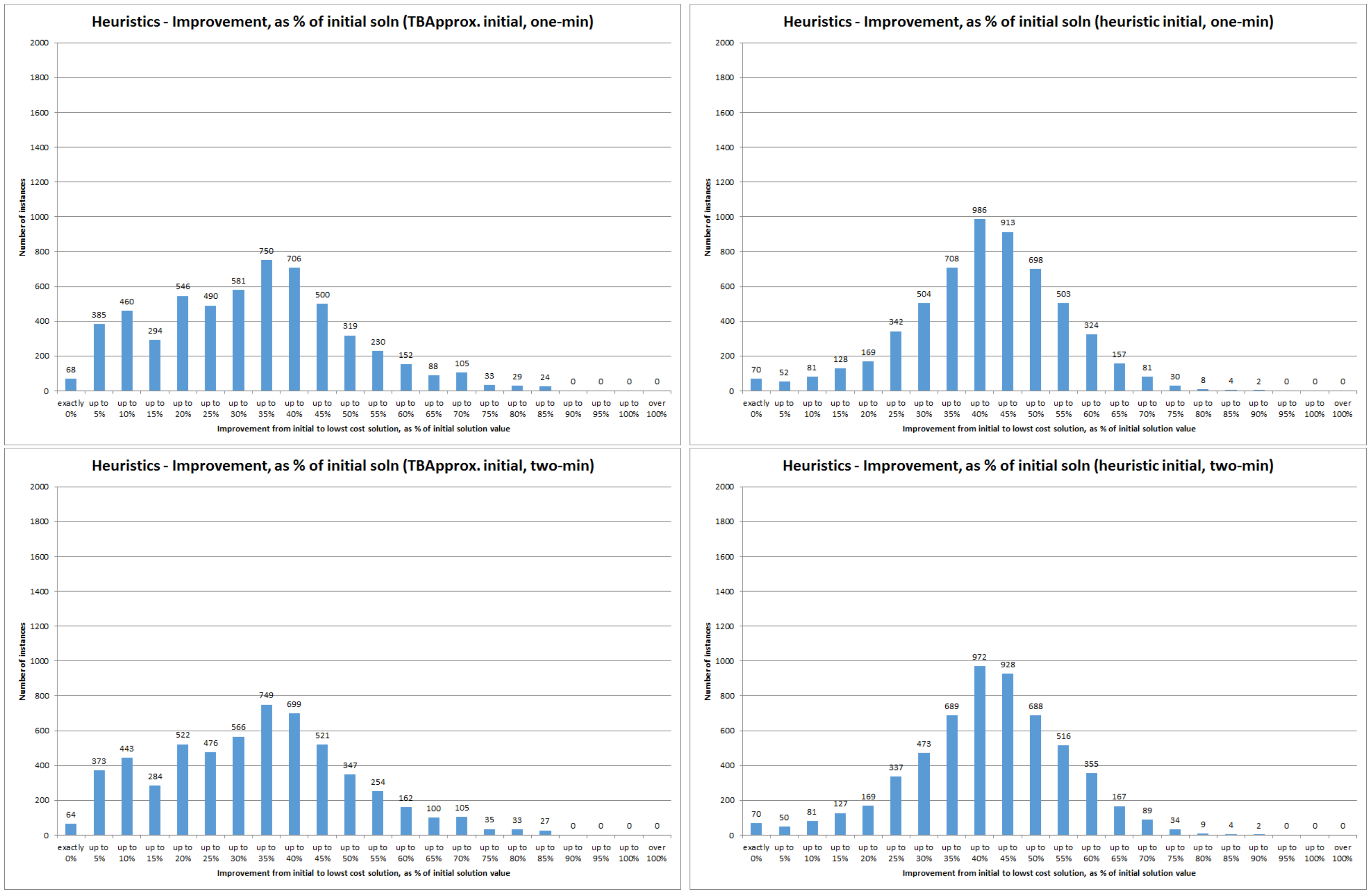


Figure D.21: Improvement on the initial solution achieved within one minute (top) and two minutes (bottom) given as a percentage of the initial solution value, broken down by whether the Task-Based Approximation (left hand side) or Heuristic (right hand side) initial solution was used.

D.2.4 Number of changes

As shown in Table 6.5 in section 6.8.4.4, four of the heuristic settings had a significant influence on the number of changes in the first lowest-cost solution found. Here we show the following graphs in relation to this:

- A breakdown according to the number of employees to examine at each iteration, for both the one-minute and two-minute results (shown together in Figure D.22);
- A breakdown of the two-minute results according to the order in which employees are examined at each iteration (Figure D.23);
- A breakdown according to the kick activation rules used, for both the one-minute results (Figure D.24) and the two-minute results (Figure D.25); and
- A breakdown according to the type of initial solution used, for both the one-minute and two-minute results (shown together in Figure D.26).

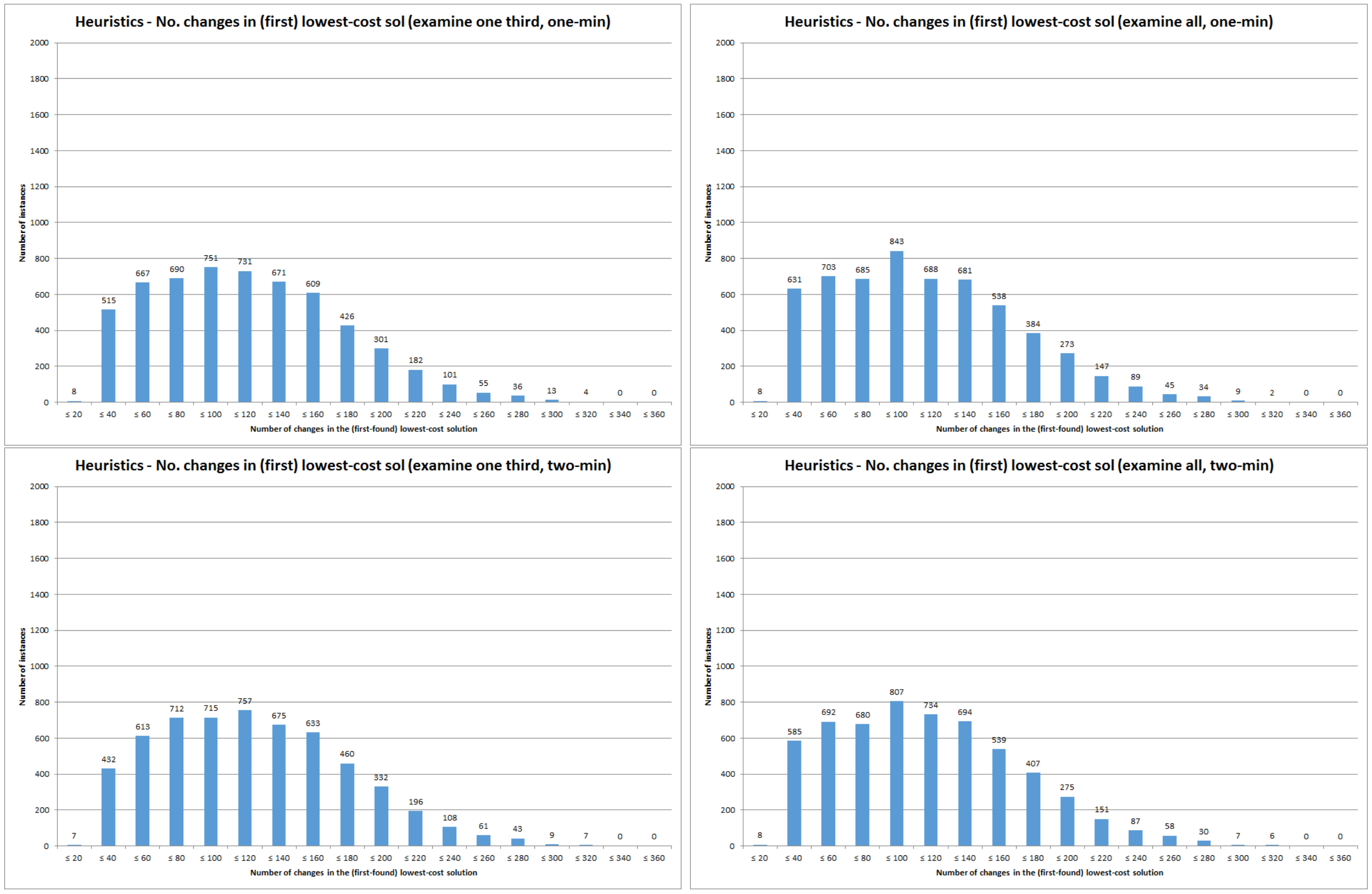


Figure D.22: Number of changes in the first lowest-cost solution found within one minute (top) and two minutes (bottom), broken down by whether one third of employees are examined at each iteration (left hand side) or all employees are examined (right hand side).

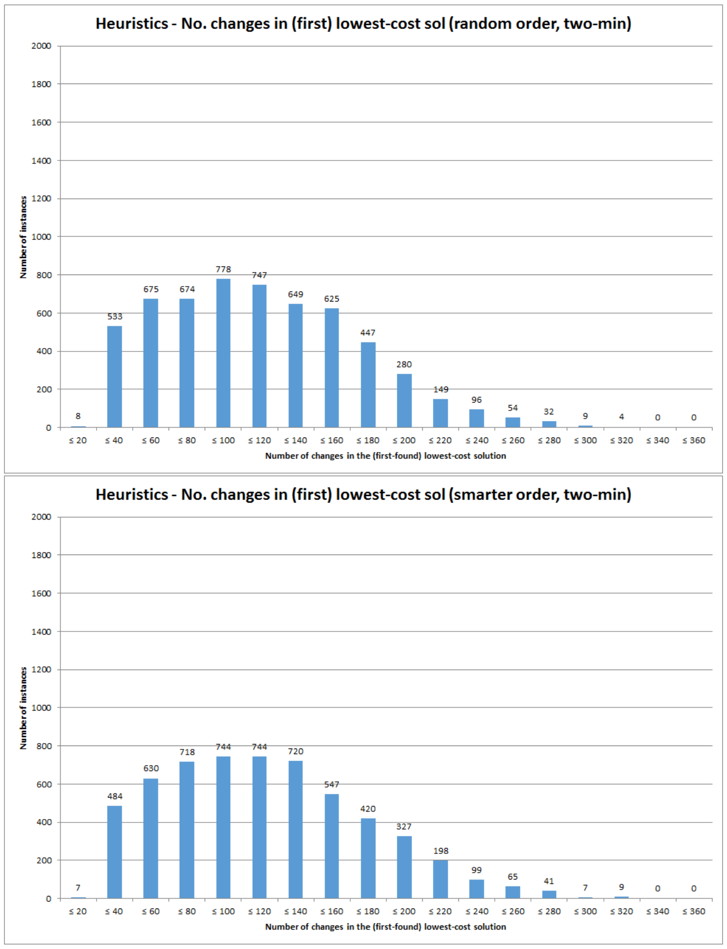


Figure D.23: Number of changes in the first lowest-cost solution found within two minutes, broken down by whether random ordering (top) or more tailored ‘smarter’ ordering (bottom) of employees is used at each iteration.

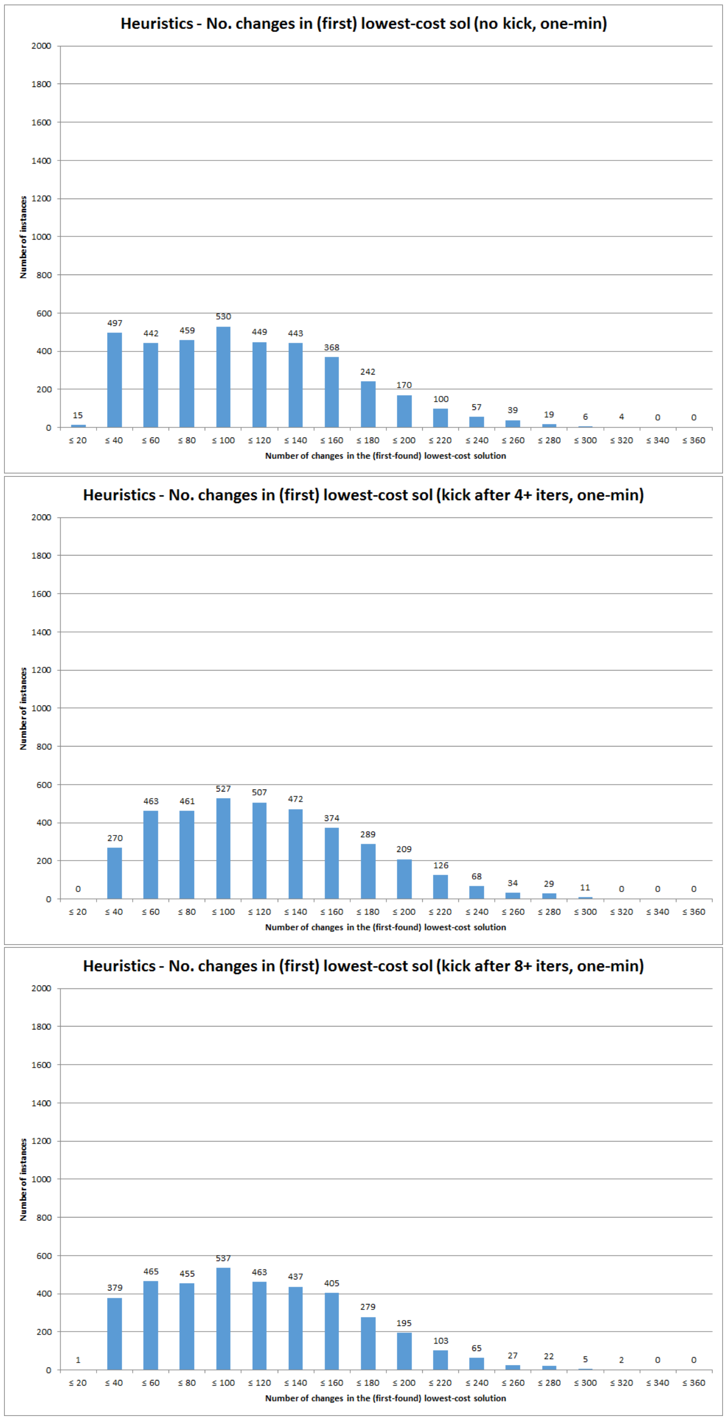


Figure D.24: Number of changes in the first lowest-cost solution found within one minute, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

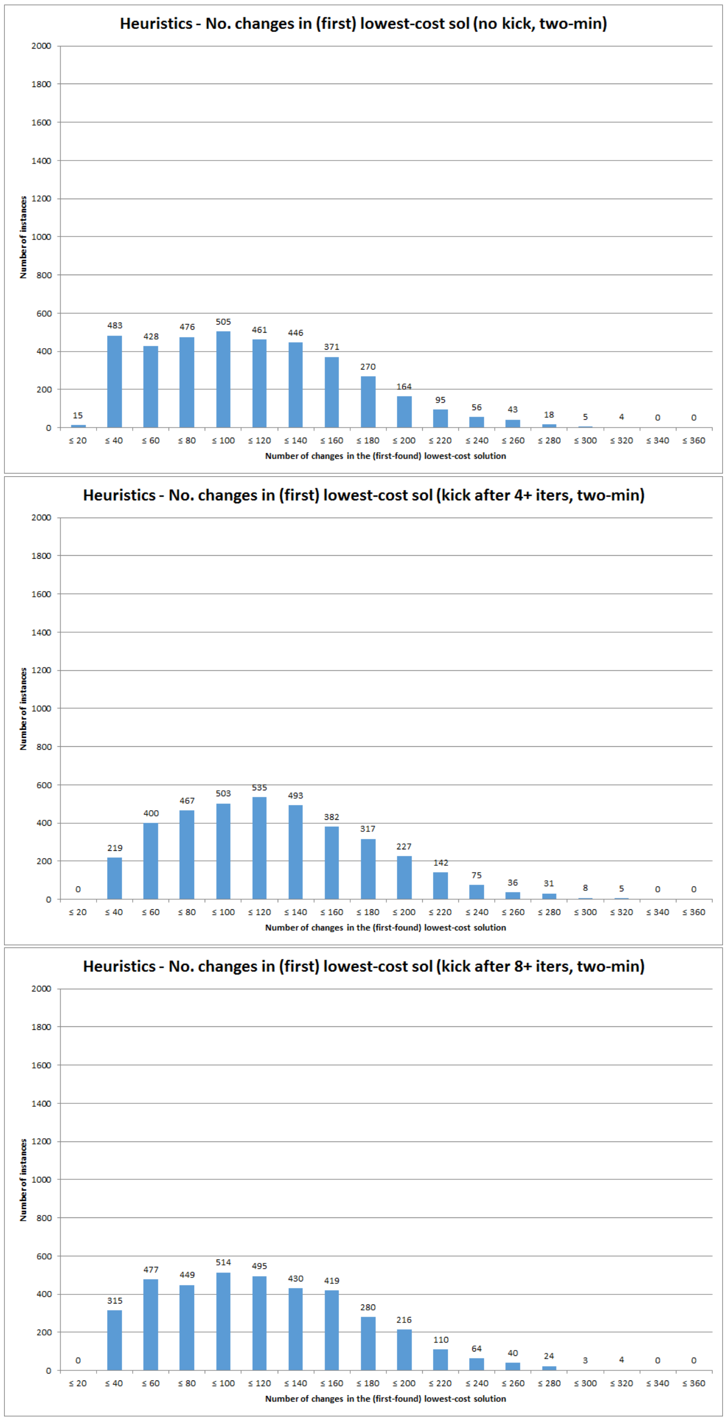


Figure D.25: Number of changes in the first lowest-cost solution found within two minutes, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

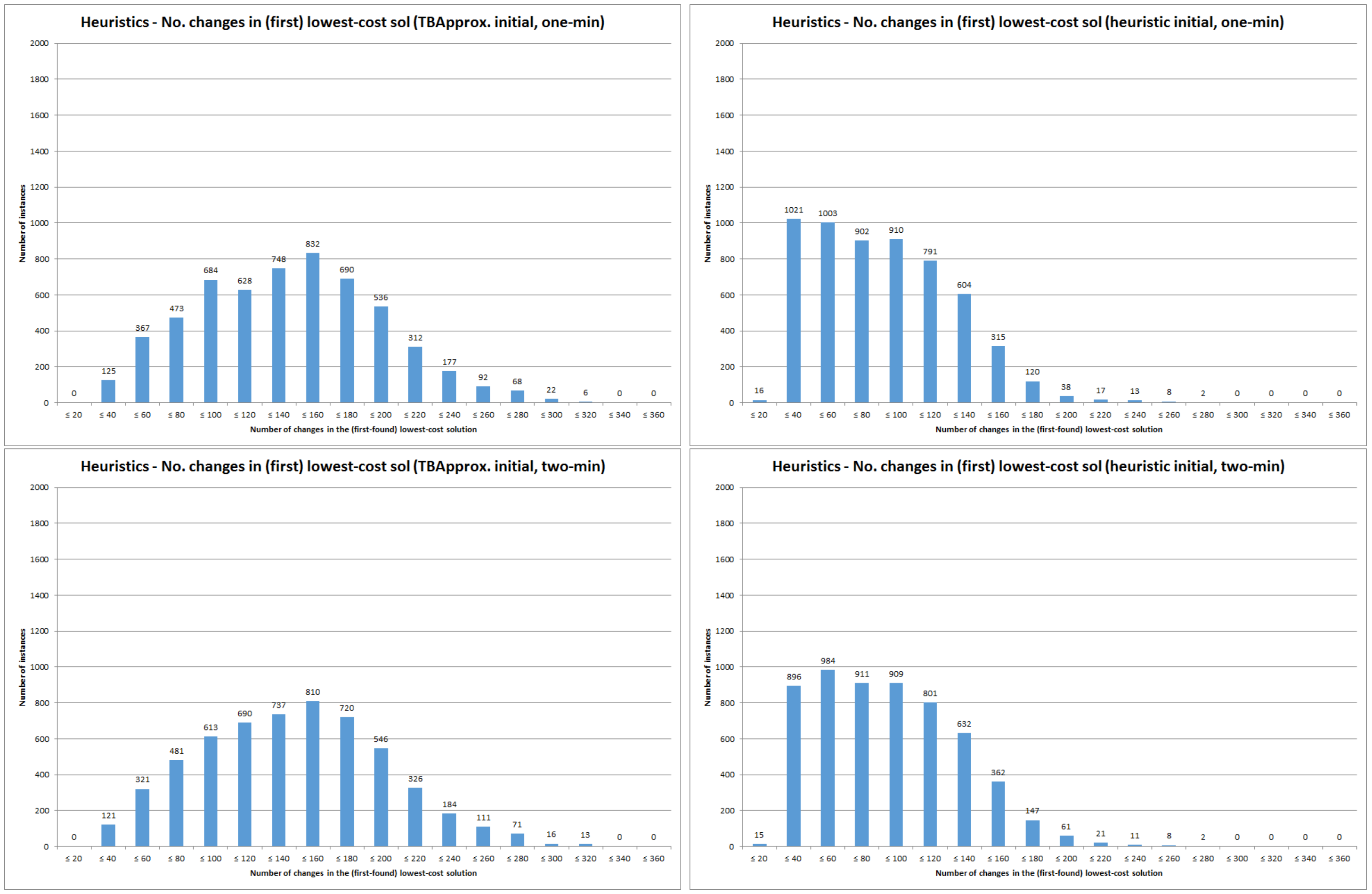


Figure D.26: Number of changes in the first lowest-cost solution found within one minute (top) and two minutes (bottom), broken down by whether the Task-Based Approximation (left hand side) or Heuristic (right hand side) initial solution was used.

D.2.5 Number of best solutions

As shown in Table 6.6 in section 6.8.4.5, four of the heuristic settings had a significant influence on the number of solutions found with equal-best cost. Here we show the following graphs in relation to this:

- A breakdown according to the number of employees to examine at each iteration, for both the one-minute and two-minute results (shown together in Figure D.27);
- A breakdown according to the order in which employees are examined at each iteration, for both the one-minute and two-minute results (shown together in Figure D.28);
- A breakdown according to the kick activation rules used, for both the one-minute results (Figure D.29) and the two-minute results (Figure D.30); and
- A breakdown according to the type of initial solution used, for both the one-minute and two-minute results (shown together in Figure D.31).

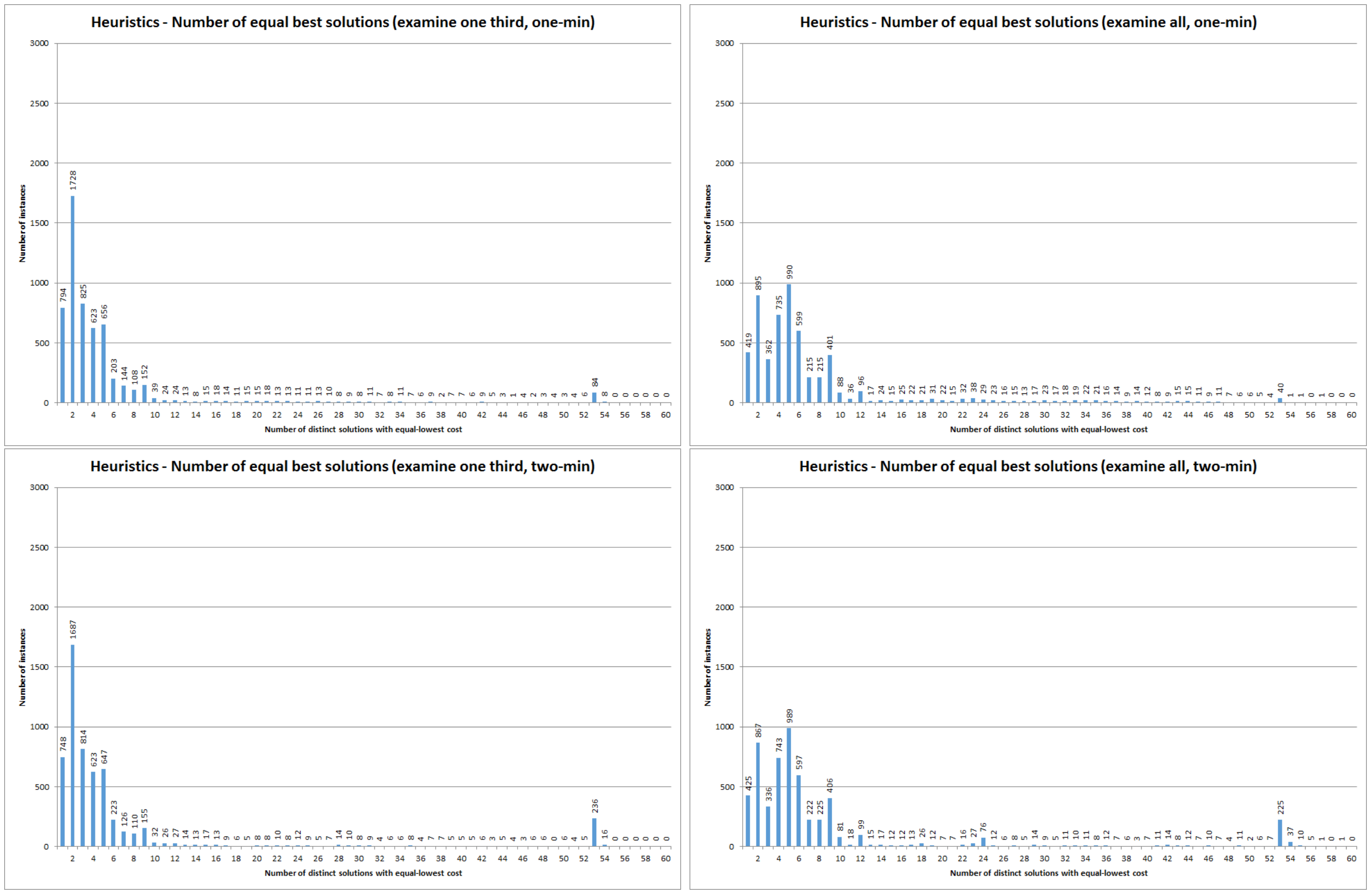


Figure D.27: Number of solutions found with equal-best cost after one minute (top) and after two minutes (bottom), broken down by whether one third of employees are examined at each iteration (left hand side) or all employees are examined (right hand side).

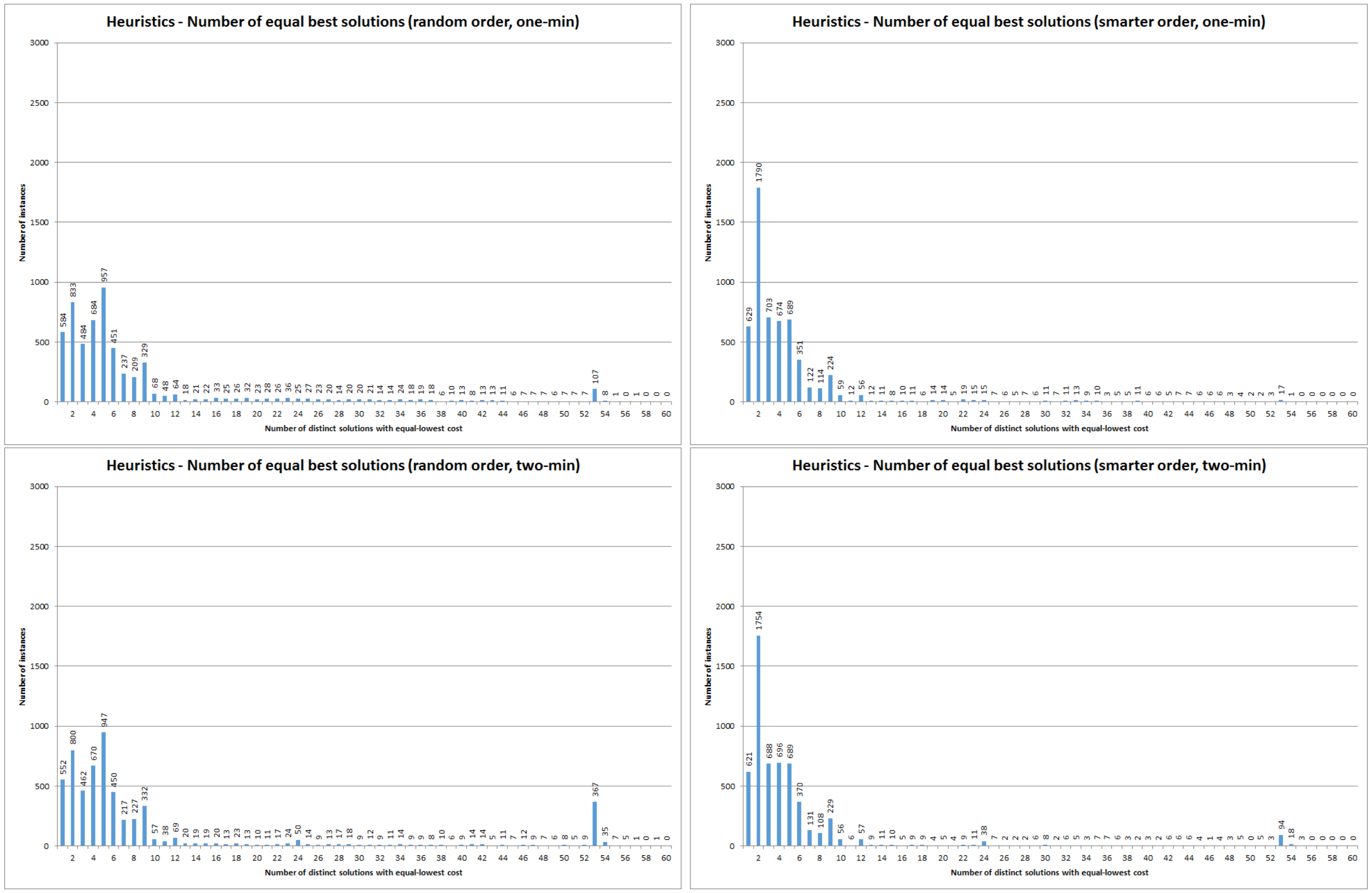


Figure D.28: Number of solutions found with equal-best cost after one minute (top) and after two minutes (bottom), broken down by whether random ordering (left hand side) or more tailored ‘smarter’ ordering (right hand side) of employees is used at each iteration.

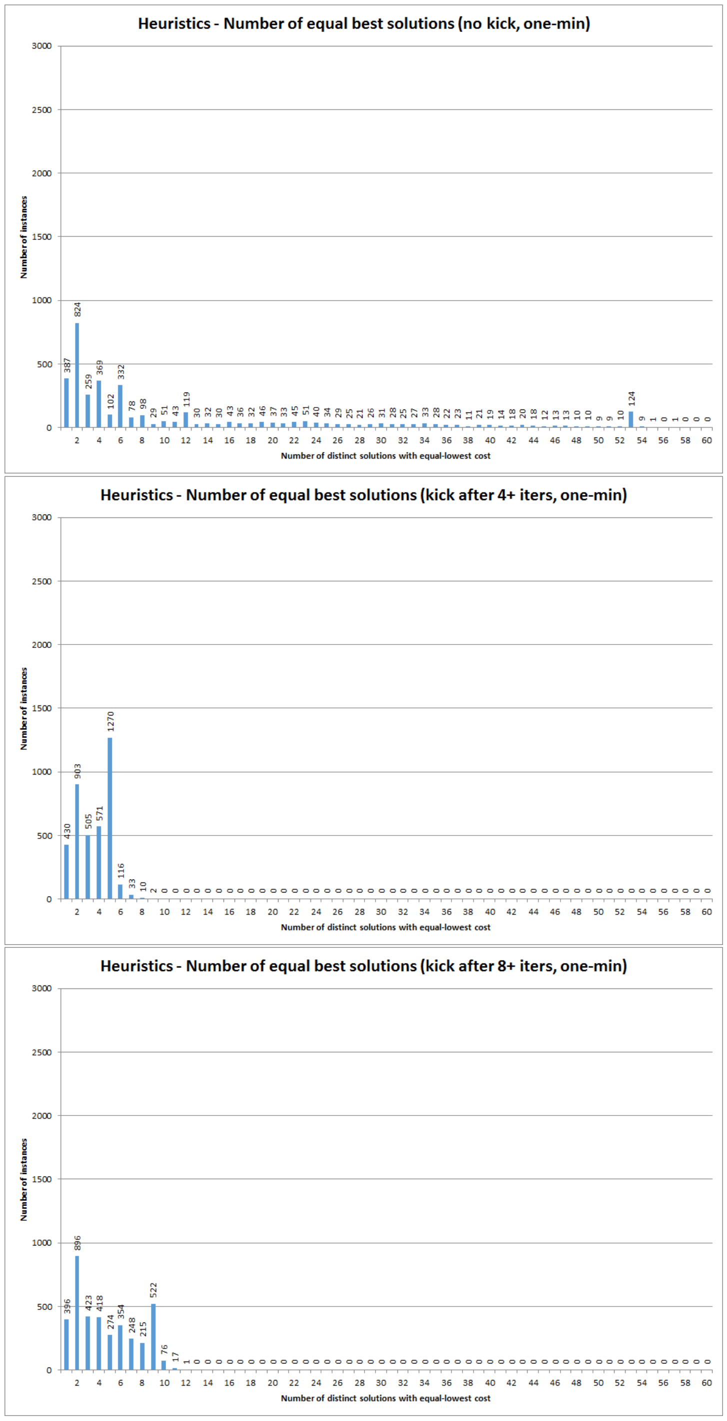


Figure D.29: Number of solutions found with equal-best cost after one minute, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

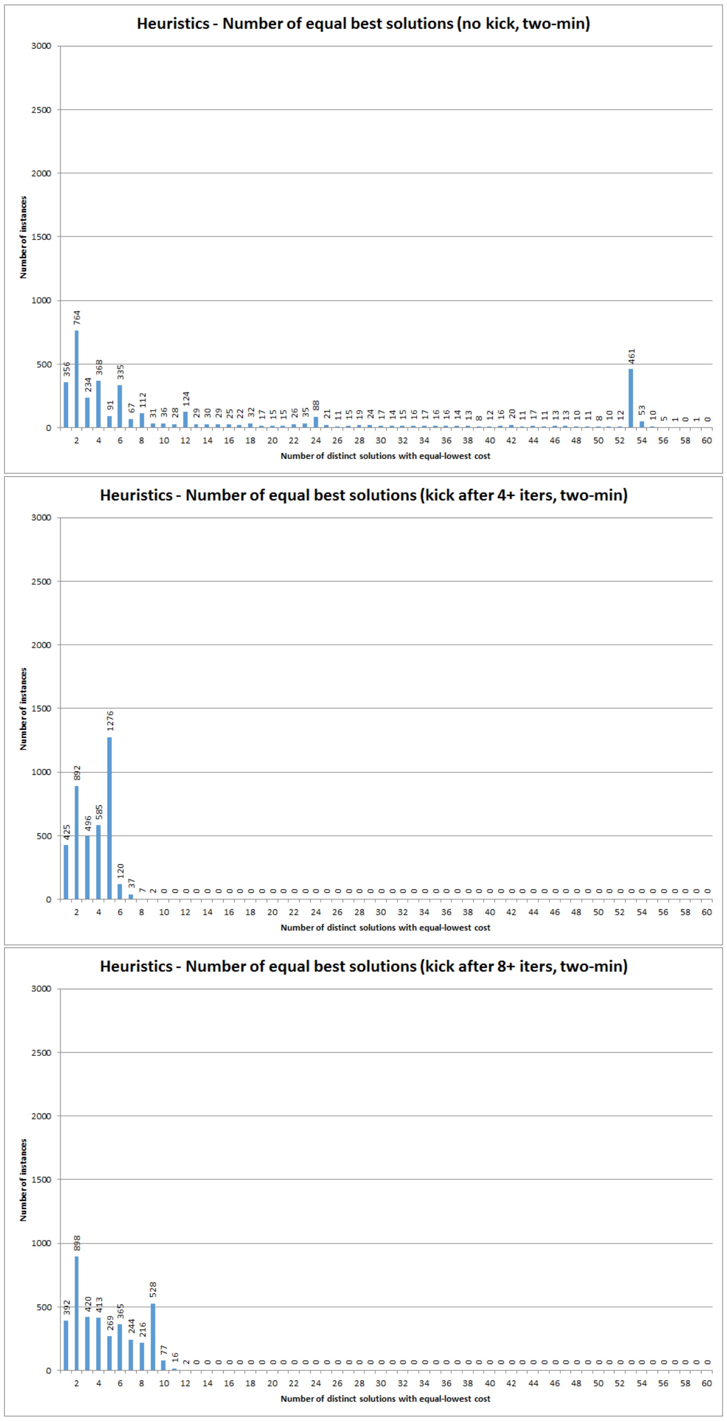


Figure D.30: Number of solutions found with equal-best cost after two minutes, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

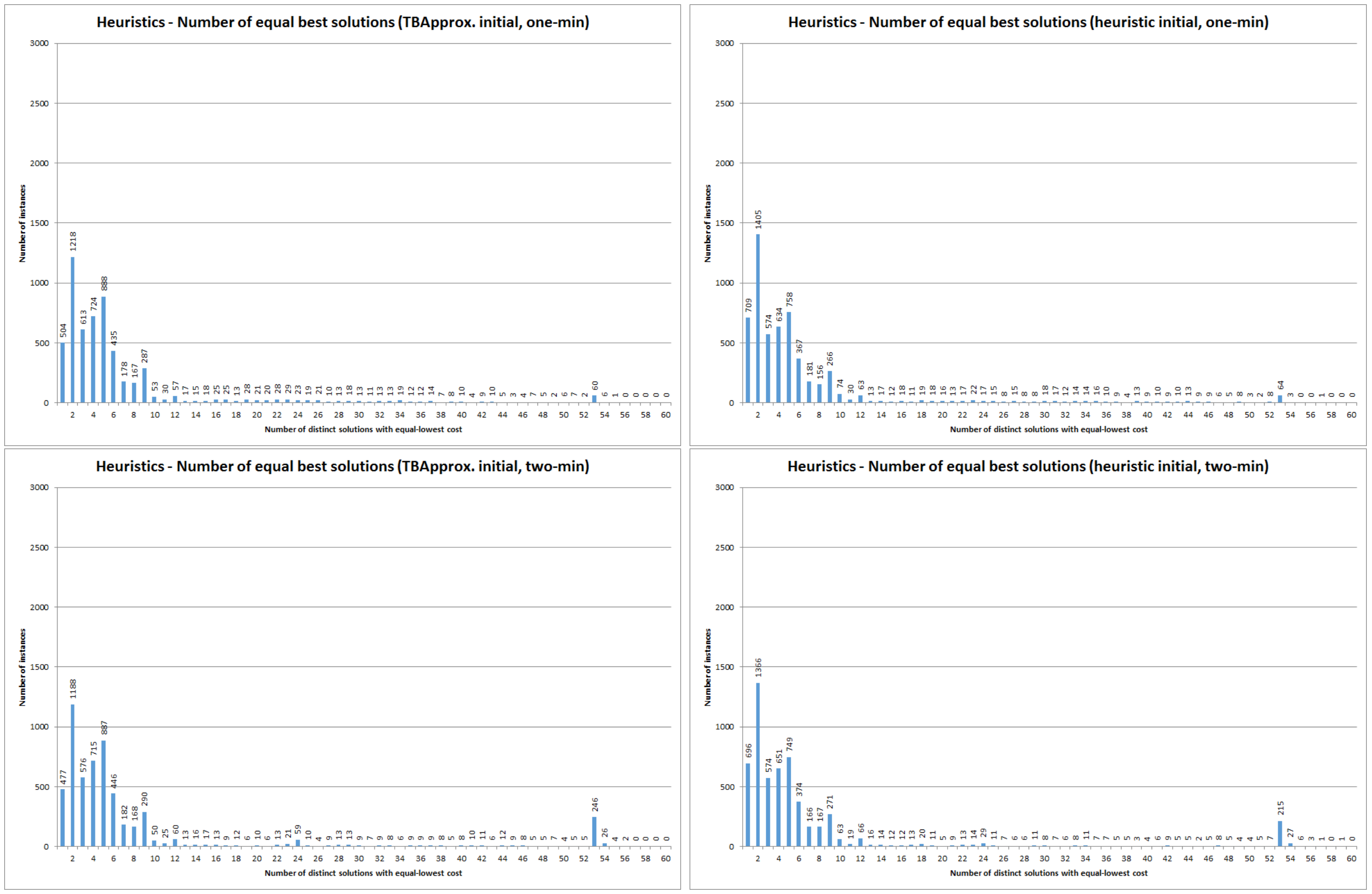


Figure D.31: Number of solutions found with equal-best cost after one minute (top) and after two minutes (bottom), broken down by whether the Task-Based Approximation (left hand side) or Heuristic (right hand side) initial solution was used.

D.2.6 Number of iterations

As shown in Table 6.7 in section 6.8.4.6, four of the heuristic settings had a significant influence on the number of iterations carried out by the algorithm. Here we show the following graphs in relation to this:

- A breakdown according to the number of employees to examine at each iteration, for both the one-minute and two-minute results (shown together in Figure D.32);
- A breakdown according to the order in which employees are examined at each iteration, for both the one-minute and two-minute results (shown together in Figure D.33);
- A breakdown according to the solution acceptance criteria used, for both the one-minute and two-minute results (shown together in Figure D.34);
- A breakdown of the two-minute results according to the the kick activation rules used (Figure D.35).

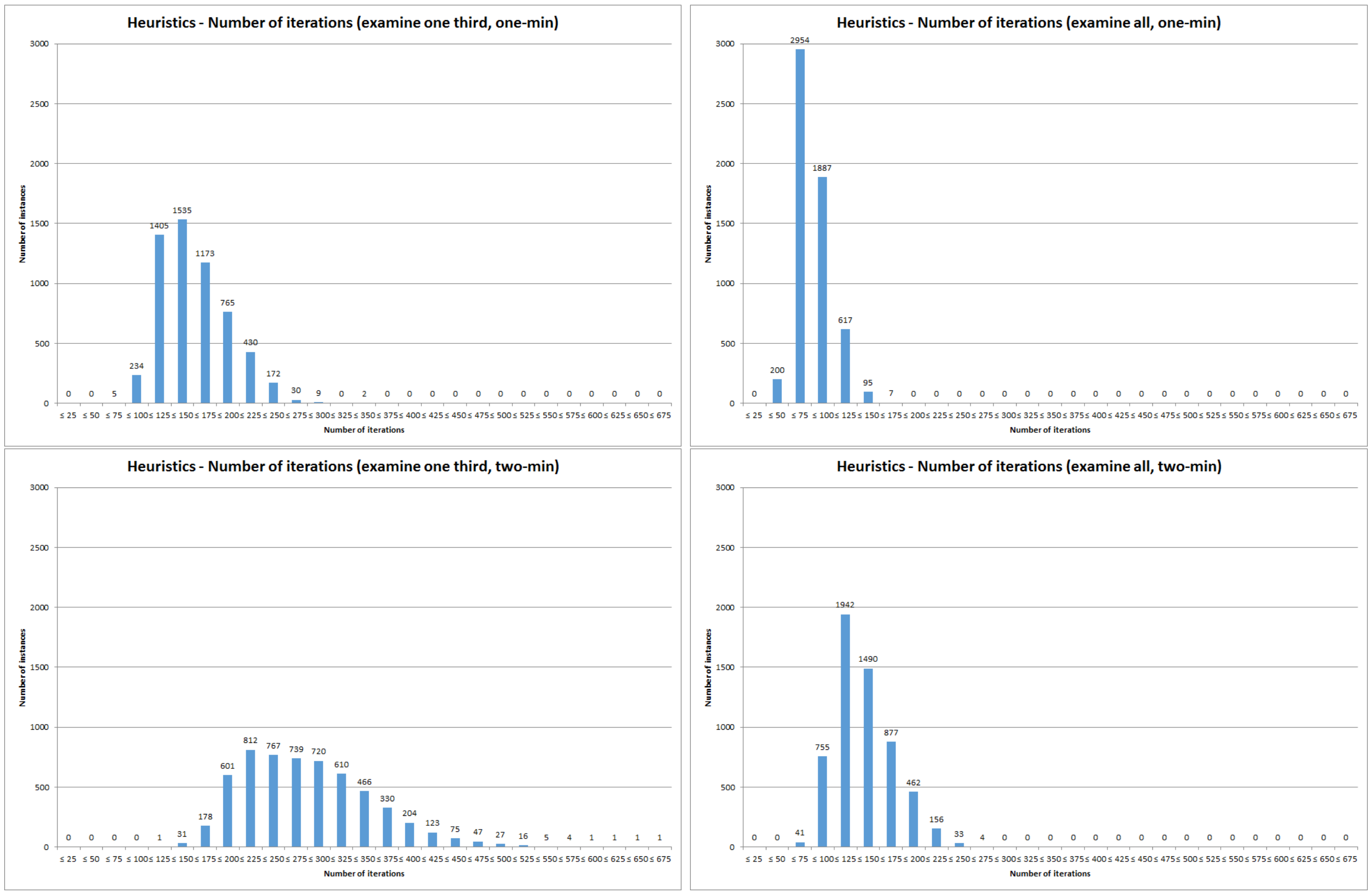


Figure D.32: Number of iterations carried out by the algorithm within one minute (top) and within two minutes (bottom), broken down by whether one third of employees are examined at each iteration (left hand side) or all employees are examined (right hand side).

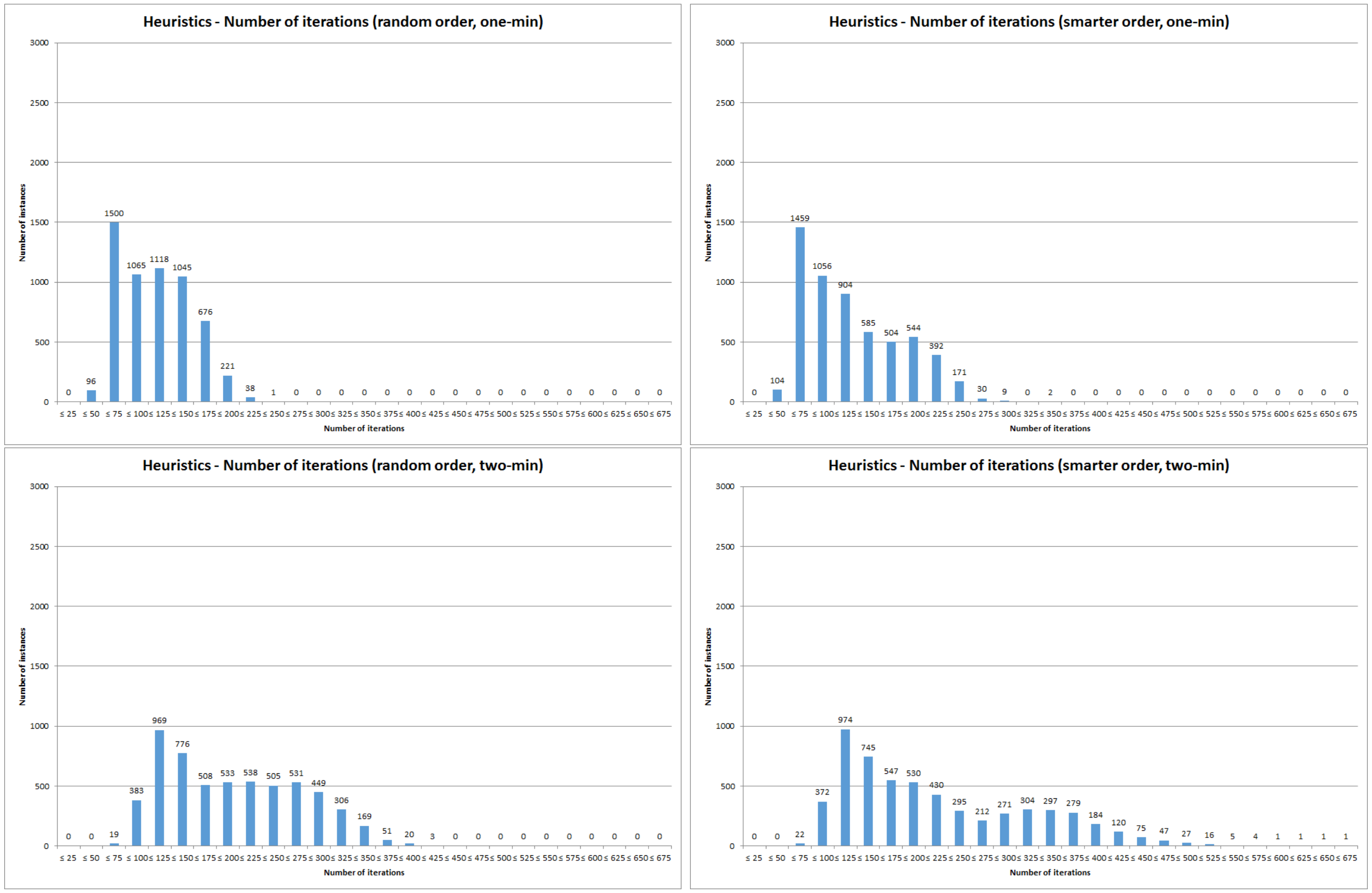


Figure D.33: Number of iterations carried out by the algorithm within one minute (top) and within two minutes (bottom), broken down by whether random ordering (left hand side) or more tailored ‘smarter’ ordering (right hand side) of employees is used at each iteration.

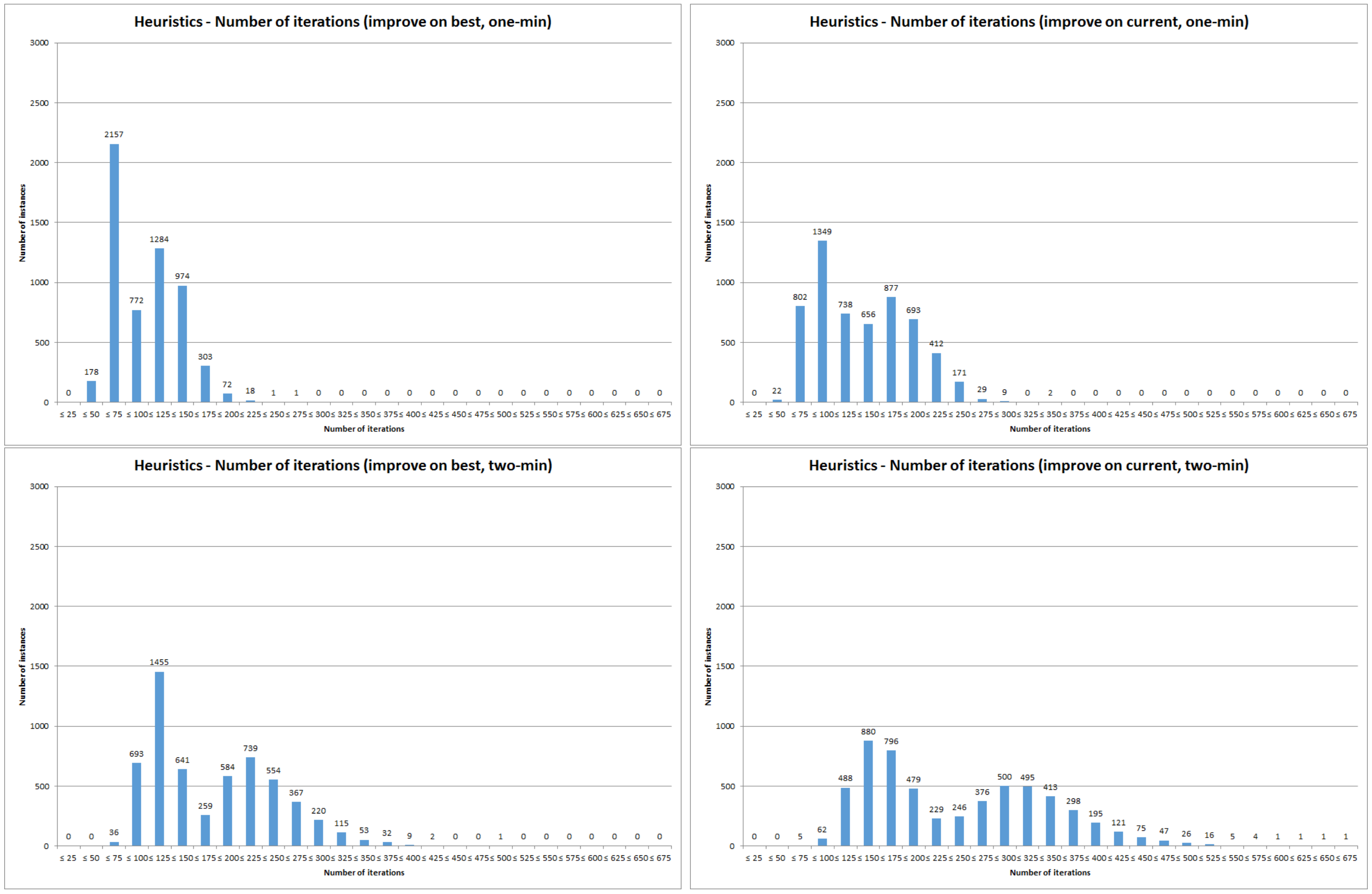


Figure D.34: Number of iterations carried out by the algorithm within one minute (top) and within two minutes (bottom), broken down by whether algorithm accepts first non-tabu solution which improves on the best solution so far (left hand side) or on the current solution (right hand side).

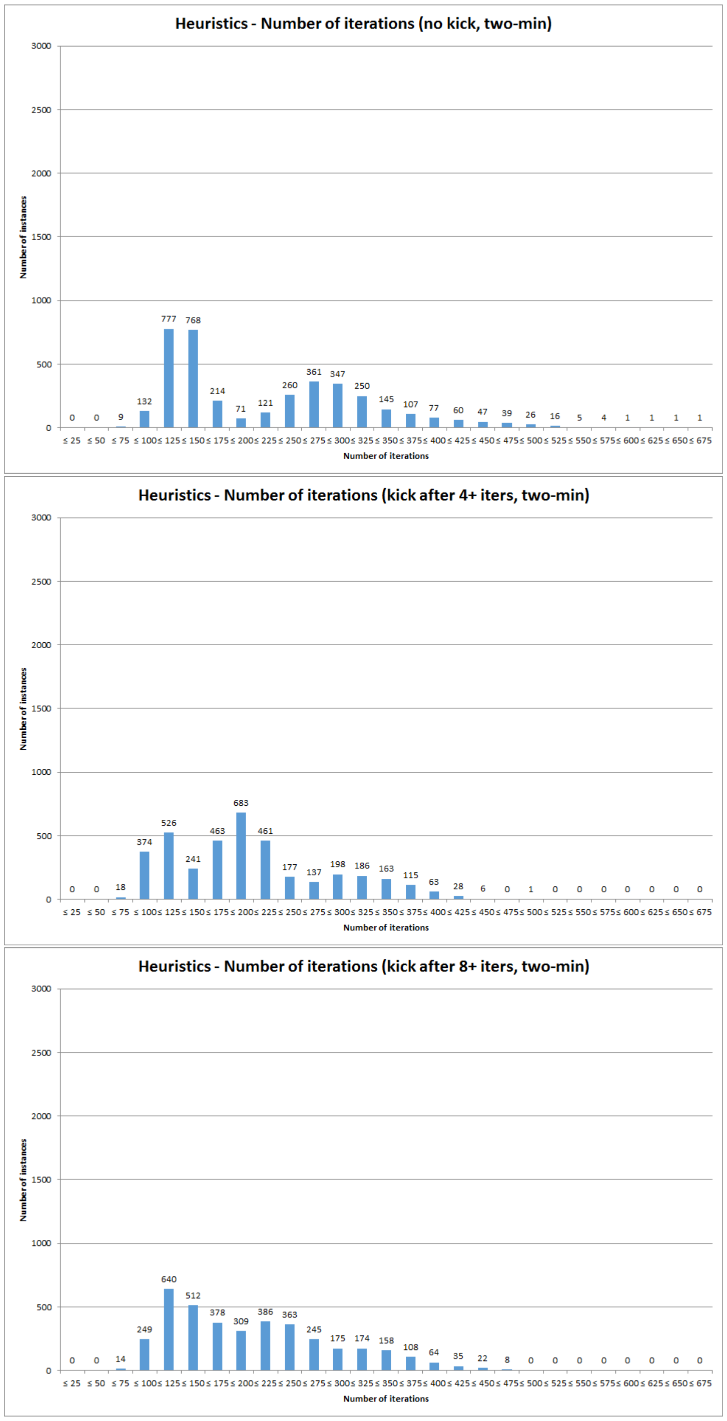


Figure D.35: Number of iterations carried out by the algorithm within two minutes, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

D.2.7 Time of best solution

As shown in Table 6.8 in section 6.8.4.7, all five of the heuristic settings had a significant influence on the estimated time at which the algorithm found the best solution. Here we show the following graphs in relation to this:

- A breakdown of the two-minute results according to the number of employees to examine at each iteration (Figure D.36);
- A breakdown of the one-minute results according to the order in which employees are examined at each iteration (Figure D.37);
- A breakdown according to the solution acceptance criteria used, for both the one-minute and two-minute results (shown together in Figure D.38);
- A breakdown according to the kick activation rules used, for both the one-minute results (Figure D.39) and the two-minute results (Figure D.40); and
- A breakdown according to the type of initial solution used, for both the one-minute and two-minute results (shown together in Figure D.41).

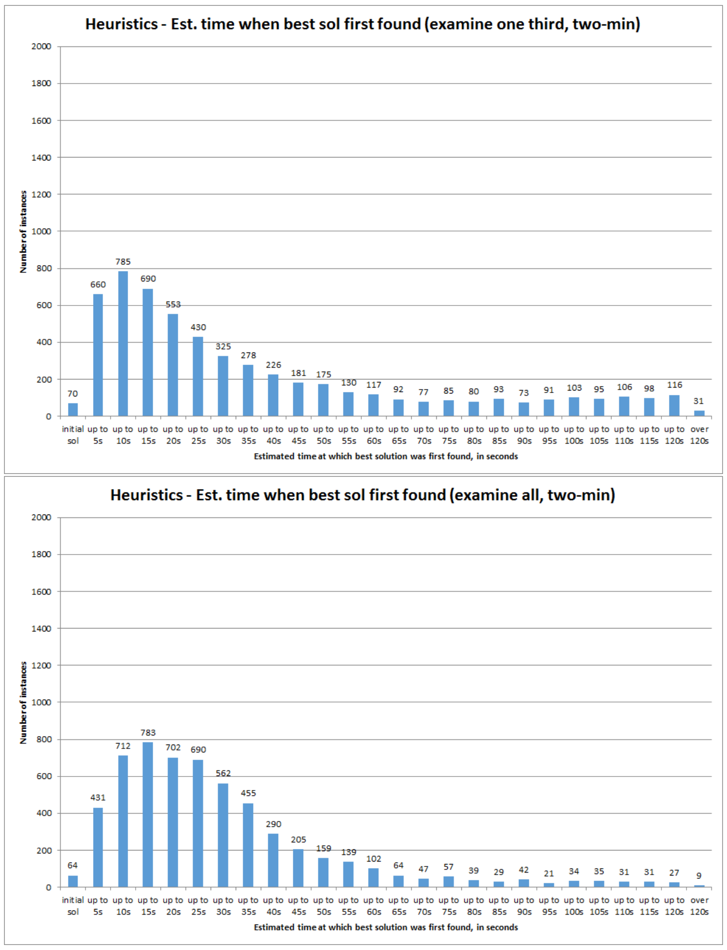


Figure D.36: Estimated time at which the best solution within two minutes was first found, broken down by whether one third of employees are examined at each iteration (top) or all employees are examined (bottom).

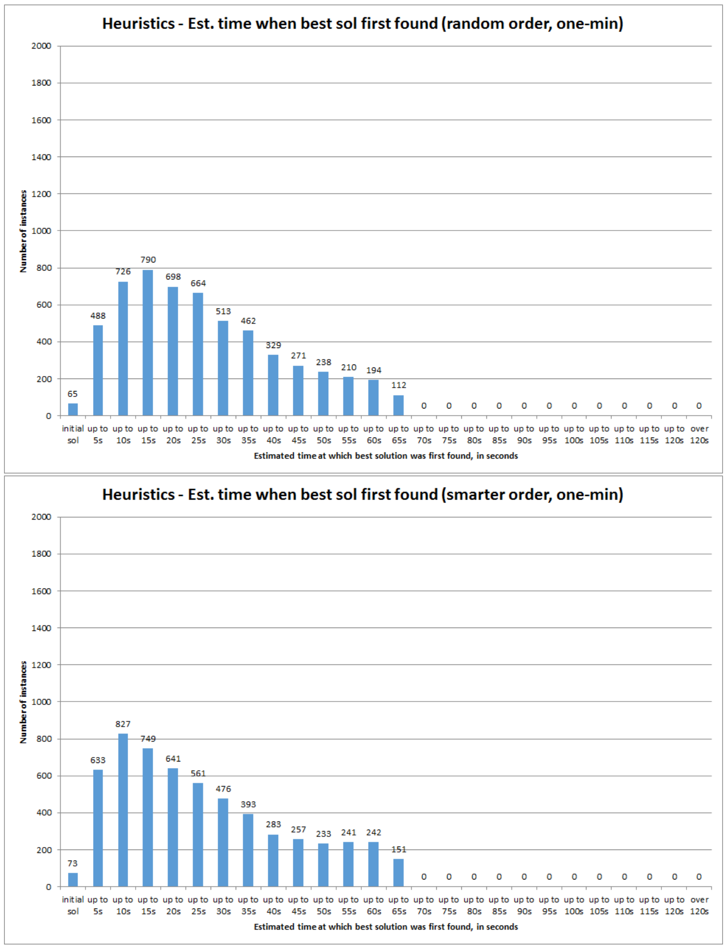


Figure D.37: Estimated time at which the best solution within one minute was first found, broken down by whether random ordering (top) or more tailored ‘smarter’ ordering (bottom) of employees is used at each iteration.

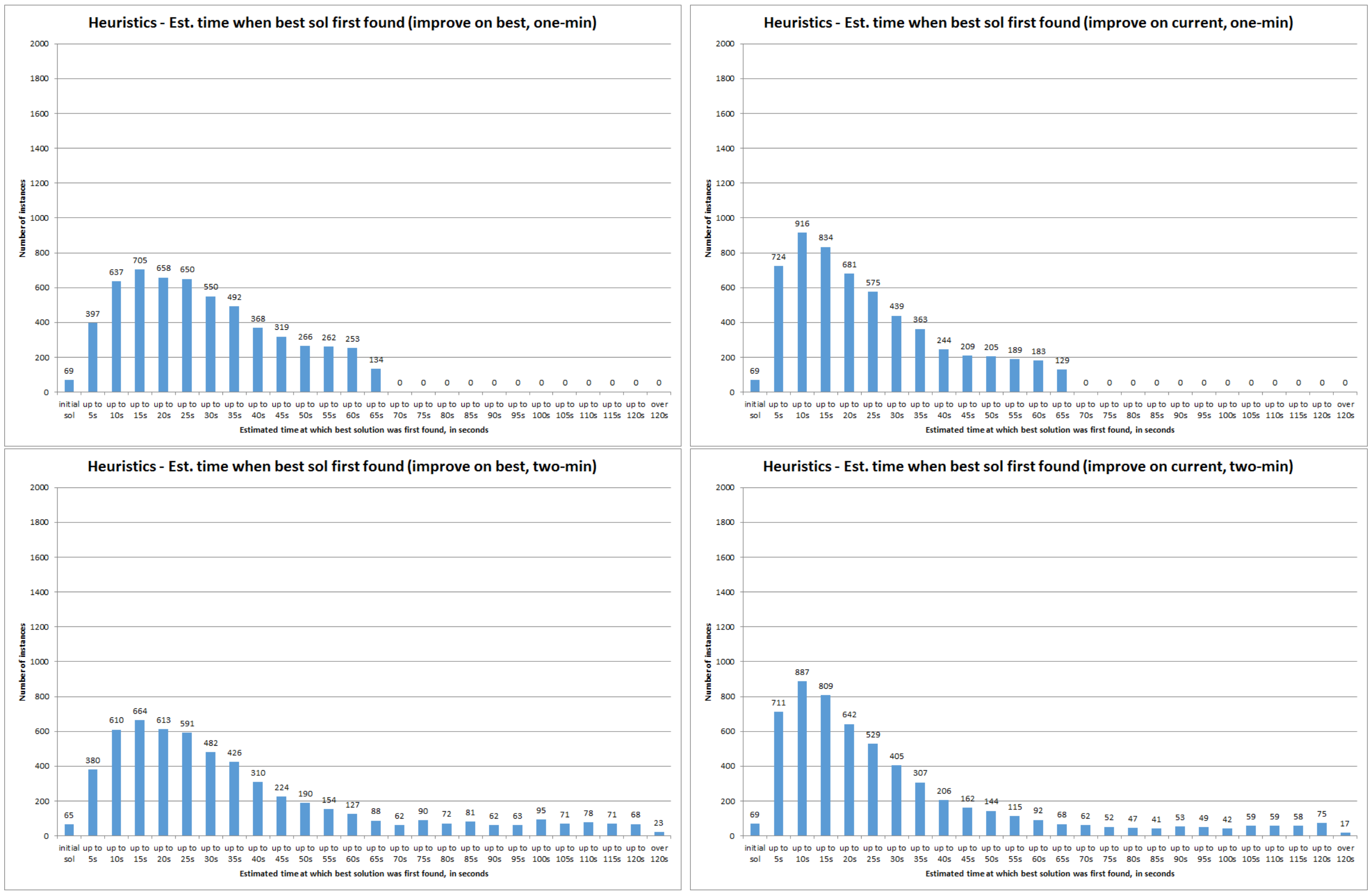


Figure D.38: Estimated time at which the best solution within one minute (top) and within two minutes (bottom) was first found, broken down by whether algorithm accepts first non-tabu solution which improves on the best solution so far (left hand side) or on the current solution (right hand side).

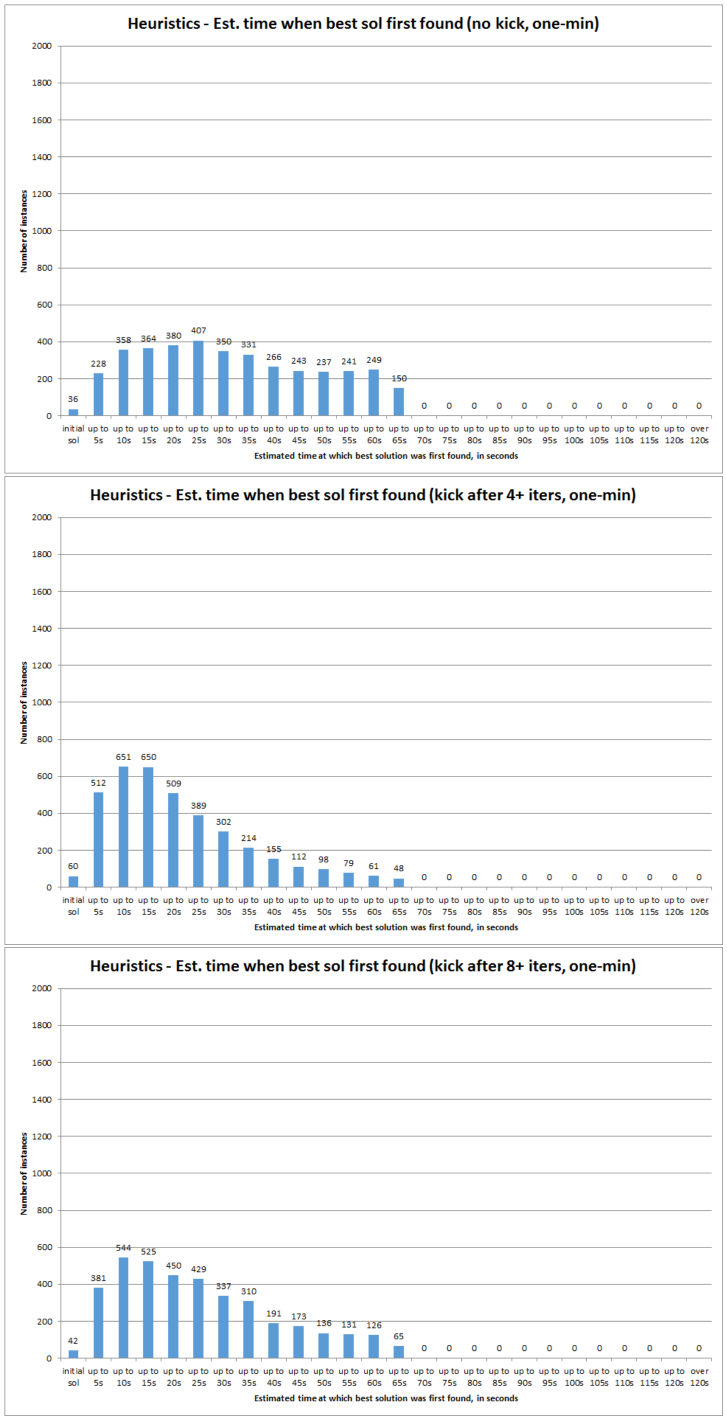


Figure D.39: Estimated time at which the best solution within one minute was first found, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

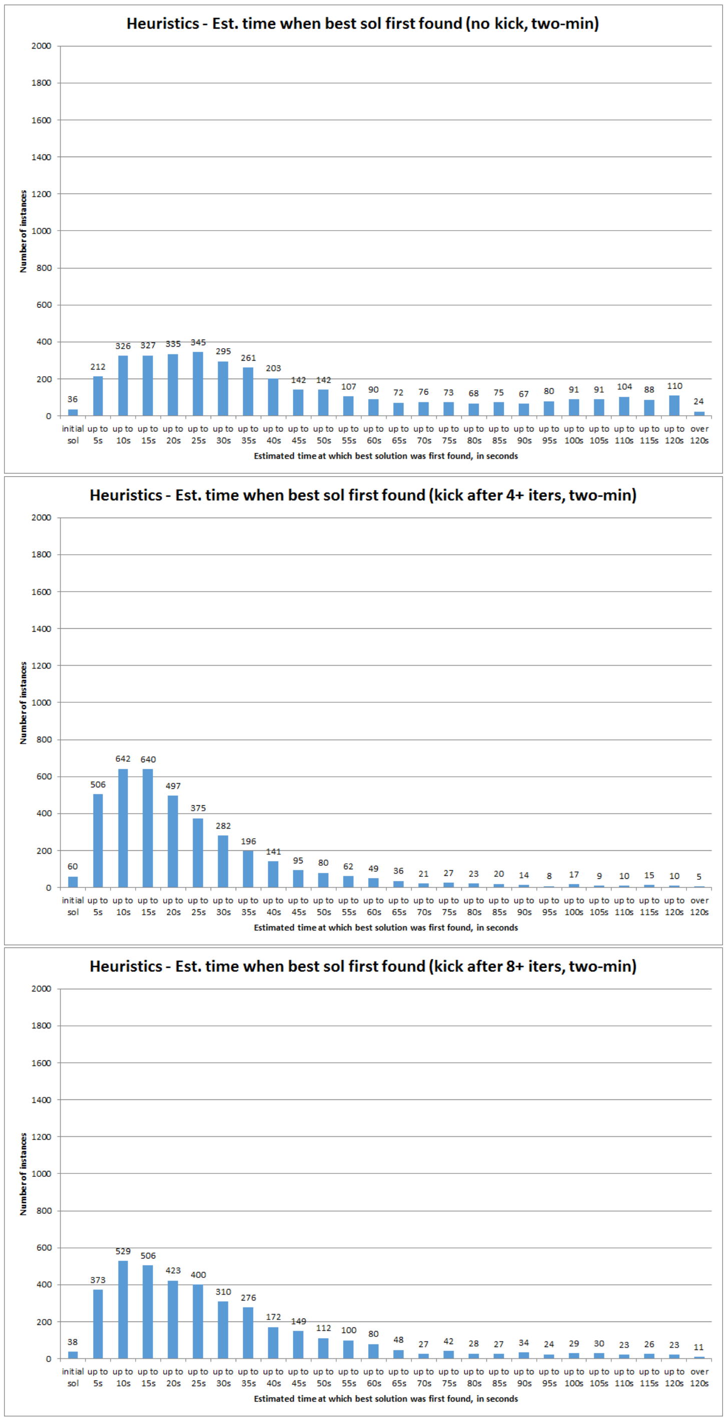
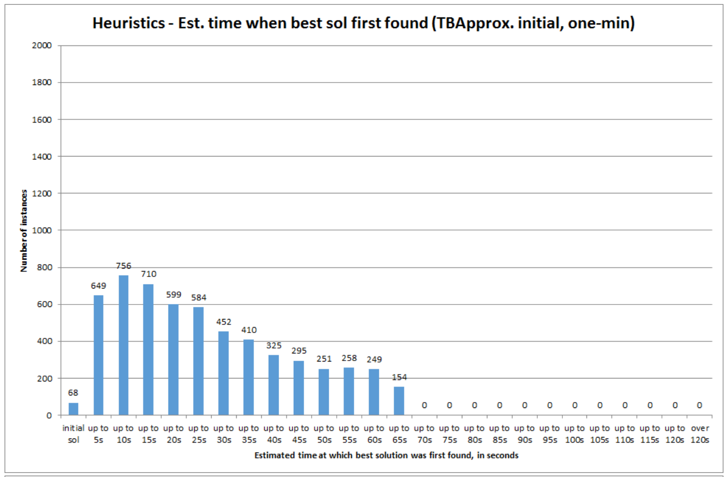
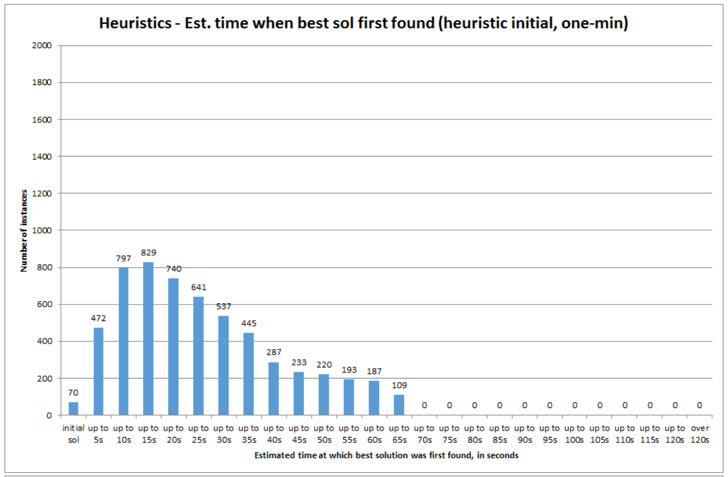
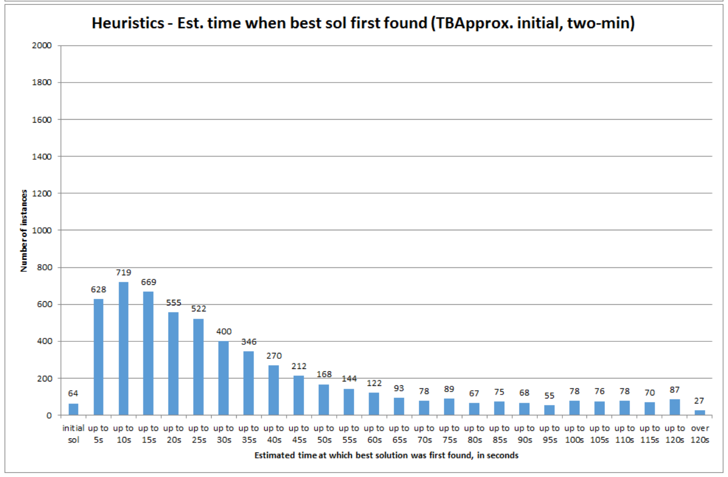


Figure D.40: Estimated time at which the best solution within two minutes was first found, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).







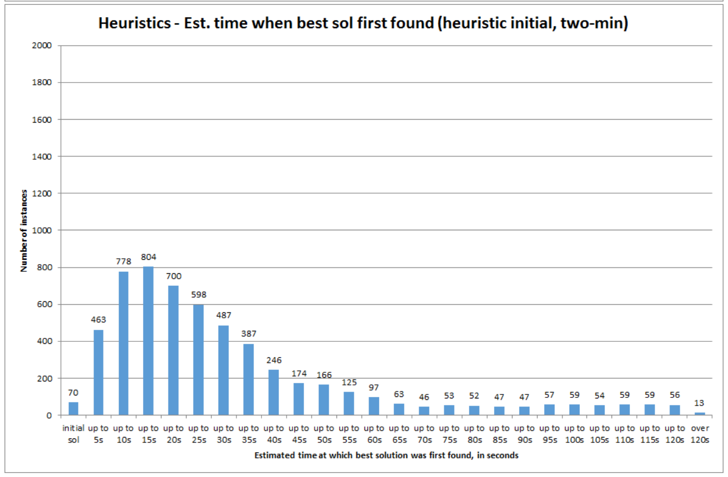


Figure D.41: Estimated time at which the best solution within one minute (top) and within two minutes (bottom) was first found, broken down by whether the Task-Based Approximation (left hand side) or Heuristic (right hand side) initial solution was used.

D.2.8 Effect of cutting off algorithm early

As shown in Table 6.9 in section 6.8.4.8, all five of the heuristic settings had a significant influence on the gap between the estimated early cut-off solutions and the final (i.e. two-minute) solution. Here we show the following graphs in relation to this:

- A breakdown according to the number of employees to examine at each iteration, for the one-minute and estimated 30-second cut-off solutions (shown together in Figure D.42) and for the estimated 20-second and estimated 10-second cut-off solutions (shown together in Figure D.43);

- A breakdown of the one-minute cut-off solutions according to the order in which the employees are examined at each iteration (Figure D.44);

- A breakdown according to the solution acceptance criteria used, for the one-minute and estimated 30-second cut-off solutions (shown together in Figure D.45) and for the estimated 20-second and estimated 10-second cut-off solutions (shown together in Figure D.46);

- A breakdown according to the kick activation rules used, for the one-minute cut-off solutions (Figure D.47), the estimated 30-second cut-off solutions (Figure D.48), the estimated 20-second cut-off solutions (Figure D.49) and the estimated 10-second cut-off solution (Figure D.50); and

- A breakdown according to the type of initial solution used, for the one-minute and estimated 30-second cut-off solutions (shown together in Figure D.51) and for the estimated 10-second cut-off solutions (shown together in Figure D.52).

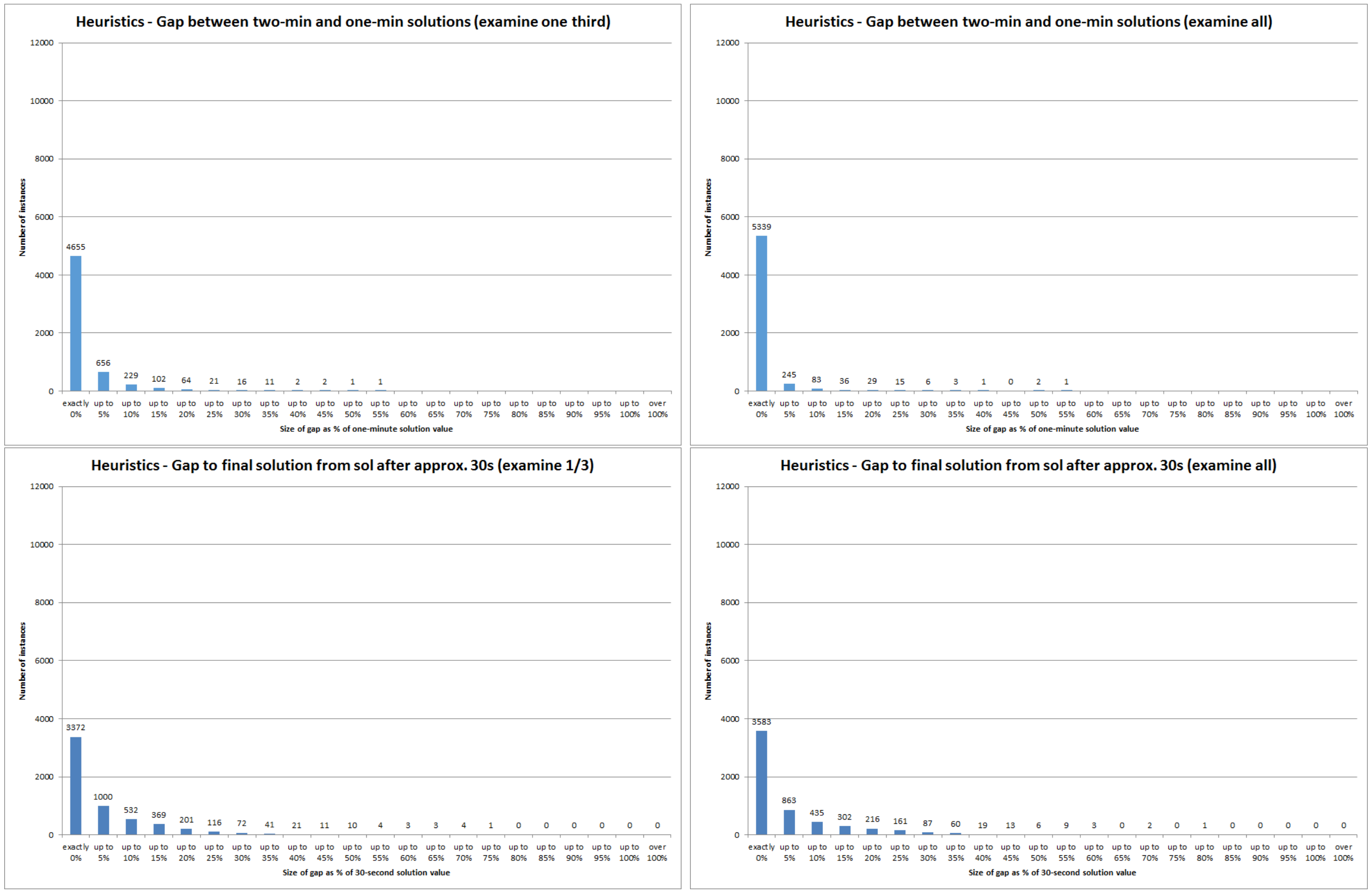


Figure D.42: Gap to two-minute solution value from one-minute solution (top) and from estimated 30-second solution (bottom), broken down by whether one third of employees are examined at each iteration (left hand side) or all employees are examined (right hand side).

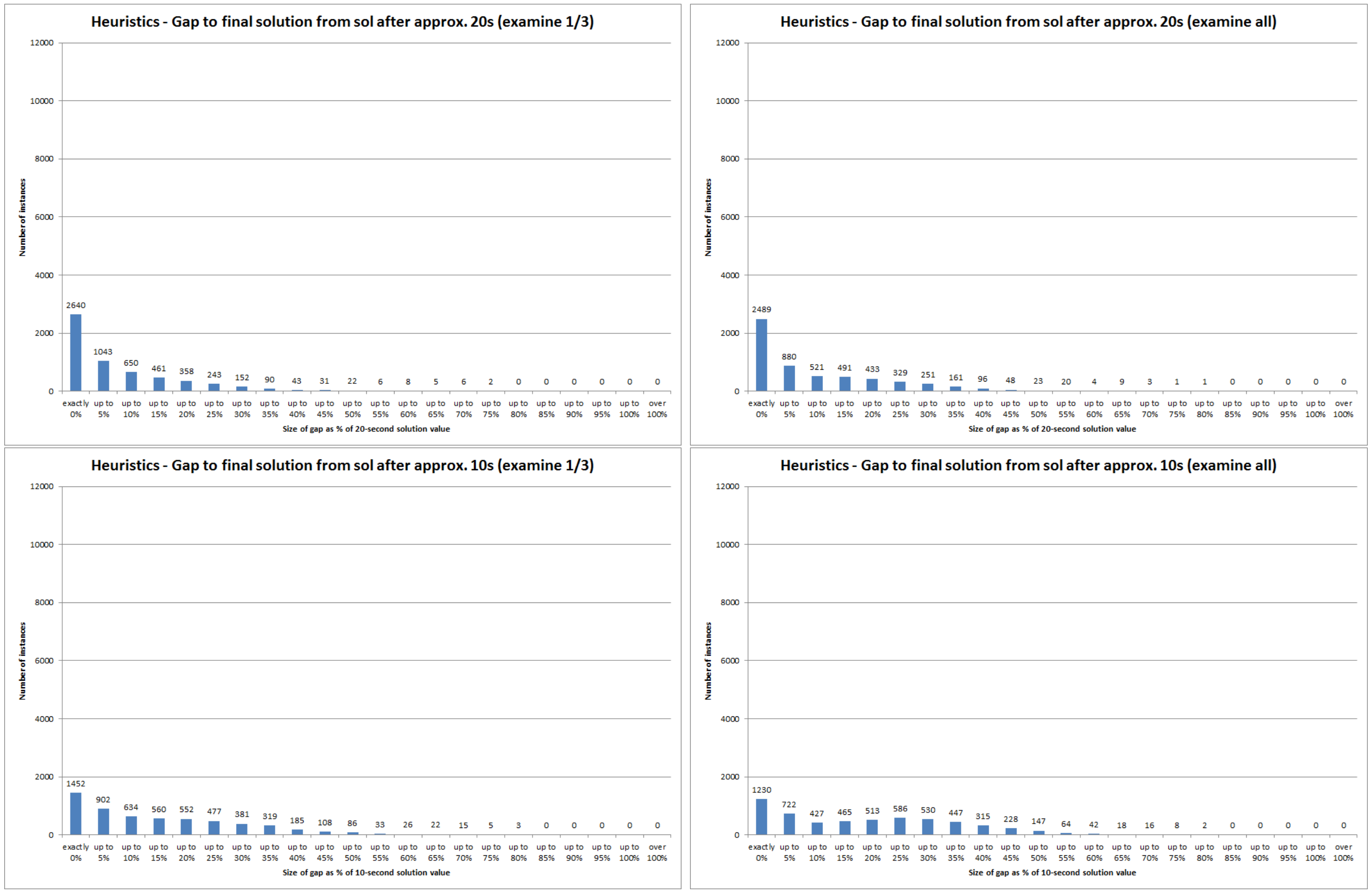


Figure D.43: Gap to two-minute solution value from estimated 20-second solution (top) and from estimated 10-second solution (bottom), broken down by whether one third of employees are examined at each iteration (left hand side) or all employees are examined (right hand side).

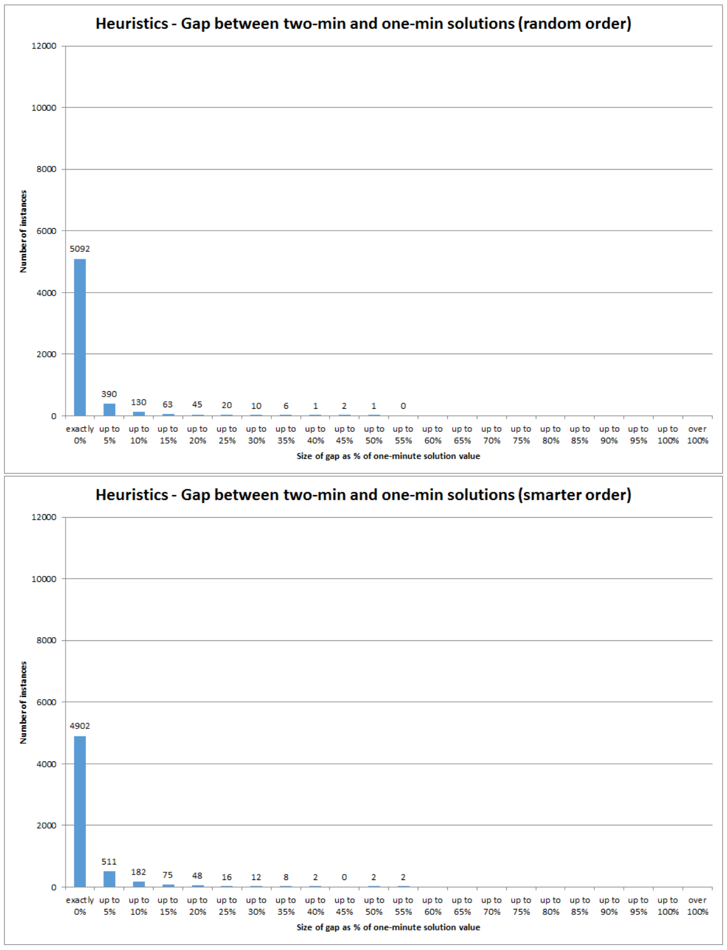


Figure D.44: Gap to two-minute solution value from one-minute solution, broken down by whether random ordering (top) or more tailored ‘smarter’ ordering (bottom) of employees is used at each iteration.

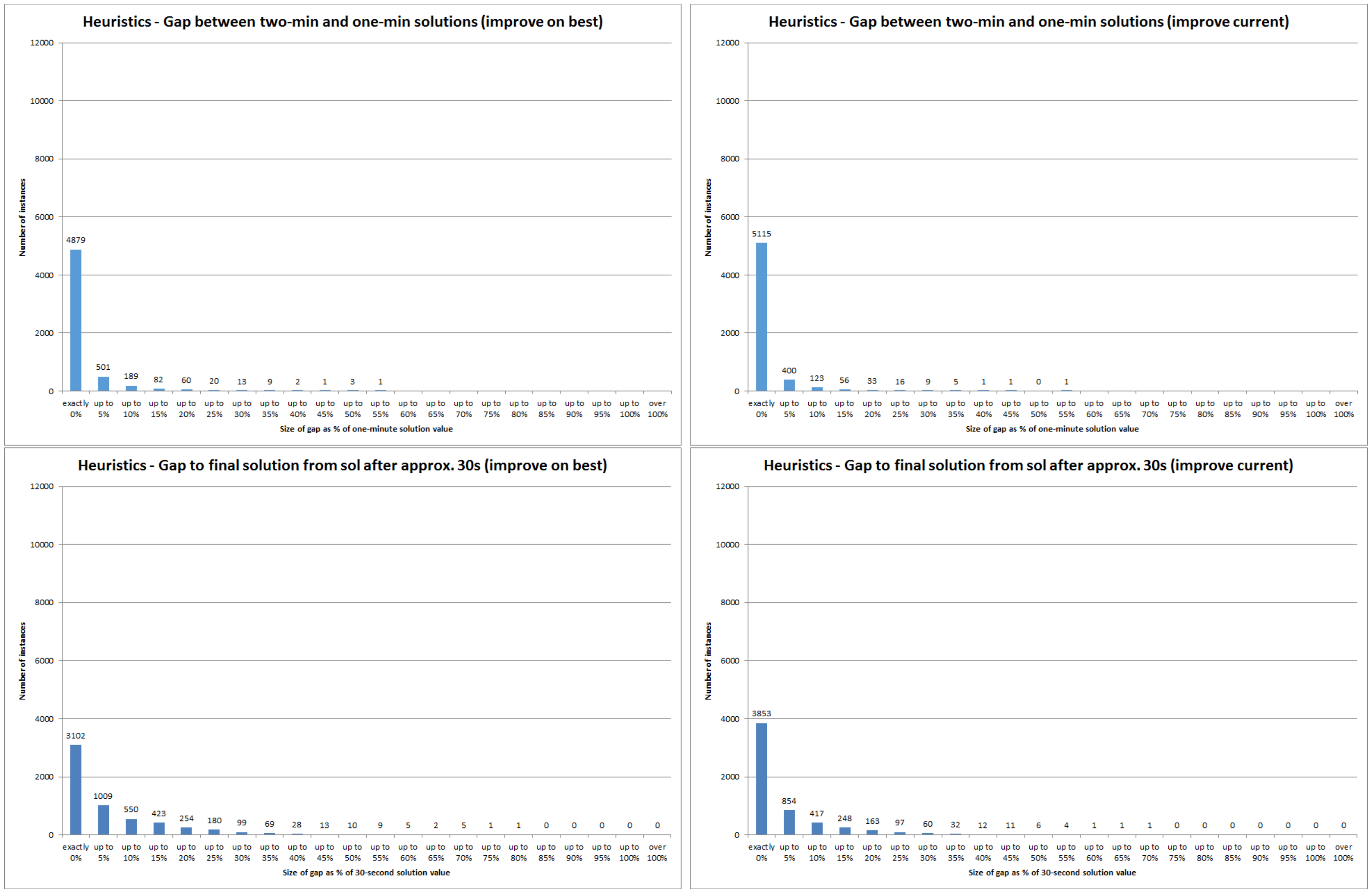


Figure D.45: Gap to two-minute solution value from one-minute solution (top) and from estimated 30-second solution (bottom), broken down by whether algorithm accepts first non-tabu solution which improves on the best solution so far (left hand side) or on the current solution (right hand side).

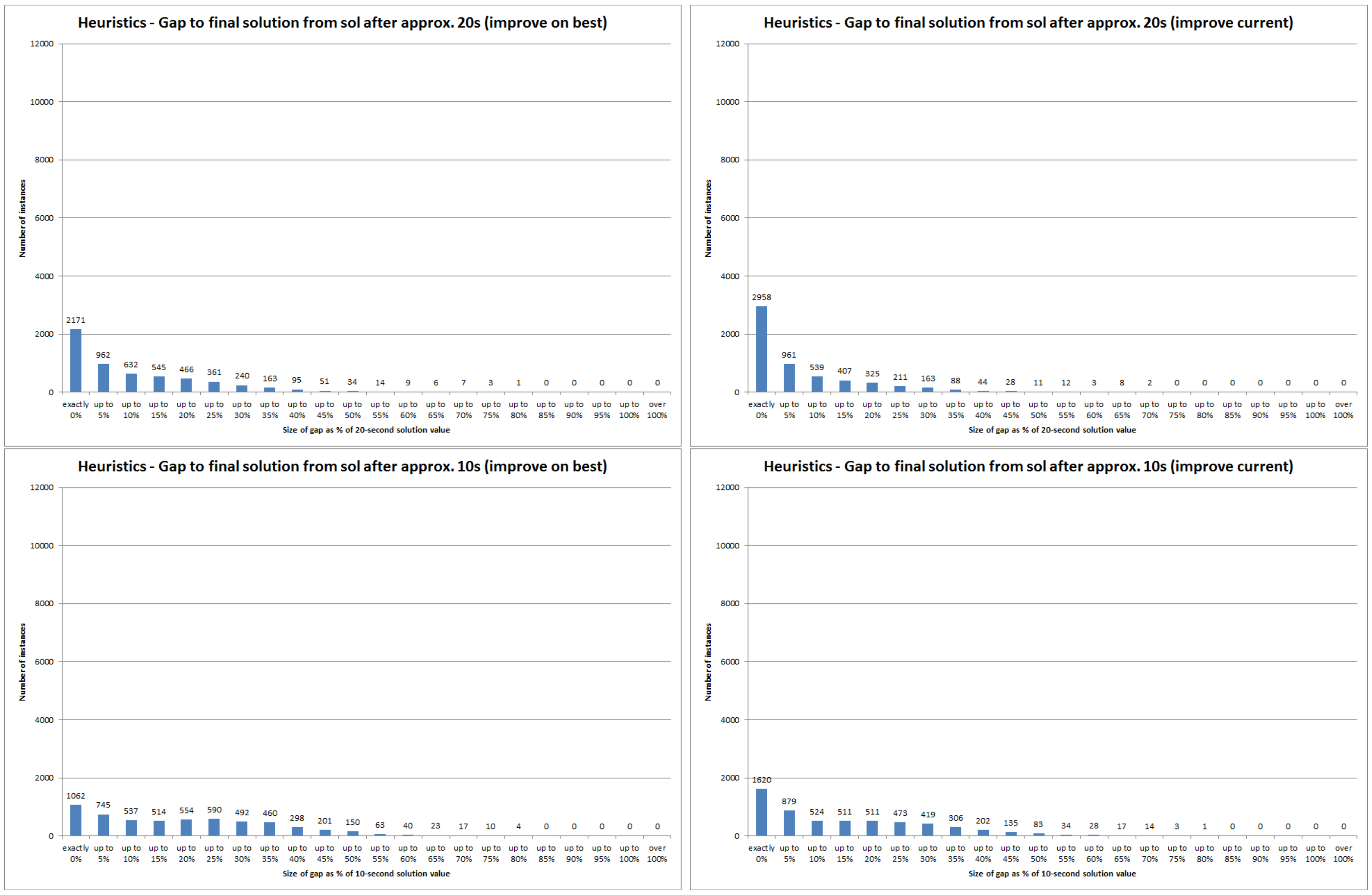


Figure D.46: Gap to two-minute solution value from estimated 20-second solution (top) and from estimated 10-second solution (bottom), broken down by whether algorithm accepts first non-tabu solution which improves on the best solution so far (left hand side) or on the current solution (right hand side).

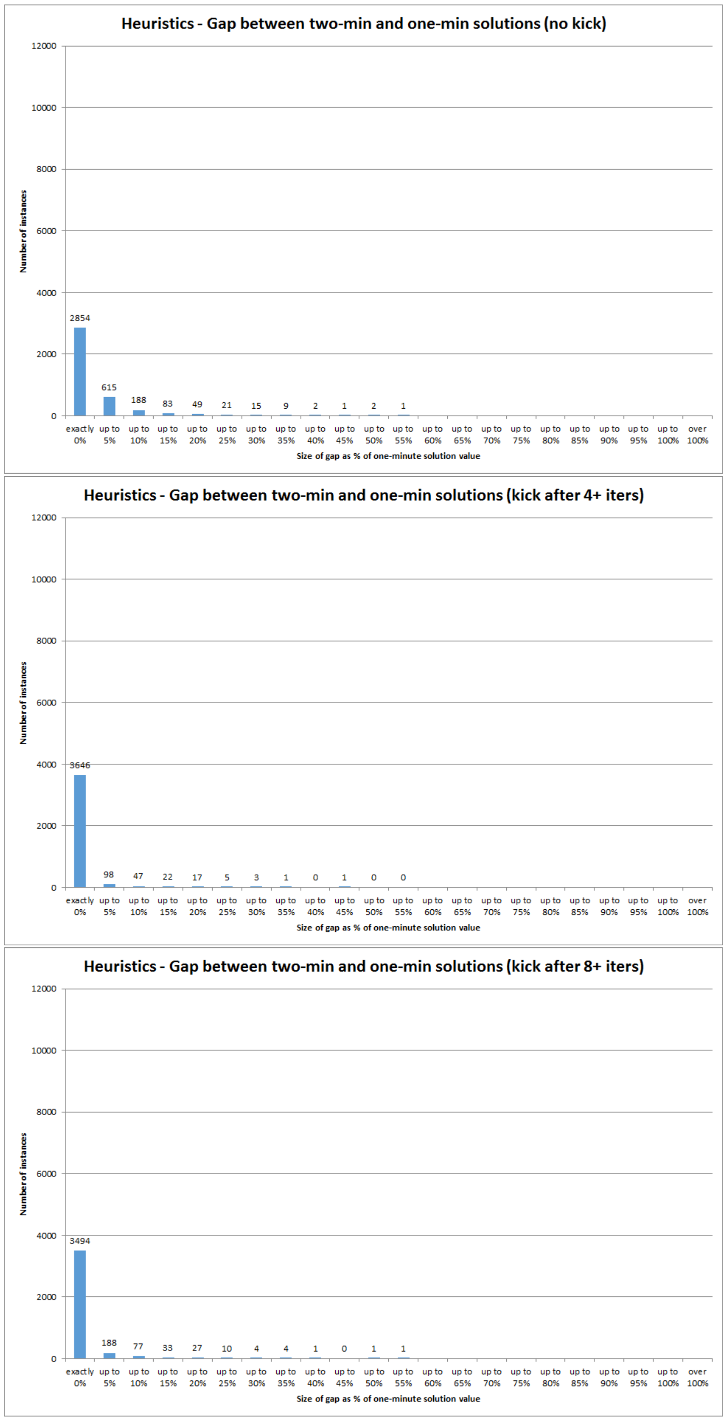


Figure D.47: Gap to two-minute solution value from one-minute solution, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

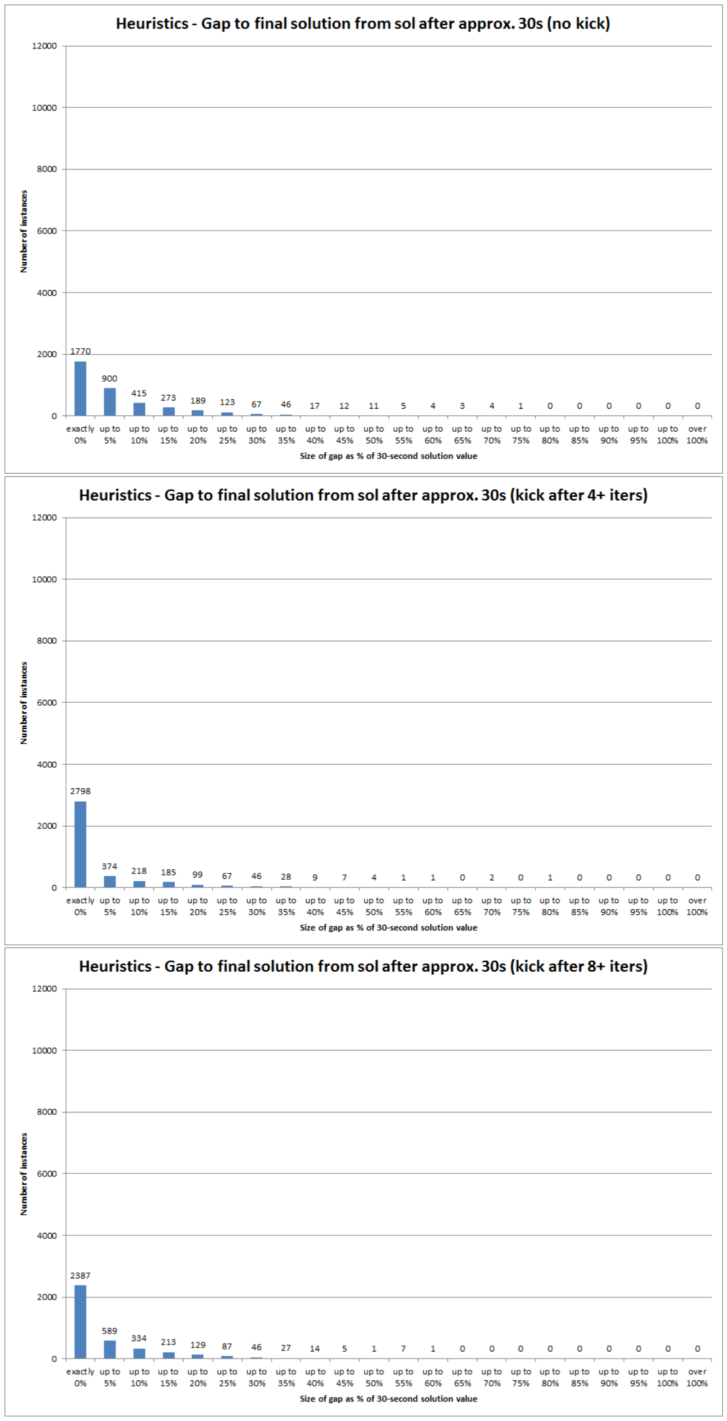


Figure D.48: Gap to two-minute solution value from estimated 30-second solution, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

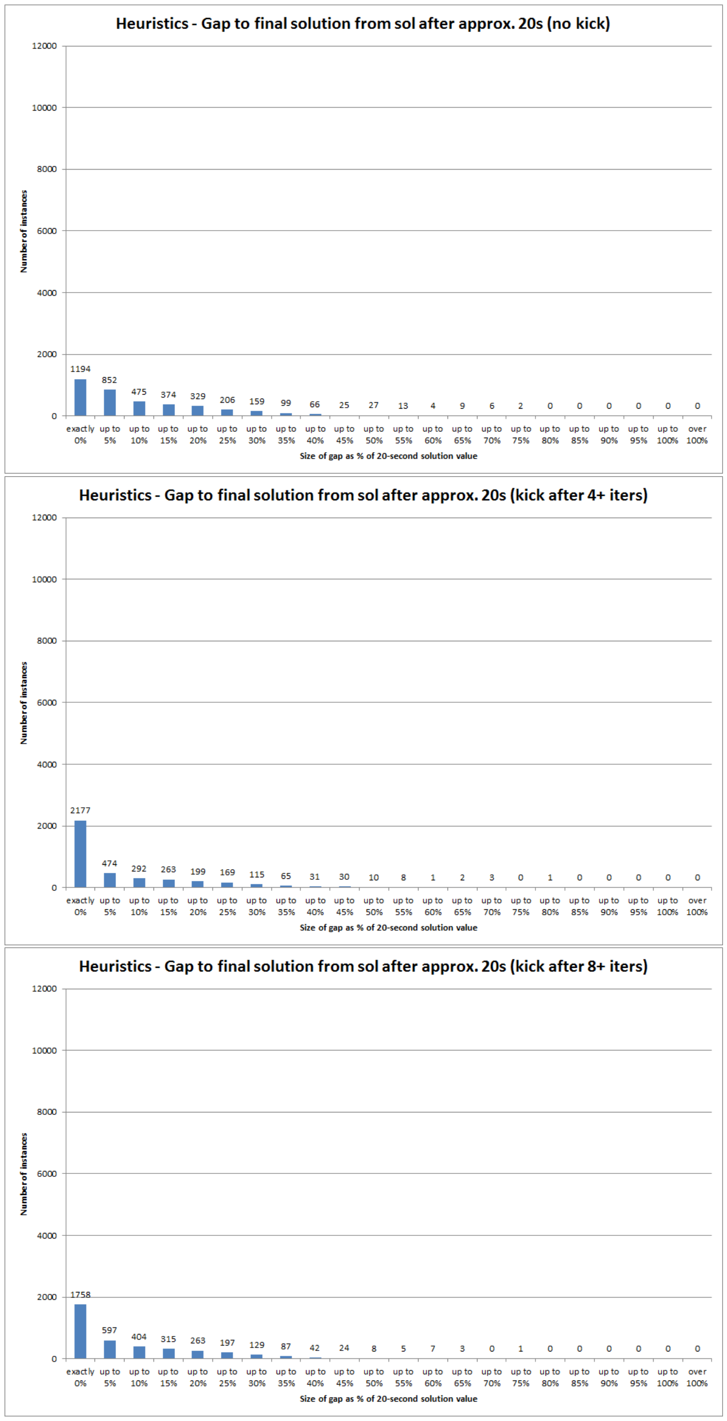


Figure D.49: Gap to two-minute solution value from estimated 20-second solution, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).

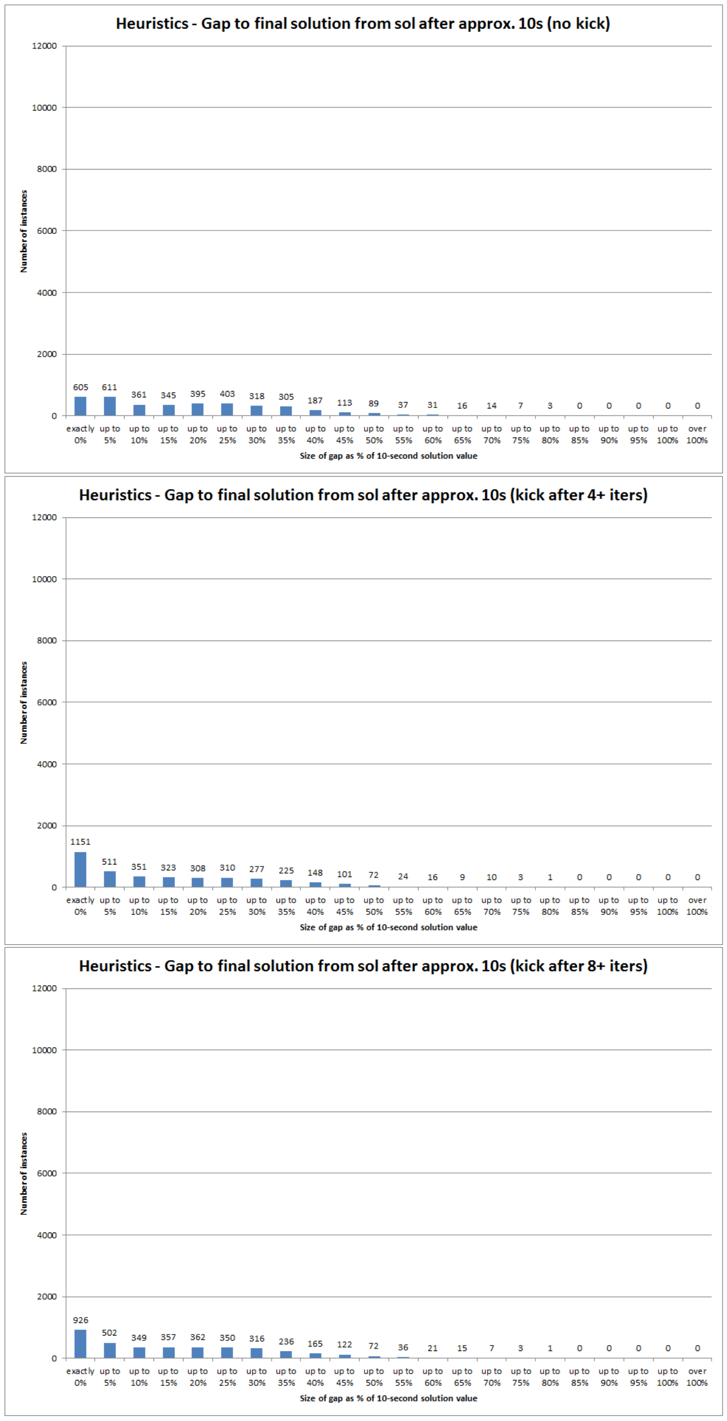
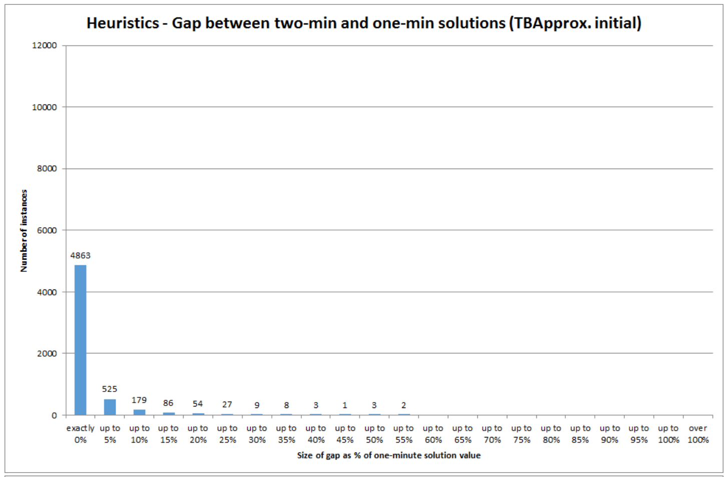
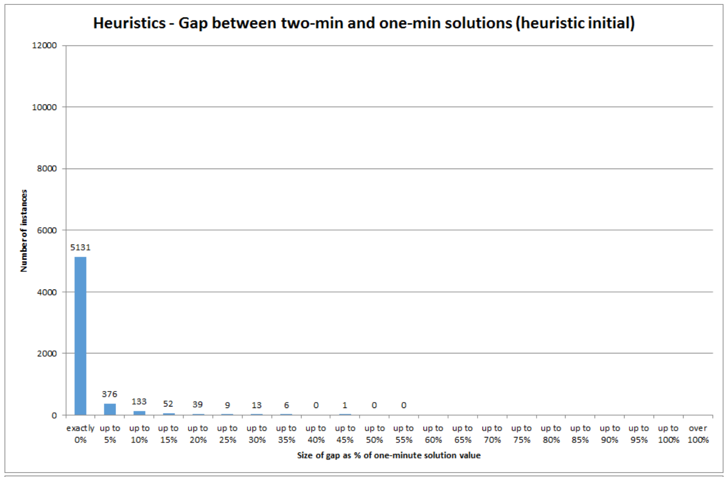
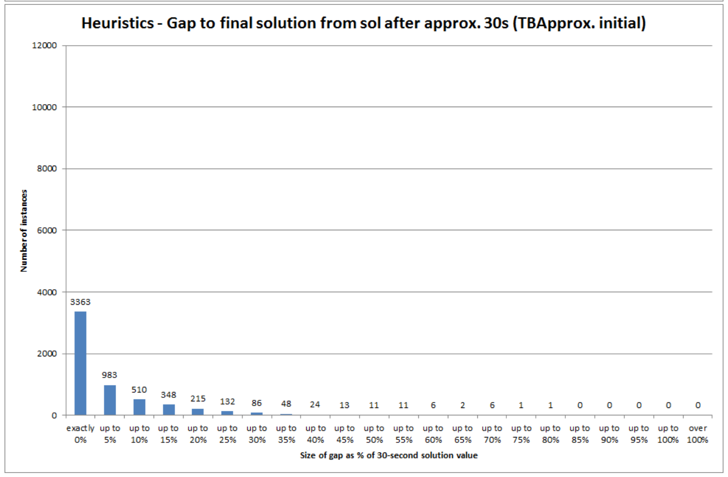


Figure D.50: Gap to two-minute solution value from estimated 10-second solution, broken down by whether no kick was used (top), the kick was activated if no improvement was found after 4 or more iterations (middle), or activated if no improvement was found after 8 or more iterations (bottom).







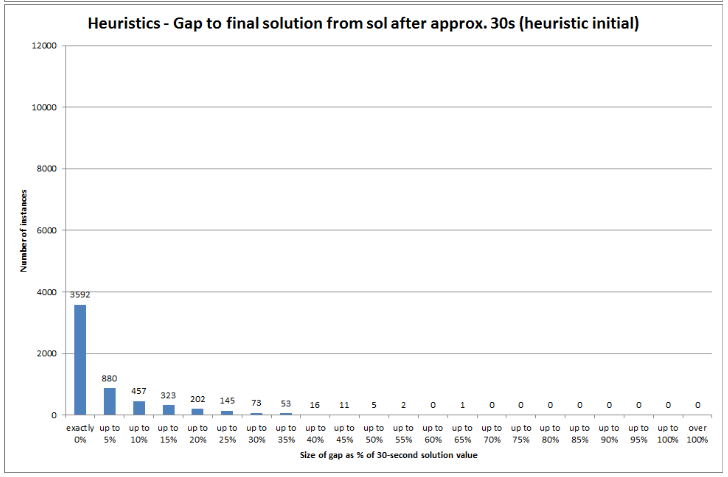
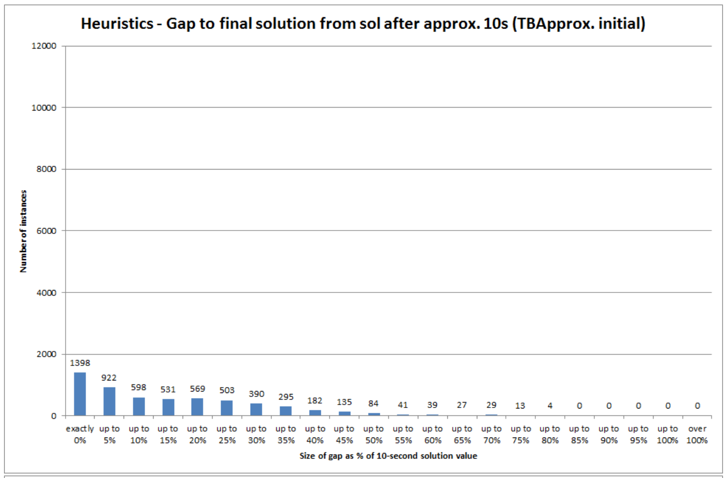


Figure D.51: Gap to two-minute solution value from one-minute solution (top) and from estimated 30-second solution (bottom), broken down by whether the Task-Based Approximation (left hand side) or Heuristic (right hand side) initial solution was used.



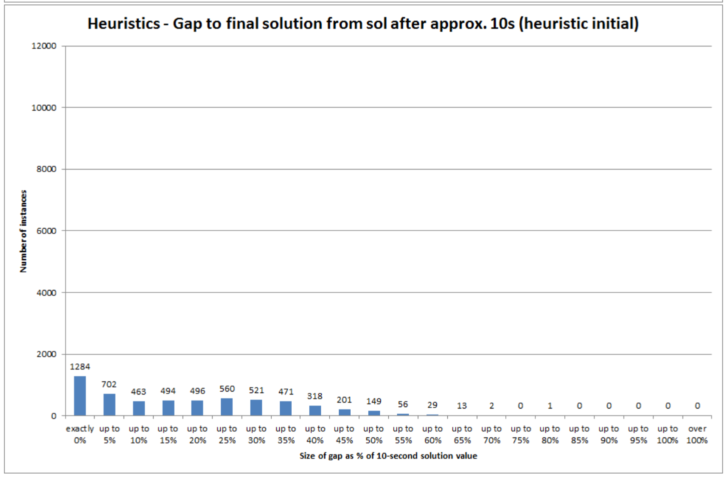
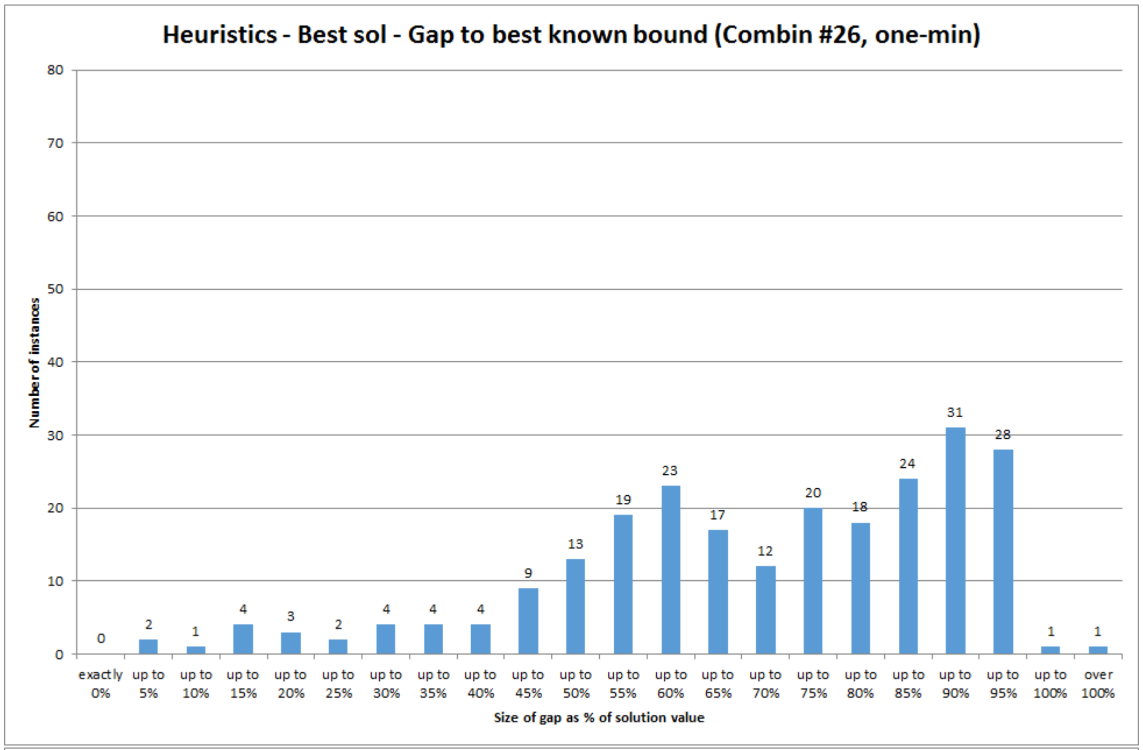


Figure D.52: Gap to two-minute solution value from estimated 10-second solution, broken down by whether the Task-Based Approximation (top) or Heuristic (bottom) initial solution was used.

D.2.9 Results for recommended combination

It is discussed in section 6.8.4.9 that the combination of settings labelled as Combination #26 (see Table D.2) is the most likely to produce good quality results, taking into account considerations such as solution cost, number of changes, number of (good) solutions, and the possibility of terminating the algorithm earlier than the two-minute time limit. Here, we give results for the measures of interest for Combination #26 only, which comprises summary histograms of the following:

- The gap to the best known bound, for both the one-minute and two-minute solutions (shown together in Figure D.53);
- The gap to best known solution, for both the one-minute and two-minute solutions (shown together in Figure D.54);
- Improvement made on the cost of the initial solution, for both the one-minute and two-minute solutions (shown together in Figure D.55);
- Number of changes in the first solution found with the lowest cost, both within one minute and within two minutes (shown together in Figure D.56);
- Number of distinct solutions found with solution value equal to the best cost, both within one minute and within two minutes (shown together in Figure D.57);
- Number of iterations carried out by the algorithm, for both the one-minute and two-minute results (shown together in Figure D.58);
- Estimated time at which the first solution with the best cost value was found, for both the one-minute and two-minute results (shown together in Figure D.59); and
- Estimated gap to the final (i.e. two-minute) solution if the algorithm was terminated after one minute, 30 seconds, 20 seconds or 10 seconds (shown together in Figure D.60).



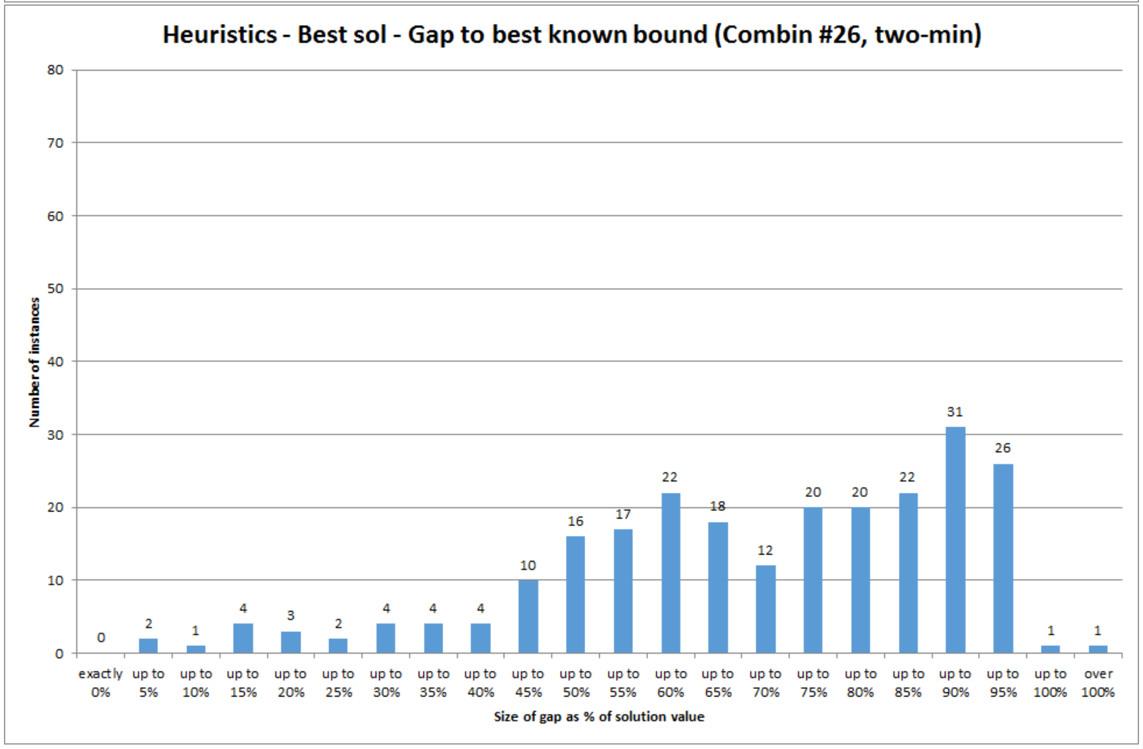
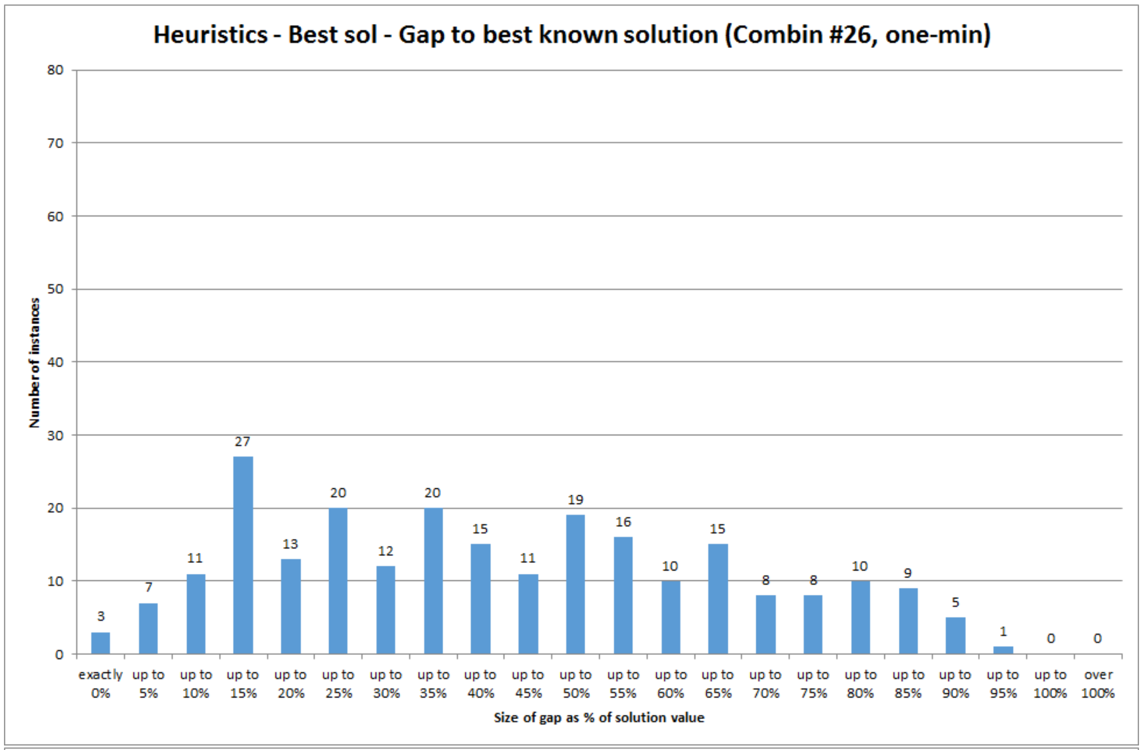


Figure D.53: Gap to best known bound from solutions found after one minute (top) and after two minutes (bottom), for setting combination #26 only.



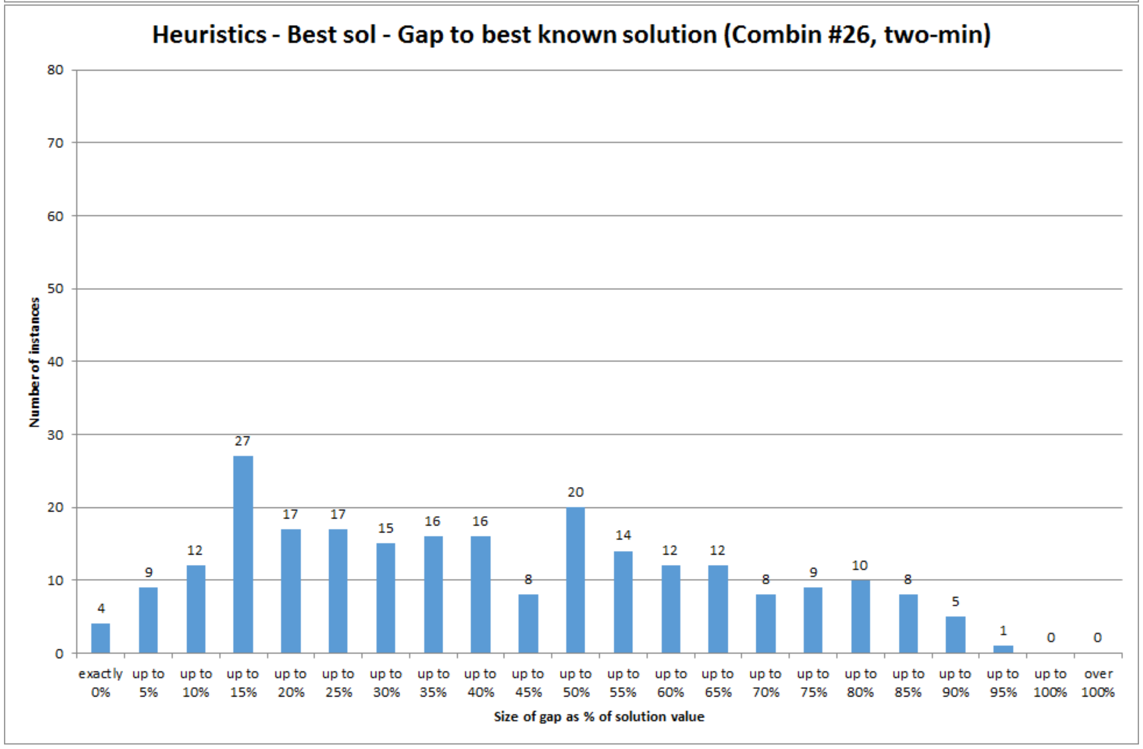
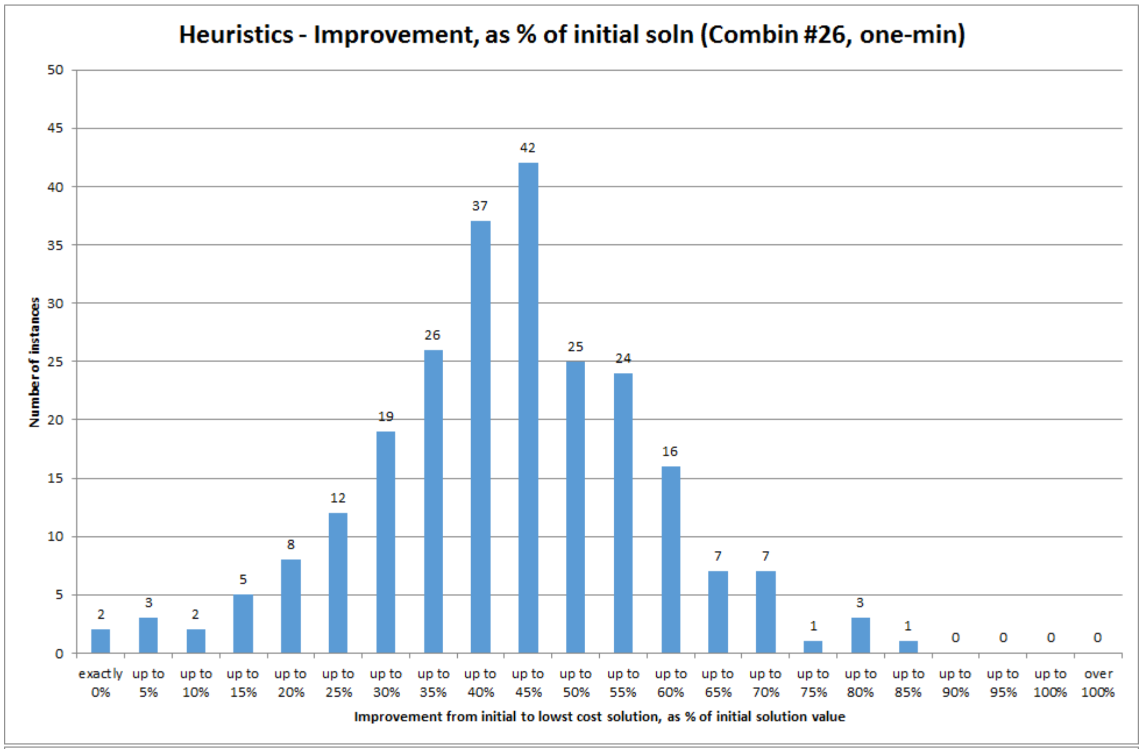


Figure D.54: Gap to best known solution overall from solutions found after one minute (top) and two minutes (bottom), for setting combination #26 only.



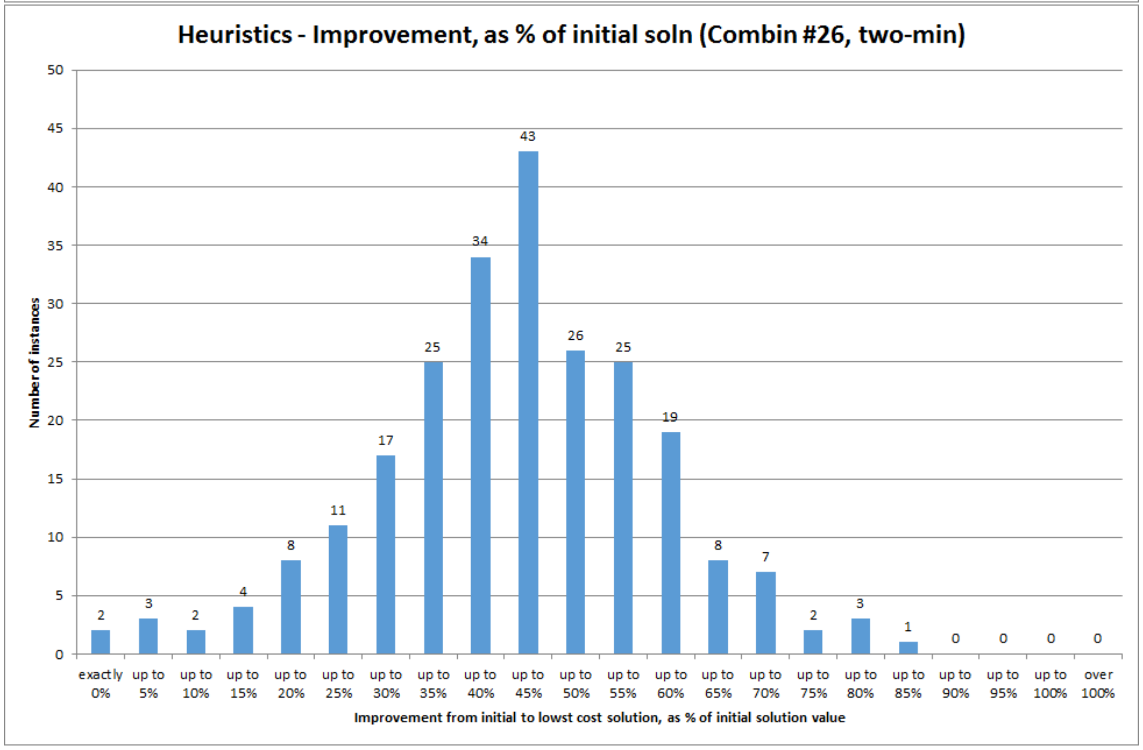
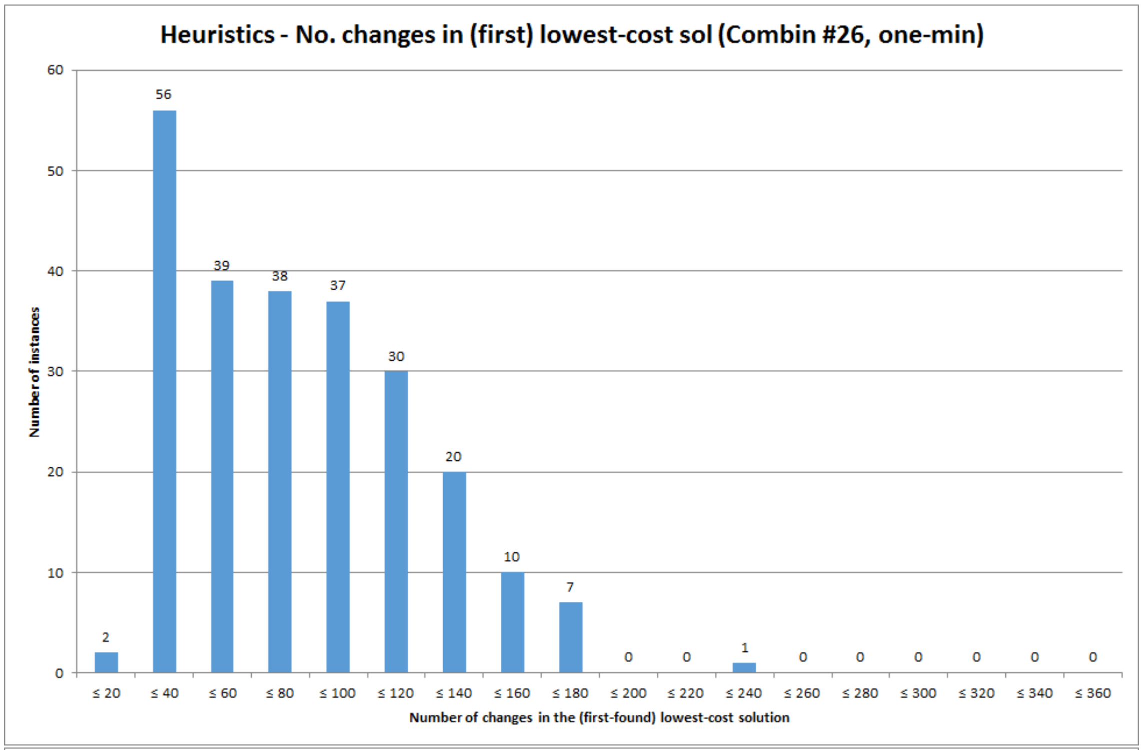


Figure D.55: Improvement on the initial solution achieved within one minute (top) and two minutes (bottom) given as a percentage of the initial solution value, for setting combination #26 only.



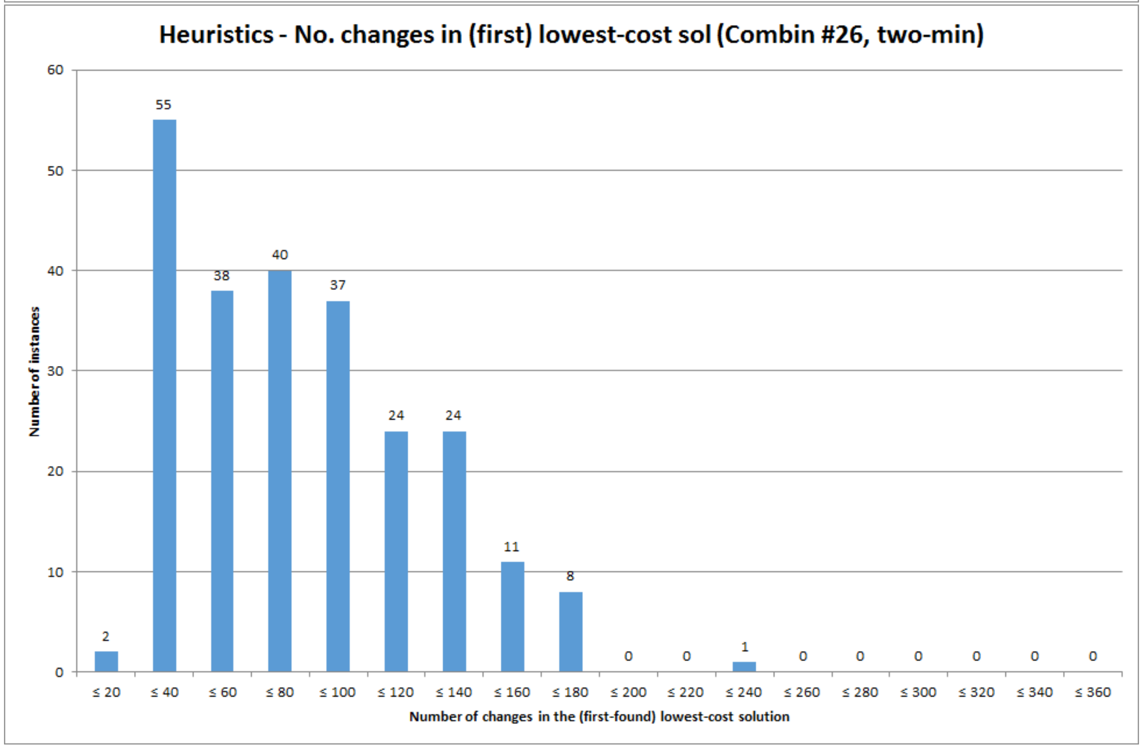
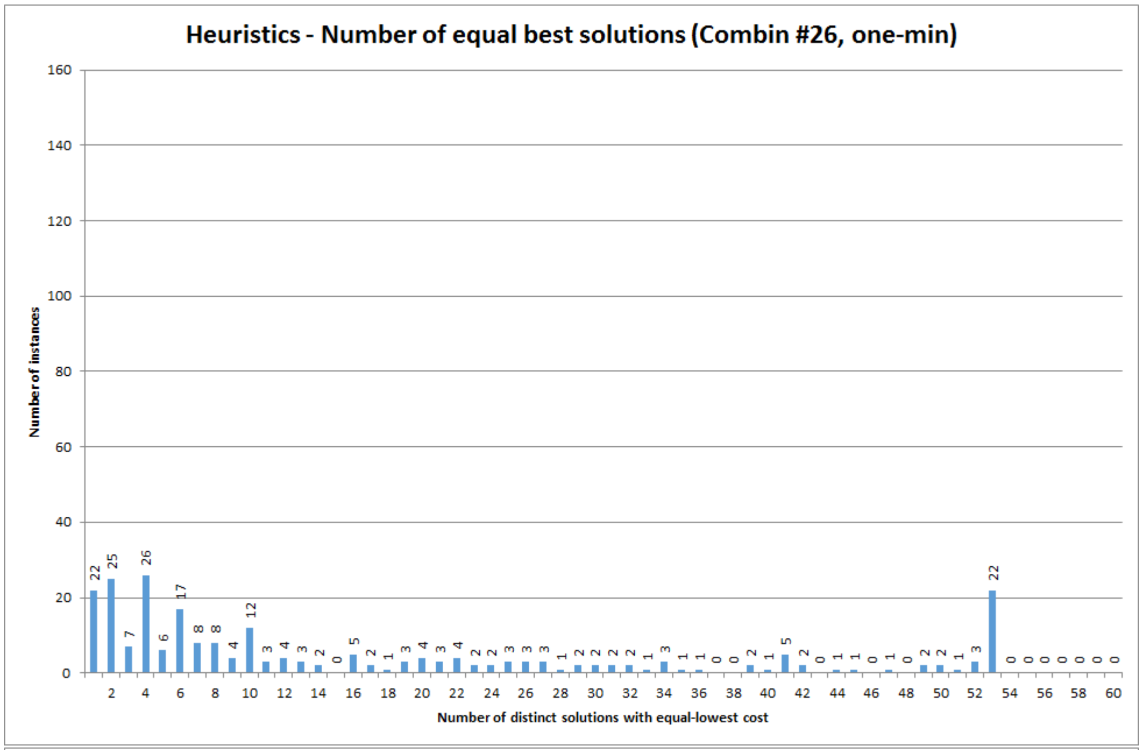


Figure D.56: Number of changes in the first lowest-cost solution found within one minute (top) and two minutes (bottom), for setting combination #26 only.



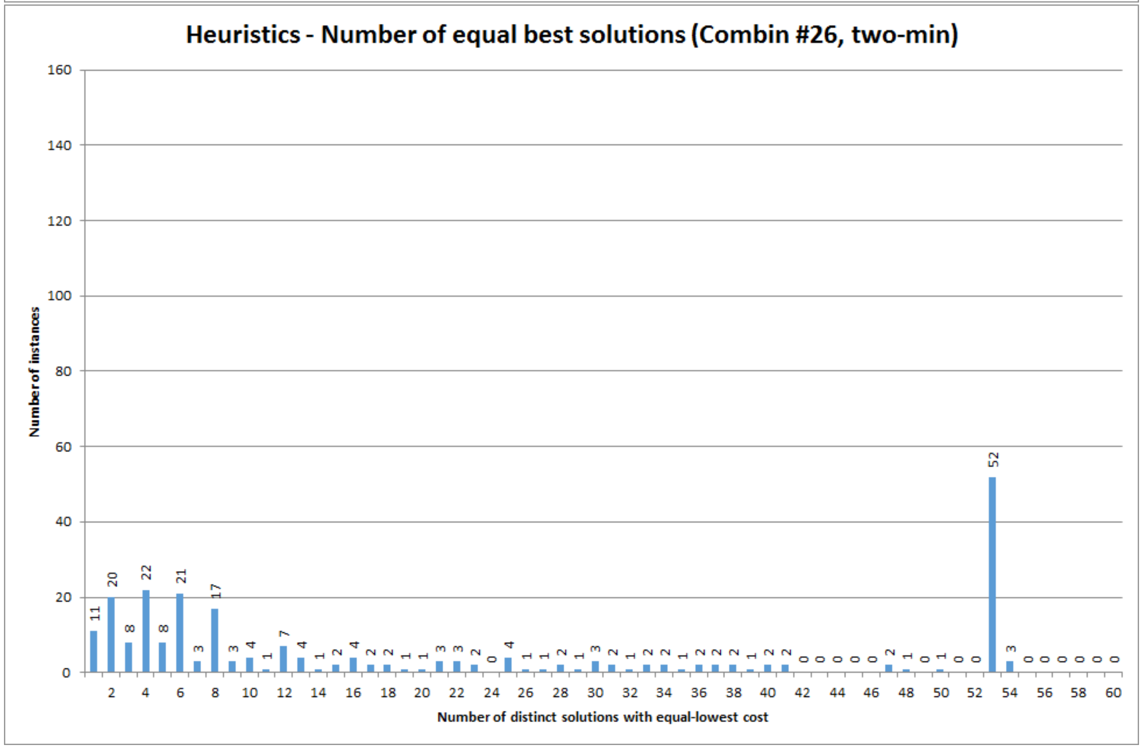
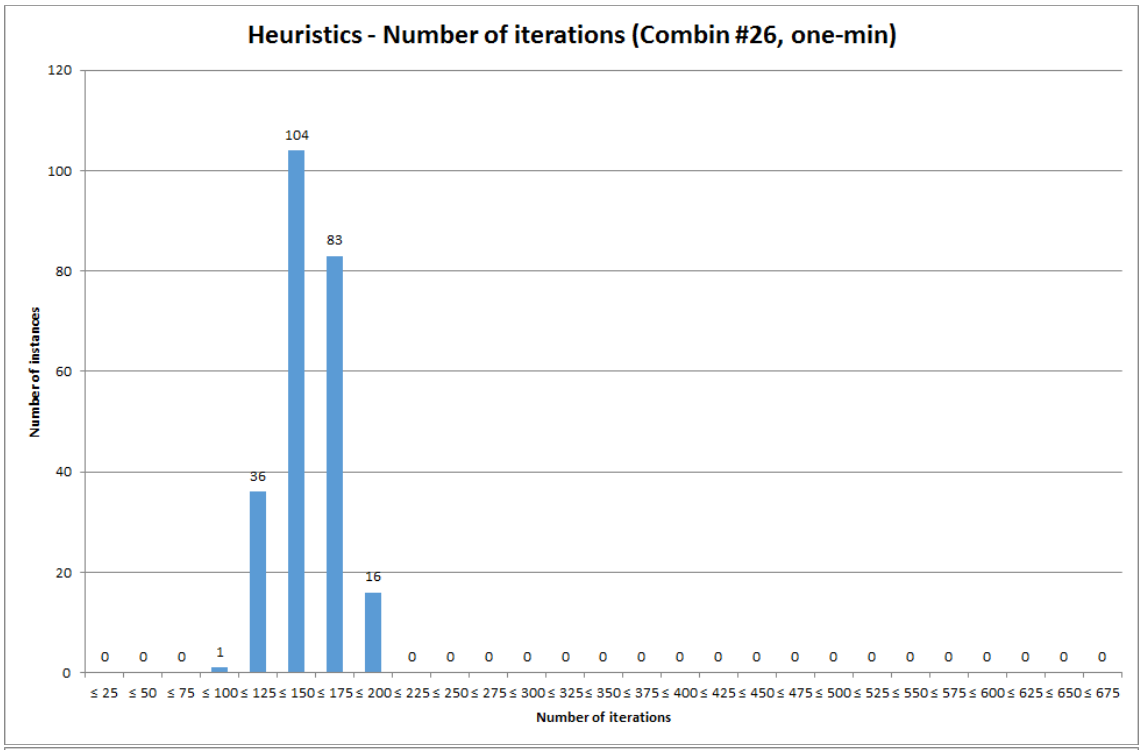


Figure D.57: Number of solutions found with equal-best cost after one minute (top) and after two minutes (bottom), for setting combination #26 only.



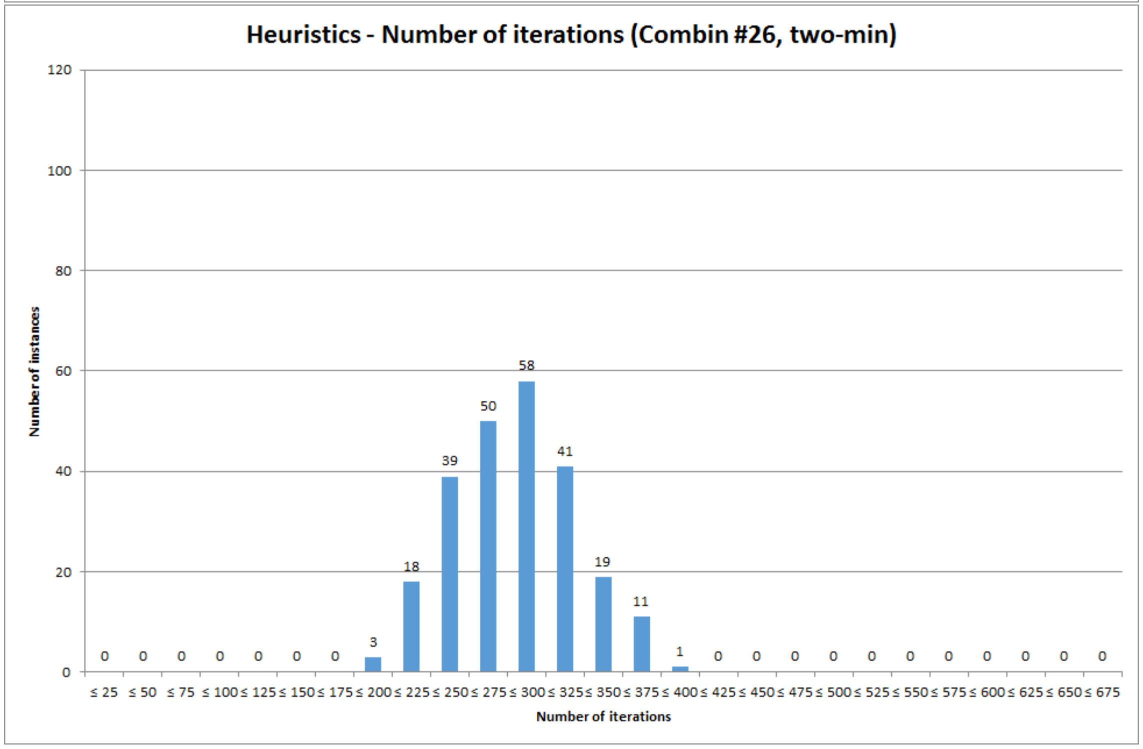
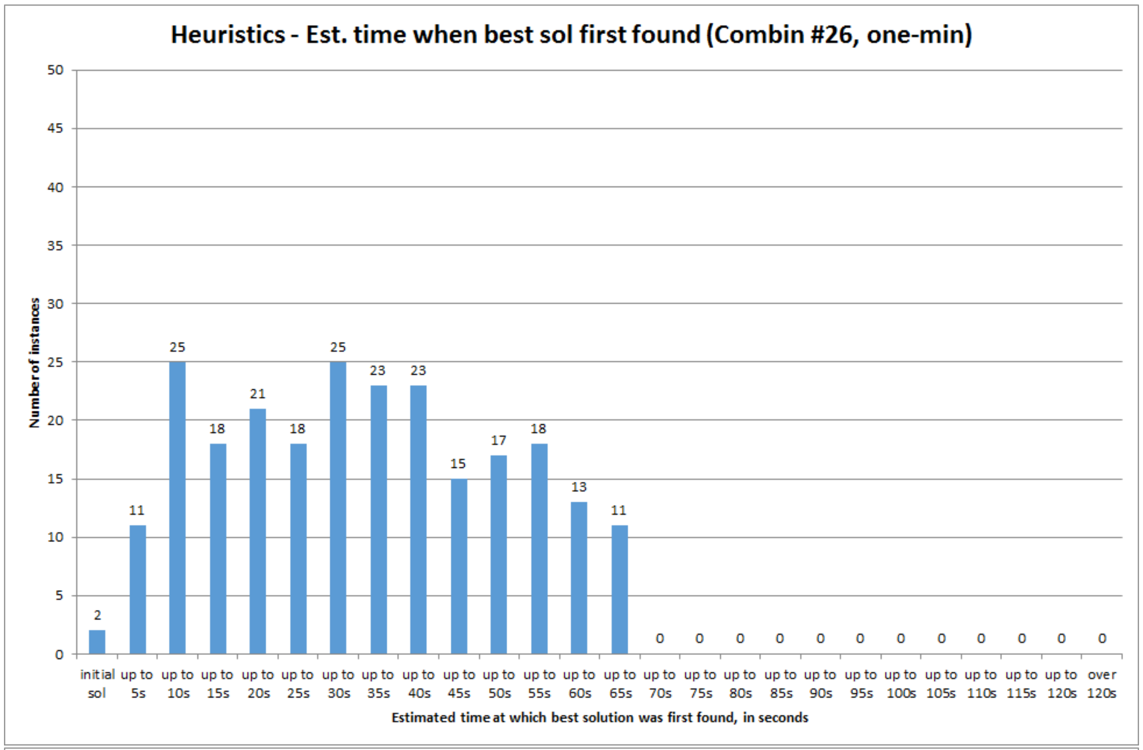


Figure D.58: Number of iterations carried out by the algorithm within one minute (top) and within two minutes (bottom), for setting combination #26 only.



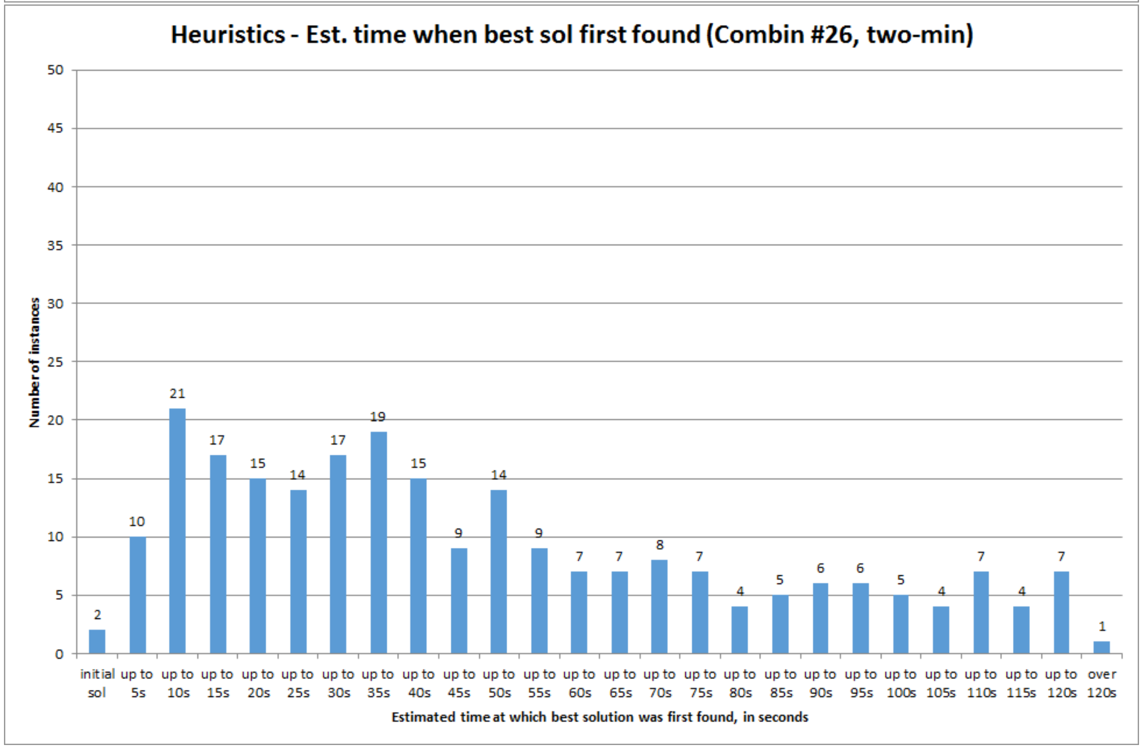


Figure D.59: Estimated time at which the best solution within one minute (top) and within two minutes (bottom) was first found, for setting combination #26 only.

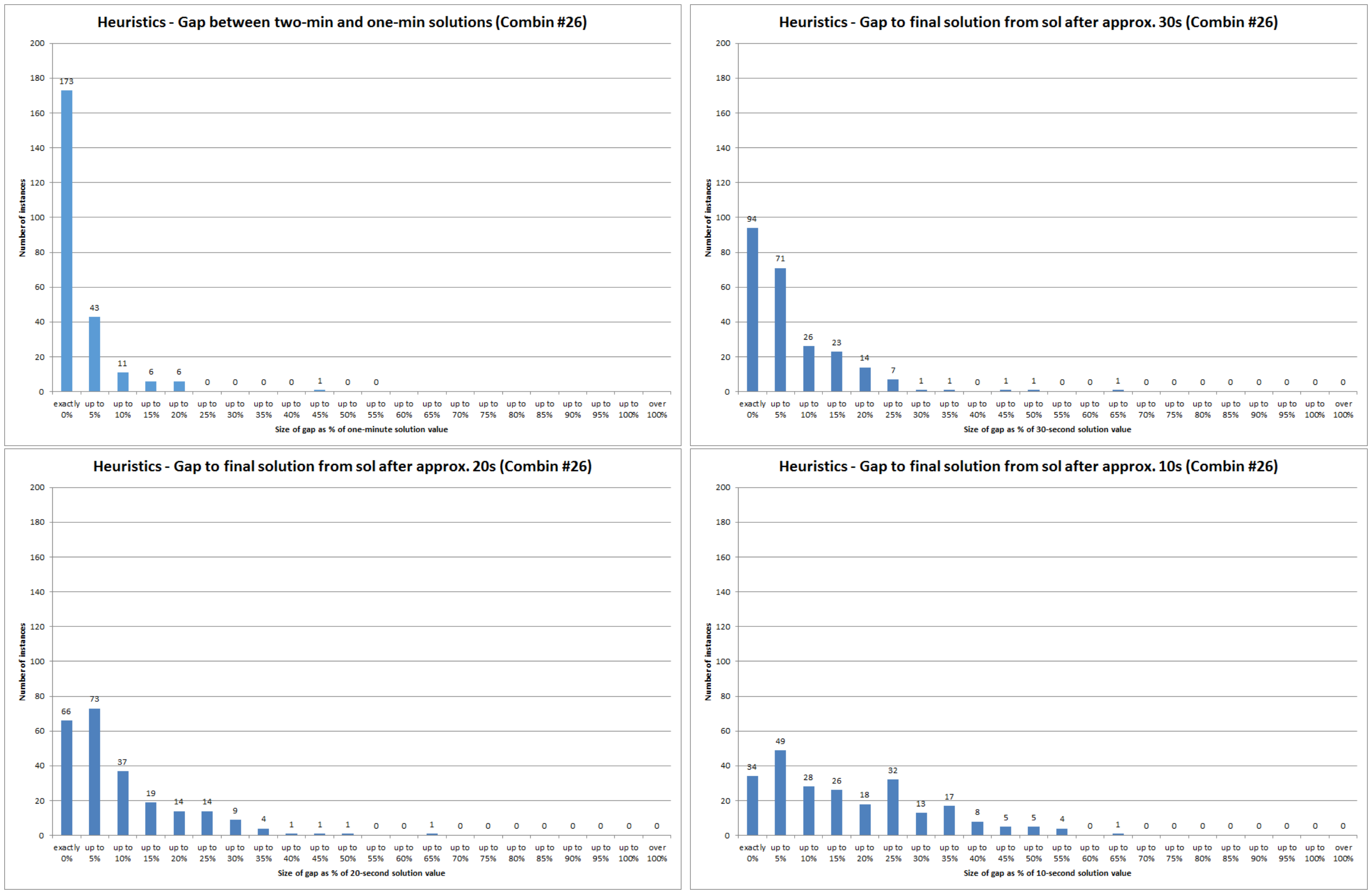


Figure D.60: Gap to two-minute solution value from one-minute solution (top left), estimated 30-second solution (top right), estimated 20-second solution (bottom left) and estimated 10-second solution (bottom right), for setting combination #26 only.

D.3 Relating to the effect of data generating parameters

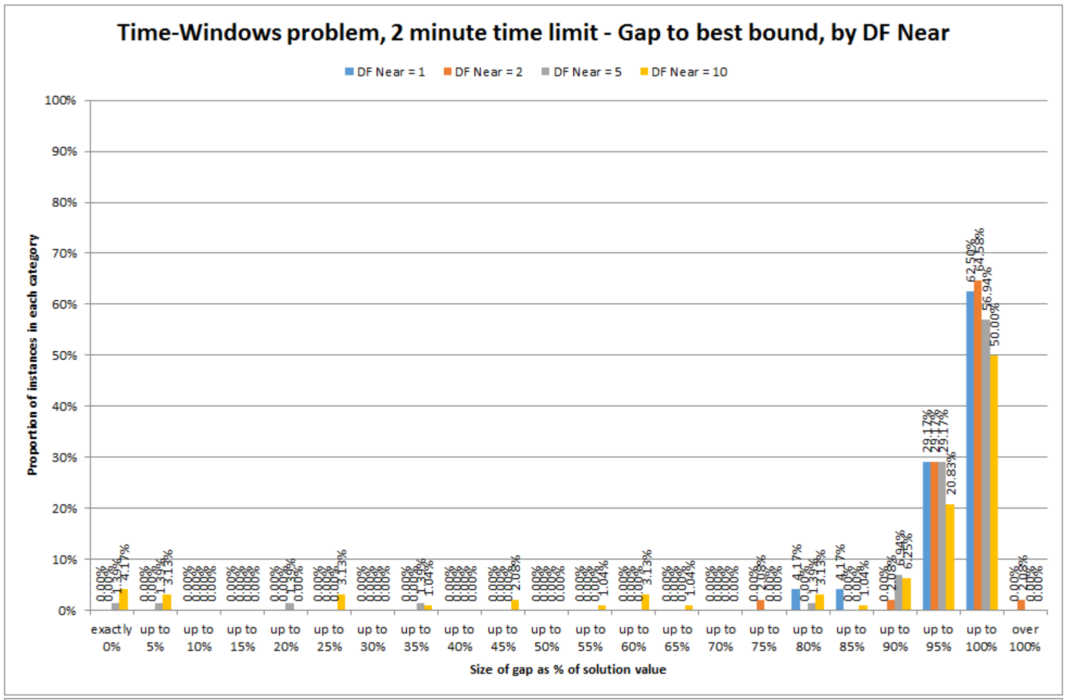
Section 6.8.5 gave results of Analysis of Variance tests for the influence of the values of the parameters used in data generation on each of the solution approaches tested for the Time-Windows formulation. This section gives histograms showing the breakdown of the outputs by those parameters which have a significant influence. This is divided into subsections in the same way as section 6.8.5, according to the types of results which are being analysed. Firstly, section D.3.1 shows the breakdown of significant parameters for the initial computational results (and the associated further analysis) which is discussed in section 6.8.5.1; then in sections D.3.2 and D.3.3 we look at the details of the influence of parameters on the two initial solution approaches as determined in sections 6.8.5.2 for the Task-Based Approximation approach and section 6.8.5.3 for the heuristic initial solution approach respectively. Finally, details of the parameter value influence on the main heuristic algorithm, both for the aggregated results for all settings and for Combination #26 specifically, as discussed in section 6.8.5.4, are given in section D.3.4.

D.3.1 Breakdown of Initial Computational results

Here we present selected histograms which can aid our understanding of the Kruskall-Wallis test results given in Table 6.15. These show the same Initial computational results as in sections 6.3 and 6.8.1, but broken down according to the values of the parameters which this table highlights as having a significant influence.

D.3.1.1 Relating to 'direct' gap to best known bound

We begin with breakdowns of the gap to the best known bound for the 'direct' solution approach:



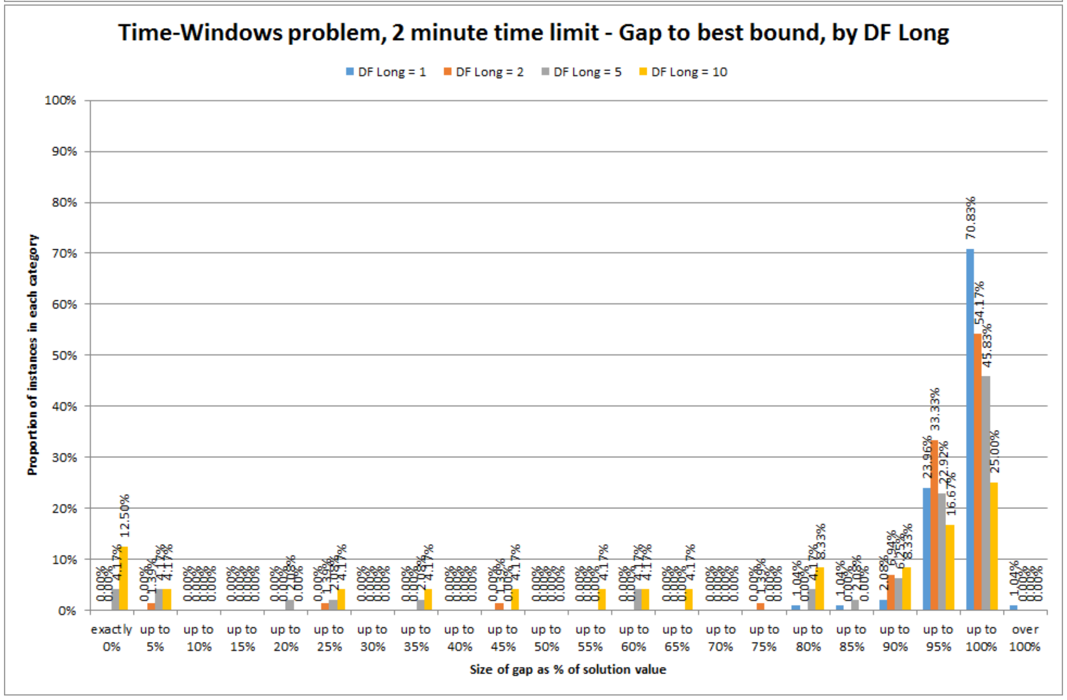
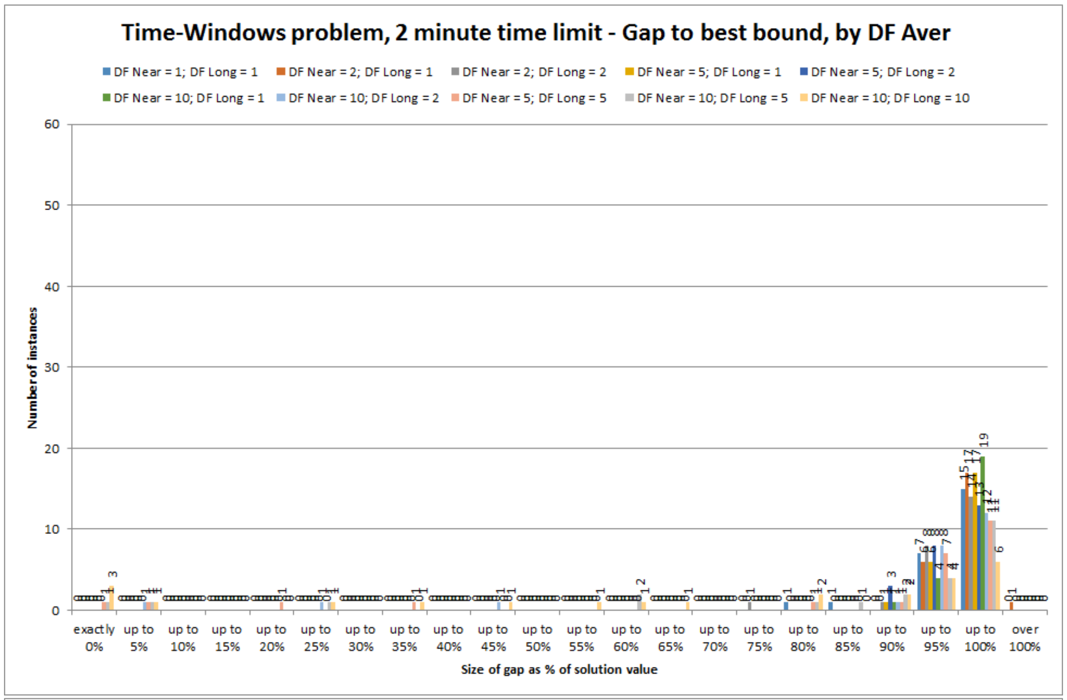


Figure D.61: Histogram of percentage gaps to best bounds for 'direct' solution of Time-Windows instances using FICO Xpress, 2 minute time limit, broken down by values of the near-term disruption penalty K_N (top) and by values of the long-term disruption penalty K_L (bottom).



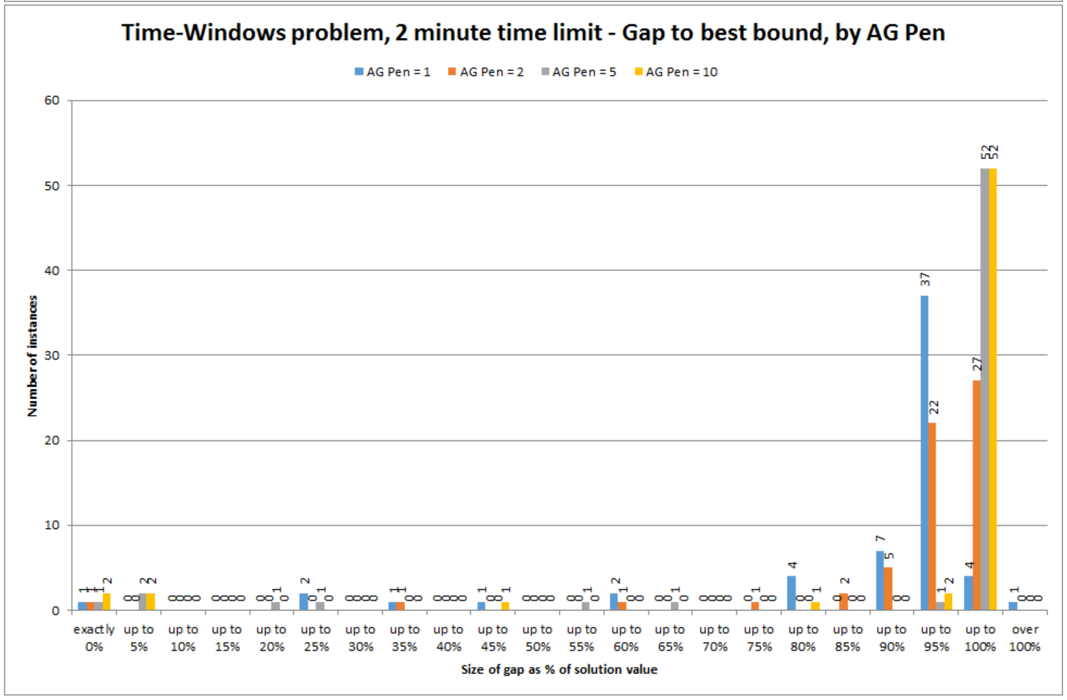


Figure D.62: Histogram of percentage gaps to best bounds for 'direct' solution of Time-Windows instances using FICO Xpress, 2 minute time limit, broken down by values of the weighted average disruption penalty $\hat{K}$ (top) and by values of the agency penalty factor K_{AG} (bottom).

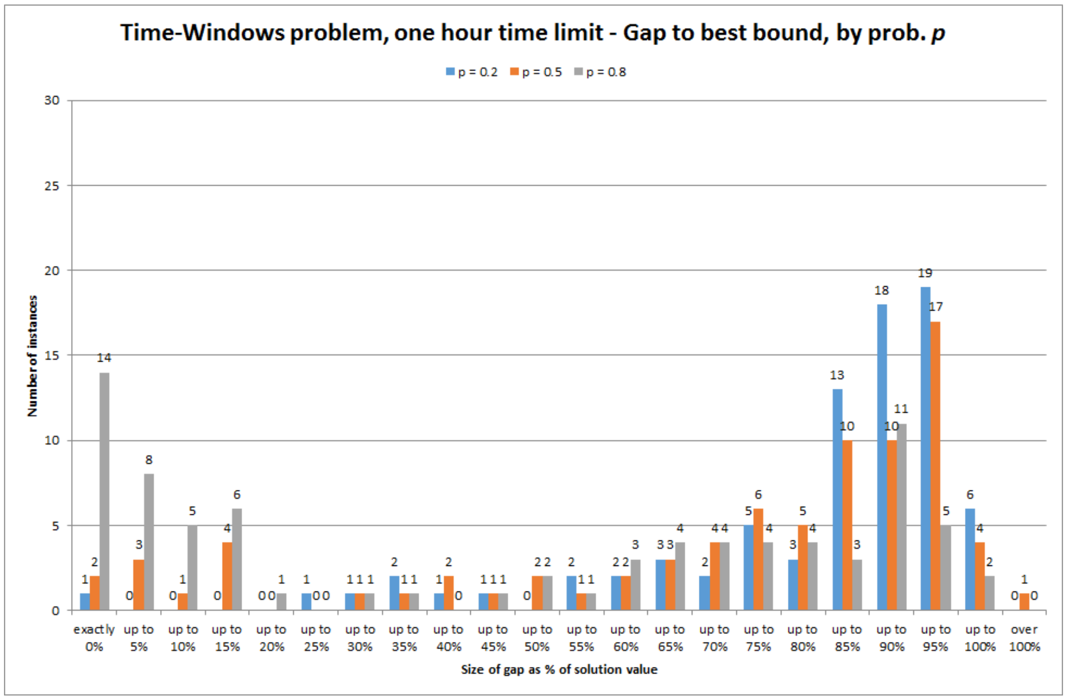
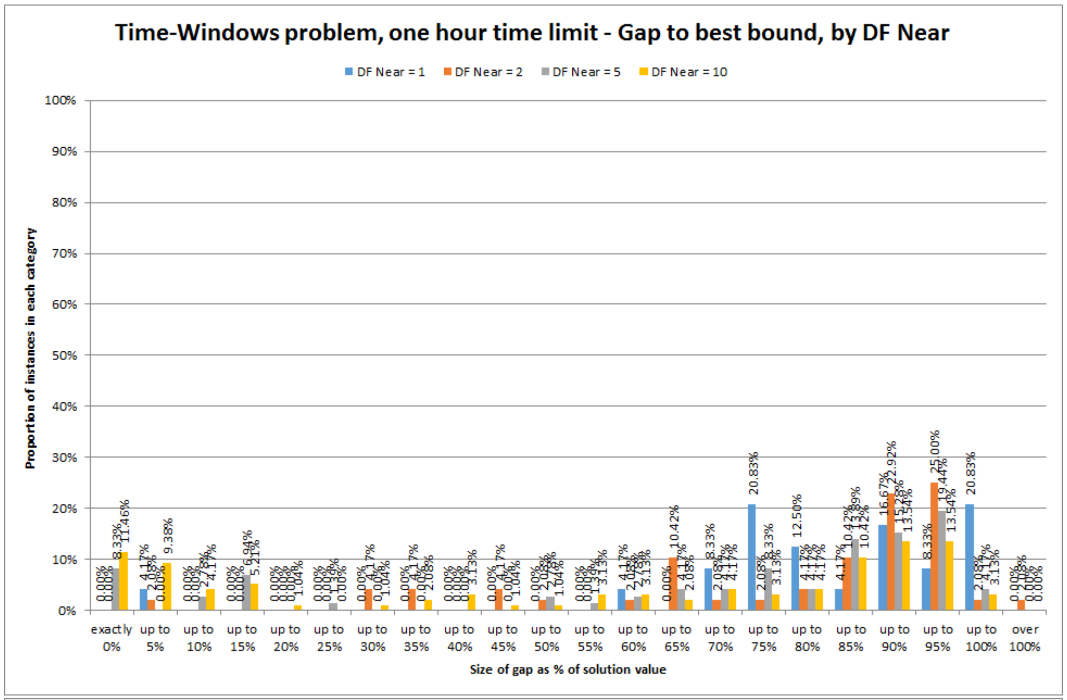


Figure D.63: Histogram of percentage gaps to best bounds for ‘direct’ solution of Time-Windows instances using FICO Xpress, 1 hour time limit, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



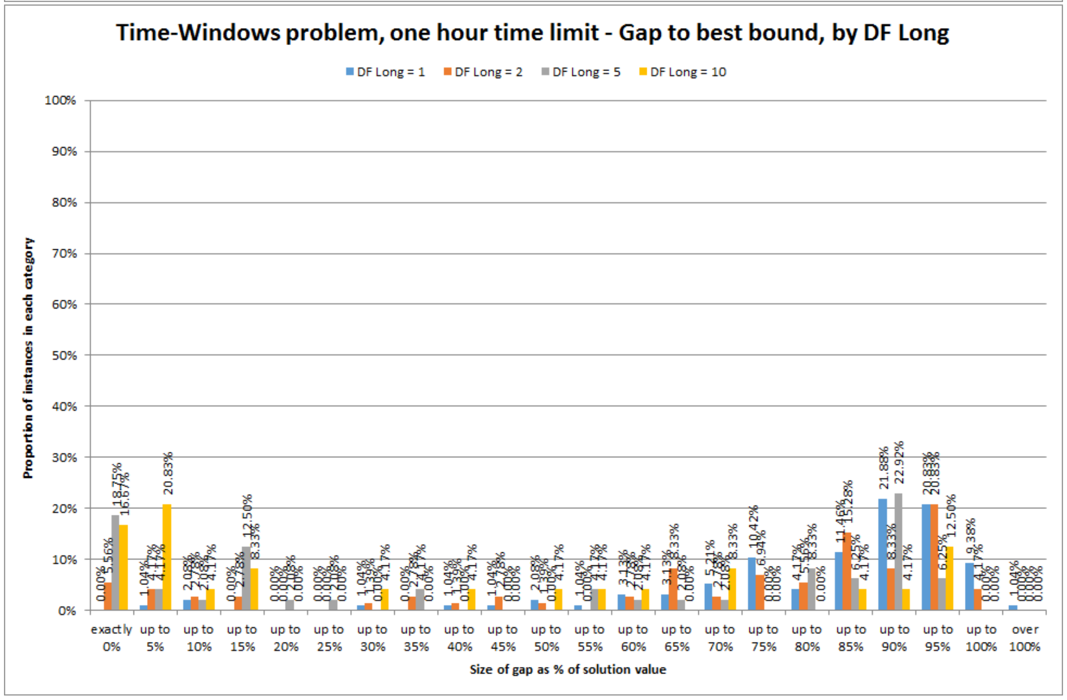
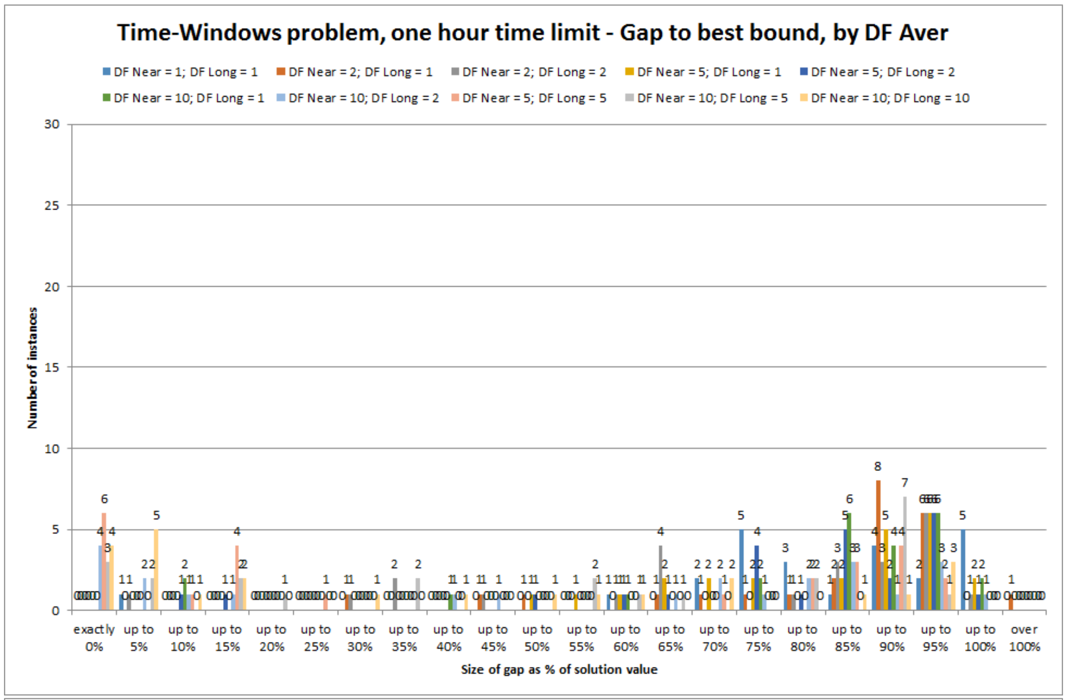


Figure D.64: Histogram of percentage gaps to best bounds for 'direct' solution of Time-Windows instances using FICO Xpress, 1 hour time limit, broken down by values of the near-term disruption penalty K_N (top) and by values of the long-term disruption penalty K_L (bottom).



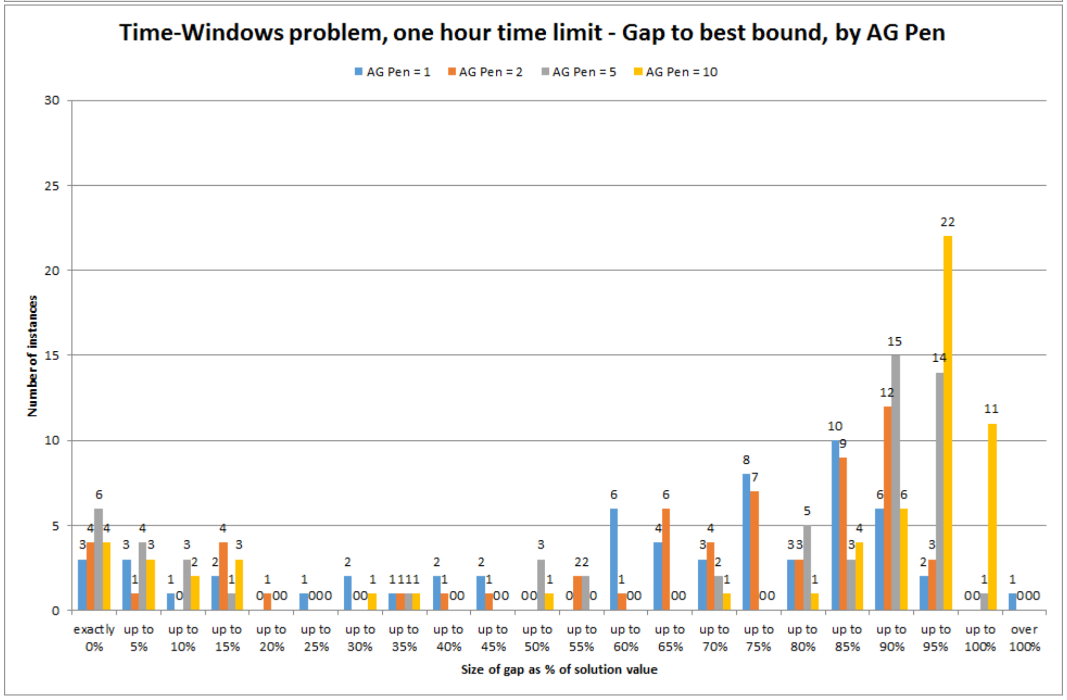


Figure D.65: Histogram of percentage gaps to best bounds for ‘direct’ solution of Time-Windows instances using FICO Xpress, 1 hour time limit, broken down by values of the weighted average disruption penalty $\hat{K}$ (top) and by values of the agency penalty factor K_{AG} (bottom).

D.3.1.2 Relating to ‘direct’ gap to best known solution

The following describe breakdowns of the gap to the best known solution for the ‘direct’ solution approach:

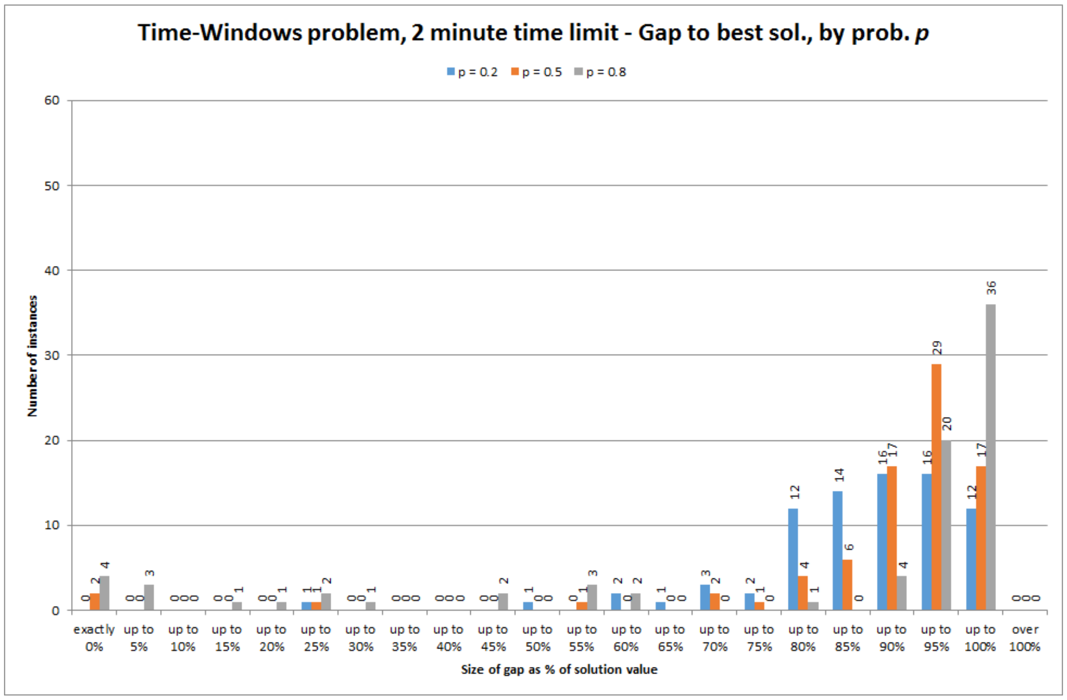
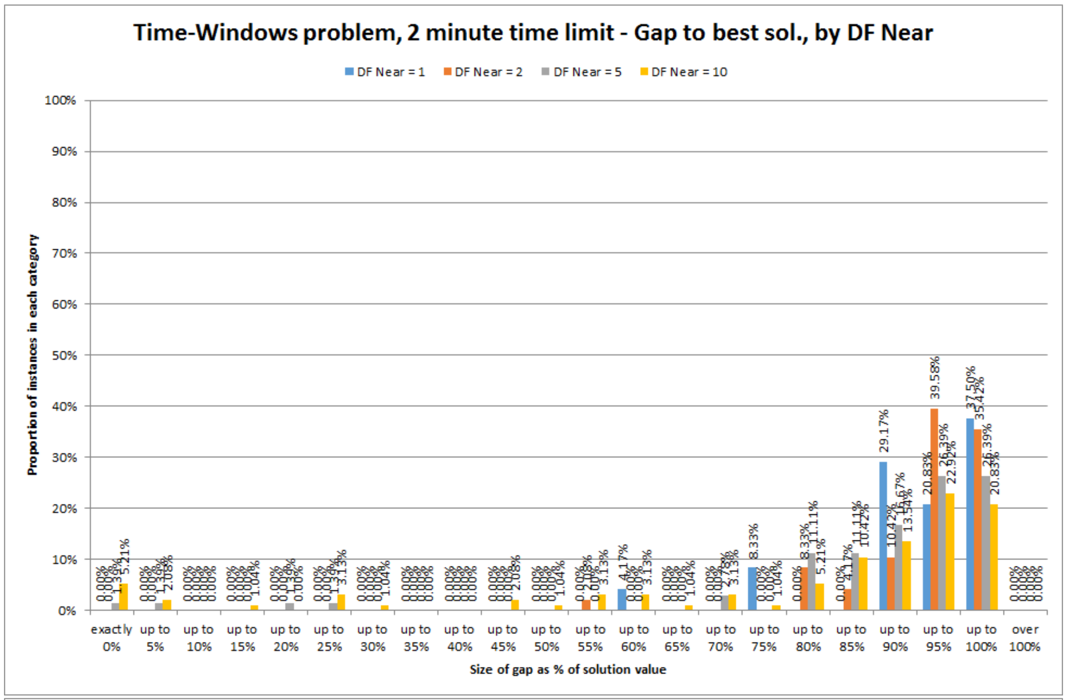


Figure D.66: Histogram of percentage gaps to best known solution for ‘direct’ solution of Time-Windows instances using FICO Xpress, 2 minute time limit, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



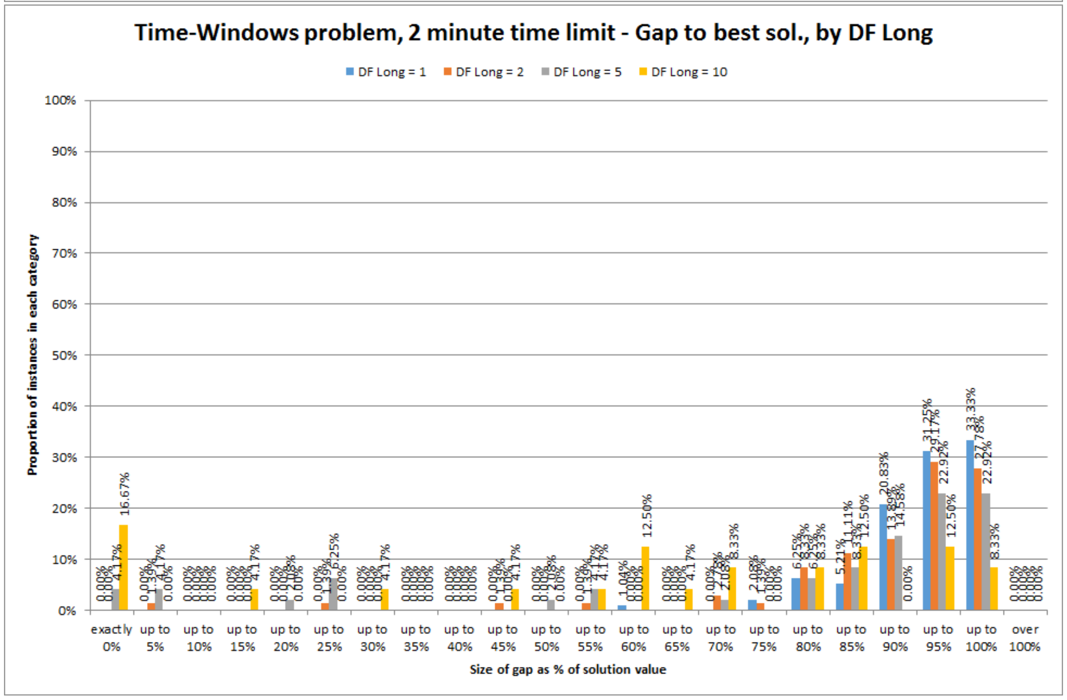
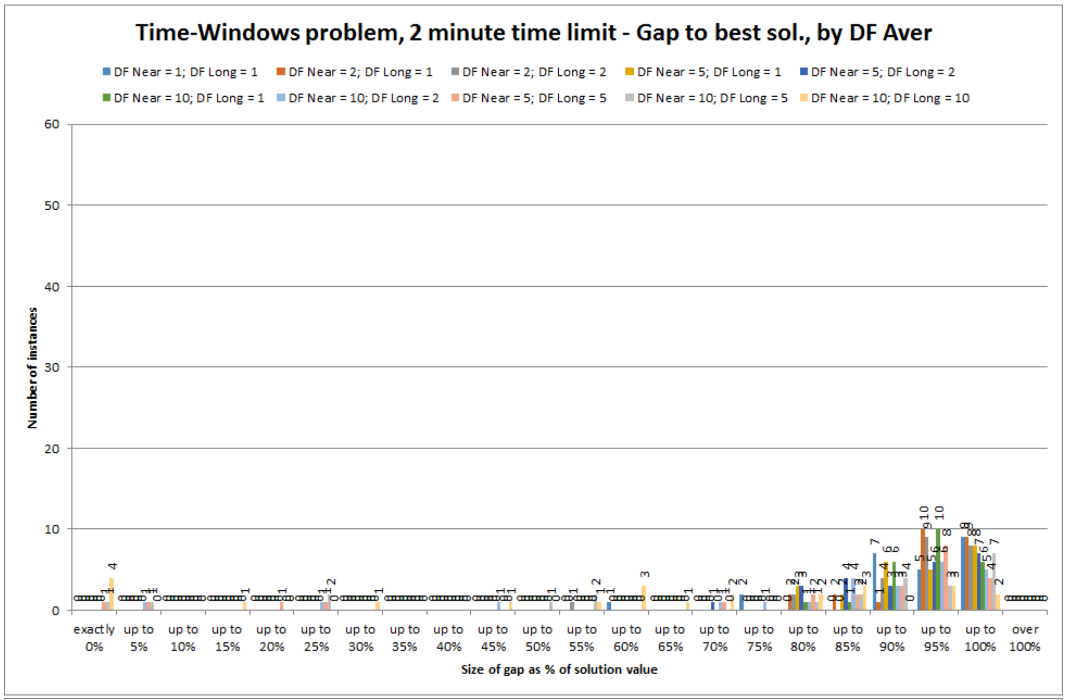


Figure D.67: Histogram of percentage gaps to best known solution for ‘direct’ solution of Time-Windows instances using FICO Xpress, 2 minute time limit, broken down by values of the near-term disruption penalty K_N (top) and by values of the long-term disruption penalty K_L (bottom).



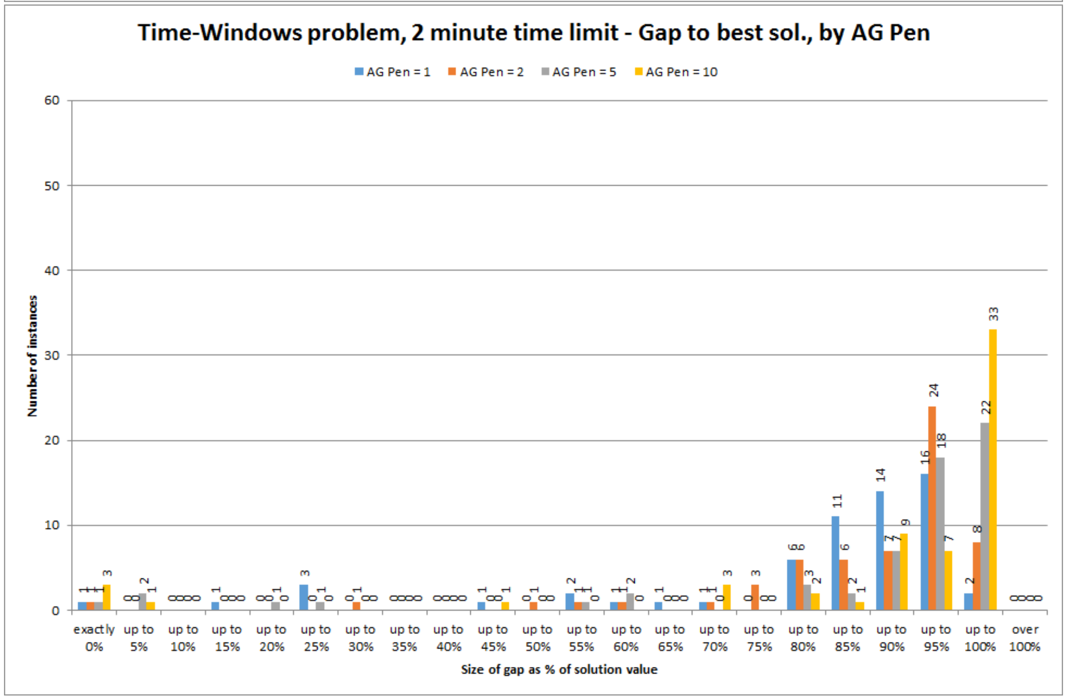
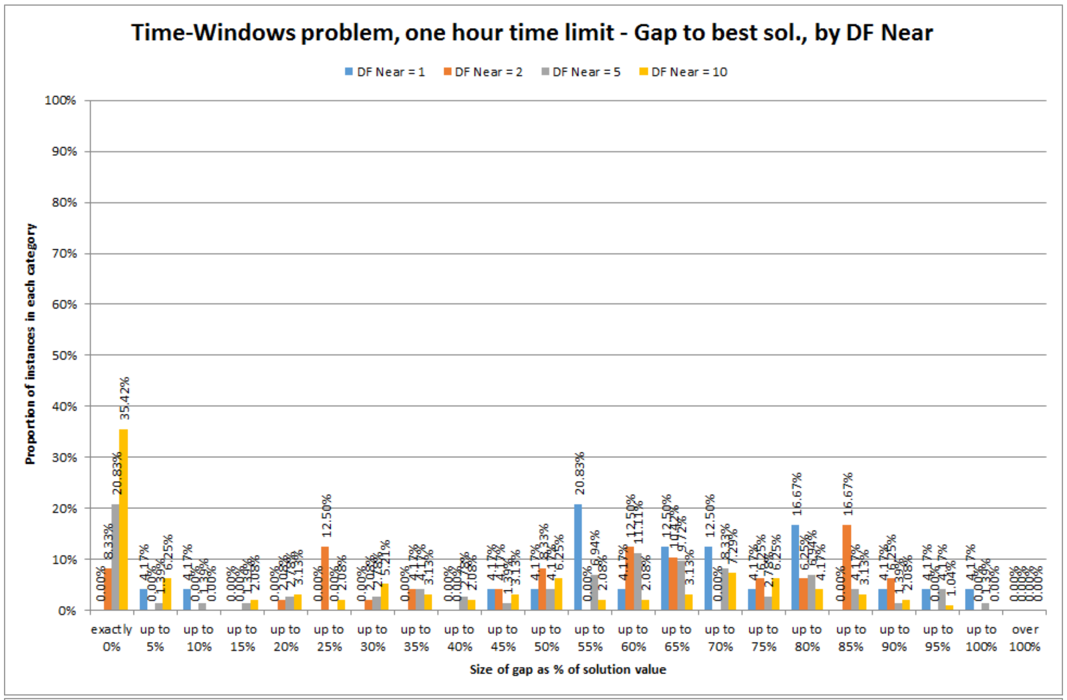


Figure D.68: Histogram of percentage gaps to best known solution for 'direct' solution of Time-Windows instances using FICO Xpress, 2 minute time limit, broken down by values of the weighted average disruption penalty $\hat{K}$ (top) and by values of the agency penalty factor K_{AG} (bottom).



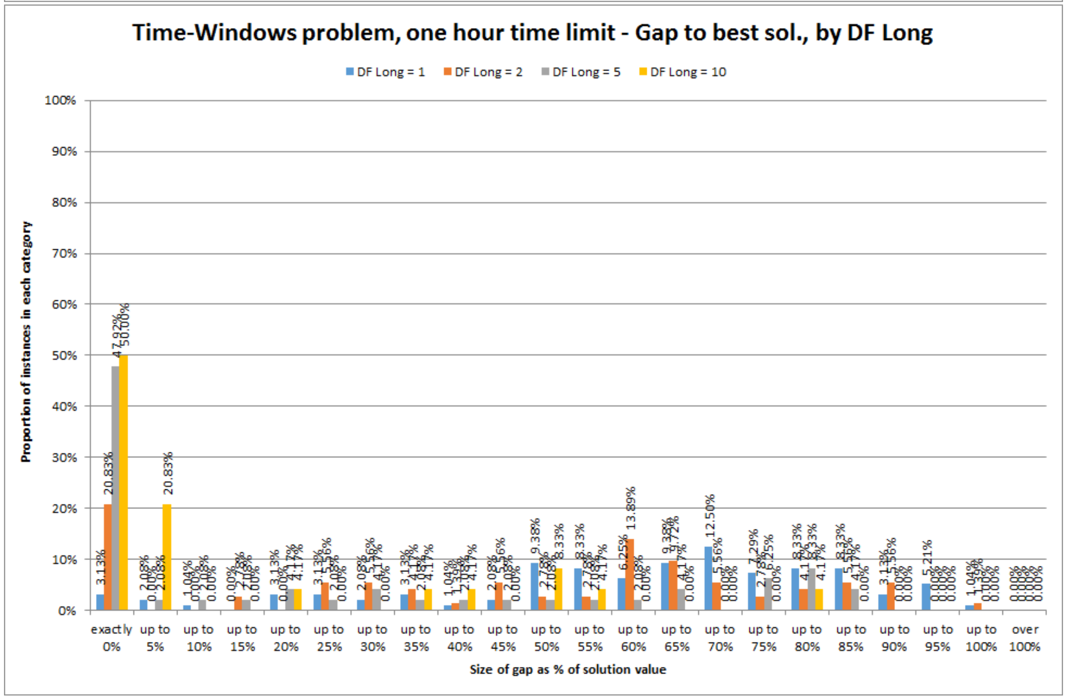


Figure D.69: Histogram of percentage gaps to best known solution for 'direct' solution of Time-Windows instances using FICO Xpress, 1 hour time limit, broken down by values of the near-term disruption penalty K_N (top) and by values of the long-term disruption penalty K_L (bottom).

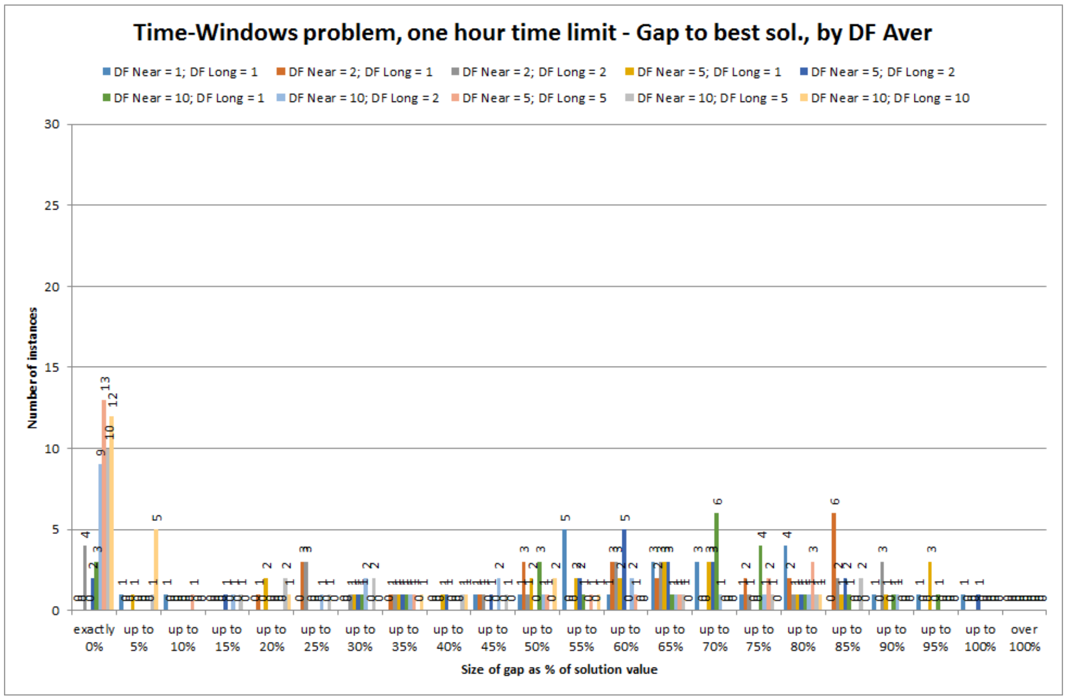
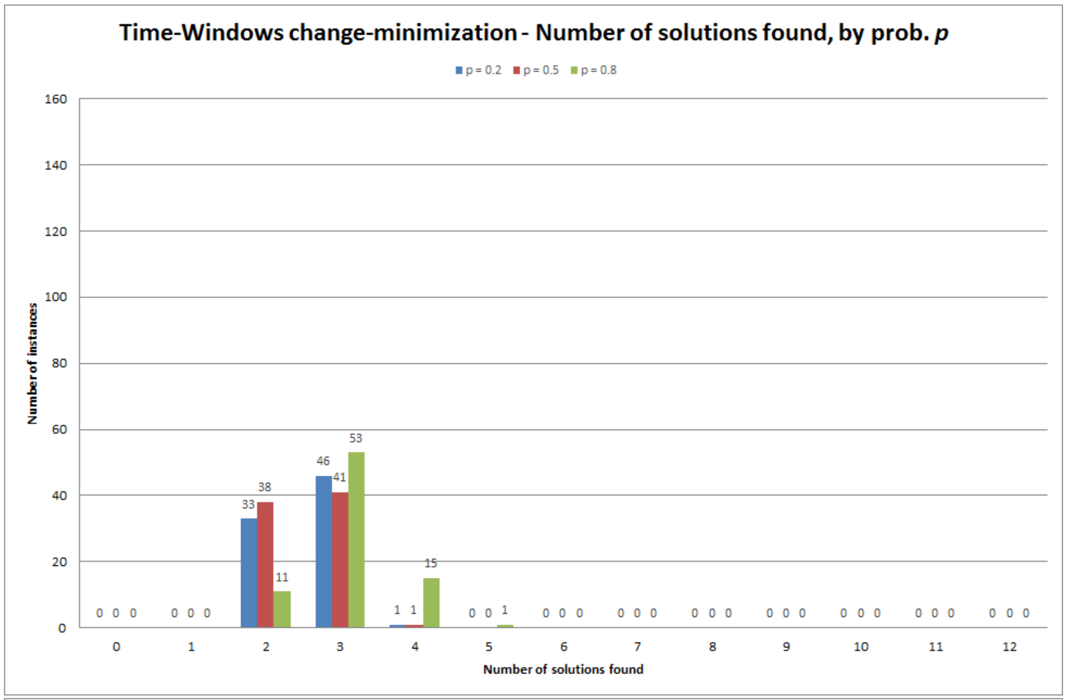


Figure D.70: Histogram of percentage gaps to best known solution for 'direct' solution of Time-Windows instances using FICO Xpress, 1 hour time limit, broken down by values of the weighted average disruption penalty $\hat{K}$.

D.3.1.3 Relating to change minimization - solutions and iterations

The following describe breakdowns of the number of solutions and number of iterations for the change-minimization approach:



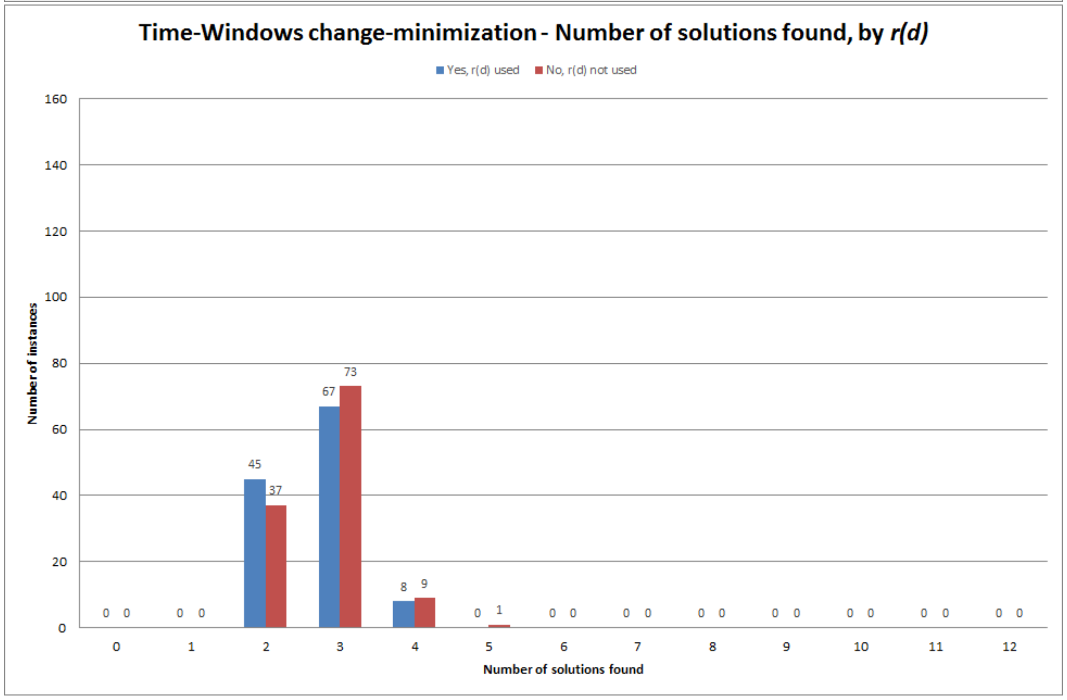


Figure D.71: Number of solutions found by each instance using algorithm for change-minimization approach, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day) (top) and by whether or not time reduction factor $r(d)$ was used (bottom).

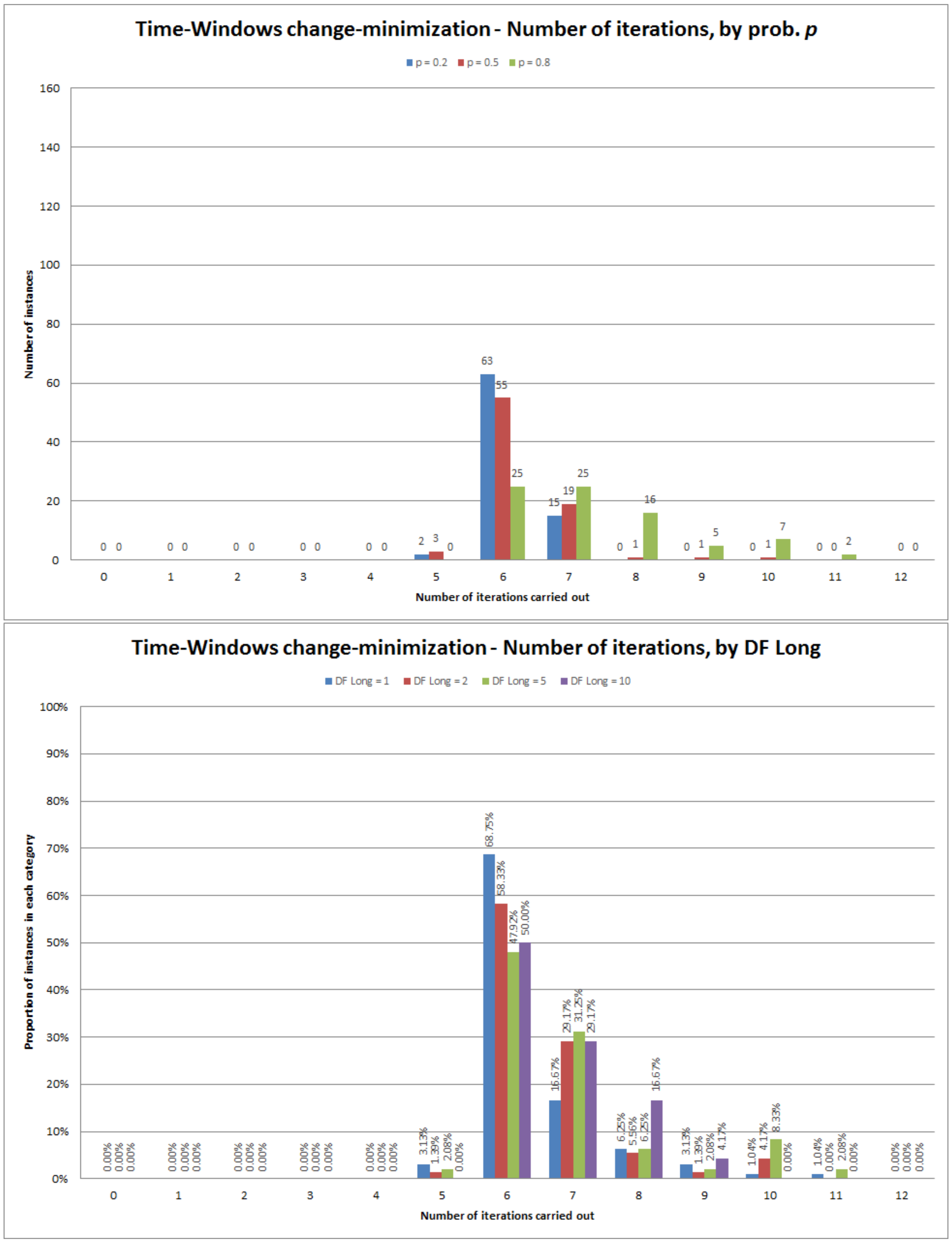


Figure D.72: Number of iterations of change-minimization algorithm carried out for each instance, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day) (top) and by values of the long-term disruption penalty K_L (bottom).

D.3.1.4 Relating to change minimization - cost gaps

The following describe breakdowns of the gap to the best known bound and best known solution for the change-minimization solutions:

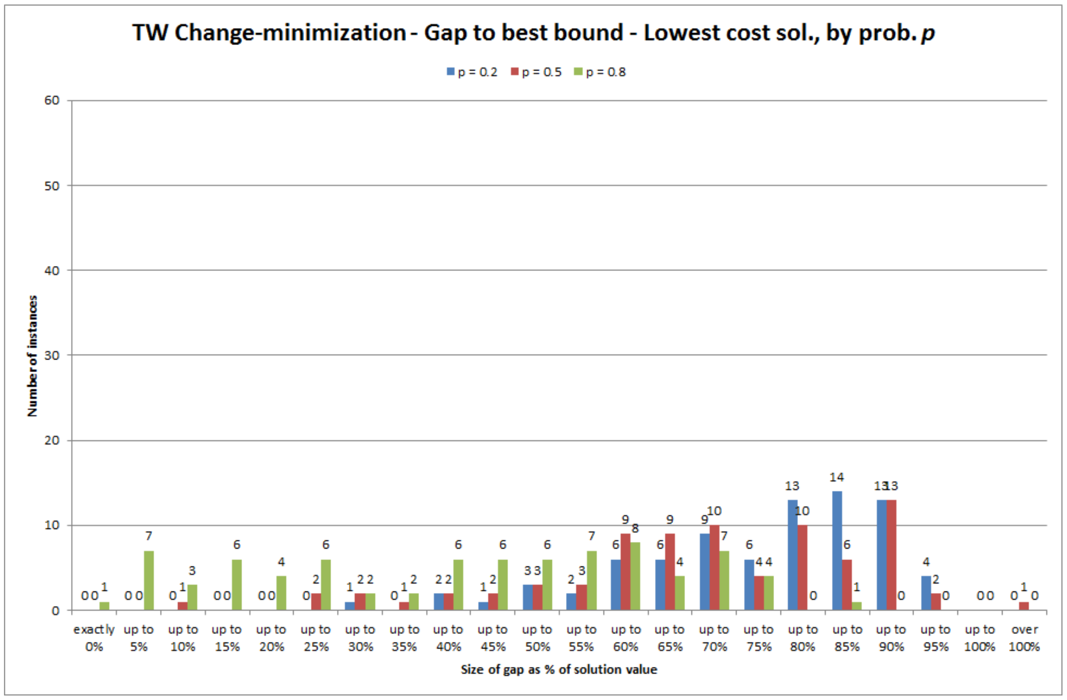


Figure D.73: Gap to best bound for lowest-cost solutions found in two minute Change-minimization test run for the Time-Windows formulation, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day)

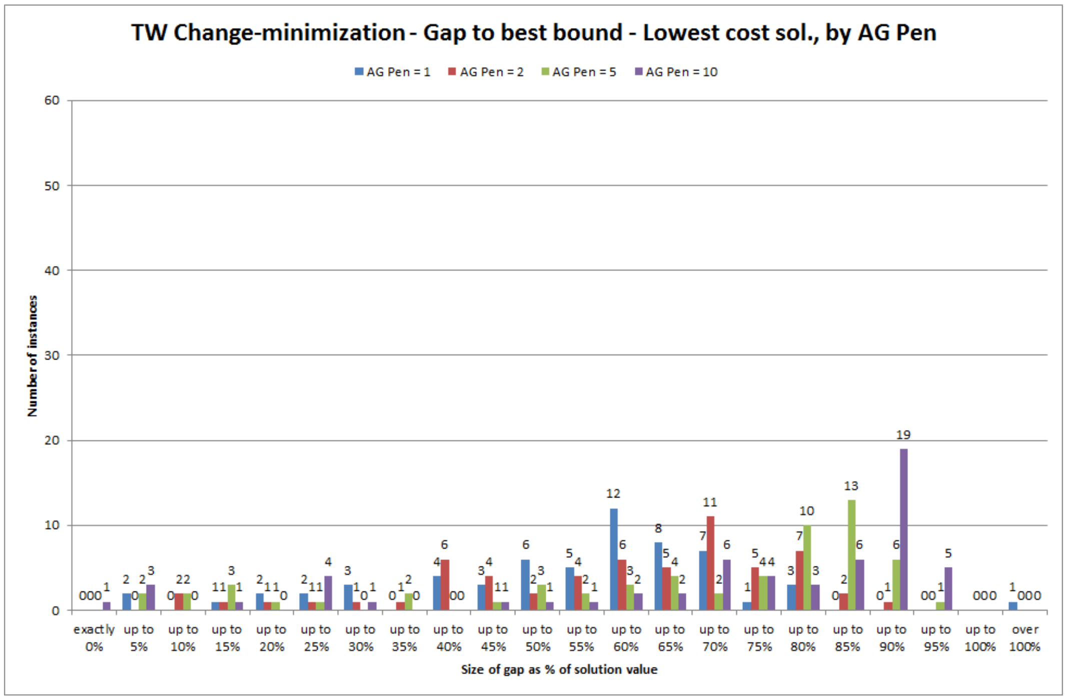
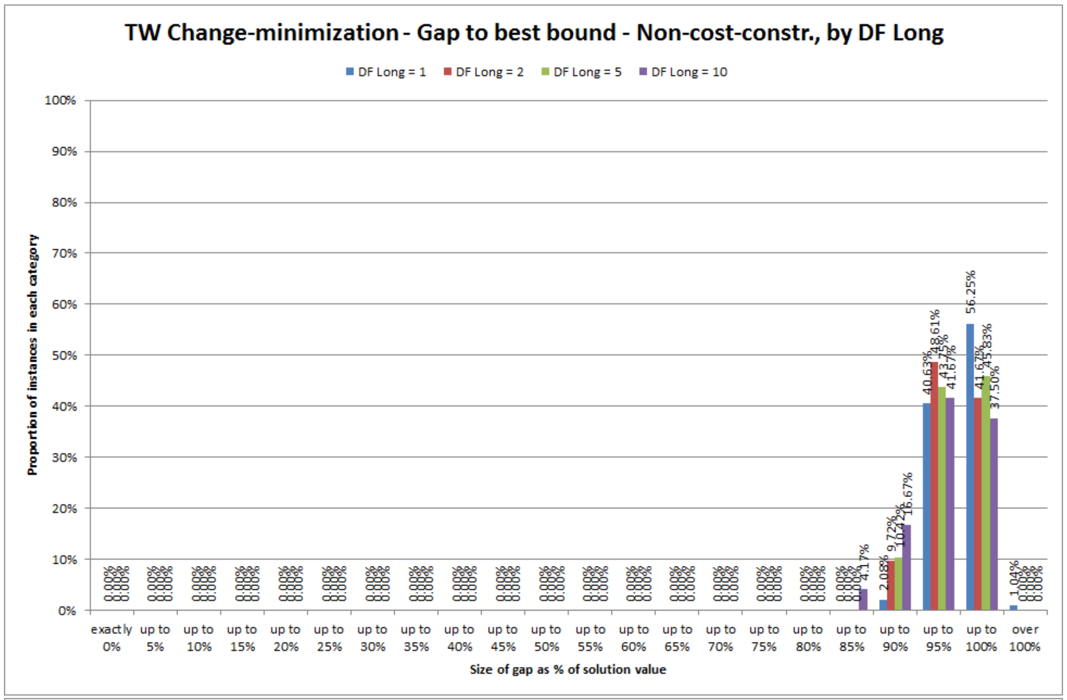


Figure D.74: Gap to best bound for lowest-cost solutions found in two minute Change-minimization test run for the Time-Windows formulation, broken down by values of the agency penalty factor K_{AG}.



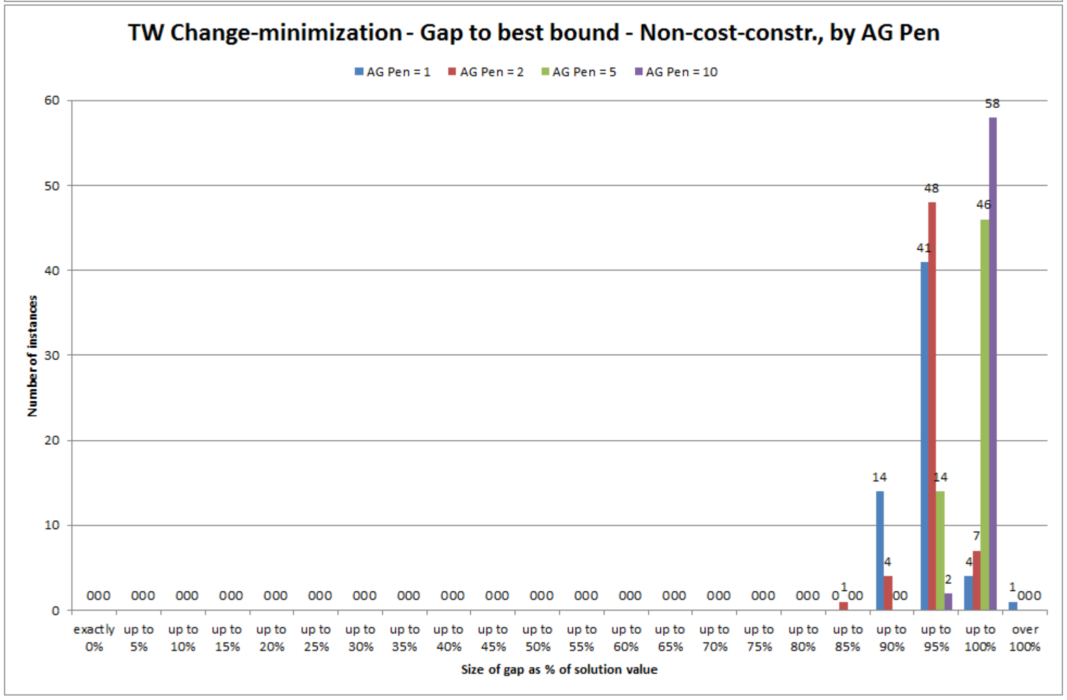
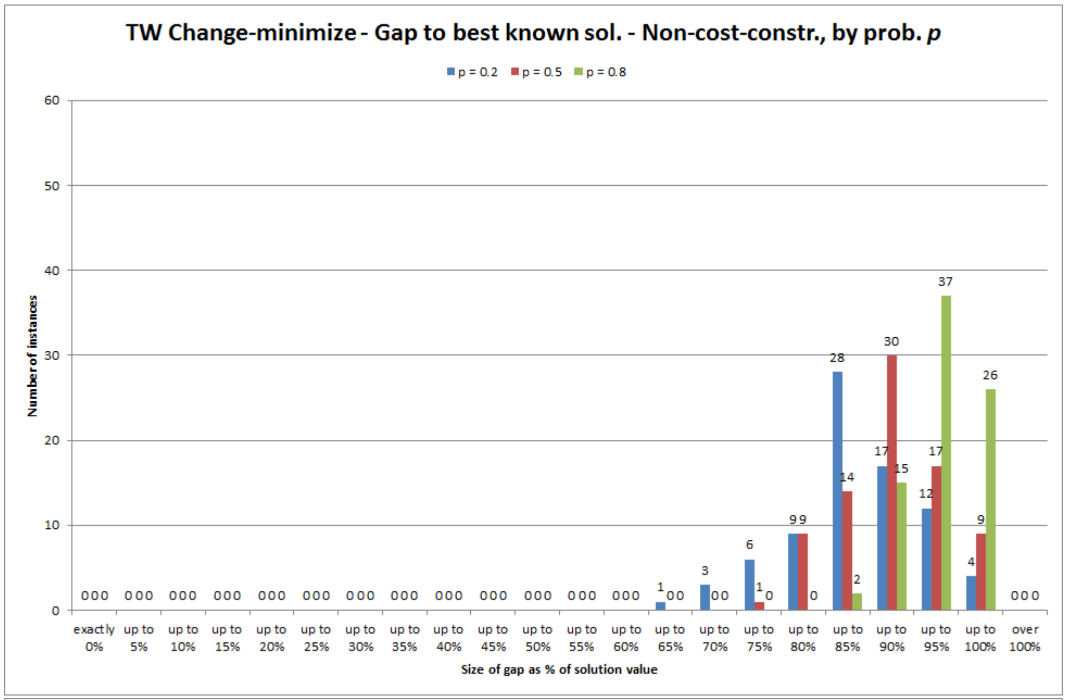


Figure D.75: Gap to best bound for the non-cost-constrained solutions found in two minute Change-minimization test run for the Time-Windows formulation, broken down by values of the long-term disruption penalty K_L (top) and by values of the agency penalty factor K_{AG} (bottom).



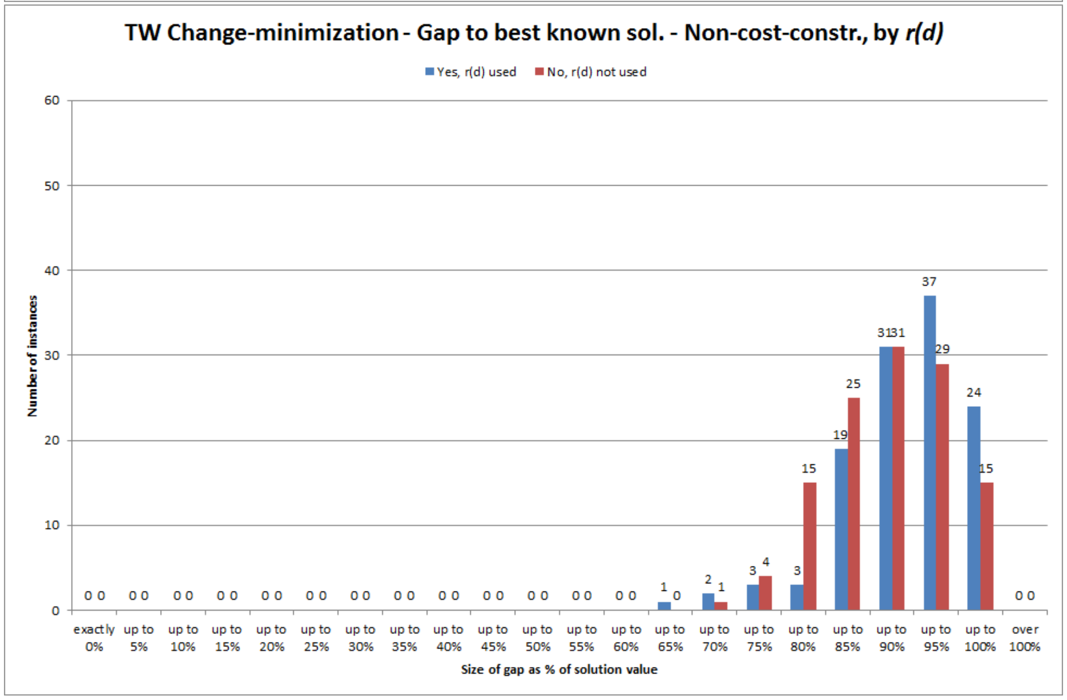


Figure D.76: Gap to best known solution for non-cost-constrained solution found in two minute Change-minimization test run for the Time-Windows formulation, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day) (top) and by whether or not time reduction factor $r(d)$ was used (bottom).

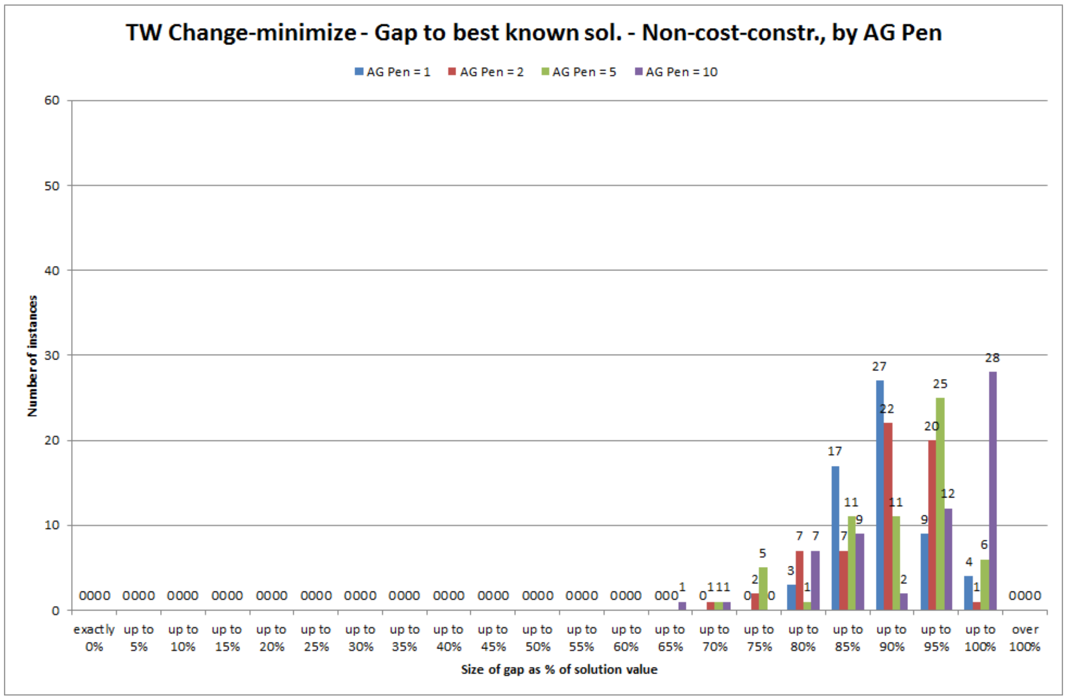


Figure D.77: Gap to best known solution for non-cost-constrained solution found in two minute Change-minimization test run for the Time-Windows formulation, broken down by values of the agency penalty factor K_{AG}.

D.3.2 Breakdown of Task-Based Approximation results

Here we present selected histograms which can aid our understanding of the F-test and Kruskall-Wallis test results given in Table 6.16. These show the same Task-Based Approximation results as in section 6.8.2, but broken down according to the values of the parameters which this table highlights as having a significant influence.

D.3.2.1 Relating to cost gaps

We begin with breakdowns of the gaps to the best known bound and best known solution:

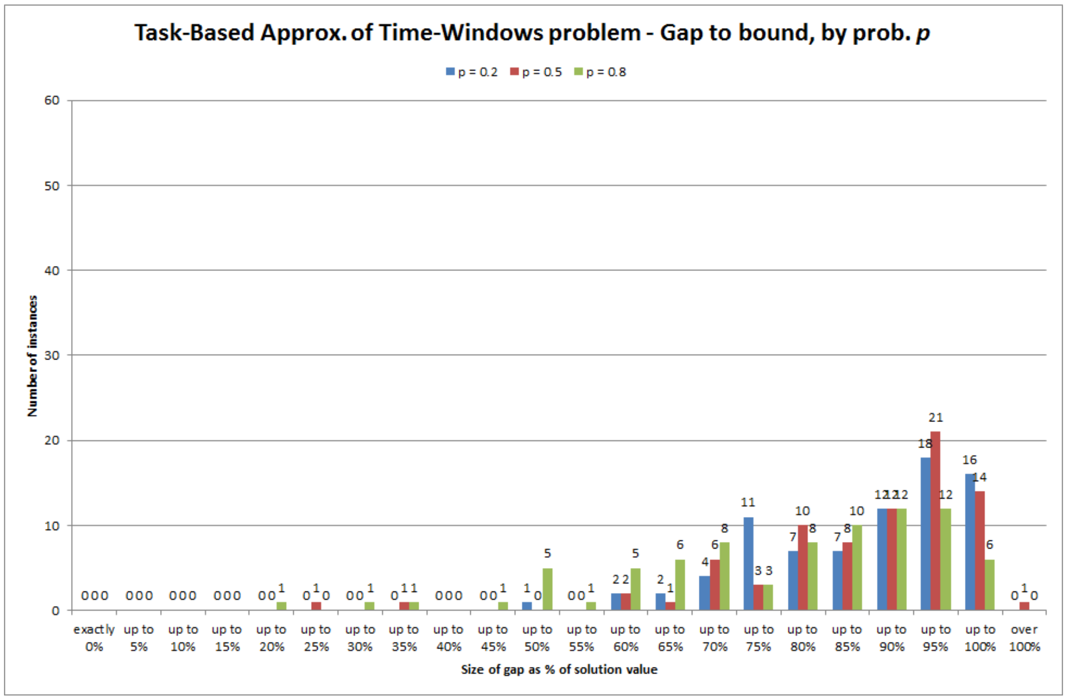


Figure D.78: Histogram of percentage gaps to best known bounds for solutions found using the Task-Based formulation as an approximation, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).

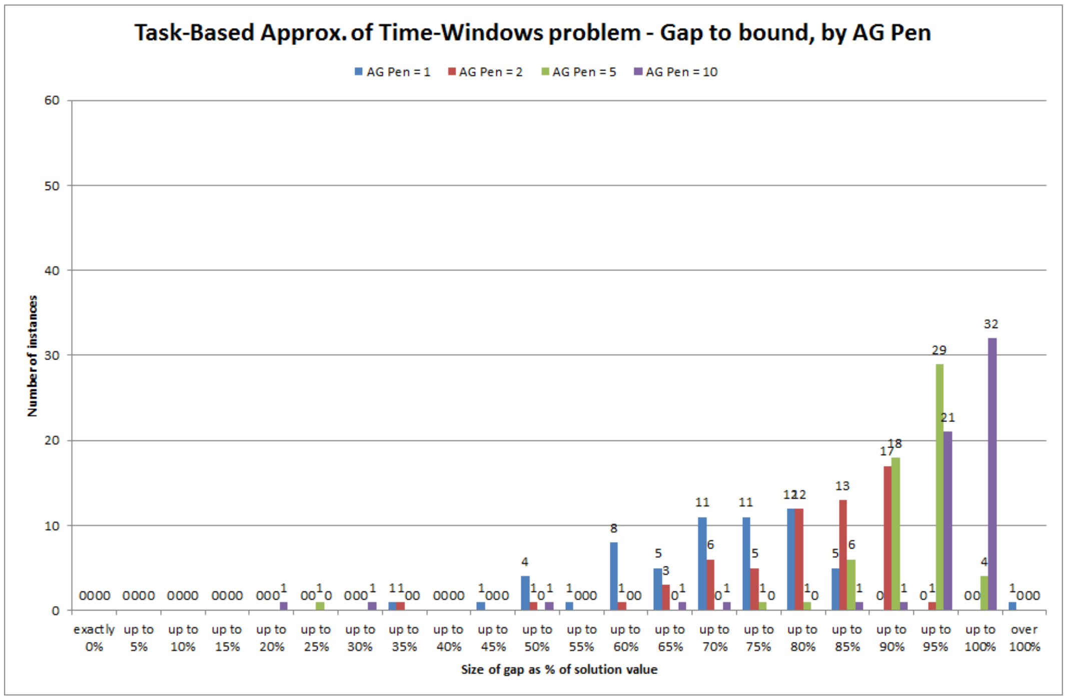
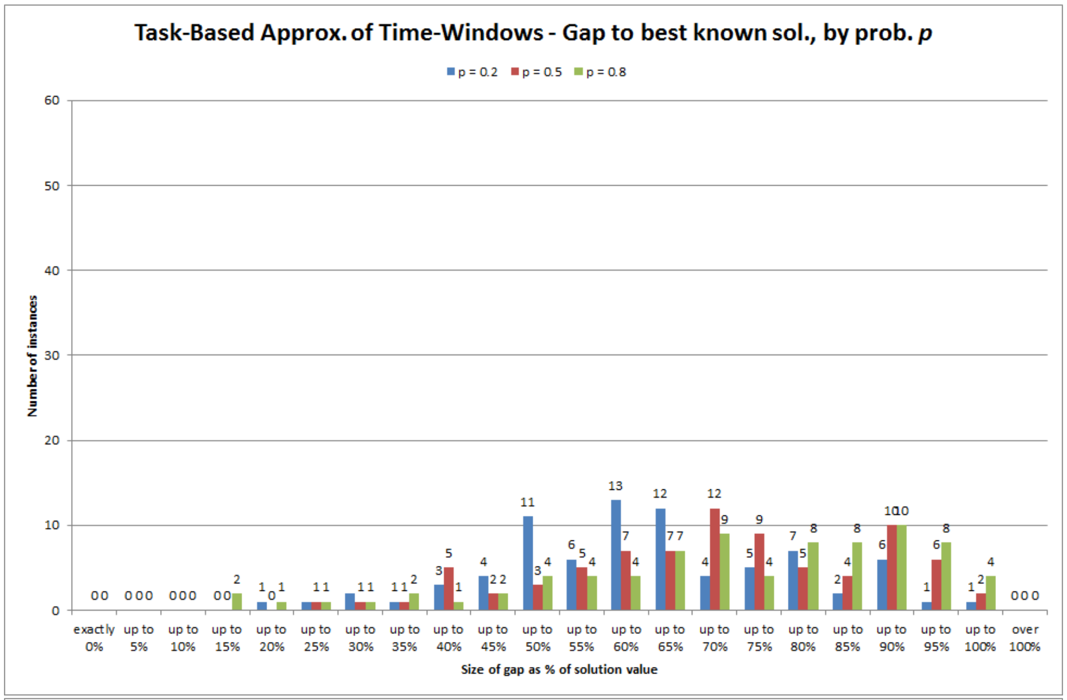


Figure D.79: Histogram of percentage gaps to best known bounds for solutions found using the Task-Based formulation as an approximation, broken down by values of the agency penalty factor K_{AG}.



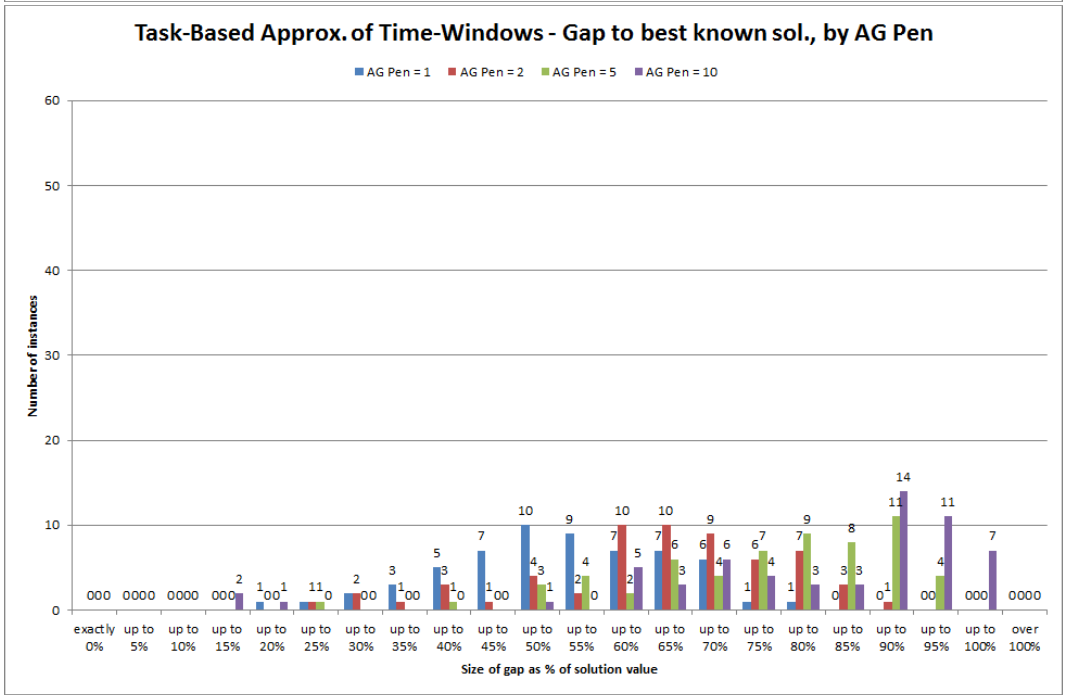


Figure D.80: Histogram of percentage gaps to best known bounds for solutions found using the Task-Based formulation as an approximation, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day) (top) and by values of the agency penalty factor K_{AG} (bottom).

D.3.2.2 Relating to improvements

The following describe breakdowns of the improvement the Task-Based Approximation makes on the direct and change-minimization approaches:

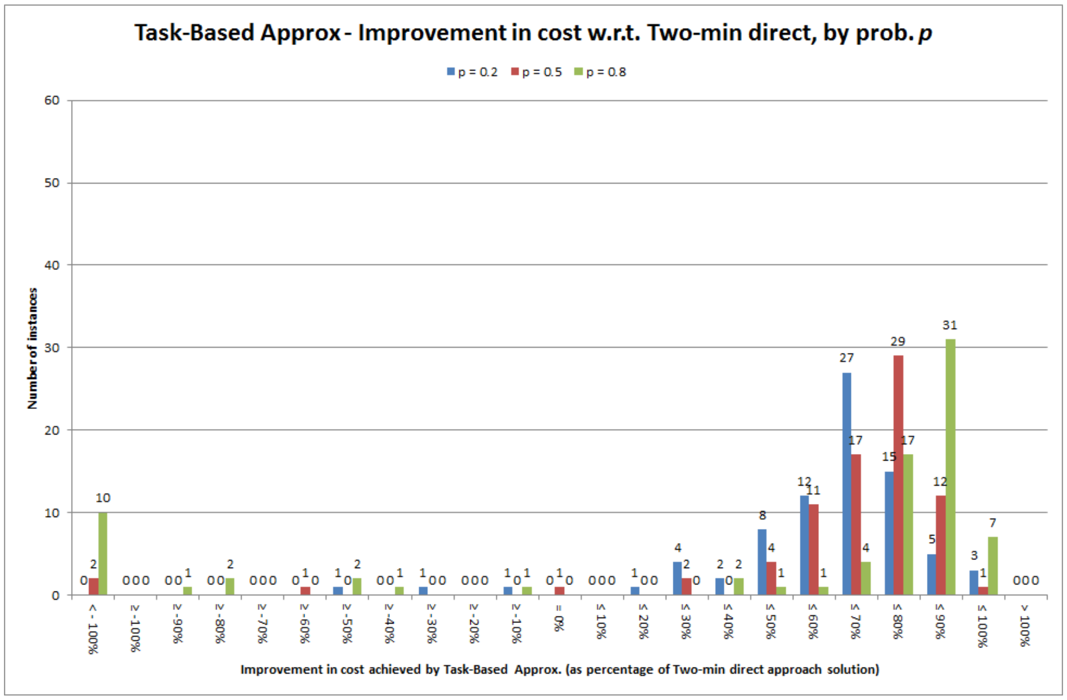
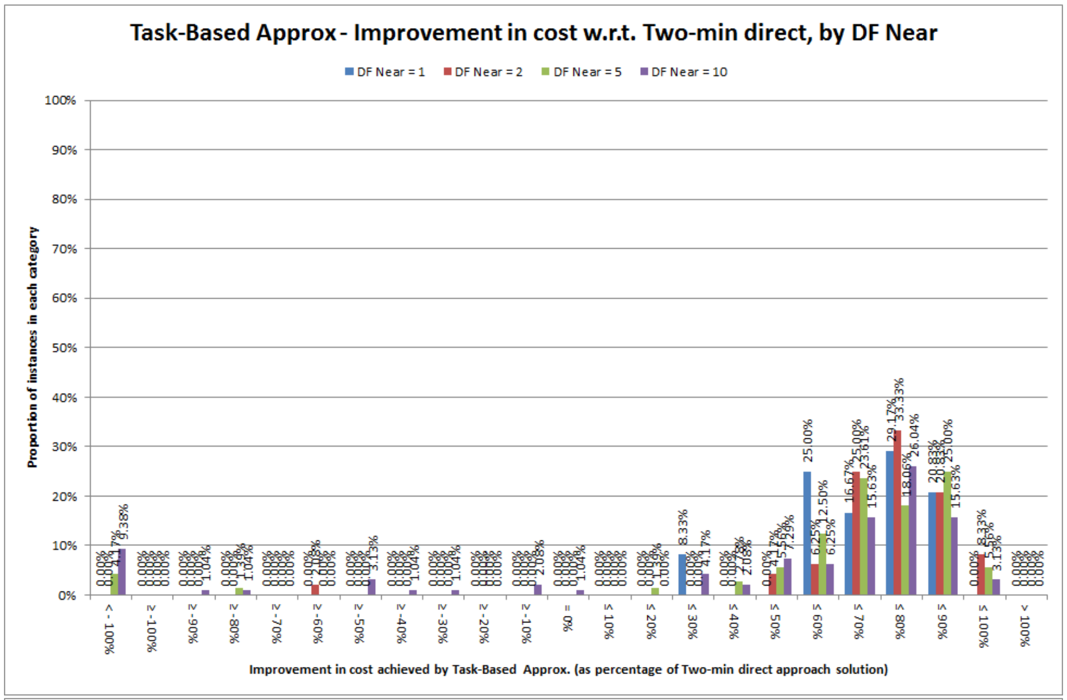


Figure D.81: Histogram summarising improvement in the solution for each instance using the Task-Based Approximation approach compared to the solution cost for the two-minute 'direct' approach, shown as a percentage of the two-minute direct approach solution value, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



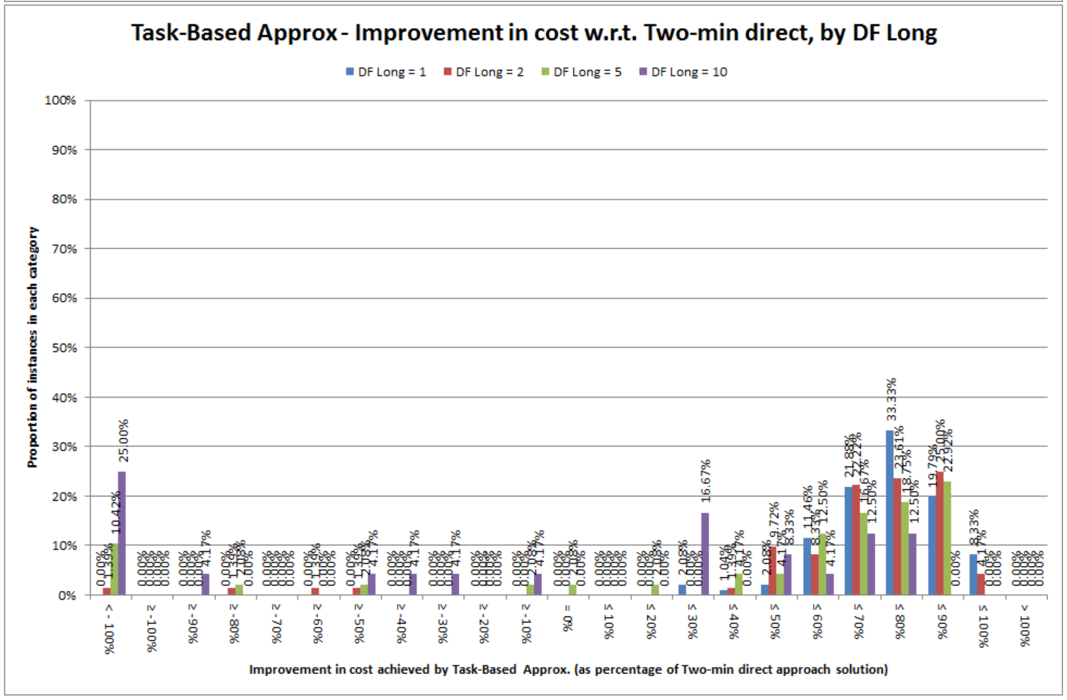


Figure D.82: Histogram summarising improvement in the solution for each instance using the Task-Based Approximation approach compared to the solution cost for the two-minute ‘direct’ approach, shown as a percentage of the two-minute direct approach solution value, broken down by values of the near-term disruption penalty K_N (top) and by values of the long-term disruption penalty K_L (bottom).

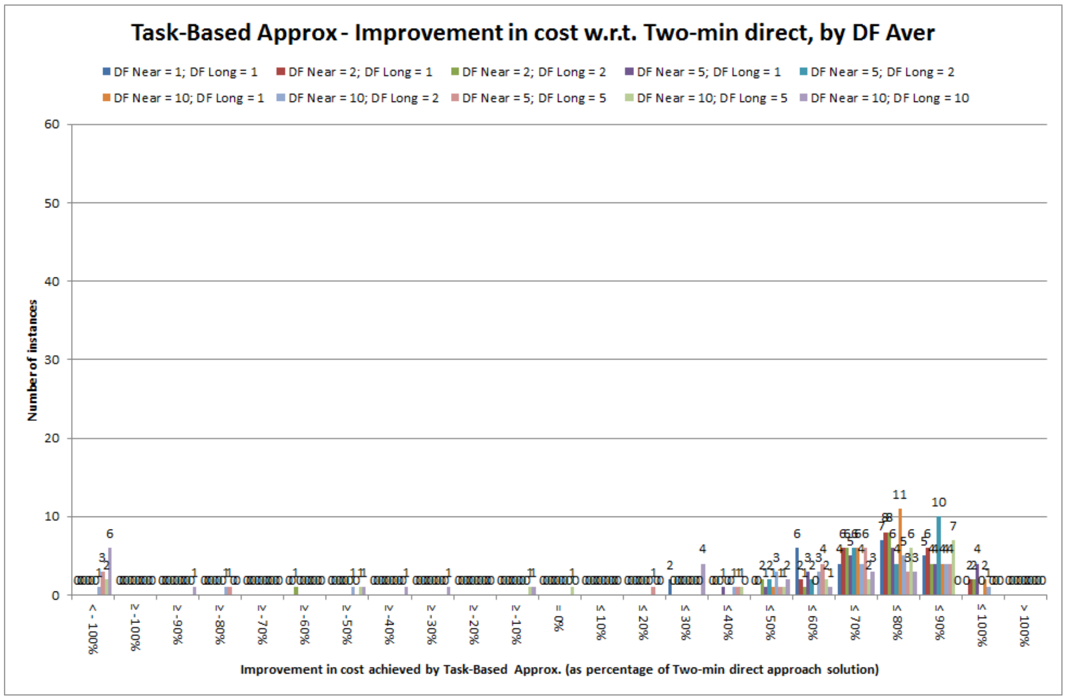
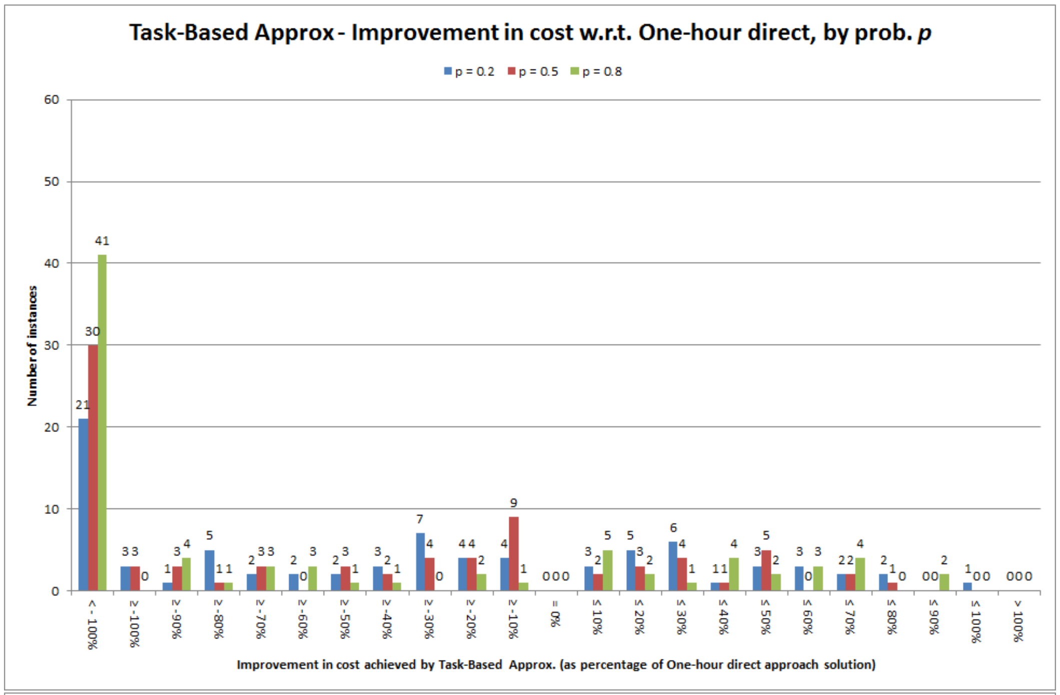


Figure D.83: Histogram summarising improvement in the solution for each instance using the Task-Based Approximation approach compared to the solution cost for the two-minute 'direct' approach, shown as a percentage of the two-minute direct approach solution value, broken down by values of the weighted average disruption penalty $\hat{K}$.



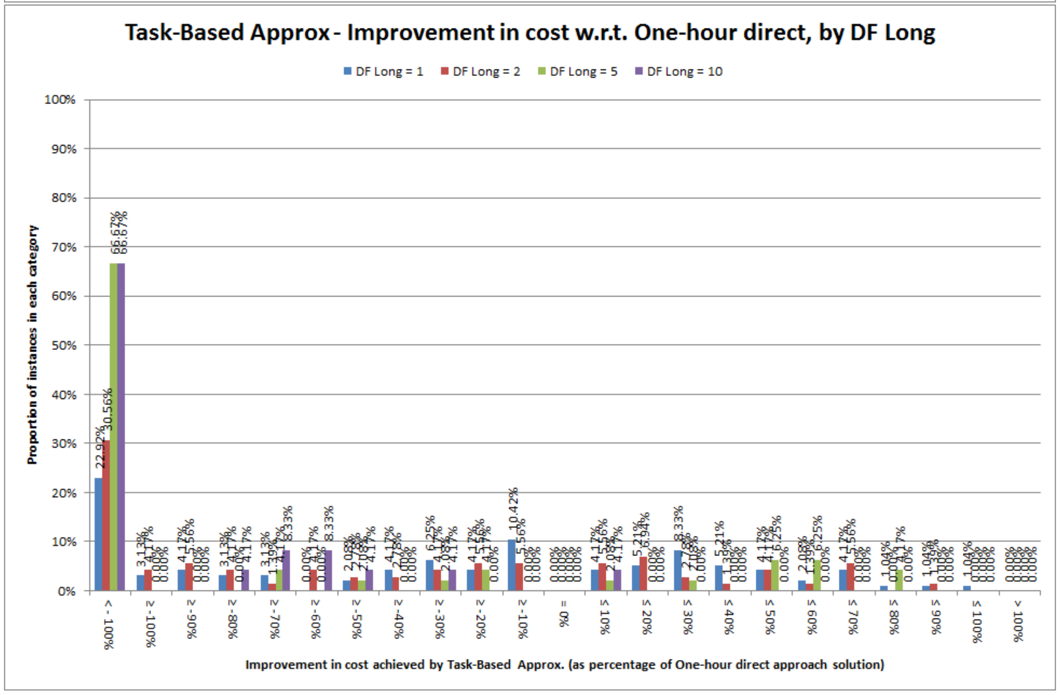
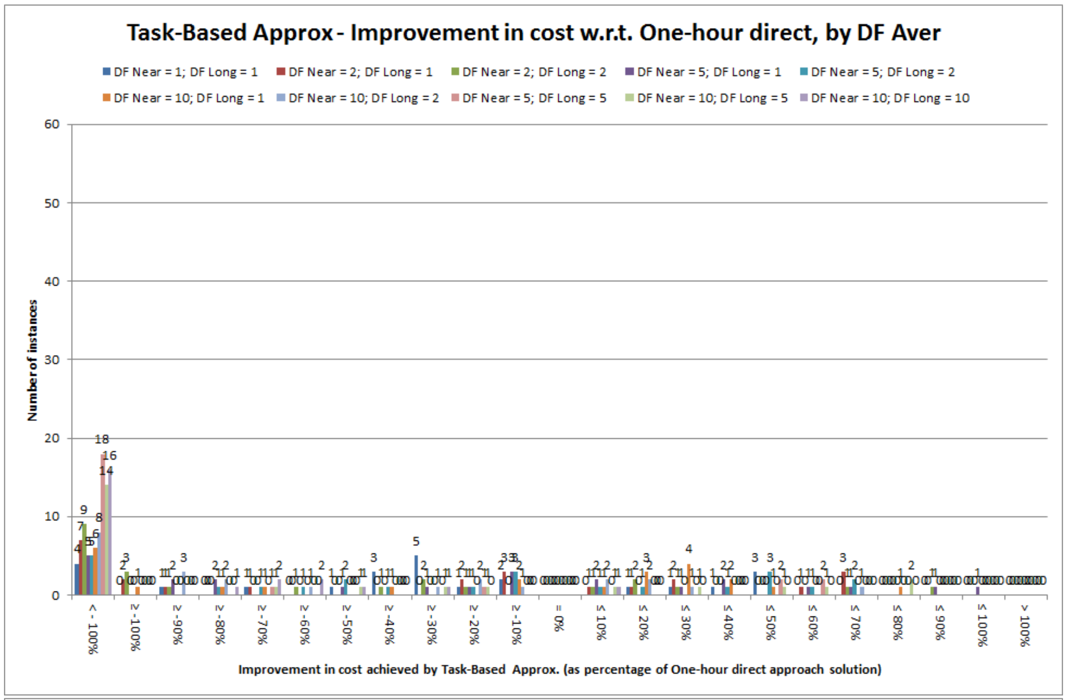


Figure D.84: Histograms summarising improvement in the solution for each instance using the Task-Based Approximation approach compared to the solution cost for the one-hour ‘direct’ approach, shown as a percentage of the one-hour direct approach solution value, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day) (top) and by values of the long-term disruption penalty K_L (bottom).



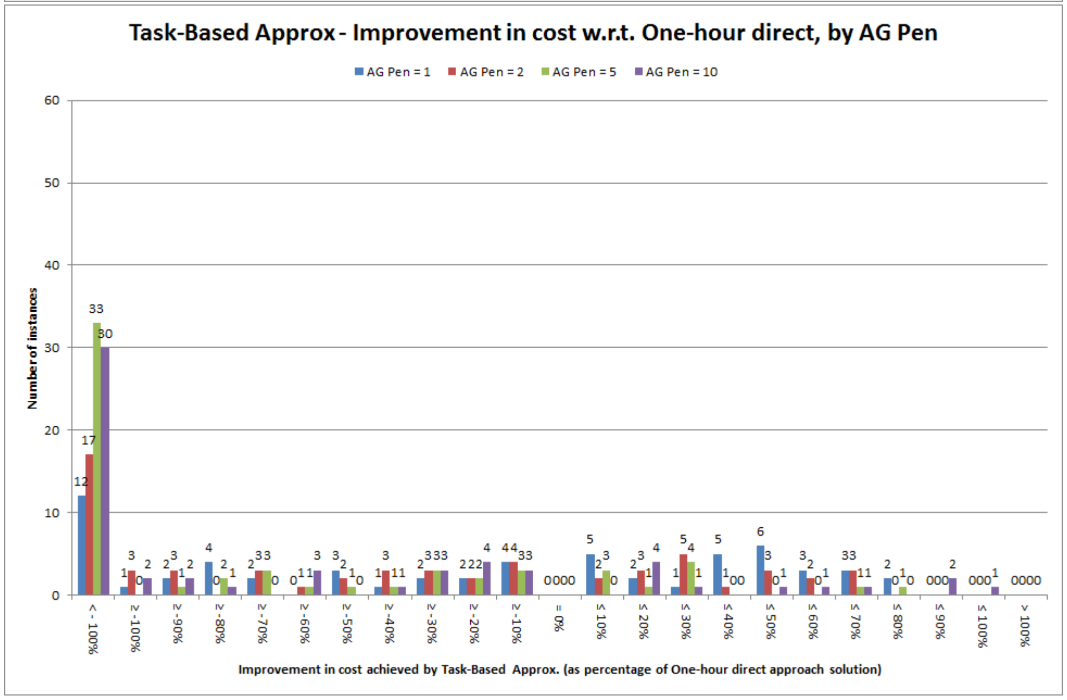
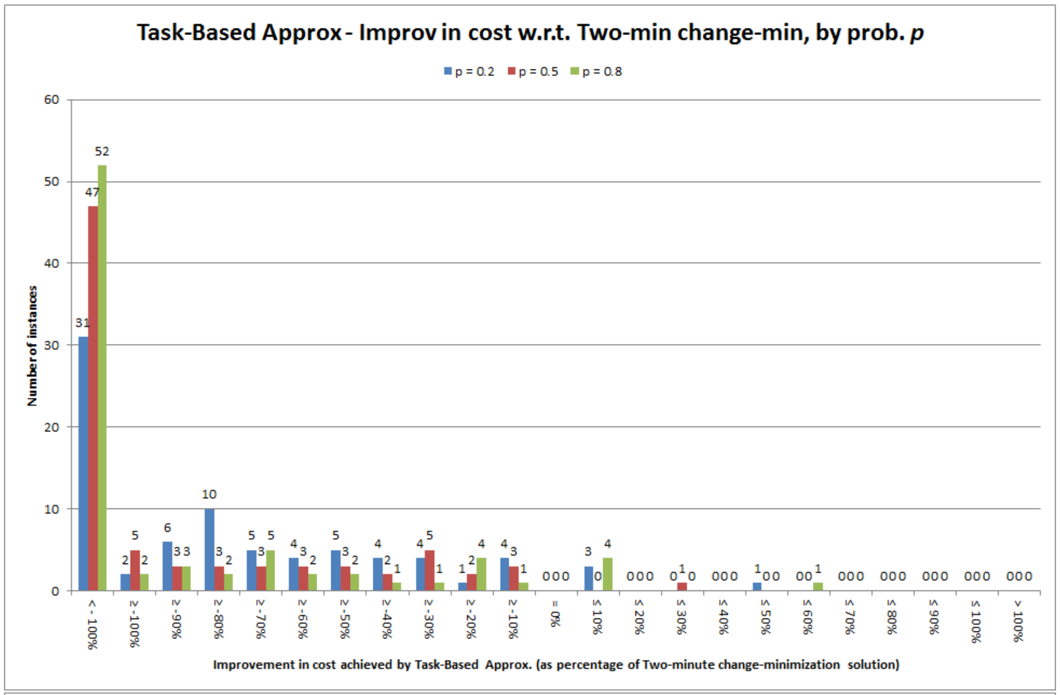


Figure D.85: Histograms summarising improvement in the solution for each instance using the Task-Based Approximation approach compared to the solution cost for the one-hour 'direct' approach, shown as a percentage of the one-hour direct approach solution value, broken down by values of the weighted average disruption penalty $\hat{K}$ (top) and by values of the agency penalty factor K_{AG} (bottom).



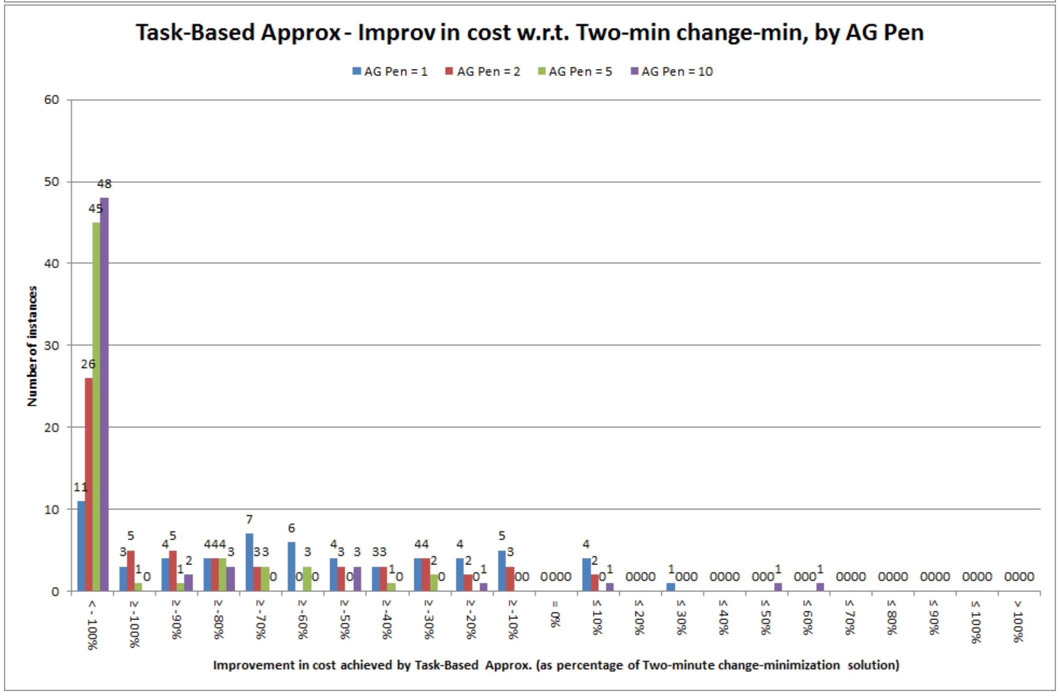


Figure D.86: Histogram summarising improvement in the solution for each instance using the Task-Based Approximation approach compared to the solution cost for the two-minute change-minimization approach, shown as a percentage of the change-minimization solution value, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day) (top) and by values of the agency penalty factor K_{AG} (bottom).

D.3.2.3 Relating to running time

The following describe breakdowns of the running time of the Task-Based Approximation approach:

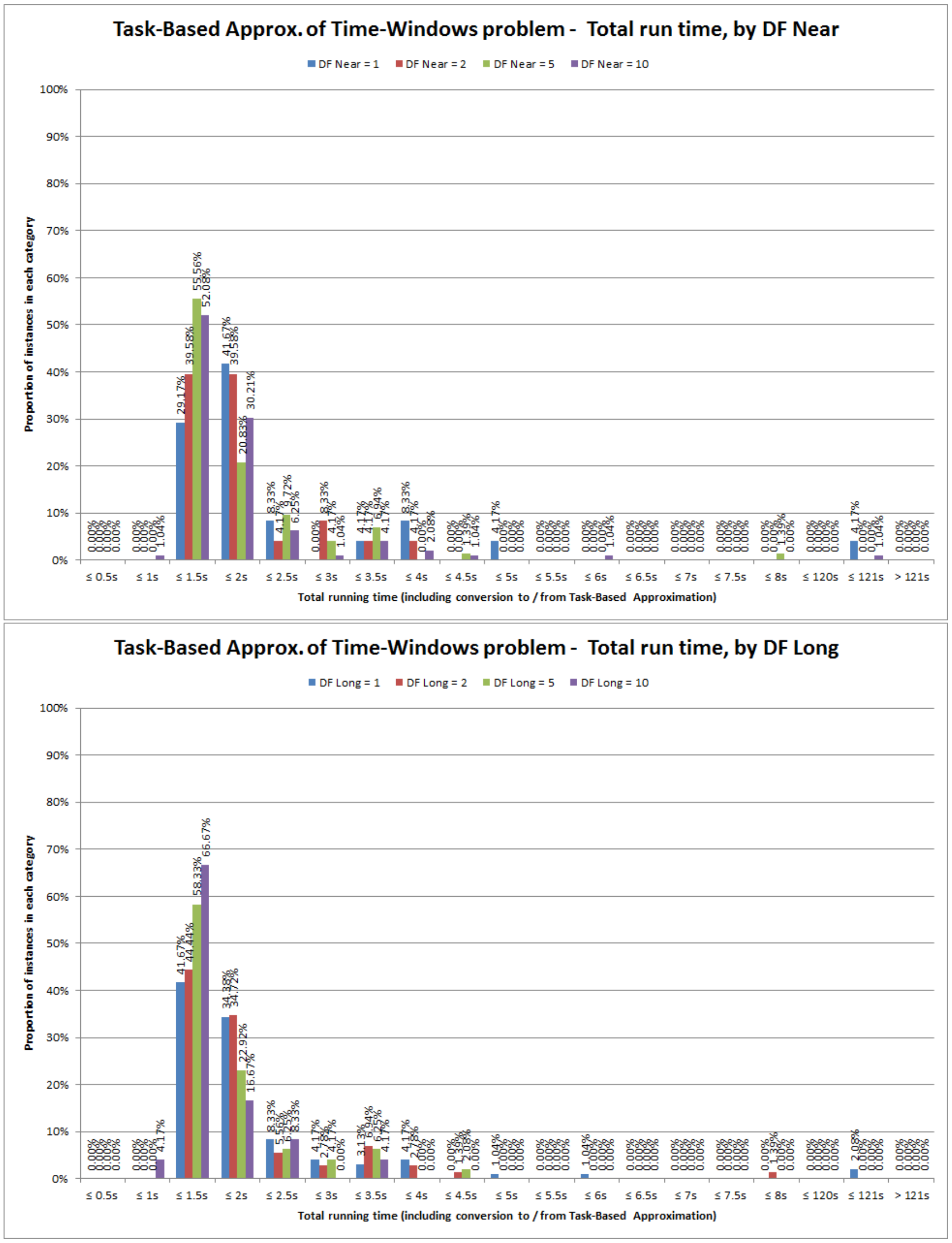


Figure D.87: Histogram summarising total running time to find a Task-Based approximation solution for each instance, including conversion time, broken down by values of the near-term disruption penalty K_N (top) and by values of the long-term disruption penalty K_L (bottom).

D.3.2.4 Relating to number of changes

The following describe breakdowns of the number of changes in the Task-Based Approximation solutions:

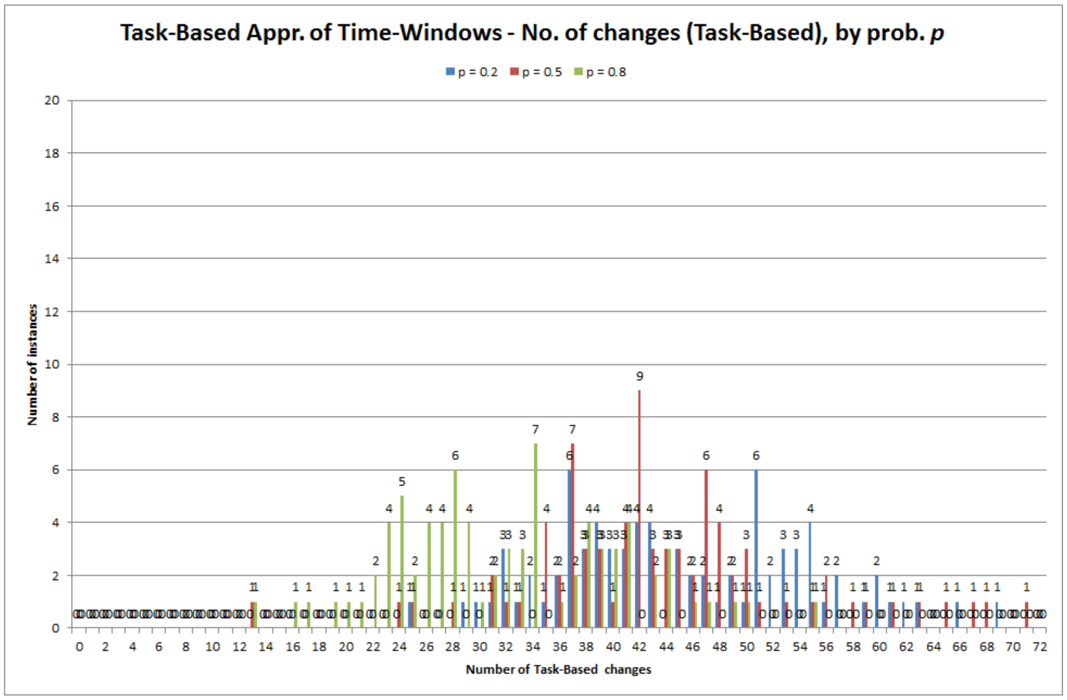


Figure D.88: Histograms summarising the number of changes in the Task-Based representation of the solution for each instance, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).

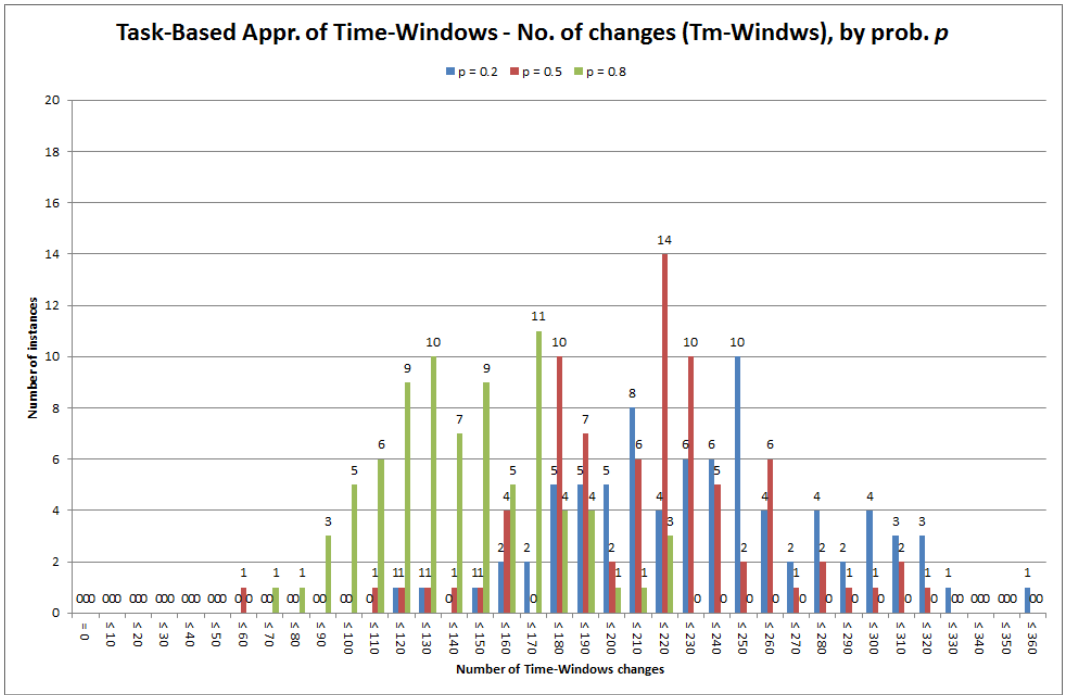


Figure D.89: Histograms summarising the number of changes in the Time-Windows representation of the solution for each instance, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).

D.3.3 Breakdown of Heuristic initial solution results

Here we present selected histograms which can aid our understanding of the Kruskall-Wallis test results given in Table 6.18. These show the same Heuristic initial solution results as in section 6.8.3, but broken down according to the values of the parameters which this table highlights as having a significant influence.

D.3.3.1 Relating to cost gaps

We begin with breakdowns of the gaps to the best known bound and best known solution:

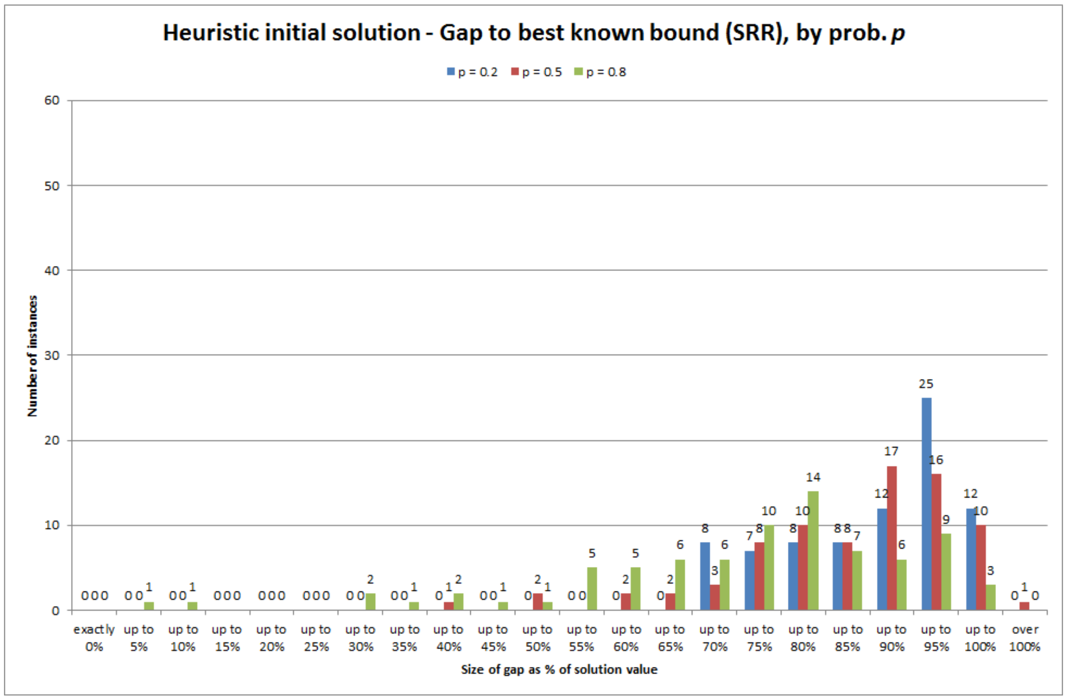
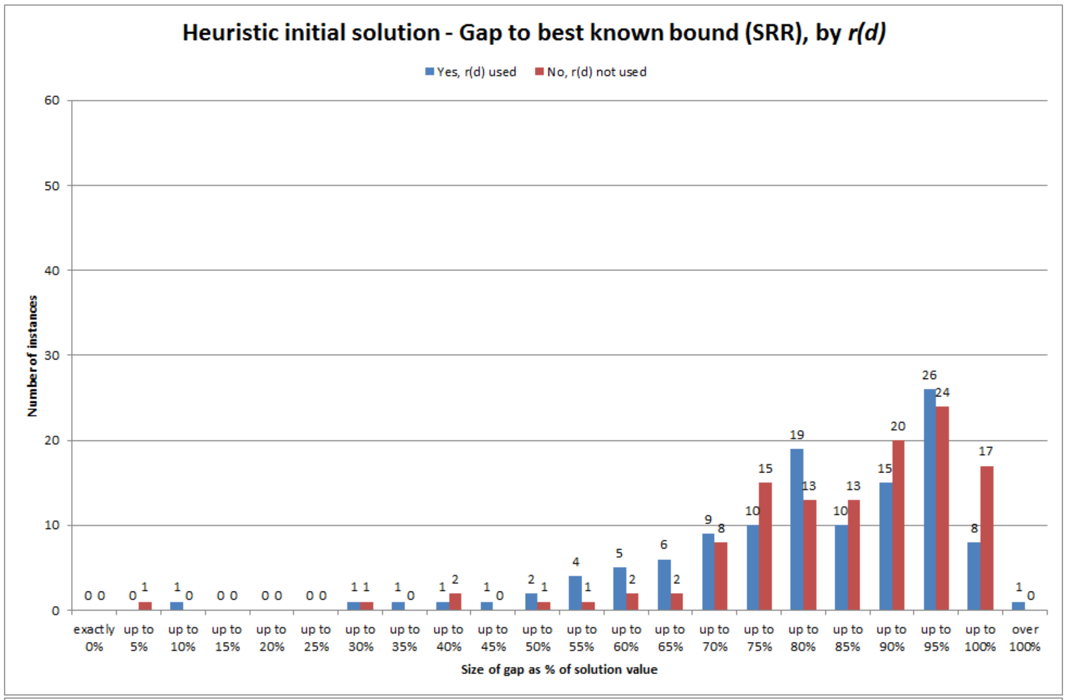


Figure D.90: Histogram of percentage gaps to best known bounds for solutions found using the Heuristic initial solution approach, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



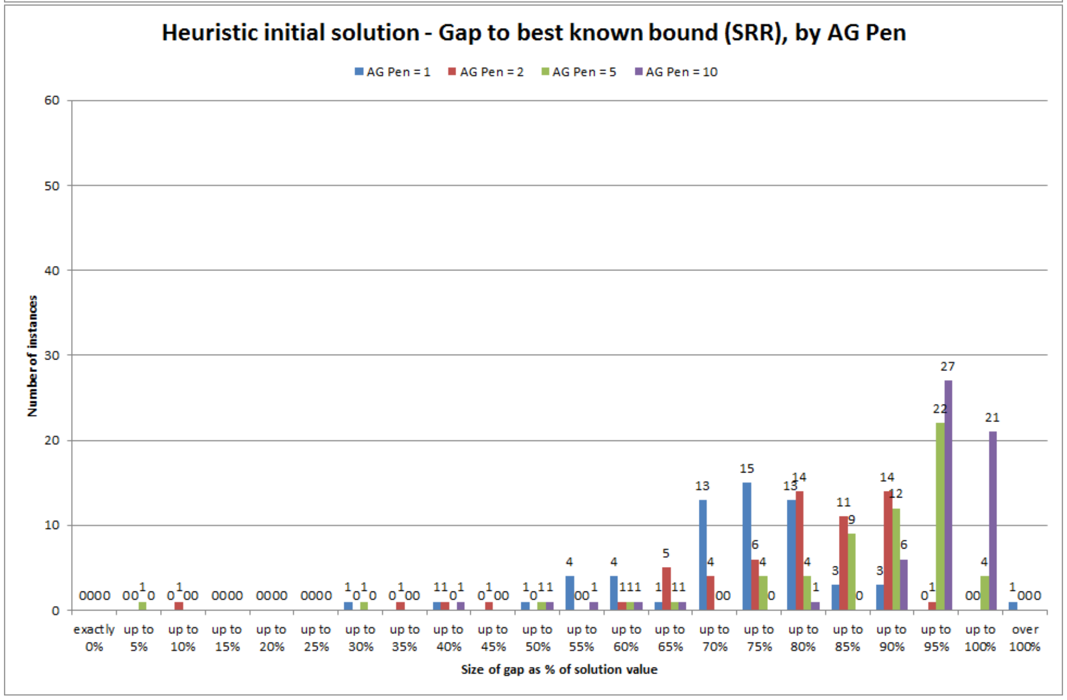


Figure D.91: Histogram of percentage gaps to best known bounds for solutions found using the Heuristic initial solution approach, broken down by whether or not time reduction factor $r(d)$ was used (top) and by values of the agency penalty factor K_{AG} (bottom).

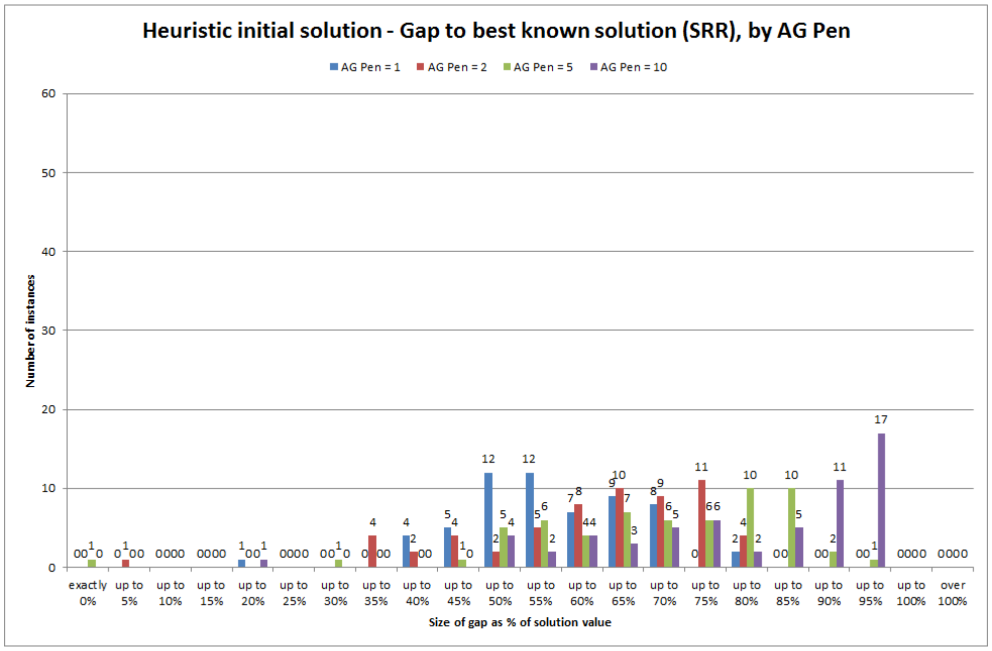


Figure D.92: Histogram of percentage gaps to best known solution for solutions found using the Heuristic initial solution approach, broken down by values of the agency penalty factor K_{AG}.

D.3.3.2 Relating to improvements

The following describe breakdowns of the improvement the Heuristic initial solution makes on the direct and change-minimization approaches:

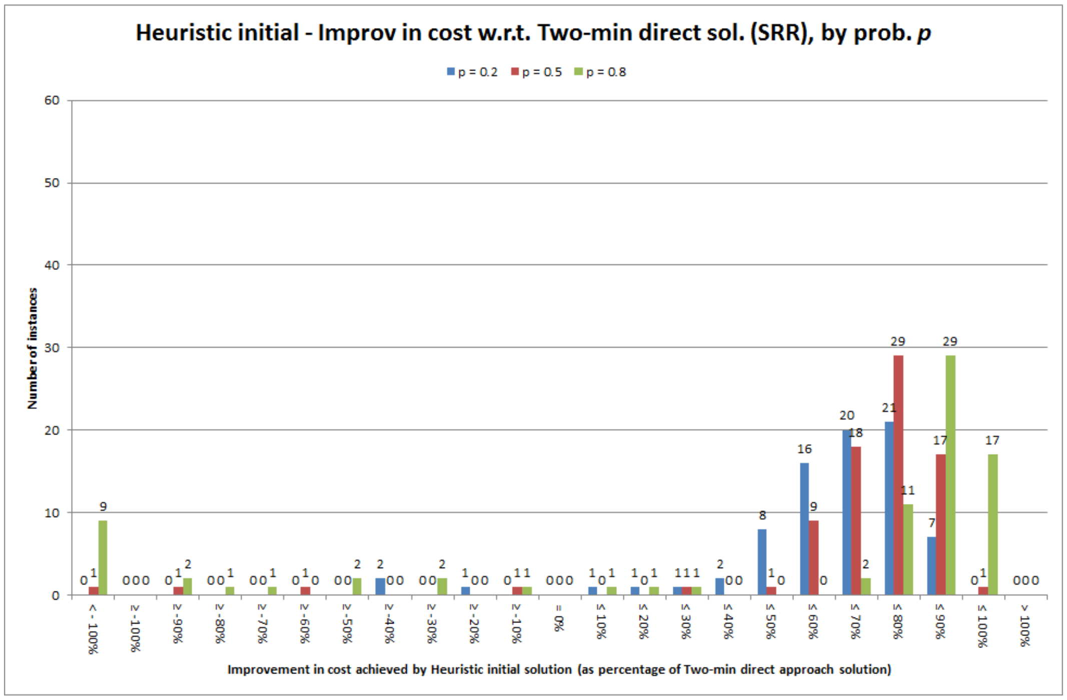
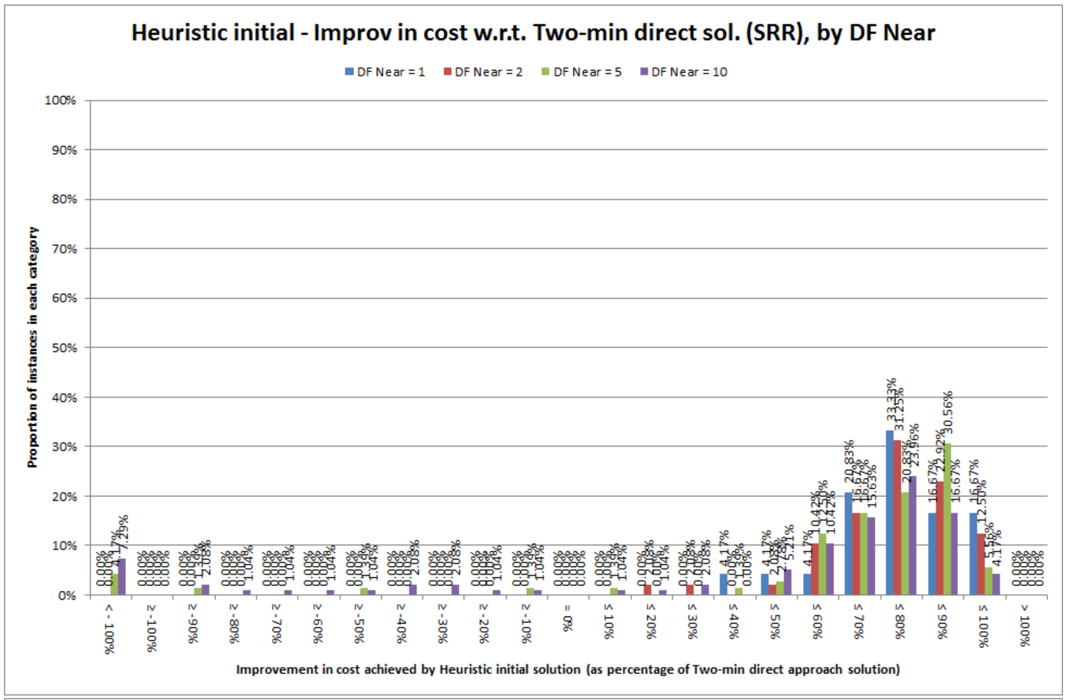


Figure D.93: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the two-minute 'direct' approach, shown as a percentage of the two-minute direct approach solution value, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



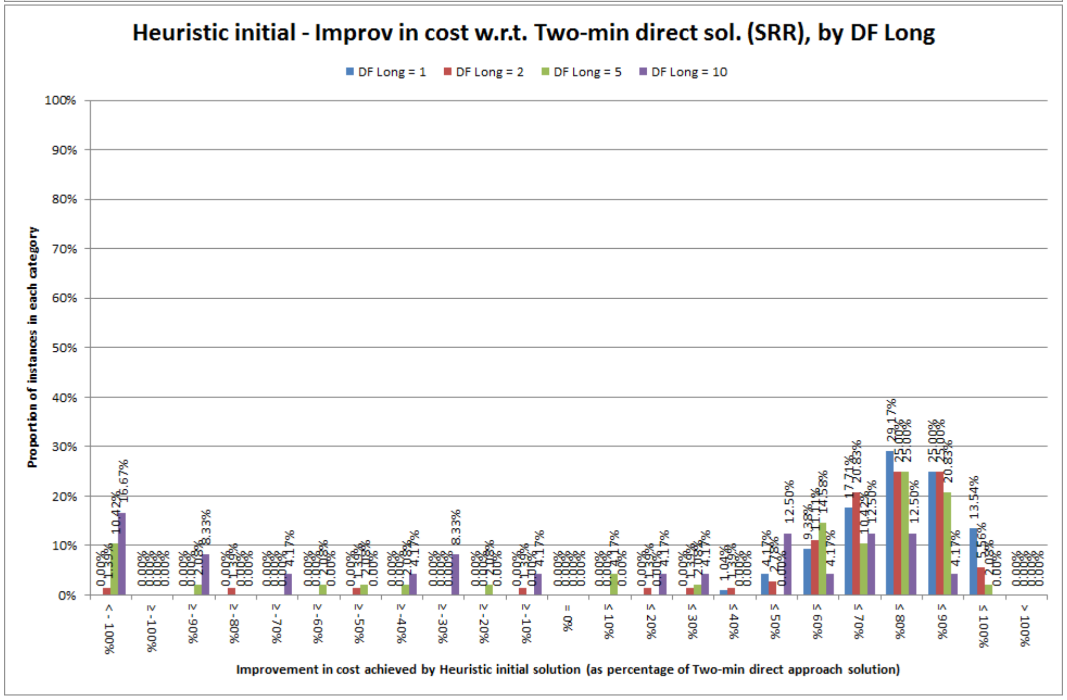


Figure D.94: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the two-minute 'direct' approach, shown as a percentage of the two-minute direct approach solution value, broken down by values of the near-term disruption penalty K_N (top) and by values of the long-term disruption penalty K_L (bottom).

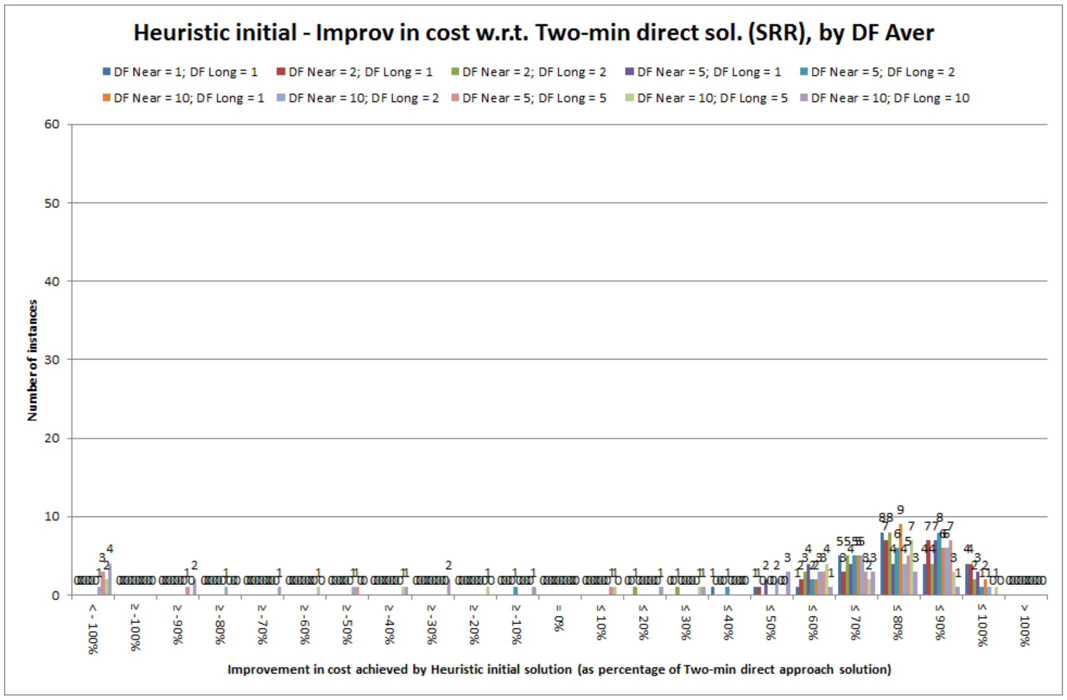
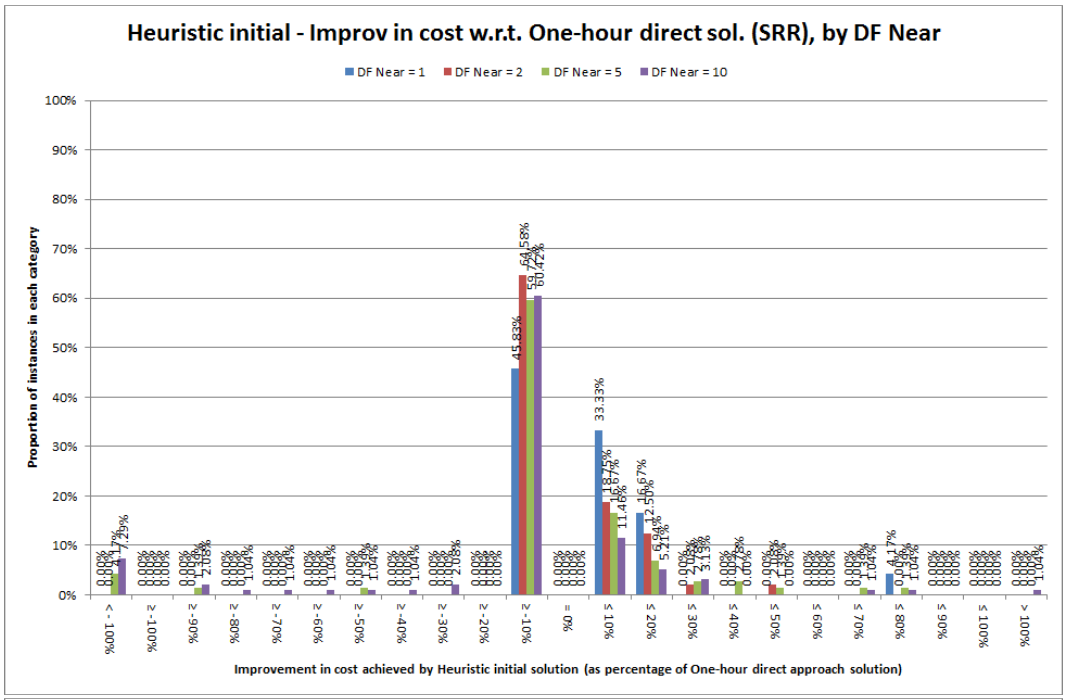


Figure D.95: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the two-minute ‘direct’ approach, shown as a percentage of the two-minute direct approach solution value, broken down by values of the weighted average disruption penalty $\hat{K}$.



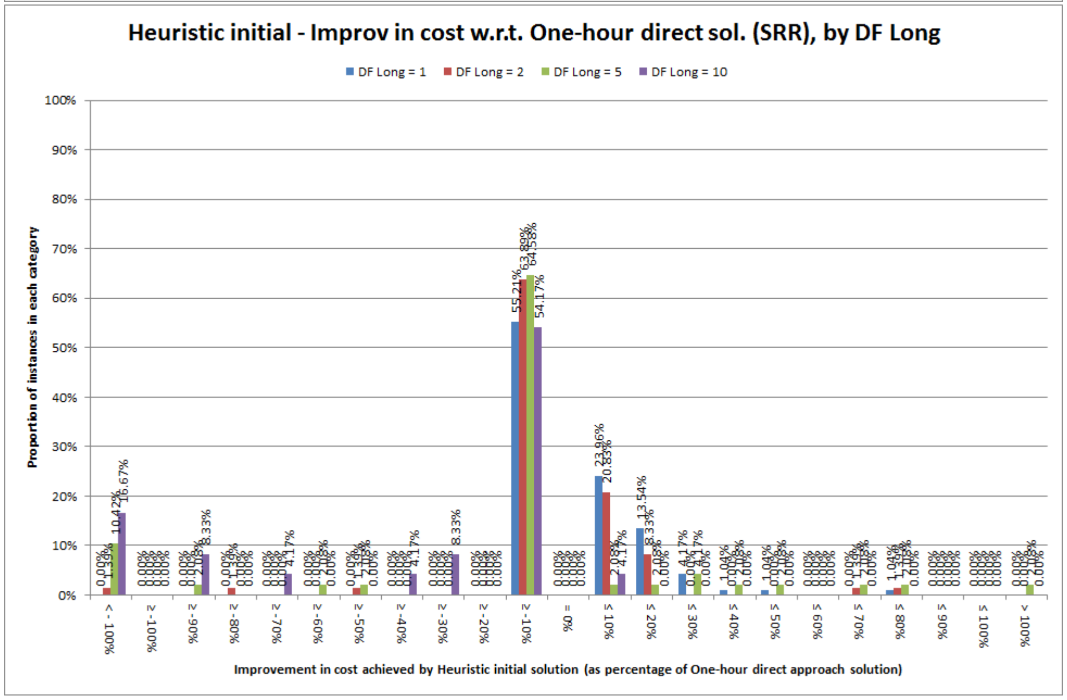
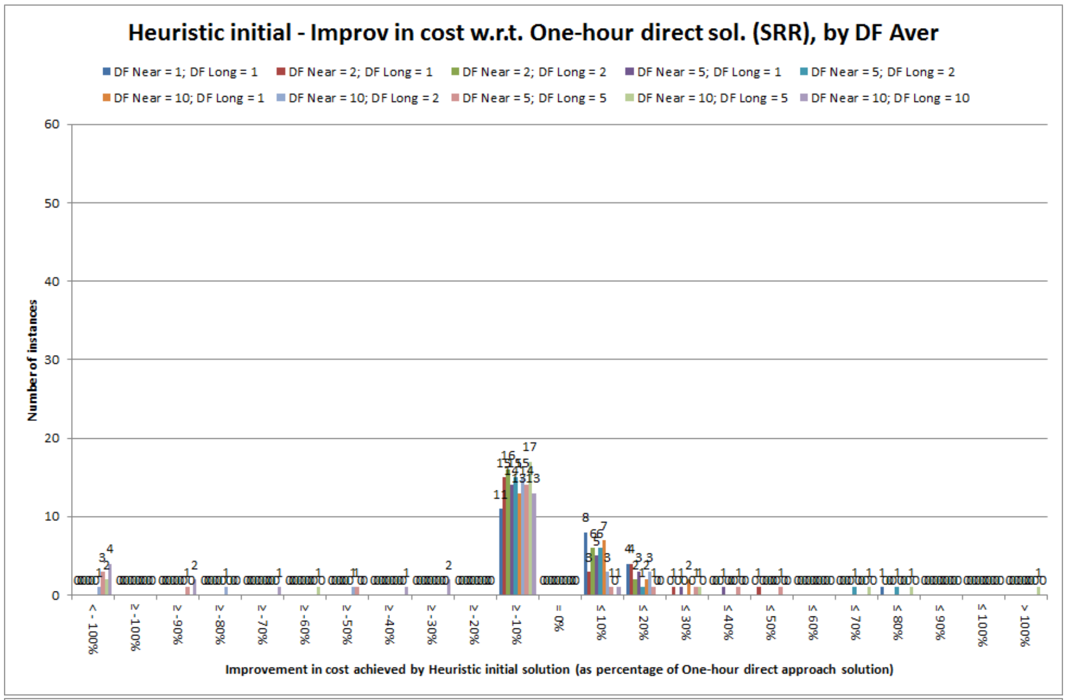


Figure D.96: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the one-hour 'direct' approach, shown as a percentage of the one-hour direct approach solution value, broken down by values of the near-term disruption penalty K_N (top) and by values of the long-term disruption penalty K_L (bottom).



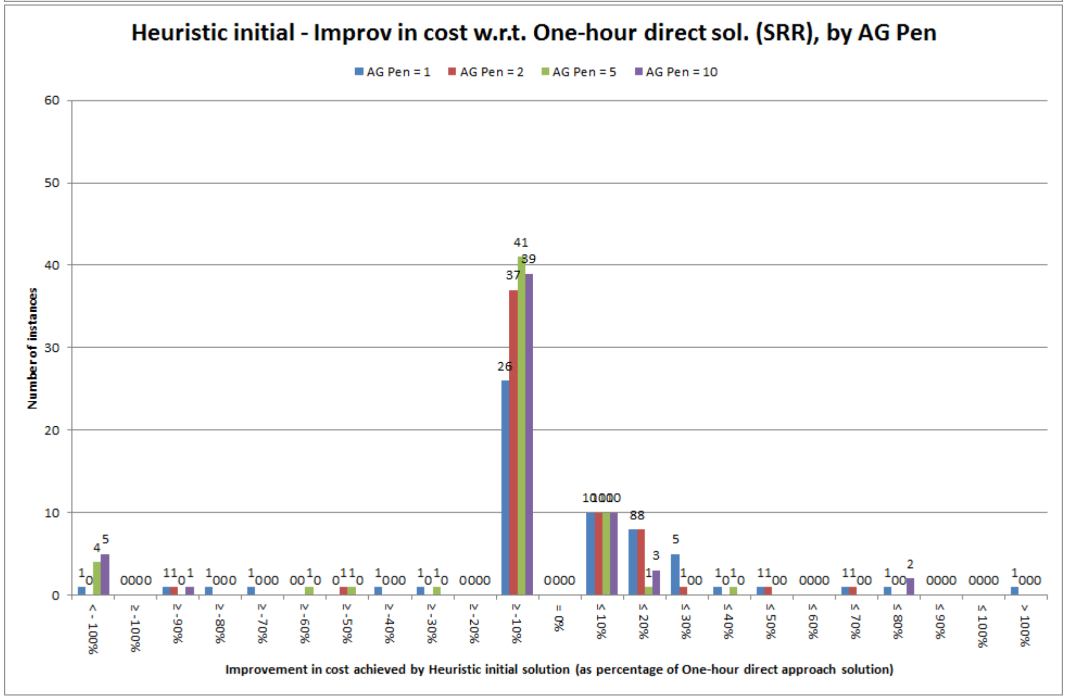


Figure D.97: Histograms summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the one-hour ‘direct’ approach, shown as a percentage of the one-hour direct approach solution value, broken down by values of the weighted average disruption penalty $\hat{K}$ (top) and by values of the agency penalty factor K_{AG} (bottom).

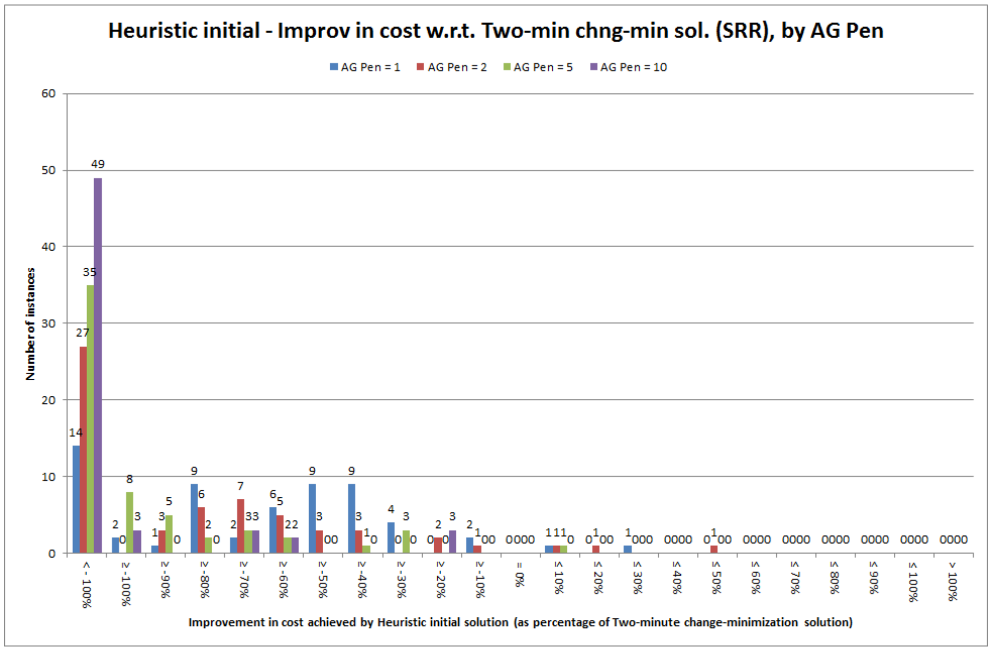
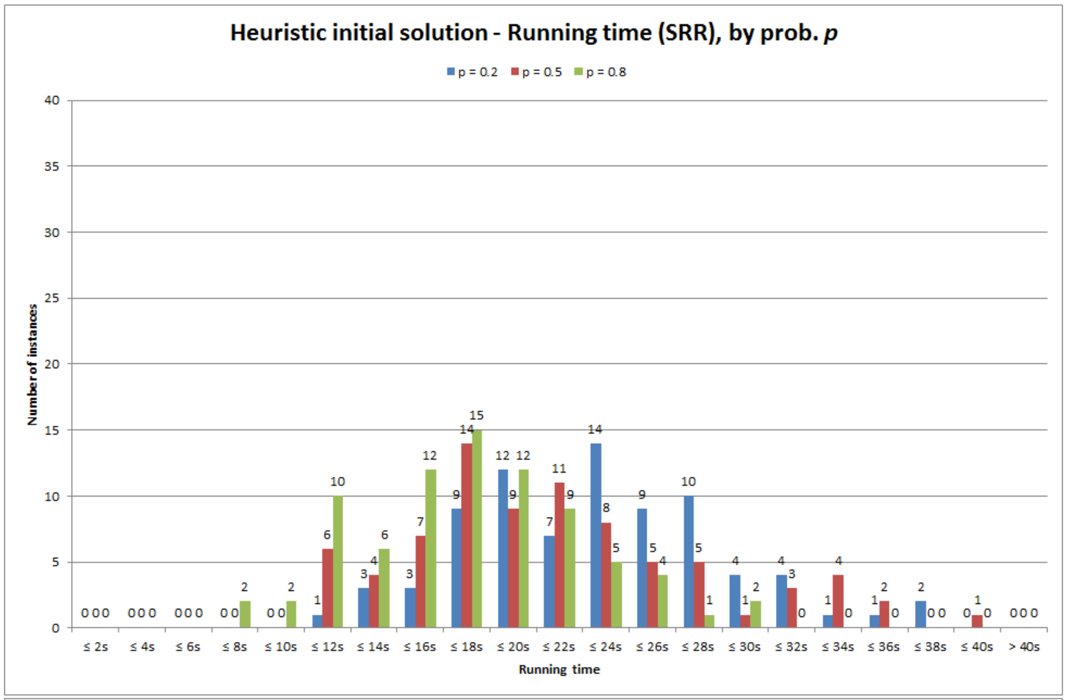


Figure D.98: Histogram summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the two-minute change-minimization approach, shown as a percentage of the change-minimization solution value, broken down by values of the agency penalty factor K_{AG}.

D.3.3.3 Relating to running time

The following describe breakdowns of the running time of the Heuristic initial solution approach:



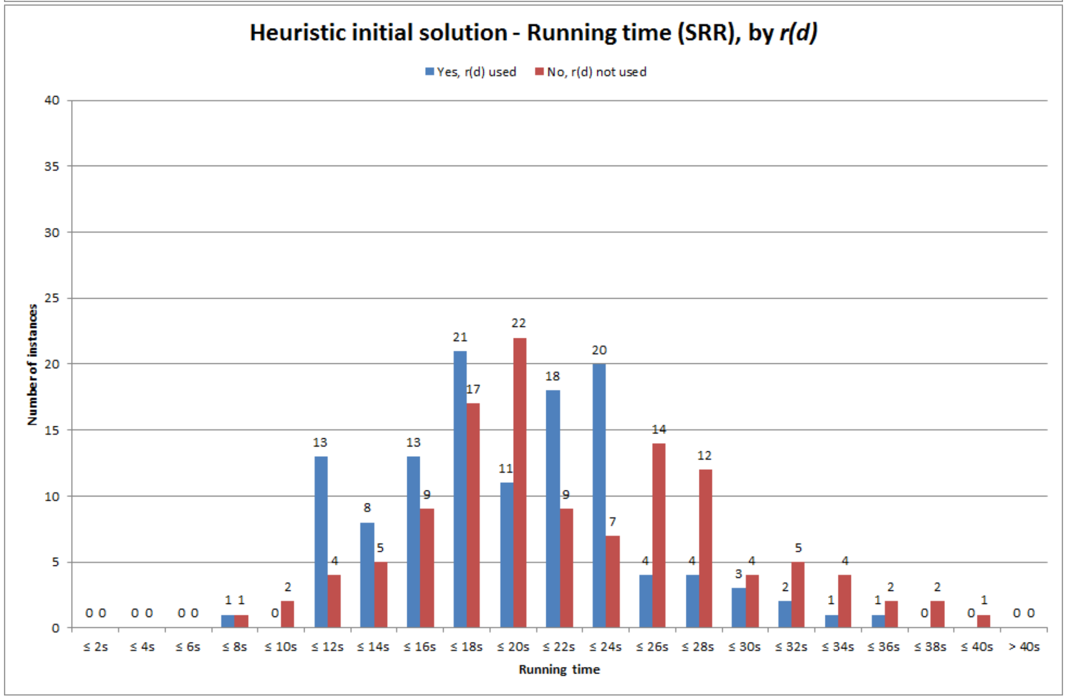


Figure D.99: Histogram summarising total running time of the Heuristic initial solution algorithm for each instance, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day) (top) and by whether or not time reduction factor $r(d)$ was used (bottom).

D.3.3.4 Relating to numbers of iterations and changes

The following describe breakdowns of the number of iterations and number of changes in the Heuristic initial solutions:

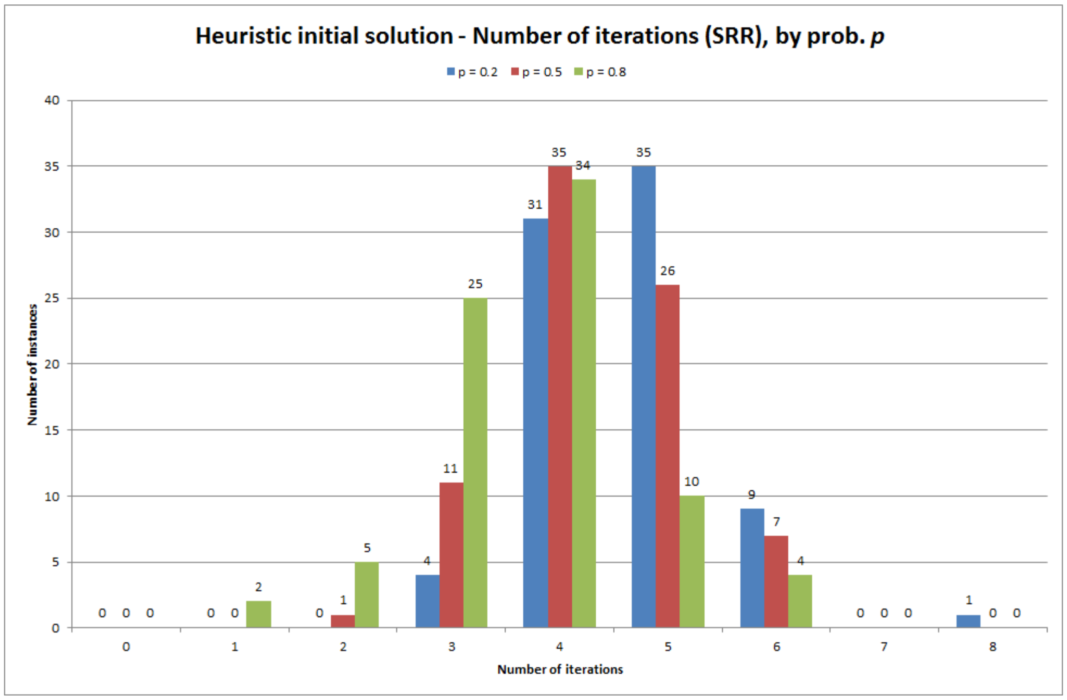
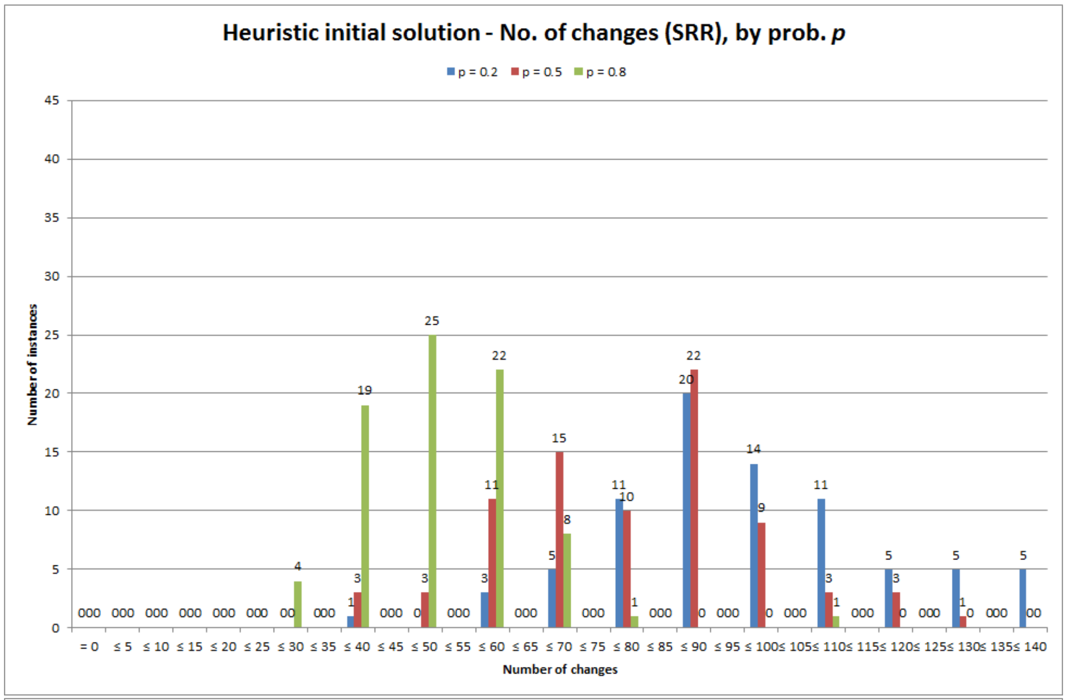


Figure D.100: Histogram summarising the number of iterations carried out by the Heuristic initial solution algorithm for each instance, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



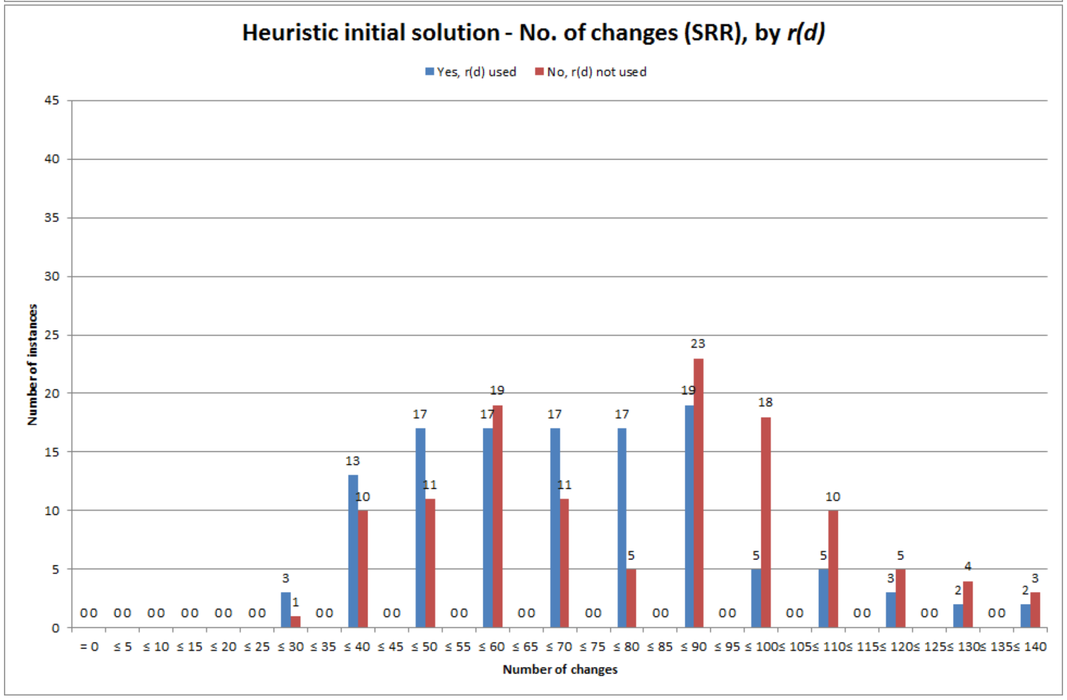


Figure D.101: Histogram summarising the number of changes in the Heuristic initial solution for each instance, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day) (top) and by whether or not time reduction factor $r(d)$ was used (bottom).

D.3.3.5 Relating to improvement on the Task-Based Approximation

The following describe breakdowns of the improvement the Heuristic initial solution makes on the Task-Based Approximation approach:

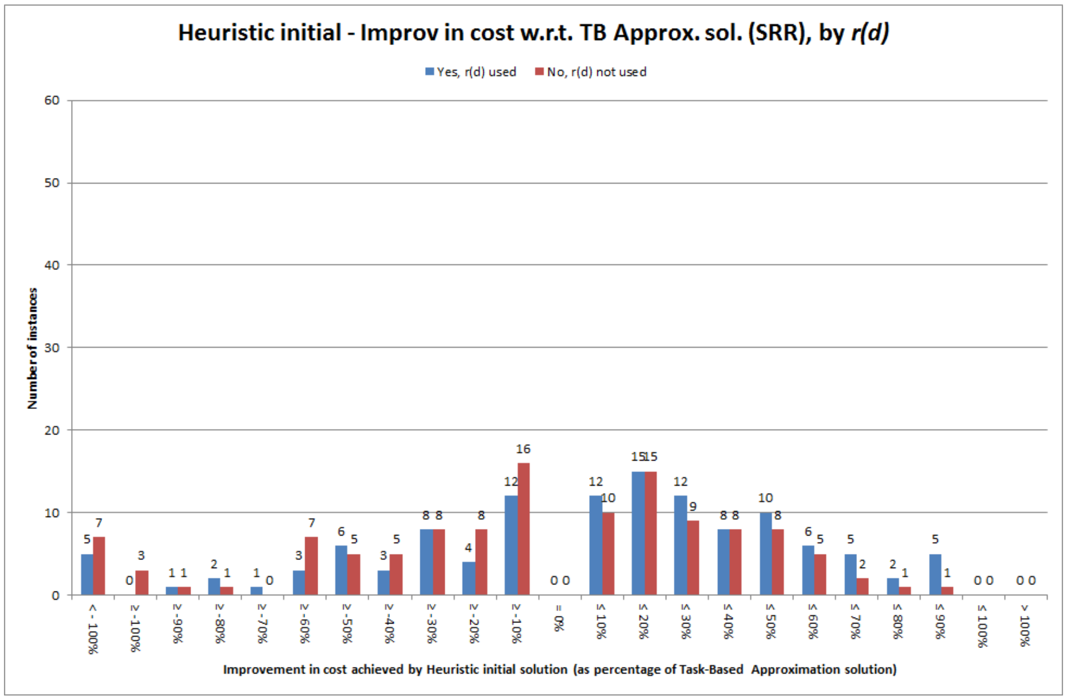


Figure D.102: Histogram summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the Task-Based Approximation solution, shown as a percentage of the Task-Based Approximation solution value, broken down by whether or not time reduction factor $r(d)$ was used.

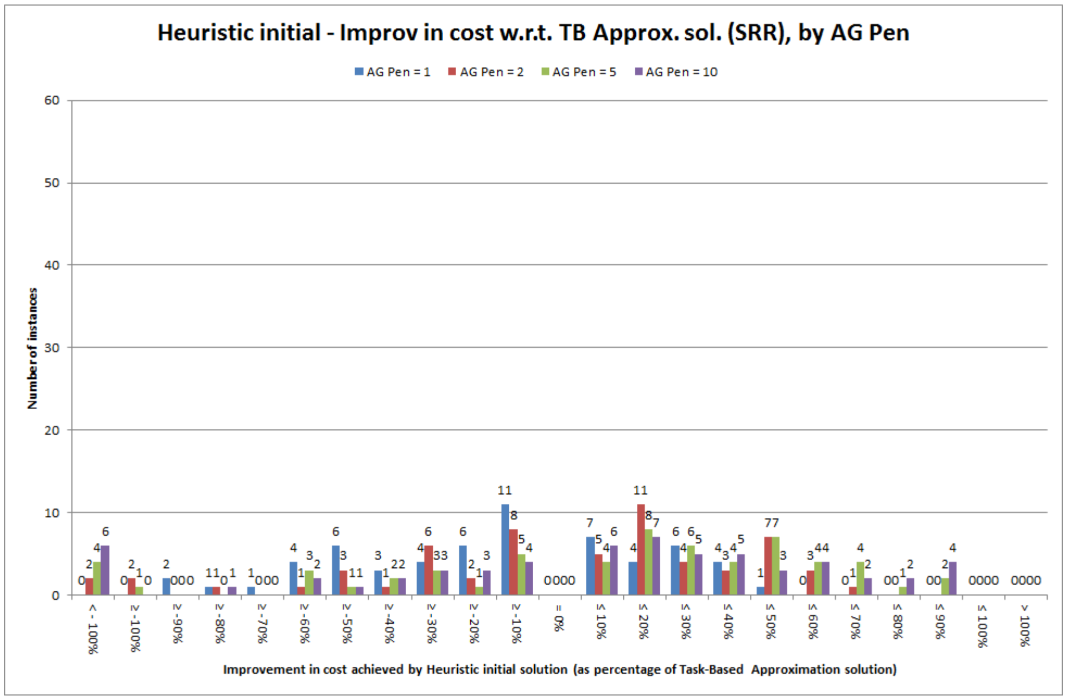


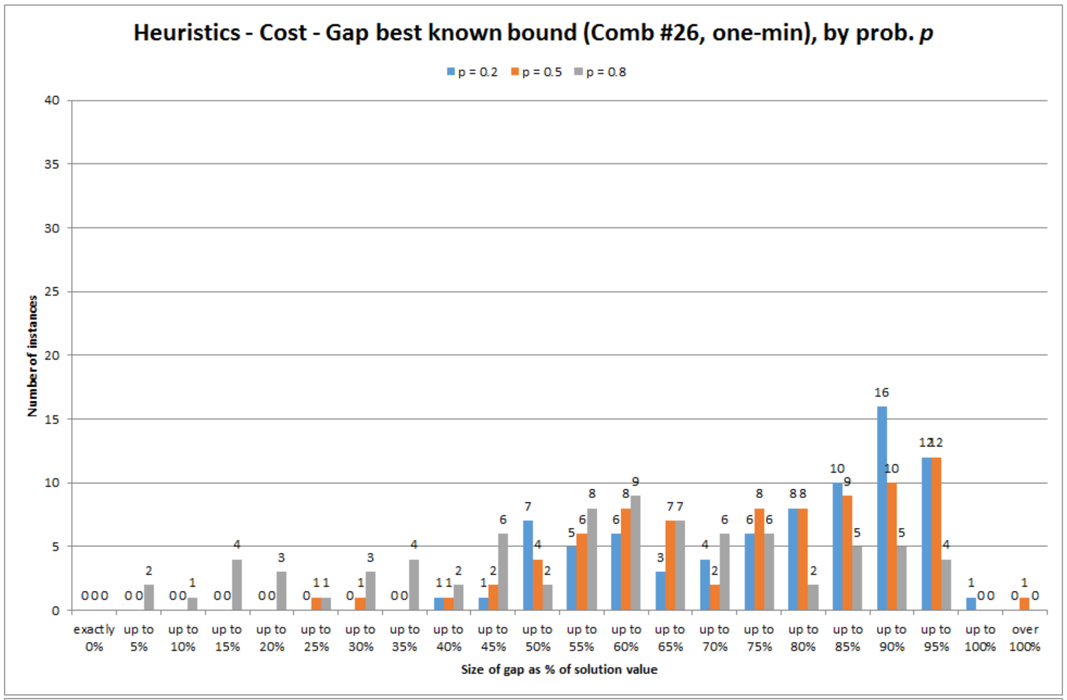
Figure D.103: Histogram summarising improvement in the solution for each instance using the Heuristic initial solution approach compared to the solution cost for the Task-Based Approximation solution, shown as a percentage of the Task-Based Approximation solution value, broken down by values of the agency penalty factor K_{AG}.

D.3.4 Breakdown of Heuristic algorithm results

Here we present selected histograms which can aid our understanding of the F-test and Kruskall-Wallis test results given in Table 6.19. These show the same results as in section D.2.9 for the Heuristic algorithm using setting Combination #26, but broken down according to the values of the parameters which this table highlights as having a significant influence.

D.3.4.1 Relating to cost measures

We begin with breakdowns of the gaps to the best known bound and best known solution, and the improvement made on the initial solution:



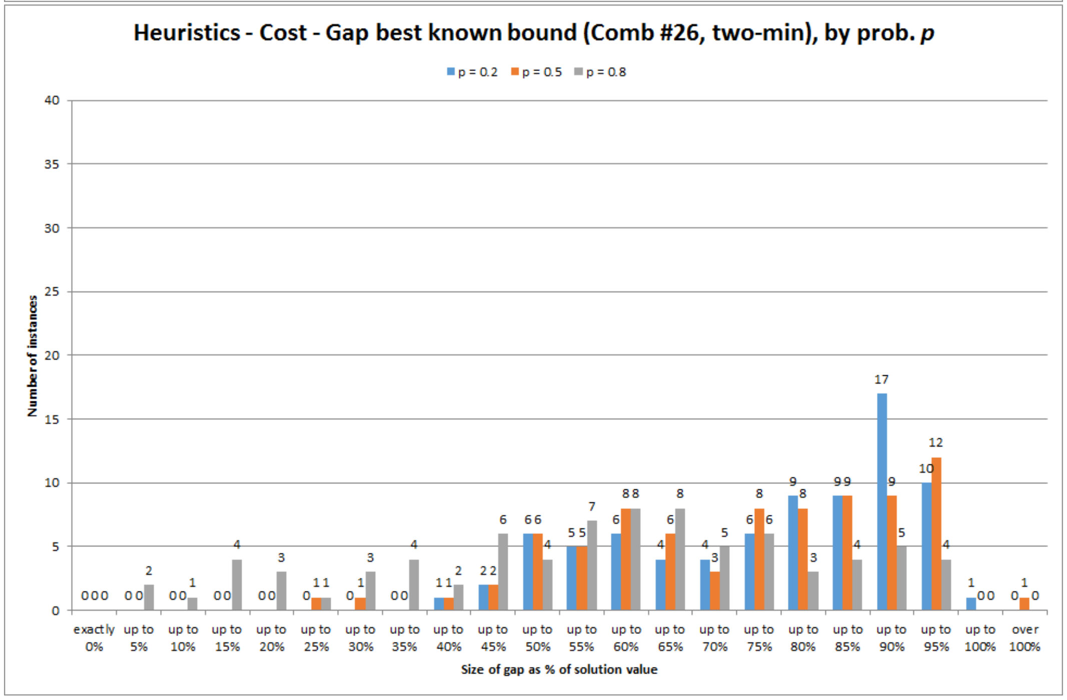
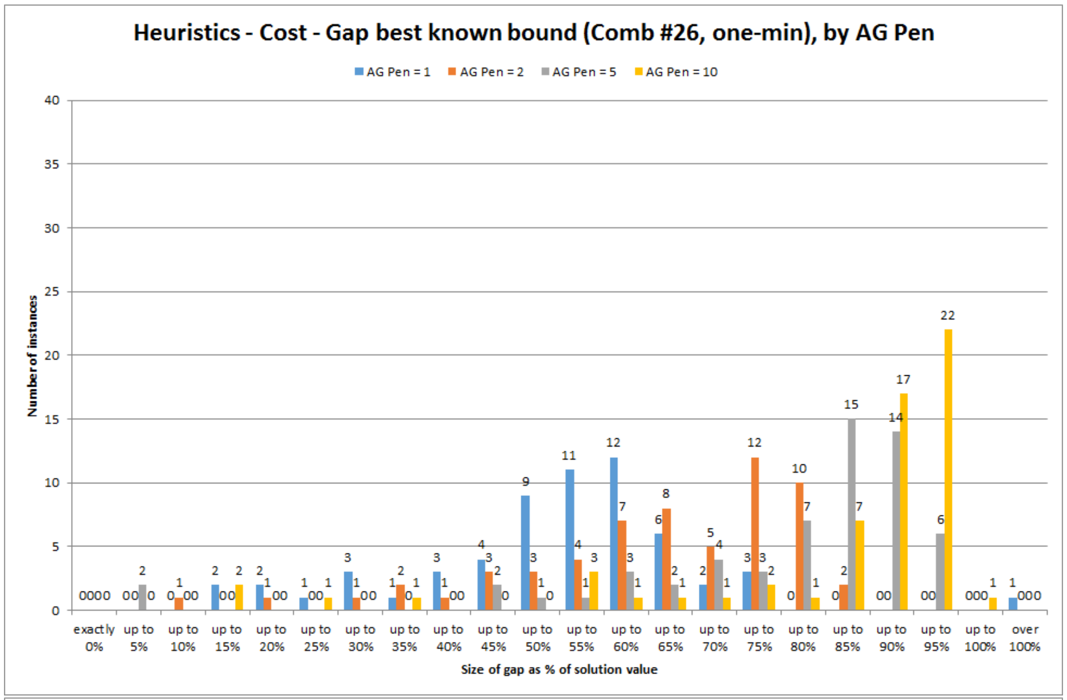


Figure D.104: Gap to best known bound from solutions found after one minute (top) and after two minutes (bottom), for setting combination #26 only, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



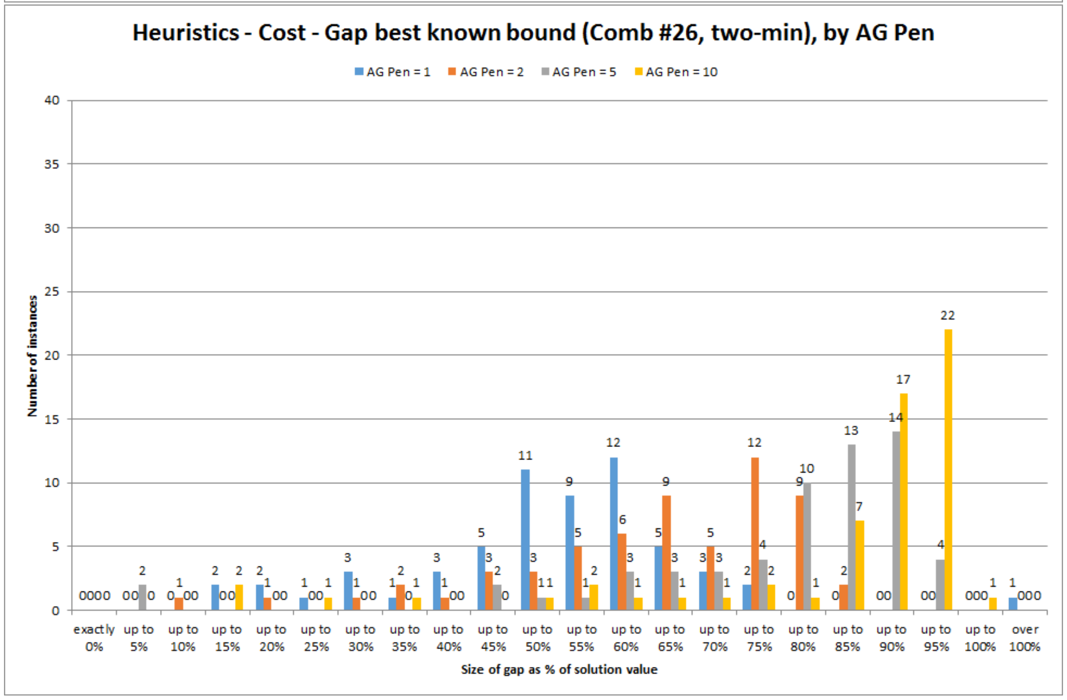
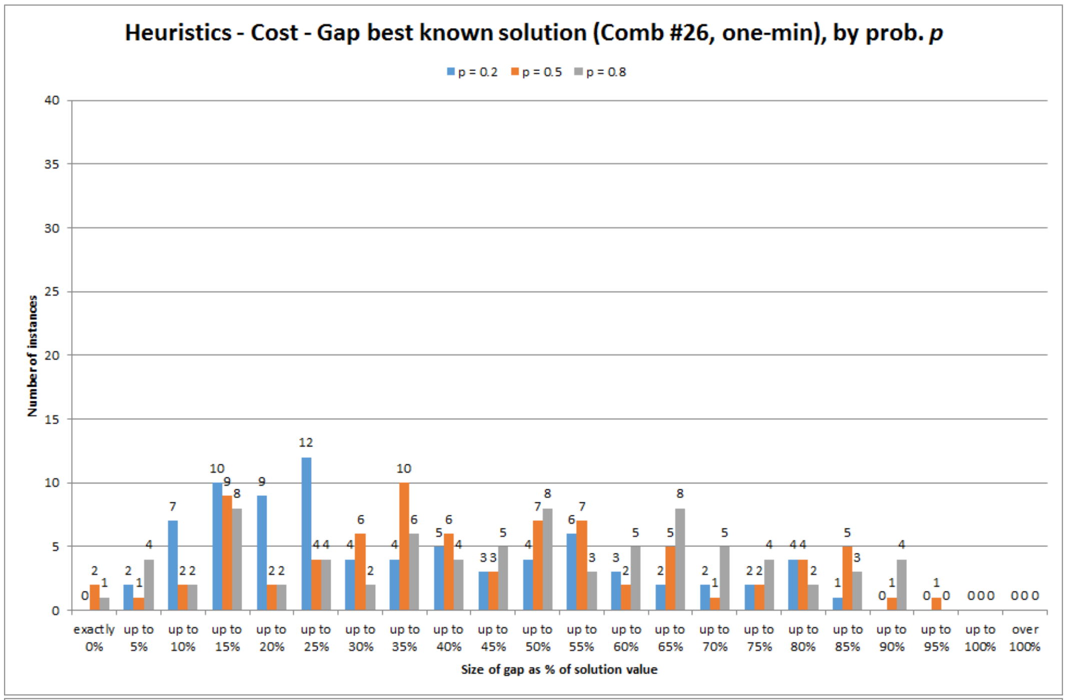


Figure D.105: Gap to best known bound from solutions found after one minute (top) and after two minutes (bottom), for setting combination #26 only, broken down by values of the agency penalty factor K_{AG}.



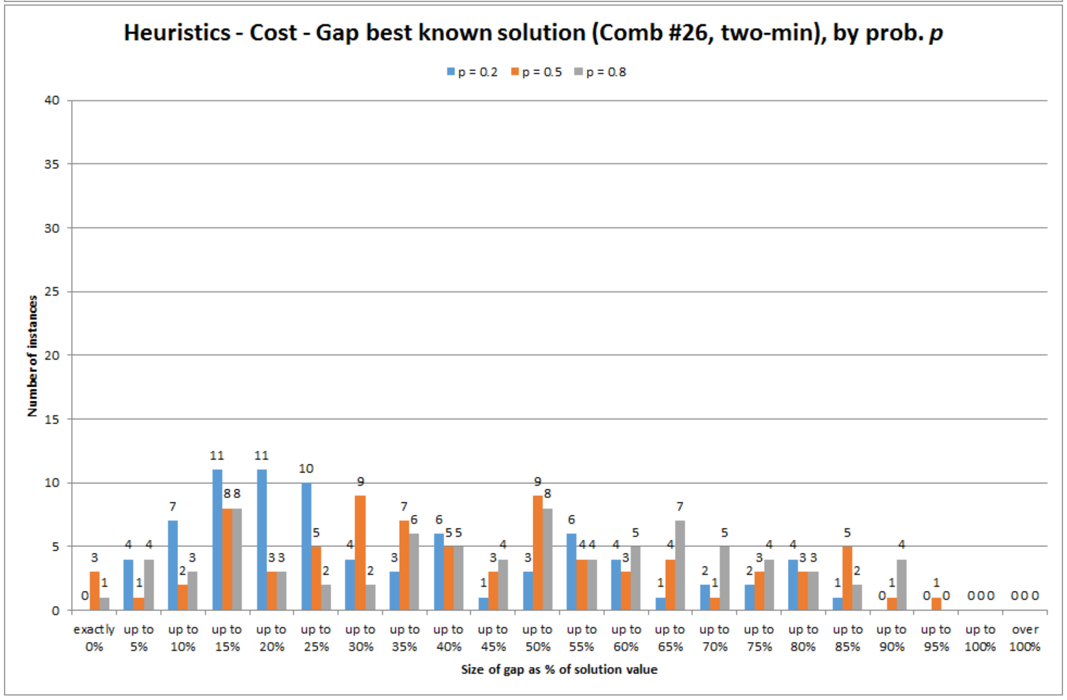
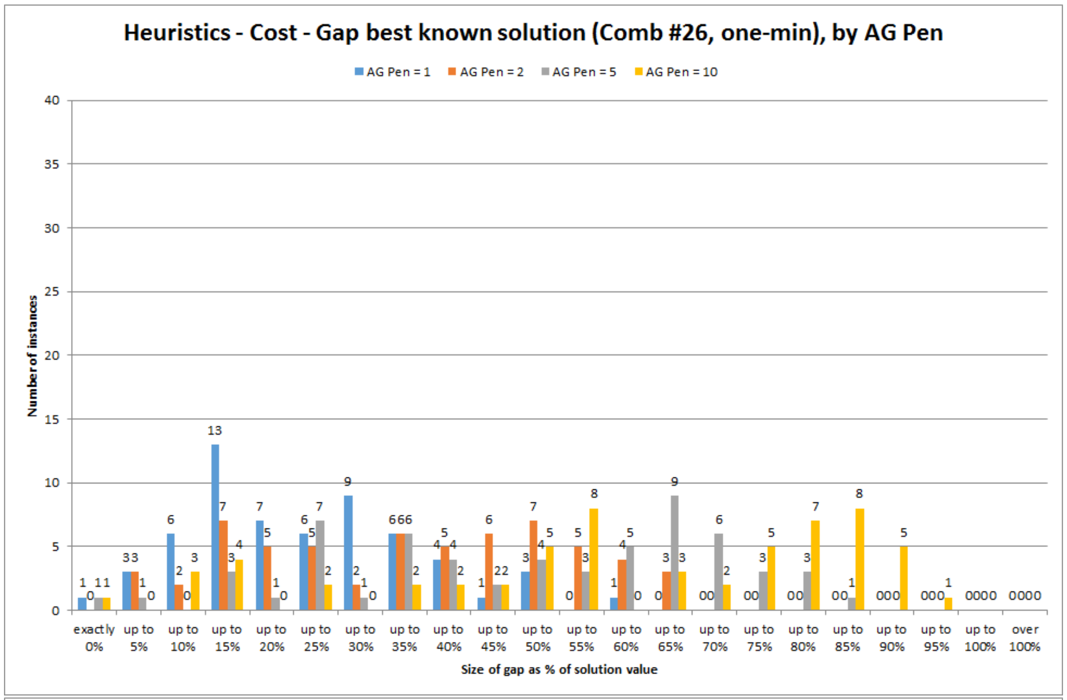


Figure D.106: Gap to best known solution overall from solutions found after one minute (top) and two minutes (bottom), for setting combination #26 only, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



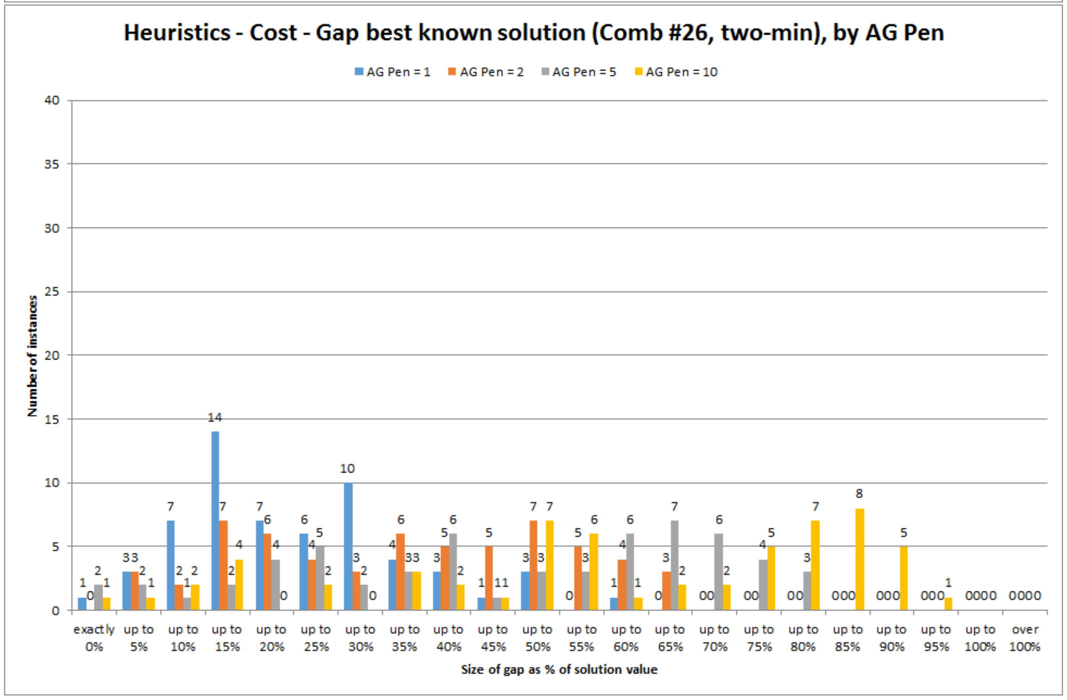
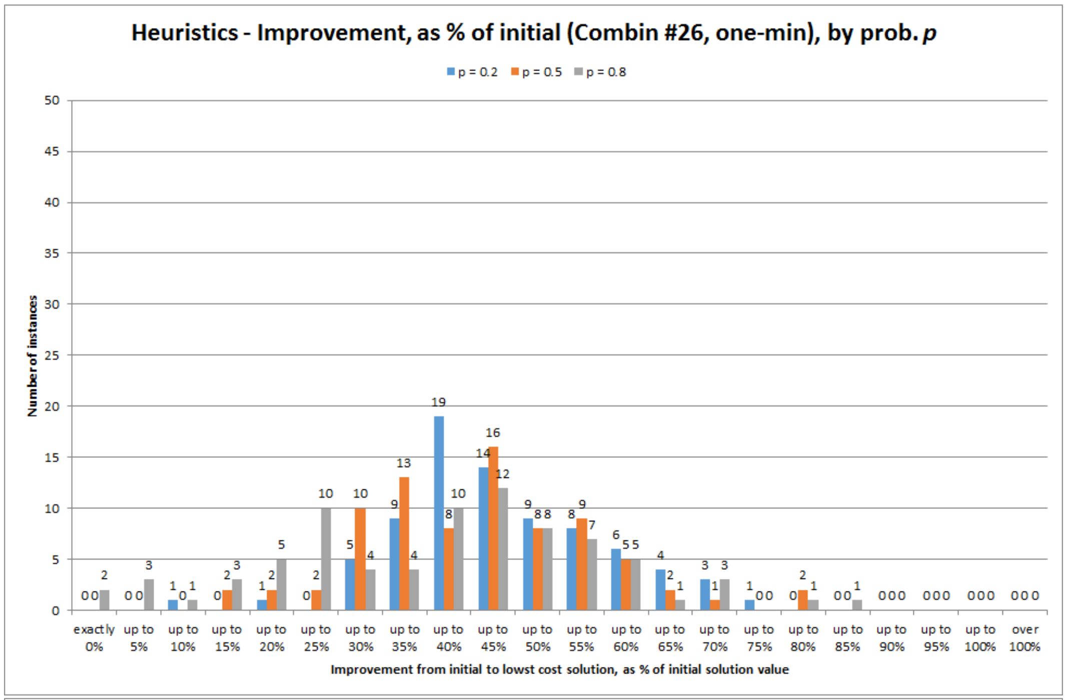


Figure D.107: Gap to best known solution overall from solutions found after one minute (top) and two minutes (bottom), for setting combination #26 only, broken down by values of the agency penalty factor K_{AG}.



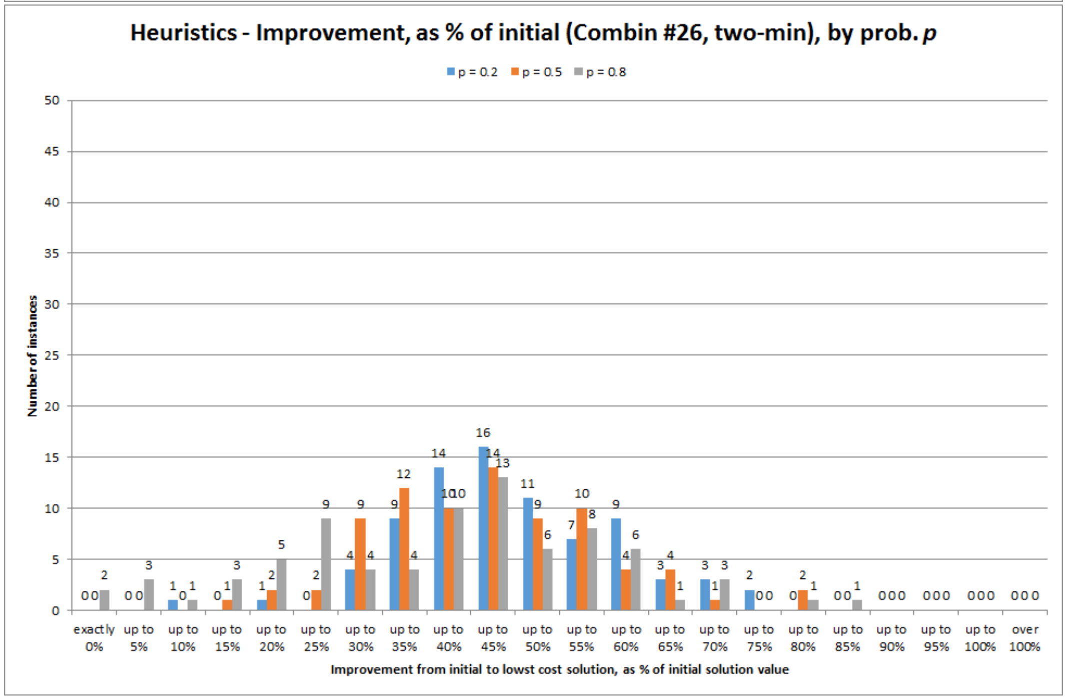


Figure D.108: Improvement on the initial solution achieved within one minute (top) and two minutes (bottom) given as a percentage of the initial solution value, for setting combination #26 only, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).

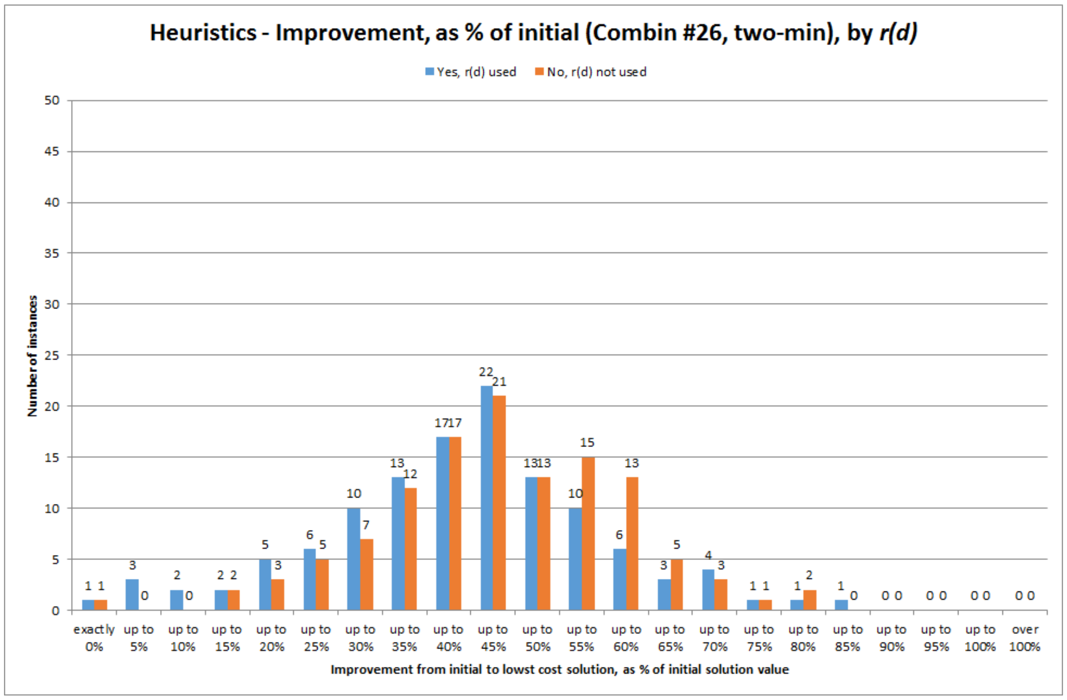
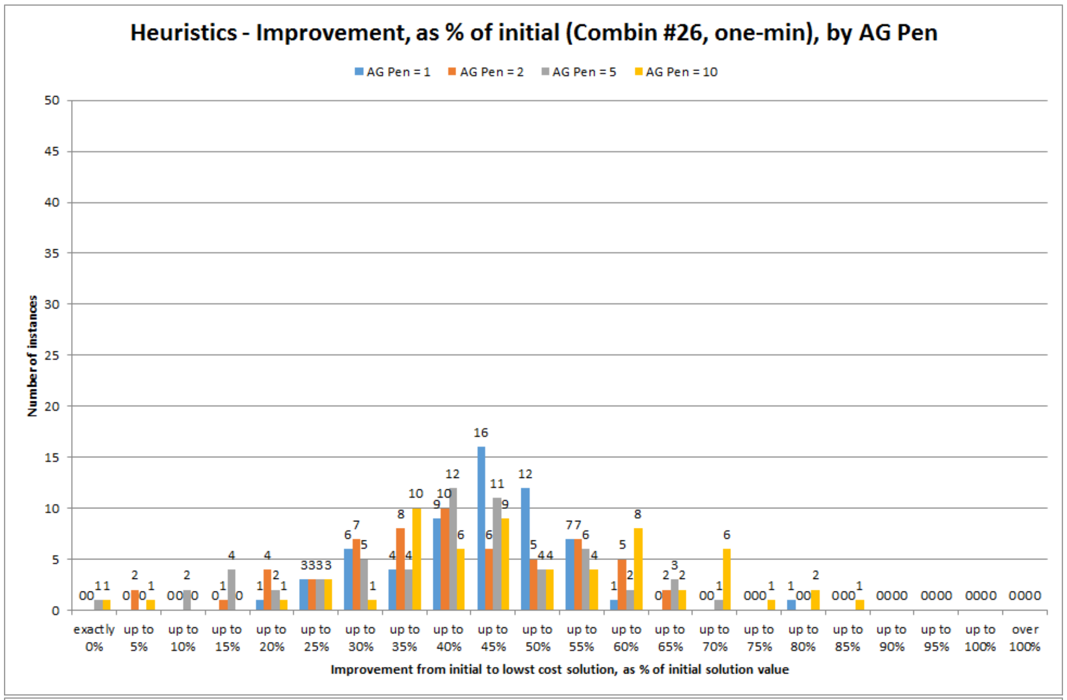


Figure D.109: Improvement on the initial solution achieved within two minutes given as a percentage of the initial solution value, for setting combination #26 only, broken down by whether or not time reduction factor $r(d)$ was used.



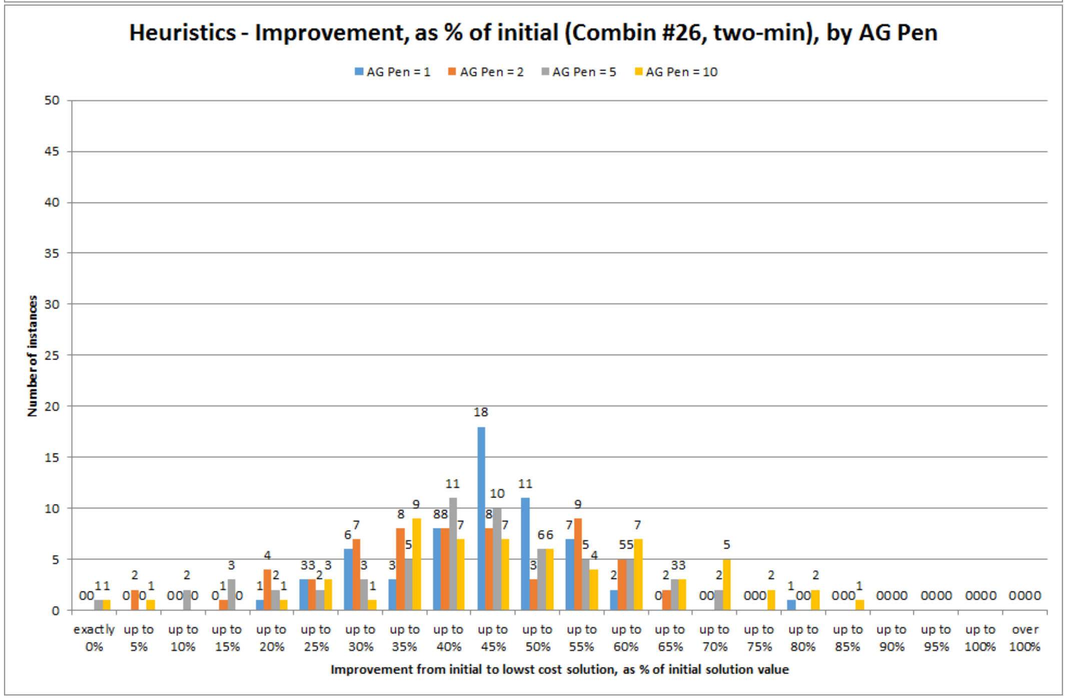
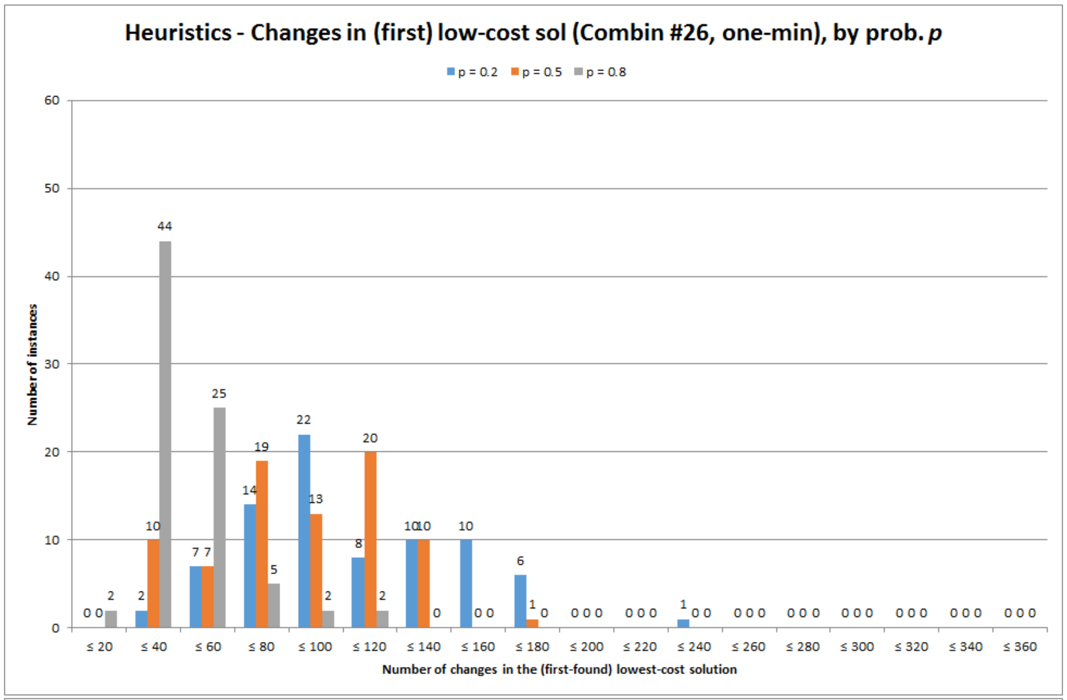


Figure D.110: Improvement on the initial solution achieved within one minute (top) and two minutes (bottom) given as a percentage of the initial solution value, for setting combination #26 only, broken down by values of the agency penalty factor K_{AG}.

D.3.4.2 Relating to numbers of changes and iterations

The following describe breakdowns of the number of changes in the solution and the number of iterations carried out by the algorithm:



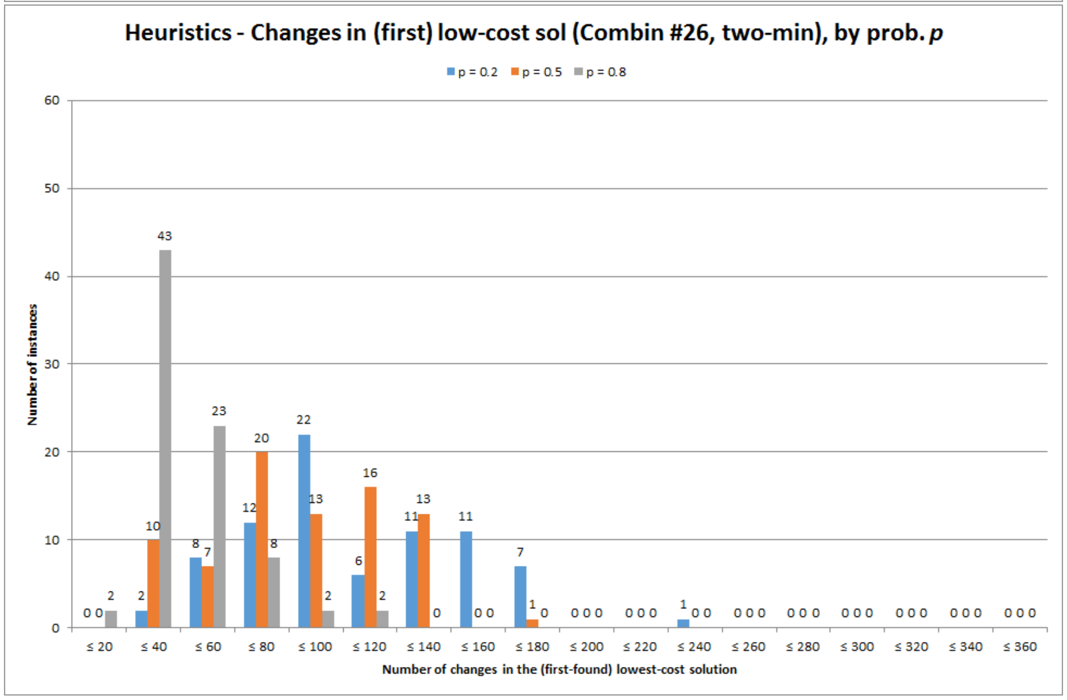
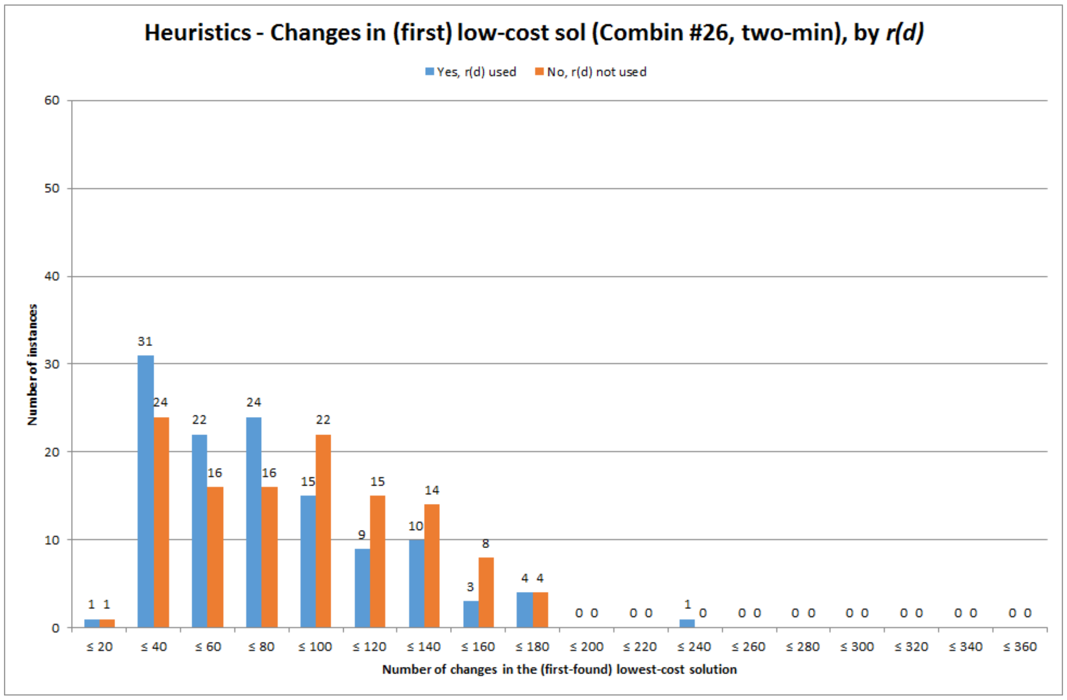


Figure D.111: Number of changes in the first lowest-cost solution found within one minute (top) and two minutes (bottom), for setting combination #26 only, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



	≤ 20	≤ 40	≤ 60	≤ 80	≤ 100	≤ 120	≤ 140	≤ 160	≤ 180	≤ 200	≤ 220	≤ 240	≤ 260	≤ 280	≤ 300	≤ 320	≤ 340	≤ 360
Yes, r(d) used	1	31	22	24	15	9	10	3	4	0	0	1	0	0	0	0	0	0
No, r(d) not used	1	24	16	16	22	15	14	8	4	0	0	0	0	0	0	0	0	0

Figure D.112: Number of changes in the first lowest-cost solution found within two minutes, for setting combination #26 only, broken down by whether or not time reduction factor $r(d)$ was used.

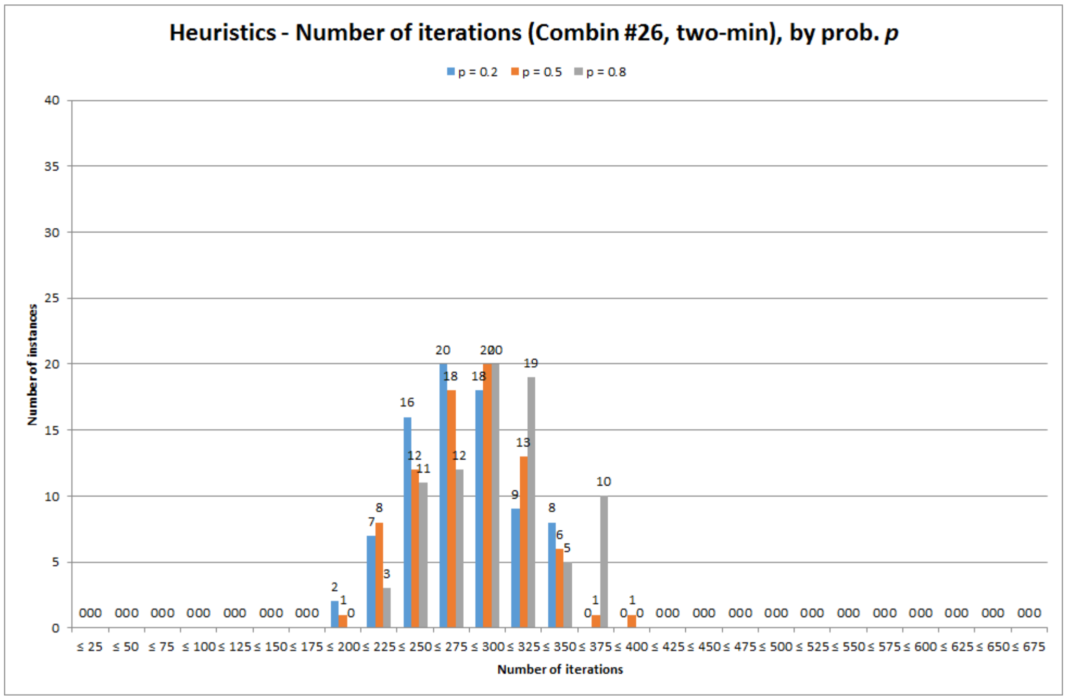
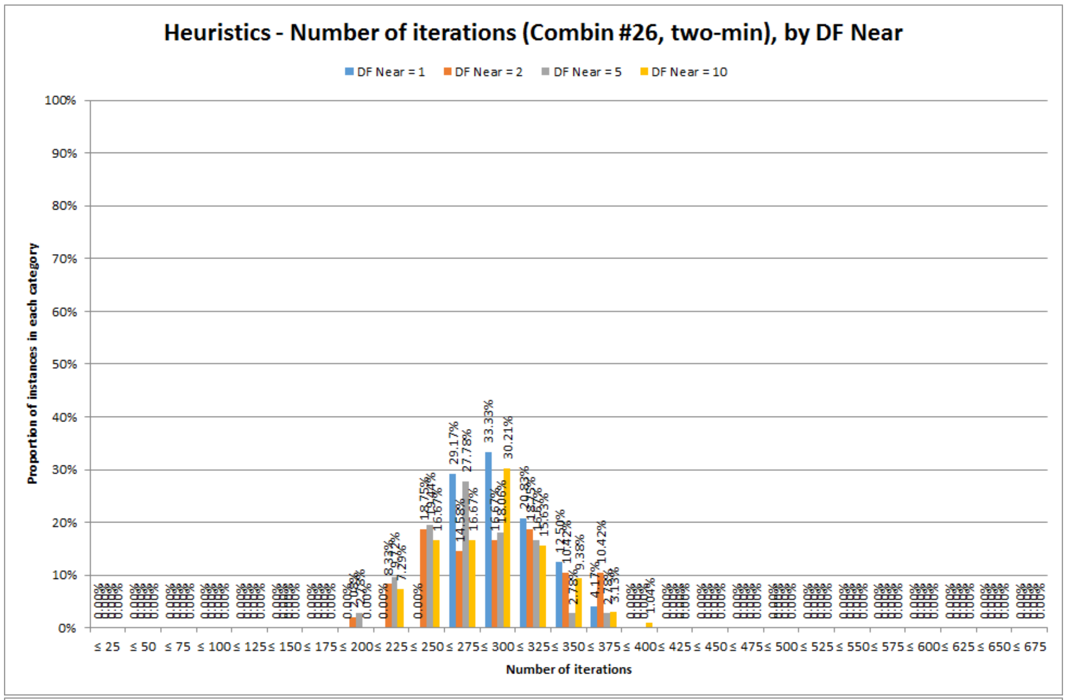


Figure D.113: Number of iterations carried out by the algorithm within two minutes, for setting combination #26 only, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



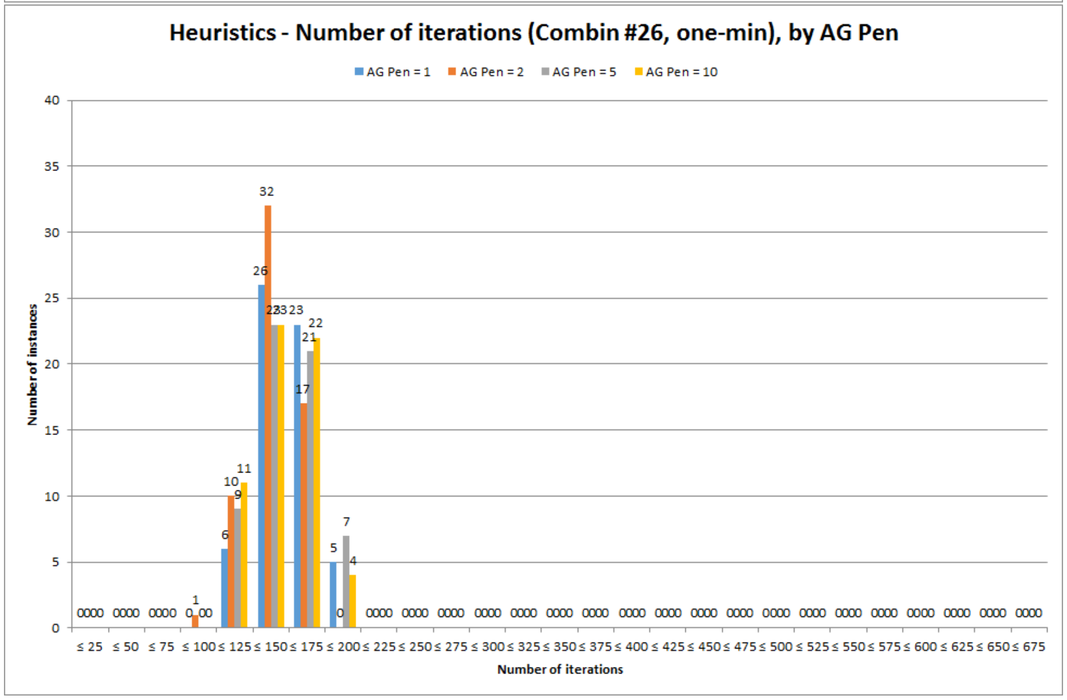
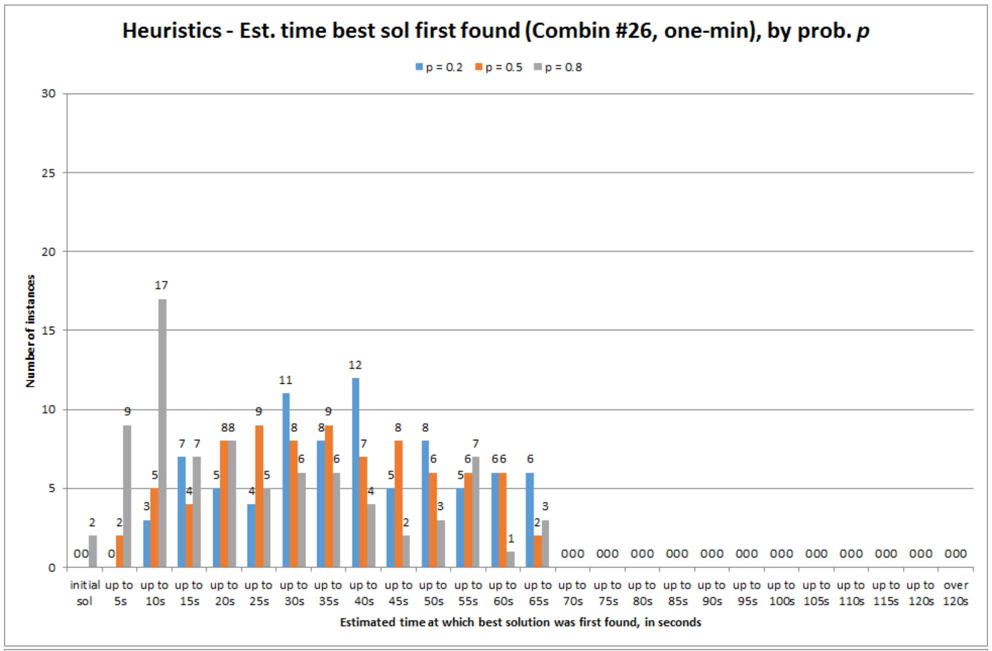


Figure D.114: Number of iterations carried out by the algorithm within two minutes, for setting combination #26 only, broken down by values of the near-term disruption penalty K_N (top) and by values of the agency penalty factor K_{AG} (bottom).

D.3.4.3 Relating to solution time

The following describe breakdowns of the estimated time to the best solution and the estimated gaps to the final solution if the algorithm is terminated early:



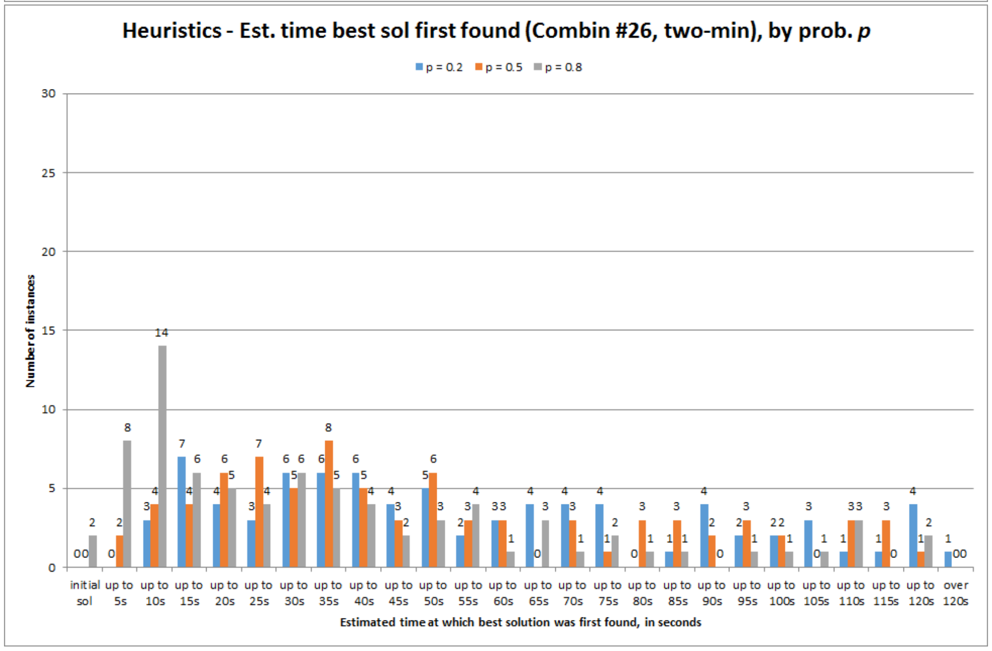


Figure D.115: Estimated time at which the best solution within one minute (top) and within two minutes (bottom) was first found, for setting combination #26 only, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).

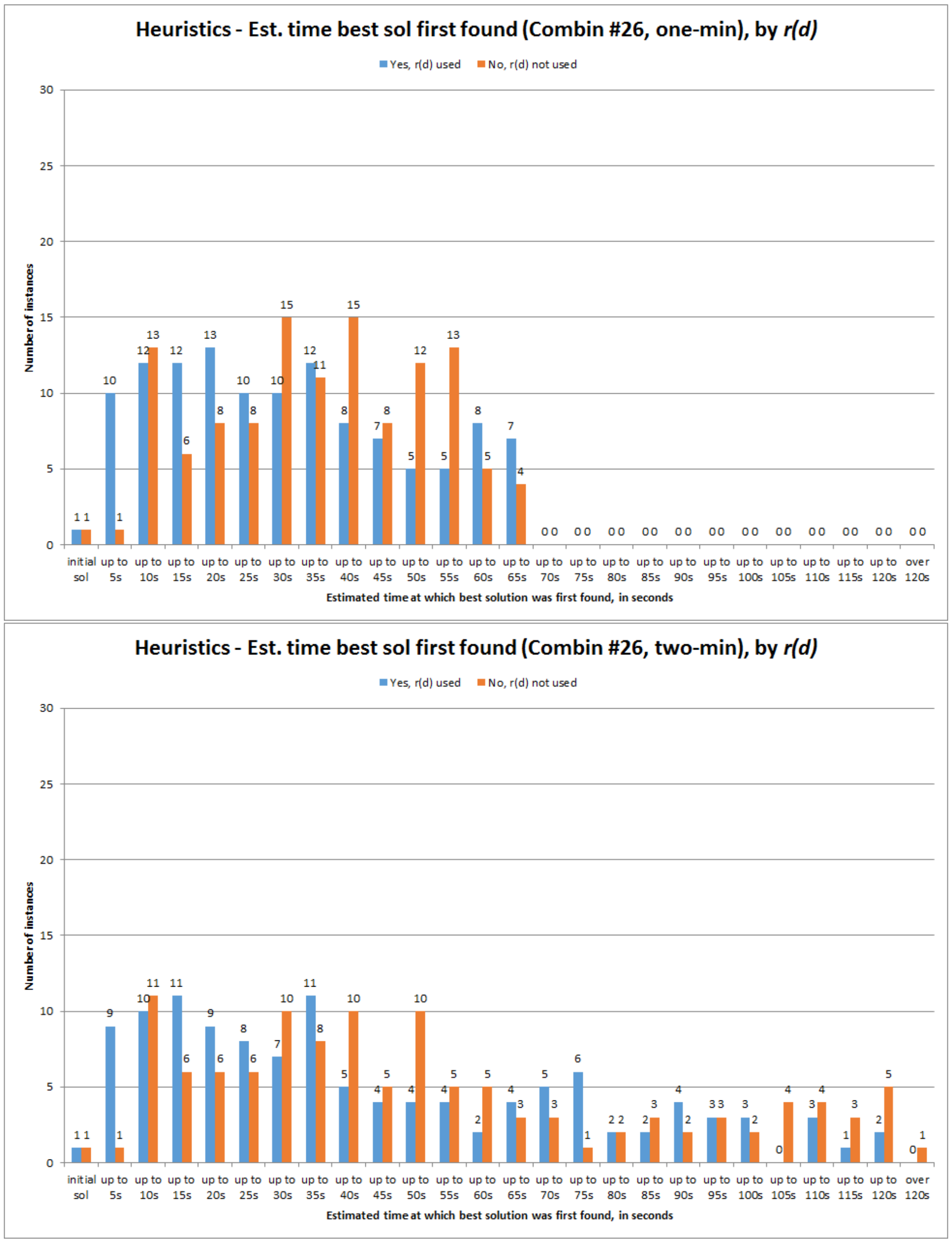
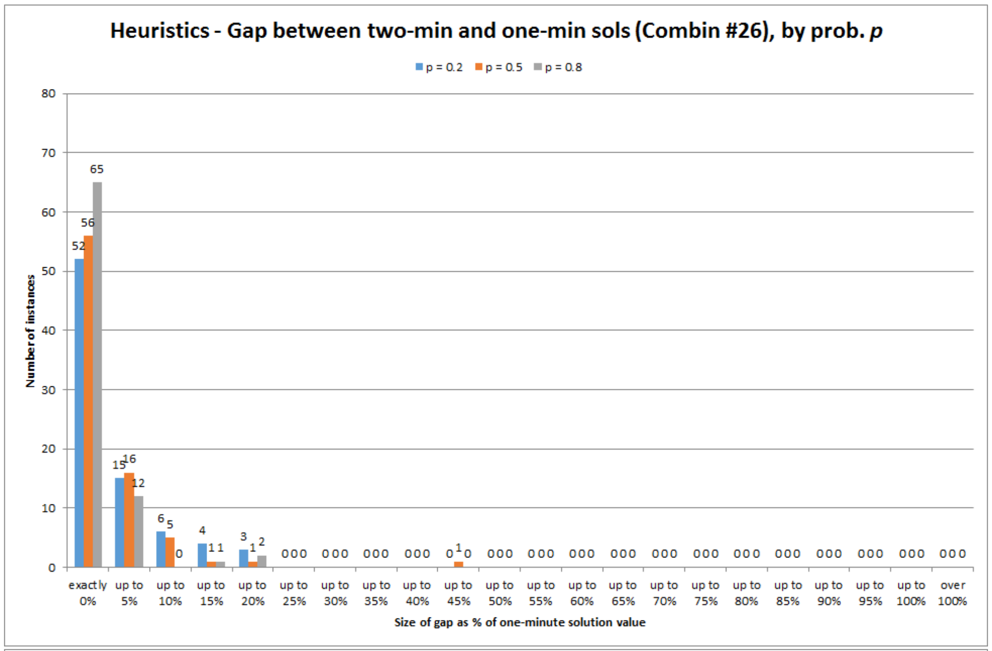


Figure D.116: Estimated time at which the best solution within one minute (top) and within two minutes (bottom) was first found, for setting combination #26 only, broken down by whether or not time reduction factor $r(d)$ was used.



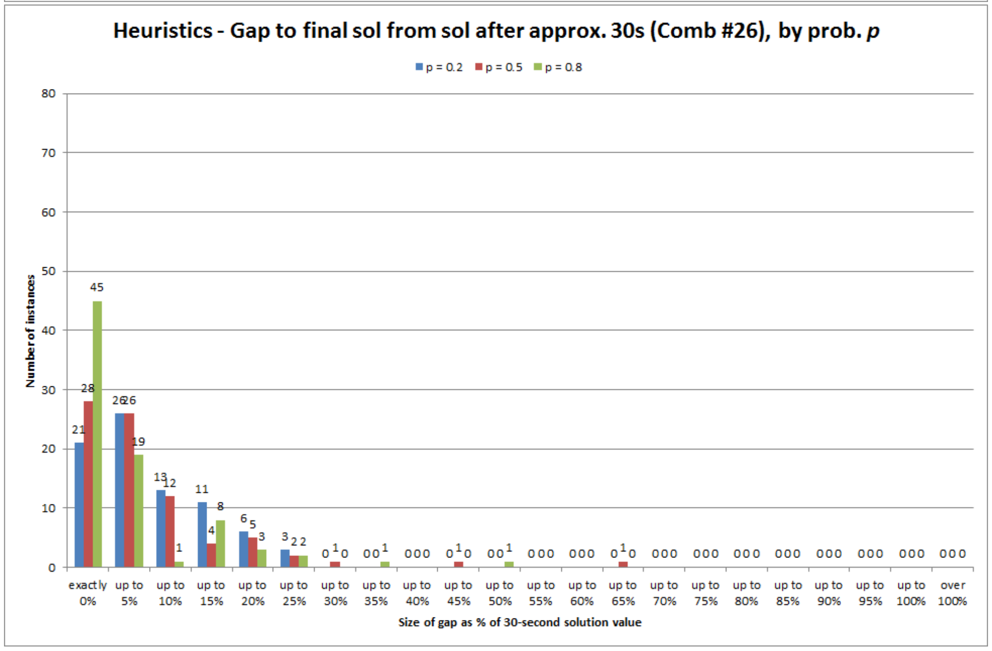
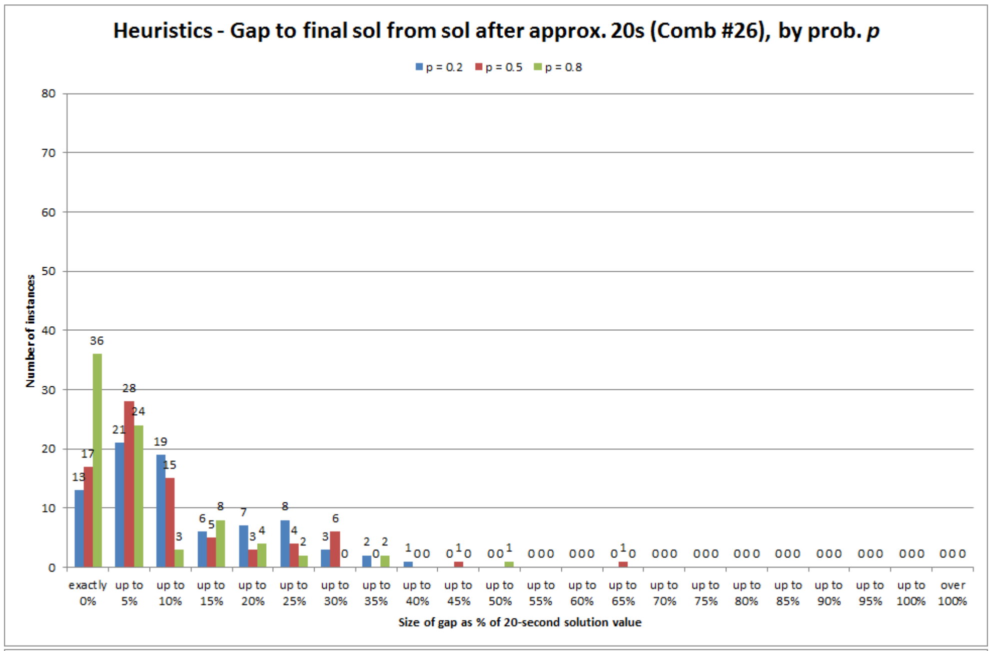


Figure D.117: Gap to two-minute solution value from one-minute solution (top) and estimated 30-second solution (bottom), for setting combination #26 only, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).



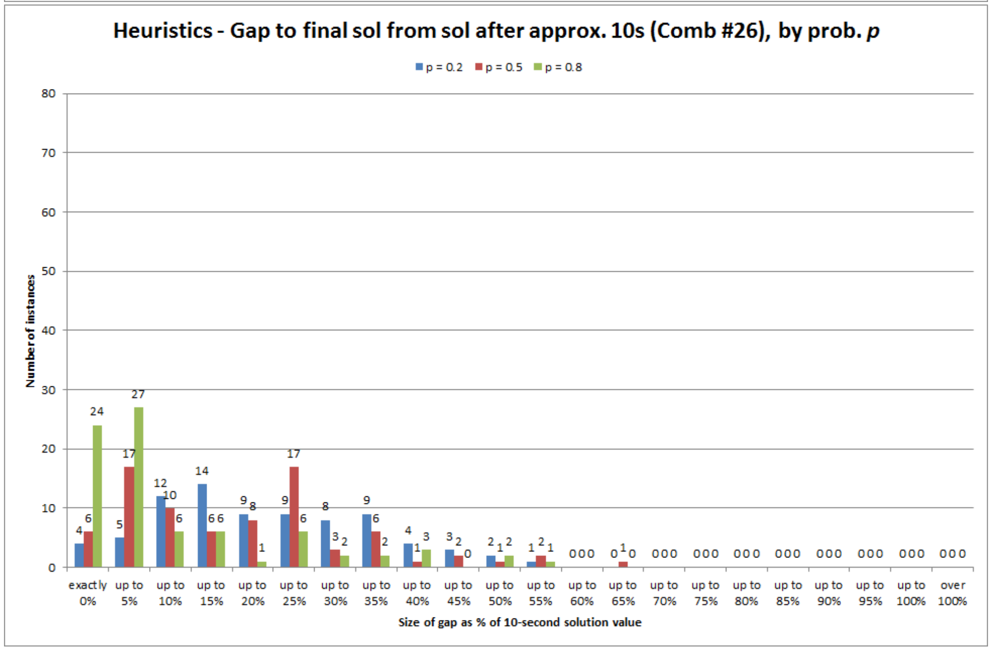


Figure D.118: Gap to two-minute solution value from estimated 20-second solution (top) and estimated 10-second solution (bottom), for setting combination #26 only, broken down by values of availability probability p (probability employee unavailable given they were unavailable the previous day).

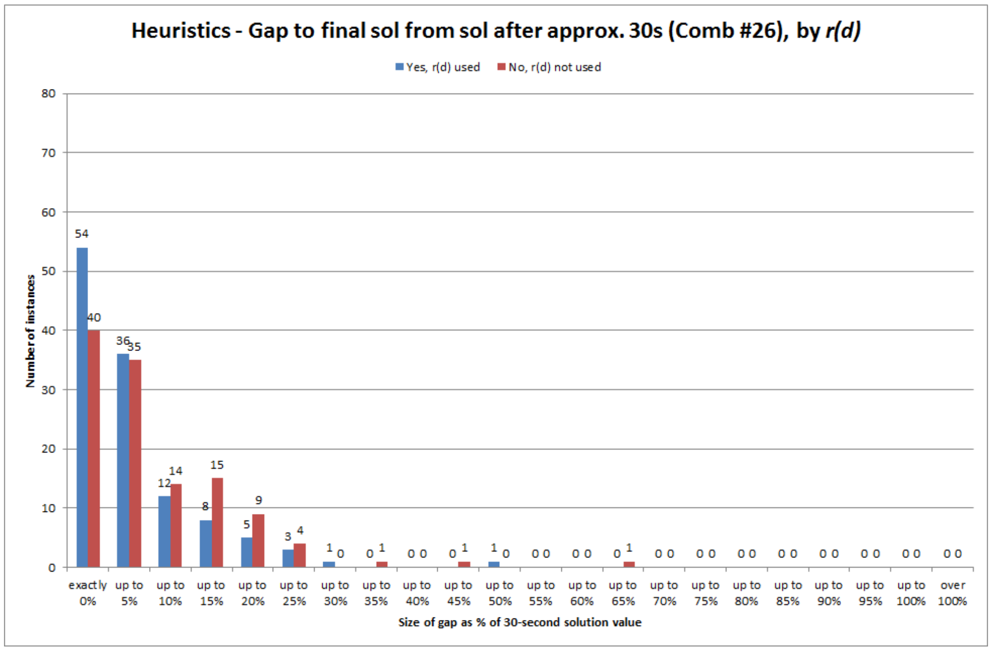
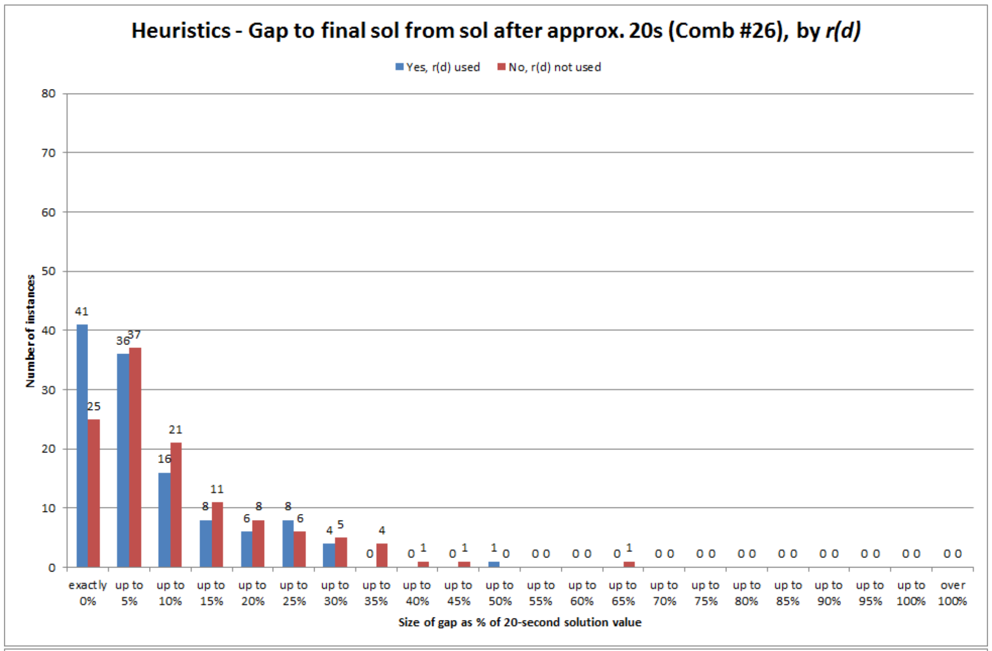


Figure D.119: Gap to two-minute solution value from estimated 30-second solution, for setting combination #26 only, broken down by whether or not time reduction factor $r(d)$ was used.



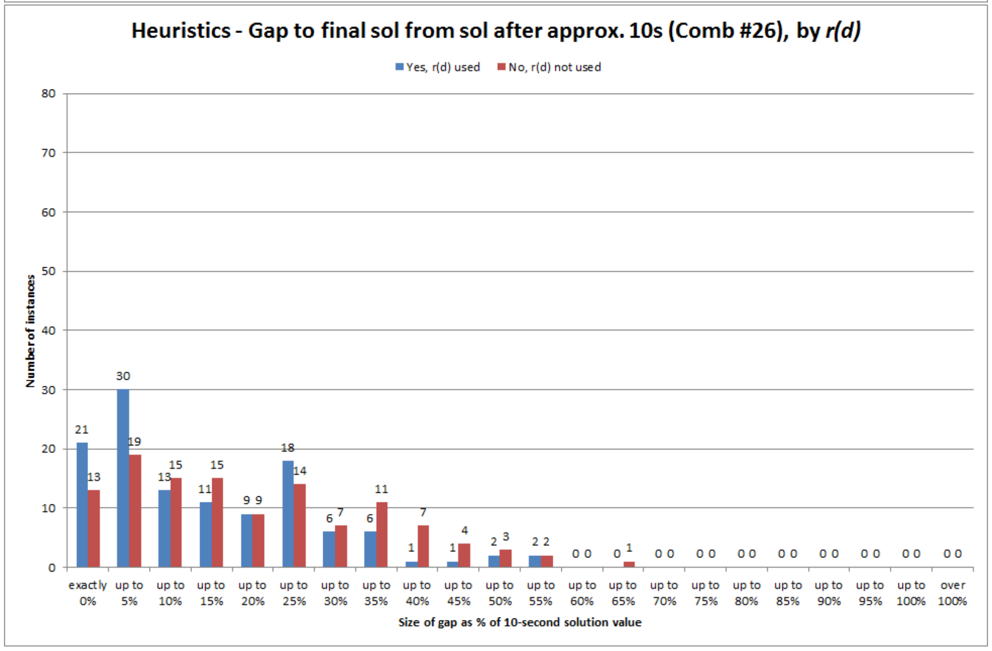


Figure D.120: Gap to two-minute solution value from estimated 20-second solution (top) and estimated 10-second solution (bottom), for setting combination #26 only, broken down by whether or not time reduction factor $r(d)$ was used.

Appendix E

Code

This chapter gives the code used for the computational tests carried out on both the Task-Based formulation (as described in chapter 5) and the Time-Windows formulation (as described in chapter 6) of the problem, as well as the code used to run the data-generating algorithms for each formulation.

E.1 For the Task-Based formulation

Here we give the code used for generating the Task-Based datasets and for testing the cost-minimization and change-minimization approaches to solving the problem.

E.1.1 Generating datasets

Here we give the code used to generate the Task-Based datasets, as described in section 5.4. This was implemented using the FICO Xpress software.

```
model ModelName
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver

! use the following when generating multiple data sets:
!parameters
!       PROJECTS = FALSE
!       INSTANCELIST = "Task-Based - Captains - Instance List.txt"
!       DATAFILE = "SS7 real data - Captains.txt"
!end-parameters
!
!declarations
!       NO_OF_INSTANCES: integer
!       OUTFILE: string
```

```
!end-declarations
!
!initialisations from INSTANCELIST
!       NO_OF_INSTANCES
!end-initialisations
!
!declarations
!       INSTANCES = 1..NO_OF_INSTANCES
!       instance_name: array(INSTANCES) of string
!       ill_given_ill: array(INSTANCES) of real
!       ill_given_fit: array(INSTANCES) of real
!       use_TR: array(INSTANCES) of boolean
!       DF_near: array(INSTANCES) of real
!       DF_long: array(INSTANCES) of real
!       AG_pen: array(INSTANCES) of real
!end-declarations
!
!initialisations from INSTANCELIST
!       instance_name
!       ill_given_ill
!       ill_given_fit
!       use_TR
!       DF_near
!       DF_long
!       AG_pen
!end-initialisations
!
!
!
!declarations
!       ! Parameters to be varied for various data sets:
!       ill_given_prev_fit: real
!       ill_given_prev_ill: real
!
!       use_time_reduction: boolean
!
!       disruption_factor_near: real
!       disruption_factor_long: real
!       agency_penalty: real
!
!end-declarations
!------------------------------------------------------
```

```
! NOTE - Above section used when generating multiple data sets
!              Alternatively, use section as given below:

parameters
       OUTFILE = "Task-Based - Captains - Trial runs.txt"
       DATAFILE = "SS7 real data - Captains.txt"
       PROJECTS = FALSE

       ! Parameters to be varied for various data sets:
       ill_given_prev_fit = 0.004
       ill_given_prev_ill = 0.5

       use_time_reduction = TRUE

       disruption_factor_near = 1
       disruption_factor_long = 1
       agency_penalty = 1

end-parameters



! Notes on Parameters:
!------------------------------------------------------------------------
! Probabilities of being ill or fit to work given previous day condition:
! Firstly, assume that the probability of being ill on a given day is 0.8% (=
   0.008)
! Then, can use the following pairs of data:
!      Prob(ill | ill on previous day)     Prob(ill | fit on previous day)
!                    0.8
   0.0016
!                    0.75                                             0.002
!                    0.7
   0.0024
!                    0.65                                             0.0028
!                    0.6
   0.0032
!                    0.55                                             0.0036
!                    0.5
   0.004         --> currently selected
!                    0.45                                             0.0044
!                    0.4
   0.0048
```

```
!                    0.35                                              0.0052
!                    0.3
   0.0056
!                    0.25                                              0.0061
!                    0.2
   0.0065
!-------------------------------------------------------------------------
! Time reduction can be used to reduce the above probabilities for times further
   in the future than 4 weeks.
! For a given day index d > 28, the time reduction is calculated as
!            time_reduction := (d-28)/((7*no_of_weeks)-28)
!
! The probabilities are then multiplied by (1 - time_reduction)
! It may be useful to vary the use of the time reduction to see if there is a
   significant impact on results
!-------------------------------------------------------------------------
! Disruption factors and agency penalty are used to penalise the making of
   changes in the schedule
!  -->  Disruption factors are multipliers applied to the cost of changing the
   assignment if the cost is
!            positive, and used as a divisor if the cost is negative. They are
   defined differently for tasks which
!            start within the first four weeks (assumed to be penalised more
   harshly if any penalty is applied),
!            and those which start after this time. There is no disruption
   factor for assignments made in the final
!            week of the horizon.
!  -->  Similarly, a penalty is defined as a multiplier for positive or a divisor
   for negative costs of
!            changing the assignment of Agency crew, over and above the actual
   costs involved.
!-------------------------------------------------------------------------


!-------------------------------------------------------------------------------------

! Declarations for data to be read in:
declarations
       no_of_crew: integer
       no_of_vessels: integer
       no_of_weeks: integer
end-declarations
```

```
initialisations from DATAFILE
       no_of_crew no_of_vessels no_of_weeks
end-initialisations

declarations
       crew = 1..no_of_crew
       crew_label: array(crew) of string
       contract_type: array(crew) of string
       legal_entity: array(crew) of string
       crew_continent: array(crew) of string
       crew_nation: array(crew) of string

       vessel = 1..no_of_vessels
       vessel_name: array(vessel) of string
       vessel_label: array(vessel) of string
       required: array(vessel) of integer
       vessel_location: array(vessel) of string
!      operation_type: array(vessel) of string
end-declarations

initialisations from DATAFILE
       crew_label vessel_name
       contract_type legal_entity crew_continent crew_nation
       vessel_label required vessel_location
!      operation_type
end-initialisations



procedure MAIN_PROG

       !-------------------------------------------------------------------------
       ! Declarations for data to be calulated / generated for this instance:
       declarations
              no_of_tasks: integer

              Norway_emps, Singapore_emps, Other_emps: integer

              max_work, min_rest: integer

              vessel_max_work: array(vessel) of integer

              vessel_roles: array(vessel) of set of integer
```

```
            role_tasks: integer

            task_no, time_count: integer

            rand_no: real

    end-declarations


    !-----------------------------------------------------------------------
    ! Calculations:

    ! First, need to know task lengths:
    Norway_emps := 0
    Singapore_emps := 0
    Other_emps := 0

    forall(c in crew) do
            if(legal_entity(c) = "Subsea7 Contr (Norway) AS") then Norway_emps
                := Norway_emps +1
            elif(legal_entity(c) = "Subsea 7 (Sing) PTE-GEC") then
                Singapore_emps := Singapore_emps +1
            else Other_emps := Other_emps +1
            end-if
    end-do

    if(Singapore_emps > Norway_emps) then
            if(Singapore_emps > Other_emps) then
!                   writeln("Singapore employees are the dominant contract type
  (",Singapore_emps,", vs ",Norway_emps," Norway and ",Other_emps," others)")
                    max_work := 10
                    min_rest := 5
            else
!                   writeln("Other employees are the dominant contract type (",
  Other_emps,", vs ",Norway_emps," Norway and ",Singapore_emps," Singapore)")
                    max_work := 5
                    min_rest := 4
            end-if
    else
            if(Norway_emps > Other_emps) then
!                   writeln("Norway employees are the dominant contract type (",
  Norway_emps,", vs ",Singapore_emps," Singapore and ",Other_emps," others)")
                    max_work := 2
```

```
                        min_rest := 4
                else
!                       writeln("Other employees are the dominant contract type (",
    Other_emps,", vs ",Norway_emps," Norway and ",Singapore_emps," Singapore)")
                        max_work := 5
                        min_rest := 4
                end-if
        end-if


        ! Next need to calculate the number of roles required in total
        role_no := 0
        forall(v in vessel | required(v) > 0) do
                vessel_roles(v) := {}
                forall(r in 1..required(v)) do
                        role_no := role_no + 1
                        vessel_roles(v) += {role_no}
                end-do
        end-do

        declarations
                roles = 1..role_no
                start_duration: array(roles) of integer
        end-declarations


        ! Can now determine the starting points for all vessel tasks
        no_of_tasks := 0
!       writeln
!       writeln("At start, tasks on board vessels have the following lengths
    remaining:")

        forall(v in vessel | required(v) > 0) do

                if(max_work = 5 and vessel_location(v) = "Europe") then
                        vessel_max_work(v) := 4
                else
                        vessel_max_work(v) := max_work
                end-if

                forall(r in vessel_roles(v)) do
                        start_duration(r) := integer(vessel_max_work(v)*random)
                        if(start_duration(r) = 0) then
```

```
                              if(10*integer(no_of_weeks/vessel_max_work(v)) = 10*(
                                  no_of_weeks/vessel_max_work(v))) then
!                                     write("\tCase 1: ",10*(integer(no_of_weeks/
    vessel_max_work(v))),"  = ",10*(no_of_weeks/vessel_max_work(v)))
                                      role_tasks := integer(no_of_weeks/
                                          vessel_max_work(v))
                              else
!                                     write("\tCase 2: ",10*integer(no_of_weeks/
    vessel_max_work(v)),"  <> ",10*(no_of_weeks/vessel_max_work(v)))
                                      role_tasks := integer(no_of_weeks/
                                          vessel_max_work(v)) +1
                              end-if
                      else
                              if(10*integer((no_of_weeks - start_duration(r))/
                                  vessel_max_work(v)) = 10*(no_of_weeks -
                                  start_duration(r))/vessel_max_work(v)) then
!                                     write("\tCase 3: ",10*integer((no_of_weeks -
    start_duration(r))/vessel_max_work(v)),"  = ",10*(no_of_weeks - start_duration
    (r))/vessel_max_work(v))
                                      role_tasks := integer((no_of_weeks -
                                          start_duration(r))/vessel_max_work(v)) +1
                              else
!                                     write("\tCase 4: ",10*integer((no_of_weeks -
    start_duration(r))/vessel_max_work(v)),"  <> ",10*(no_of_weeks -
    start_duration(r))/vessel_max_work(v))
                                      role_tasks := integer((no_of_weeks -
                                          start_duration(r))/vessel_max_work(v)) +2
                              end-if
                      end-if
                      no_of_tasks := no_of_tasks + role_tasks

!                     writeln("\tVessel ",v,"\t Role ",r,":\t",start_duration(r),"
     weeks (out of ",vessel_max_work(v)," weeks)\tAnd so has ",role_tasks," tasks
     in total.")
              end-do
      end-do


      !-----------------------------------------------------------------------
      ! Generate regular assignments:
      declarations
              active_roles: integer
```

```
                regular_vessel: array(crew) of integer
                no_of_regulars_vessel: array(vessel) of integer

                regular_role: array(crew) of integer
                no_of_regulars_role: array(roles) of integer

                assigned_crew: array(roles) of integer
                assigned_role: integer

                extra, spare: integer
        end-declarations

        active_roles := 0
        forall(v in vessel) do
                active_roles := active_roles + required(v)
                no_of_regulars_vessel(v) := 2*required(v)
                forall(r in vessel_roles(v)) no_of_regulars_role(r) := 2
        end-do

!       writeln
!       writeln
!       writeln("Number of active roles:\t",active_roles)
!       writeln("Number of crew available:\t",no_of_crew)
!       writeln("\t... so number of crew per role:\t",no_of_crew/active_roles)

        if(no_of_crew/active_roles > 2) then
                if(no_of_crew/active_roles > 3) then
                        spare := no_of_crew - (3*active_roles)
                else
                        spare := 0
                end-if

                extra := 0
                while(extra < (no_of_crew - (2*active_roles) - spare)) do
                        rand_no := (active_roles*random) +1
                        if(no_of_regulars_role(integer(rand_no)) < 3) then
                                no_of_regulars_role(integer(rand_no)) :=
                                    no_of_regulars_role(integer(rand_no)) +1
                                extra := extra +1
                        end-if
                end-do
        end-if
        assigned_role := 0
        forall(c in crew) regular_vessel(c) := 0
        forall(c in crew) regular_role(c) := 0
        forall(r in roles) assigned_crew(r) := 0

        while(assigned_role < (no_of_crew - spare)) do
                forall(c in crew | regular_role(c) = 0) do
                        rand_no := (active_roles*random) +1

                        if(assigned_crew(integer(rand_no)) < no_of_regulars_role(
                            integer(rand_no))) then
                                regular_role(c) := integer(rand_no)
                                assigned_crew(integer(rand_no)) := assigned_crew(
                                    integer(rand_no)) +1
                                assigned_role := assigned_role +1

                        end-if
                end-do
        end-do

        forall(v in vessel) do
                no_of_regulars_vessel(v) := 0

                forall(r in vessel_roles(v)) do
                        forall(c in crew) do
                                if(regular_role(c) = r) then
                                        regular_vessel(c) := v
                                end-if
                        end-do

                        no_of_regulars_vessel(v) := no_of_regulars_vessel(v) +
                            no_of_regulars_role(r)
                end-do
        end-do


!       writeln
!       forall(v in vessel | required(v) > 0) do
!               write("Vessel ")
!               if(v < 10) then write(" "); end-if
!               write(v," is assigned the following as regulars:\t")
!               forall(c in crew | regular_vessel(c) = v) do
!                       if(c < 10) then write(" "); end-if
!                       write(c," ")
!               end-do
!               write("\n")
!       end-do
!       writeln
!       writeln
!       forall(r in roles) do
!               write("Role ")
!               if(r < 10) then write(" "); end-if
!               write(r," is assigned the following as regulars:\t")
!               forall(c in crew | regular_role(c) = r) do
!                       if(c < 10) then write(" "); end-if
!                       write(c," ")
!               end-do
!               write("\n")
!       end-do
!       writeln


        !-------------------------------------------------------------------------
        ! And so can determine the tasks in terms of start / end times, and the
            initial assignments to these
        declarations
                tasks = 1..no_of_tasks
                task_vessel: array(tasks) of integer
                task_role: array(tasks) of integer
                task_start, task_duration: array(tasks) of integer

                shipcrew: array(1..3) of integer
                initial: array(crew, tasks) of integer
                ag_initial: array(tasks) of integer
                previous: integer
                more_previous: integer
                back_count: integer

                start_in_role: array(roles) of integer
                work_resource_zero: array(crew) of integer
                rest_resource_zero: array(crew) of integer
                initial_rest_required: array(crew) of integer
        end-declarations

        forall(c in crew) do
                forall(t in tasks) initial(c,t) := 0
                work_resource_zero(c) := 0
                rest_resource_zero(c) := 0
                initial_rest_required(c) := 0
        end-do
        forall(t in tasks) ag_initial(t) := 1


        task_no := 0
        forall(v in vessel) do
                forall(r in vessel_roles(v)) do

                       forall(i in 1..3) shipcrew(i) := 0
                       forall(c in crew | regular_role(c) = r) do
                               if(no_of_regulars_role(r) = 2) then
                                       if(shipcrew(1) = 0 and shipcrew(2) = 0) then
                                             rand_no := random
                                             if(rand_no < 0.5) then
                                                    shipcrew(1) := c
                                             else
                                                    shipcrew(2) := c
                                             end-if
                                       else
                                             if(shipcrew(1) = 0) then
                                                    shipcrew(1) := c
                                             else
                                                    shipcrew(2) := c
                                             end-if
                                       end-if
                               else
                                       if(shipcrew(1) + shipcrew(2) + shipcrew(3) =
                                           0) then
                                             rand_no := random
                                             if(rand_no < (1/3)) then
                                                    shipcrew(1) := c
                                             else
                                                    if(rand_no < (2/3)) then
                                                            shipcrew(2) := c
                                                    else
                                                            shipcrew(3) := c
                                                    end-if
                                             end-if
                                       else
```

```
                                          if(shipcrew(1) + shipcrew(2) +
                                              shipcrew(3) = 1) then
                                                 rand_no := random
                                                 if(rand_no < 0.5) then
                                                        if(shipcrew(1) = 0) then
                                                               shipcrew(1) := c
                                                        else
                                                               shipcrew(2) := c
                                                        end-if
                                                 else
                                                        if(shipcrew(3) = 0) then
                                                               shipcrew(3) := c
                                                        else
                                                               shipcrew(2) := c
                                                        end-if
                                                 end-if
                                          else
                                                 forall(i in 1..3 | shipcrew(i)
                                                     = 0) shipcrew(i) := c
                                          end-if
                                   end-if
                            end-if
                     end-do

!                    write("The 'shipcrew' for role ",r,", in order, are:\t")
!                    forall(i in 1..3) write(shipcrew(i)," ")
!                    write("\n")


                     time_count := 0
                     previous := 0
                     start_in_role(r) := 0

                     if(start_duration(r) > 0) then
                            task_no := task_no +1
                            task_start(task_no) := 0
                            task_duration(task_no) := start_duration(r)
                            task_vessel(task_no) := v
                            task_role(task_no) := r

                            previous := previous +1
                            start_in_role(r) := shipcrew(previous)
                            if(shipcrew(previous) > 0) then
```

```
                work_resource_zero(shipcrew(previous)) := (
                    vessel_max_work(v) - start_duration(r))
                initial(shipcrew(previous), task_no) := 1
                ag_initial(task_no) := 0
        end-if

        time_count := time_count + start_duration(r)

else
        previous := previous +1
        start_in_role(r) := shipcrew(previous)
        if(shipcrew(previous) > 0) then
                work_resource_zero(shipcrew(previous)) :=
                    vessel_max_work(v)
        end-if
end-if


! Calculate the backwards-worked rest periods required...
more_previous := previous
if(start_duration(r) = 0) then
        if(shipcrew(more_previous) > 0) then
                rest_resource_zero(shipcrew(more_previous))
                    := 1
        end-if
end-if

back_count := vessel_max_work(v) - start_duration(r)
while(back_count < min_rest) do
        if(more_previous = 1) then
                more_previous := no_of_regulars_role(r)
        else
                more_previous := more_previous - 1
        end-if

        if(shipcrew(more_previous) > 0) then
                initial_rest_required(shipcrew(more_previous)
                    ) := min_rest - back_count
        end-if

        back_count := back_count + vessel_max_work(v)
end-do
```

```
                while(time_count < no_of_weeks) do
                        task_no := task_no +1
                        task_start(task_no) := time_count

                        if(time_count + vessel_max_work(v) < no_of_weeks)
                            then
                                task_duration(task_no) := vessel_max_work(v)
                        else
                                task_duration(task_no) := no_of_weeks -
                                    time_count
                        end-if

                        task_vessel(task_no) := v
                        task_role(task_no) := r


                        if(previous = no_of_regulars_role(r)) then
                                previous := 1
                        else
                                previous := previous +1
                        end-if
                        if(shipcrew(previous) > 0) then
                                if(task_start(task_no) < no_of_weeks-1) then
                                        initial(shipcrew(previous), task_no)
                                            := 1
                                end-if
                                ag_initial(task_no) := 0
                        end-if

                        time_count := time_count + vessel_max_work(v)
                end-do
        end-do
end-do


!       writeln
!       writeln("Total number of tasks is ",no_of_tasks)
!       writeln("... and the set of tasks are as follows:")
!       forall(t in tasks) do
!               if(t < 10) then
!                       write("Task ",t,": ")
!               else
!                   write("Task ",t,":")
!               end-if
!
!               if(task_vessel(t) < 10) then
!                   write("   Vessel ",task_vessel(t)," ")
!               else
!                   write("   Vessel ",task_vessel(t)," ")
!               end-if
!
!               if(task_role(t) < 10) then
!                   write("   Role ",task_vessel(t)," (")
!               else
!                   write("   Role ",task_vessel(t)," (")
!               end-if
!
!
!               write(vessel_label(task_vessel(t)),")\tStart: ",task_start(t),"\
   tDuration: ",task_duration(t),"\tEnd: ",task_start(t)+task_duration(t),"\t\
   tInitially assigned: ")
!               forall(c in crew | initial(c,t) = 1) do
!                   write(crew_label(c))
!               end-do
!               write("\n")
!
!       end-do
!       writeln
!       writeln



        !----------------------------------------------------------------------
        ! Generate employee availabilities:
        declarations
                days = 0..(7*no_of_weeks)
                available_day: array(crew,days) of integer
                available_task: array(crew,tasks) of integer
                available_sum: integer
        !       ill_given_prev_fit: real
        !       ill_given_prev_ill: real
                time_reduction: real
        end-declarations
```

```
forall(c in crew) do

        rand_no := random
        if(rand_no < 0.008) then
                available_day(c,0) := 0
        else
                available_day(c,0) := 1
        end-if

        forall(d in days | d > 0) do

                time_reduction := 0
                if(use_time_reduction = TRUE) then
                        if(d > 28) then
                                time_reduction := (d-28)/((7*no_of_weeks)-28)
                        end-if
                end-if

                rand_no := random
                if(available_day(c,d-1) = 1) then
                        if(rand_no < ill_given_prev_fit*(1-time_reduction))
                            then
                                available_day(c,d) := 0
                        else
                                available_day(c,d) := 1
                        end-if
                else
                        if(rand_no < ill_given_prev_ill*(1-time_reduction))
                            then
                                available_day(c,d) := 0
                        else
                                available_day(c,d) := 1
                        end-if
                end-if
        end-do

        forall(t in tasks) do
                available_sum := 0
                forall(d in ((7*task_start(t))+1)..(7*(task_start(t) +
                    task_duration(t)))) do
                        available_sum := available_sum + available_day(c,d)
                end-do
                if(available_sum = (7*task_duration(t))) then
                        available_task(c,t) := 1
                else
                        available_task(c,t) := 0
                end-if
        end-do



        ! Must also factor in the 'initial rest periods' required
        !       - no employee can be eligible for a task in the first r
            periods of the planning horizon
        if(initial_rest_required(c) > 0) then
                forall(t in tasks | task_start(t) <= initial_rest_required(c
                    )) do
                        available_task(c,t) := 0
                end-do
        end-if

end-do


!-----------------------------------------------------------------------
! Generate costs and fixed-contract terms:

declarations
        change_cost: array(crew, tasks) of real
        ag_change_cost: array(tasks) of real

        board_addition: real
        depart_addition: real
        working_addition: real

        disruption_factor: real


        under_rate, over_rate: array(crew) of real
        current_excess, exp_worktime: array(crew) of real
        current_worktime: array(crew) of real
        g_weeks: real
end-declarations
```

```
forall(t in tasks) do
        forall(c in crew) do

                board_addition := 0
                if(initial(c,t) = 0) then
                        if(crew_continent(c) = "European") then
                                if(vessel_location(task_vessel(t)) = "Europe
                                    ") then board_addition := 3
                                elif(vessel_location(task_vessel(t)) = "
                                    Africa" or vessel_location(task_vessel(t)
                                    ) = "USA") then board_addition := 6
                                else board_addition := 8
                                end-if
                        elif(crew_continent(c) = "North American") then
                                if(vessel_location(task_vessel(t)) = "USA")
                                    then board_addition := 4
                                elif(vessel_location(task_vessel(t)) = "
                                    Brazil" or vessel_location(task_vessel(t)
                                    ) = "Europe") then board_addition := 6
                                else board_addition := 9
                                end-if
                        elif(crew_continent(c) = "Asian" or crew_continent(c
                            ) = "Australasian") then
                                if(vessel_location(task_vessel(t)) = "AsiaPac
                                    ") then board_addition := 4
                                else board_addition := 9
                                end-if
                        else board_addition := 9
                        end-if

                else
                        if(task_start(t) > 0) then
                                if(crew_continent(c) = "European") then
                                        if(vessel_location(task_vessel(t)) = "
                                            Europe") then board_addition := -2
                                        elif(vessel_location(task_vessel(t)) =
                                             "Africa" or vessel_location(
                                            task_vessel(t)) = "USA") then
                                            board_addition := -5
                                        else board_addition := -7
                                        end-if
                                elif(crew_continent(c) = "North American")
                                    then
                                        if(vessel_location(task_vessel(t)) = "
                                            USA") then board_addition := -3
                                        elif(vessel_location(task_vessel(t)) =
                                             "Brazil" or vessel_location(
                                            task_vessel(t)) = "Europe") then
                                            board_addition := -5
                                        else board_addition := -8
                                        end-if
                                elif(crew_continent(c) = "Asian" or
                                    crew_continent(c) = "Australasian") then
                                        if(vessel_location(task_vessel(t)) = "
                                            AsiaPac") then board_addition :=
                                            -3
                                        else board_addition := -8
                                        end-if
                                else board_addition := -8
                                end-if

                                if(task_start(t) < 5) then
                                        board_addition := board_addition/2
                                end-if
                        end-if
                end-if


                depart_addition := 0
                if(initial(c,t) = 0) then
                        if(crew_continent(c) = "European") then
                                if(vessel_location(task_vessel(t)) = "Europe
                                    ") then depart_addition := 3
                                elif(vessel_location(task_vessel(t)) = "
                                    Africa" or vessel_location(task_vessel(t)
                                    ) = "USA") then depart_addition := 6
                                else depart_addition := 8
                                end-if
                        elif(crew_continent(c) = "North American") then
                                if(vessel_location(task_vessel(t)) = "USA")
                                    then depart_addition := 4
                                elif(vessel_location(task_vessel(t)) = "
                                    Brazil" or vessel_location(task_vessel(t)
                                    ) = "Europe") then depart_addition := 6
                                else depart_addition := 9
                                end-if
                         elif(crew_continent(c) = "Asian" or crew_continent(c
                             ) = "Australasian") then
                                if(vessel_location(task_vessel(t)) = "AsiaPac
                                    ") then depart_addition := 4
                                else depart_addition := 9
                                end-if
                         else depart_addition := 9
                         end-if

                 else
                         if(crew_continent(c) = "European") then
                                if(vessel_location(task_vessel(t)) = "Europe
                                    ") then depart_addition := -2
                                elif(vessel_location(task_vessel(t)) = "
                                    Africa" or vessel_location(task_vessel(t)
                                    ) = "USA") then depart_addition := -5
                                else depart_addition := -7
                                end-if
                         elif(crew_continent(c) = "North American") then
                                if(vessel_location(task_vessel(t)) = "USA")
                                    then depart_addition := -3
                                elif(vessel_location(task_vessel(t)) = "
                                    Brazil" or vessel_location(task_vessel(t)
                                    ) = "Europe") then depart_addition := -5
                                else depart_addition := -8
                                end-if
                         elif(crew_continent(c) = "Asian" or crew_continent(c
                             ) = "Australasian") then
                                if(vessel_location(task_vessel(t)) = "AsiaPac
                                    ") then depart_addition := -3
                                else depart_addition := -8
                                end-if
                         else depart_addition := -8
                         end-if

                         if((task_start(t)+task_duration(t)) < 5) then
                                depart_addition := depart_addition/2
                         end-if
                 end-if


                 working_addition := 0
                 if(contract_type(c) = "Permanent") then
```

```
                working_addition := 0
        else
                if(crew_nation(c) = "NORWAY") then
                        if(task_duration(t) < vessel_max_work(
                            task_vessel(t)) and task_start(t) +
                            task_duration(t) = 7*no_of_weeks) then
                                if(initial(c,t) = 1) then
                                        working_addition := -90*
                                            vessel_max_work(task_vessel
                                            (t))
                                else
                                        working_addition := 100*
                                            vessel_max_work(task_vessel
                                            (t))
                                end-if
                        else
                                if(initial(c,t) = 1) then
                                        working_addition := -90*
                                            task_duration(t)
                                else
                                        working_addition := 100*
                                            task_duration(t)
                                end-if
                        end-if

                else
                        if(task_duration(t) < vessel_max_work(
                            task_vessel(t)) and task_start(t) +
                            task_duration(t) = 7*no_of_weeks) then
                                if(initial(c,t) = 1) then
                                        working_addition := -60*
                                            vessel_max_work(task_vessel
                                            (t))
                                else
                                        working_addition := 70*
                                            vessel_max_work(task_vessel
                                            (t))
                                end-if
                        else
                                if(initial(c,t) = 1) then
                                        working_addition := -60*
                                            task_duration(t)
                                else
```

```
                                        working_addition := 70*
                                            task_duration(t)
                                end-if
                        end-if
                end-if
        end-if


        change_cost(c,t) := board_addition + depart_addition +
            working_addition

        if(task_start(t) < 5) then
                if(change_cost(c,t) > 0) then
                        change_cost(c,t) := disruption_factor_near*
                            change_cost(c,t)
                elif(change_cost(c,t) < 0) then
                        change_cost(c,t) := (1/disruption_factor_near
                            )*change_cost(c,t)
                end-if
        elif(task_start(t) < (no_of_weeks-1)) then
                if(change_cost(c,t) > 0) then
                        change_cost(c,t) := disruption_factor_long*
                            change_cost(c,t)
                elif(change_cost(c,t) < 0) then
                        change_cost(c,t) := (1/disruption_factor_long
                            )*change_cost(c,t)
                end-if
        end-if

end-do


board_addition := 0
if(ag_initial(t) = 0) then
        board_addition := 2
else
        if(task_start(t) > 4) then
                board_addition := -2
        elif(task_start(t) > 0) then
                board_addition := -1
        end-if
end-if
```

```
        depart_addition := 0
        if(ag_initial(t) = 0) then
               depart_addition := 2
        else
               if((task_start(t) + task_duration(t)) > 4) then
                      depart_addition := -2
               else
                      depart_addition := -1
               end-if
        end-if


        working_addition := 0
        if(task_duration(t) < vessel_max_work(task_vessel(t)) and
            task_start(t) + task_duration(t) = 7*no_of_weeks) then
               if(ag_initial(t) = 0) then
                      working_addition := 200*vessel_max_work(task_vessel(
                          t))
               else
                      working_addition := -180*vessel_max_work(task_vessel
                          (t))
               end-if
        else
               if(ag_initial(t) = 0) then
                      working_addition := 200*task_duration(t)
               else
                      working_addition := -180*task_duration(t)
               end-if
        end-if


        ag_change_cost(t) := board_addition + depart_addition +
            working_addition

        if(ag_change_cost(t) > 0) then
               ag_change_cost(t) := agency_penalty*ag_change_cost(t)
        elif(ag_change_cost(t) < 0) then
               ag_change_cost(t) := (1/agency_penalty)*ag_change_cost(t)
        end-if

end-do
```

```
forall(c in crew | contract_type(c) = "Permanent") under_rate(c) := 70
forall(c in crew | contract_type(c) = "Permanent") do
       if(legal_entity(c) = "Subsea7 Contr (Norway) AS") then over_rate(c)
            := 500
       elif(legal_entity(c) = "Subsea 7 Norway") then over_rate(c) := 500
       elif(crew_nation(c) = "Norway") then over_rate(c) := 120
       else over_rate(c) := 70
       end-if
end-do


g_weeks := 26
forall(c in crew | contract_type(c) = "Permanent") do
       exp_worktime(c) := 26 - sum(t in tasks)(initial(c,t)*task_duration(
           t))
end-do


forall(c in crew | contract_type(c) = "Permanent") do
       current_worktime(c) := exp_worktime(c) + sum(t in tasks)(initial(c,
           t)*task_duration(t)*available_task(c,t))
end-do


forall(c in crew | contract_type(c) = "Permanent") do
       current_excess(c) := 0
       if(current_worktime(c) < 26) then
              current_excess(c) := under_rate(c)*(26-current_worktime(c))
       end-if
       if(current_worktime(c) > 26) then
              current_excess(c) := over_rate(c)*(current_worktime(c)-26)
       end-if
end-do



!-----------------------------------------------------------------------
! Print dataset into output file:
fopen(OUTFILE, F_OUTPUT)
       writeln("! Dataset generated for task-based formulation based on
           real data on Subsea 7 Captains.")
       writeln
```

```
        writeln("! Parameters used:")
        writeln
        writeln("! -> Probability employee ill given previously fit:\t",
            ill_given_prev_fit)
        writeln("! -> Probability employee ill given previously ill:\t",
            ill_given_prev_ill)
        if(use_time_reduction = TRUE) then writeln("! -> Time-reduction of
            probabilities:\t\t\tON")
        else writeln("! -> Time-reduction of probabilities:\t\t\tOFF")
        end-if
        writeln
        writeln("! -> Disruption factor for near tasks:\t\t\t",
            disruption_factor_near)
        writeln("! -> Disruption factor for more distant tasks:\t\t",
            disruption_factor_long)
        writeln("! -> Agency crew penalty:\t\t\t\t",agency_penalty)
        writeln
        writeln("!---------------------------------------------")

        writeln
        writeln
        writeln("Max_OF: 1000000")
        writeln("Max_Sol: 15")
        writeln
        writeln("DAYS_TO_PLAN: ",no_of_weeks*7)
        writeln
        writeln("JOBS_TO_PLAN: ",no_of_tasks)
        writeln("LINKED_SETS: 0")
        writeln

        if(PROJECTS = FALSE) then
               writeln("PROJECT_NUMBER: 0")
!       else
!              will have to calculate number of projects, not required here
        end-if

        writeln
        write("EMP : [")
        forall(c in crew) write(" '",crew_label(c),"' ")
        write("] \n")
        writeln
        write("GUARANTEED: [")
        forall(c in crew | contract_type(c) = "Permanent") write(" '",
            crew_label(c),"'")
        write("] \n")
        writeln

        write("start_time: [")
        forall(t in tasks) write(" ",7*task_start(t))
        write("] \n")
        write("duration: [")
        forall(t in tasks) write(" ",7*task_duration(t))
        write("] \n")
        writeln
        write("linked_matrix: [")
        forall(t in tasks) write(" 0")
        write("] \n")
        writeln

        write("min_rest: [")
        forall(c in crew) write(" ",min_rest)
        write("] \n")
        write("max_work: [")
        forall(c in crew) write(" ",max_work)
        write("] \n")
        writeln

        writeln("eligible: [")
        forall(c in crew) do
               forall(t in tasks) do
                      if(task_start(t) = 0 and initial(c,t) = 1) then
                             write(" 1")
                      else
                             write(" ",available_task(c,t))
                      end-if
               end-do
               write("\n")
        end-do
        write("] \n")
        writeln
        write("AG_eligible: [")
        forall(t in tasks) write(" 1")
        writeln("]")
        writeln
        writeln
```

```
write("work_resource_zero: [")
forall(c in crew) write(" ",work_resource_zero(c))
writeln("]")
write("rest_resource_zero: [")
forall(c in crew) write(" ",rest_resource_zero(c))
writeln("]")
writeln
writeln

writeln("initial: [")
forall(c in crew) do
       forall(t in tasks) write(" ",initial(c,t)*available_task(c,t
           ))
       write("\n")
end-do
writeln("]")
writeln
writeln("AG_initial: [")
forall(t in tasks) write(" ",ag_initial(t))
writeln("]")
writeln
writeln

writeln("change_cost: [")
forall(c in crew) do
       forall(t in tasks) write(change_cost(c,t),"\t")
       write("\n")
end-do
writeln("]")
writeln
writeln("AG_change_cost: [")
forall(t in tasks) write(ag_change_cost(t),"\t")
writeln("]")
writeln
writeln

write("under_rate:\t[")
forall(c in crew | contract_type(c) = "Permanent") write(round(
    under_rate(c)/7),"\t")
write("]\nover_rate:\t[")
forall(c in crew | contract_type(c) = "Permanent") write(round(
    over_rate(c)/7),"\t")
```

```
               write("]\ncurrent_excess:\t[")
               forall(c in crew | contract_type(c) = "Permanent") write(
                   current_excess(c),"\t")
               write("]\ng_days:\t\t[")
               forall(c in crew | contract_type(c) = "Permanent") write(g_weeks
                   *7,"\t")
               write("]\nexp_worktime:\t[")
               forall(c in crew | contract_type(c) = "Permanent") write(
                   exp_worktime(c)*7,"\t")
               write("]\n")
               writeln
               writeln

               write("project_matrix: [")
               forall(t in tasks) write(" 0")
               write("]\n")
               writeln
               write("experience: [")
               forall(c in crew) do
                      forall(t in tasks) write(" 0")
                      write("\n")
               end-do
               writeln("]")
               writeln
               write("AG_experience: [")
               forall(t in tasks) write(" 0")
               writeln("]")
               writeln
               writeln("min_experience: [ 0 ]")
               writeln


       fclose(F_OUTPUT)

end-procedure


!-----------------------------------------------

!forall(ins in INSTANCES) do
!
!       OUTFILE := instance_name(ins)
!
```

```
!       ! Parameters to be varied for various data sets:
!       ill_given_prev_fit := ill_given_fit(ins)
!       ill_given_prev_ill := ill_given_ill(ins)
!
!       use_time_reduction := use_TR(ins)
!
!       disruption_factor_near := DF_near(ins)
!       disruption_factor_long := DF_long(ins)
!       agency_penalty := AG_pen(ins)
!
        MAIN_PROG
!
!end-do


end-model
```

E.1.2 Cost-minimization algorithm

Here we give the code used to solve the Task-Based problem using the Cost-Minimization approach described in section 5.5.1.1. This was implemented using the FICO Xpress software.

```
model ModelName
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver

! Model for the Subsea 7 problem, using formulation as set out in 2nd Year Review
     document
! This program aims to solve the problem directly from this formulation

parameters
        DATE = "10-02-14"
        PARAMETERFILE = "batch-input-parameters.dat"
end-parameters

declarations
        InstanceName: string
end-declarations

initializations from PARAMETERFILE
        InstanceName
end-initializations
```

```
DATAFILE := InstanceName+"\\Task-Based - Captains - "+InstanceName+".txt"
LOGFILE := InstanceName+"\\Logfile - TB 2min - "+InstanceName+" - "+DATE+".txt"
SUMMARYFILE := "Results - TB 2min - "+DATE+".txt"


! Set parameters relating to running time and acceptable optimality gap:
setparam("XPRS_maxtime", -120)
setparam("XPRS_miprelstop",0.05)
! Set parameters for numbers of Cover and Gomory cuts to apply:
setparam("XPRS_covercuts",30)
setparam("XPRS_gomcuts",10)
!
! Use the above settings for the two-minute time limits.
!
! For ten-minute time-limits, use the following:
!       setparam("XPRS_maxtime", -600)
!       setparam("XPRS_miprelstop",0.05)
!       setparam("XPRS_covercuts",50)
!       setparam("XPRS_gomcuts",10)
!
! ... or for the extended settings, use the following:
!       setparam("XPRS_maxtime", -21600)
!       (note - do not adjust parameter XPRS_miprelstop from the default)
!       setparam("XPRS_covercuts",1000)
! setparam("XPRS_gomcuts",1000)


prog_starttime := gettime           ! get the time so that at the end, running
    time can be calculated

declarations
       EMP: set of string                  ! Employee names / numbers
       GUARANTEED:   set of string ! Set of employees on guaranteed days
           contracts
       DAYS_TO_PLAN: integer
       JOBS_TO_PLAN: integer
       PROJECT_NUMBER: integer
       status: array({XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH}) of string !
           Used to indicate the solution status of the problem
end-declarations
```

```
initializations from DATAFILE
       EMP GUARANTEED DAYS_TO_PLAN JOBS_TO_PLAN PROJECT_NUMBER
end-initializations


if(PROJECT_NUMBER = 0) then THEORETIC_PROJECTS := 1
else THEORETIC_PROJECTS := PROJECT_NUMBER
end-if

declarations
       JOB = 1..JOBS_TO_PLAN
       PROJECT = 1..THEORETIC_PROJECTS
       eligible: array(EMP, JOB) of real   ! = 1 if employee can carry out a job,
            0 otherwise
       AG_eligible: array(JOB) of real     ! eligibility of agency crew
       change_cost: array(EMP, JOB) of real ! The cost of an employee carrying
           out a job
       AG_change_cost: array(JOB) of real         ! Cost details for agency crew
       start_time, duration: array(JOB) of real
       rest_duration_list: set of real
       REST_NUMBER: integer
       initial: array(EMP, JOB) of real    ! = 1 if employee is currently
           allocated to a task, 0 otherwise
       AG_initial: array(JOB) of real
       under_rate, over_rate, current_excess, g_days, exp_worktime: array(
           GUARANTEED) of real
       project_matrix: array(PROJECT, JOB) of integer
       project_set: array(PROJECT) of set of integer
       experience: array(EMP, JOB) of real
       AG_experience: array(JOB) of real
       min_experience: array(PROJECT) of real
       min_rest: array(EMP) of real
       max_work: array(EMP) of real

       undertime, overtime: array(GUARANTEED) of mpvar
       change: dynamic array(EMP, JOB) of mpvar
       AG_change: dynamic array(JOB) of mpvar
       new_sched: dynamic array(EMP, JOB) of mpvar
       AG_new_sched: dynamic array(JOB) of mpvar
end-declarations

initializations from DATAFILE
```

```
       eligible AG_eligible change_cost AG_change_cost start_time duration
           initial AG_initial
       under_rate over_rate current_excess g_days exp_worktime
       project_matrix experience AG_experience min_experience min_rest max_work
end-initializations


!-----------------------------------------------------------------------
! Define the change and new_sched (ie y and z) variables ONLY for the
! tasks for which each crew member is eligible
!writeln("Define 'change' and 'new sched' variables:")
forall(j in JOB) do
       forall(i in EMP) do
              if eligible(i,j) = 1 then
!                     writeln("Create variables for task ",j," for ",i)
                     create(change(i,j))
                     create(new_sched(i,j))
              end-if
       end-do
       if AG_eligible(j) = 1 then
!             writeln("Create variables for task ",j," for agency")
              create(AG_change(j))
              create(AG_new_sched(j))
       end-if

!      writeln
end-do

!-----------------------------------------------------------------------
! Define 'rest' tasks such that they start directly after each work task has
    ended
! (subject to them finishing before the time-window ends)
!writeln("Create rest tasks:")
rest_duration_list := {}
forall(i in EMP) rest_duration_list += {7*min_rest(i)}
!forall(i in EMP) writeln(i," requires a minimum rest period of length ",min_rest
    (i))
!writeln

declarations
       rest_start_list: array(rest_duration_list) of set of real
end-declarations
```

```
REST_NUMBER := 0

forall(j in JOB, r in rest_duration_list) do
       if start_time(j) + duration(j) < (DAYS_TO_PLAN - r + 1) then
              rest_start_list(r) += {start_time(j) + duration(j)}
!             writeln("Task ",j," ends at time ",start_time(j) + duration(j),",
    and we can add a rest task of length ",r," starting at this time")
       end-if
end-do
!writeln
!writeln("The 'rest start list' contains the following:")
!forall(r in rest_duration_list) writeln("Of duration ",r,", starting at: ",
    rest_start_list(r))

forall(r in rest_duration_list) REST_NUMBER := REST_NUMBER + getsize(
    rest_start_list(r))
!writeln("... and so REST_NUMBER = ",REST_NUMBER)
!writeln

declarations
       REST = 1..REST_NUMBER
       rest_start_time, rest_duration: array(REST) of real
       rest_new_sched: dynamic array(EMP, REST) of mpvar
       work_content_work_arc, rest_content_work_arc: array(JOB) of real
       work_content_rest_arc, rest_content_rest_arc: array(REST) of real
end-declarations

index := 0
forall(r in rest_duration_list) do
       forall(s in rest_start_list(r)) do
              index := index + 1
              rest_start_time(index) := s
              rest_duration(index) := r
!             writeln("Rest task ",index," starts at time ",rest_start_time(index
    )," and is of length ",rest_duration(index))
       end-do
end-do
!writeln


! Note that employees can only be assigned a rest task of length equal to their
    required minimum rest period
forall(r in REST, i in EMP) do
```

```
        if(rest_duration(r) = 7*min_rest(i)) then
                create(rest_new_sched(i,r))
!               writeln("Rest task: ",r,"\tDuration: ",rest_duration(r),"\tEmployee
    : ",i,"\tMinRest: ",min_rest(i),"\t=> Create variable")
        end-if
end-do



!------------------------------------------------------------------------
! Can now define work and rest content for work and rest arcs
!writeln("Define work and rest contents:")
forall(j in JOB) do
        work_content_work_arc(j) := duration(j)
        rest_content_work_arc(j) := 1
!       rest_content_work_arc(j) := (duration(j))/14
!       writeln("For task ",j,", work content = ",work_content_work_arc(j)," and
    rest content = ",rest_content_work_arc(j))
end-do


forall(r in REST) do
        work_content_rest_arc(r) := -DAYS_TO_PLAN
        rest_content_rest_arc(r) := -1
!       writeln("For rest task ",r,", work content = ",work_content_rest_arc(r),"
    and rest content = ",rest_content_rest_arc(r))
end-do
!writeln

!------------------------------------------------------------------------
! Next we will generate the sets C_beta of overlapping tasks
!writeln("Define the sets of overlapping tasks:")
declarations
        beta: integer
        time_points: set of real
        time_points_ordered: list of real
        NUMBER_OF_POINTS: integer
        min_time: real
        beta_index: range
        overlap_set_work, overlap_set_rest: dynamic array(beta_index) of set of
            integer
        current_set_work, current_set_rest: set of integer
end-declarations
```

```
time_points := {}
forall(j in JOB) do
       time_points += {start_time(j)}
       time_points += {start_time(j) + duration(j)}
end-do
forall(j in REST) do
       time_points += {rest_start_time(j)}
       time_points += {rest_start_time(j) + rest_duration(j)}
end-do
!writeln("The set of time points is as follows:",time_points)

time_points_ordered := []
NUMBER_OF_POINTS := getsize(time_points)
forall(i in 1..NUMBER_OF_POINTS) do
       min_time := DAYS_TO_PLAN
       forall(p in time_points) do
              if(p < min_time) then min_time := p; end-if
       end-do
       time_points_ordered += [min_time]
       time_points -= {min_time}
end-do
!writeln("Giving the following ordered set:",time_points_ordered)
!writeln

!writeln("Running algorithm:")
current_set_work := {}
current_set_rest := {}
beta := 1
constraint_required := FALSE
add_constraint := FALSE
forall(t in time_points_ordered) do
!      writeln("\t dealing with time point ",t)
       ! check to see if another constraint is required
       forall(j in JOB | start_time(j) + duration(j) = t) add_constraint := TRUE
!      forall(j in JOB | start_time(j) + duration(j) = t) writeln("\t\t job ",j,"
     ends at time ",t)
       forall(j in REST | rest_start_time(j) + rest_duration(j) = t)
           add_constraint := TRUE
!      forall(j in REST | rest_start_time(j) + rest_duration(j) = t) writeln("\t\
    t rest task ",j," ends at time ",t)
       ! if so, we create an 'overlap set' and add the current contents to it
       if(constraint_required = TRUE AND add_constraint = TRUE) then
```

```
               create(overlap_set_work(beta))
               create(overlap_set_rest(beta))
               overlap_set_work(beta) := current_set_work
               overlap_set_rest(beta) := current_set_rest
!              writeln("\t\t... so we create the overlap set for beta = ",beta)
!              writeln("\t\t\t containing working tasks ",overlap_set_work(beta),"
     and rest tasks ",overlap_set_rest(beta))
               beta := beta + 1
               add_constraint := FALSE
               constraint_required := FALSE
        end-if

        ! remove these tasks from the current set
        forall(j in JOB | start_time(j) + duration(j) = t) current_set_work -= {j}
        forall(j in REST | rest_start_time(j) + rest_duration(j) = t)
            current_set_rest -= {j}

        ! if there are new tasks starting at this point, add them to the current
            set and
        ! indicate that a contraint will now be required again
        forall(j in JOB | start_time(j) = t) do
               constraint_required := TRUE
               current_set_work += {j}
!              writeln("\t\t Task ",j," starts at time ",t,", so add it to the
    current set")
        end-do
        forall(j in REST | rest_start_time(j) = t) do
               constraint_required := TRUE
               current_set_rest += {j}
!              writeln("\t\t Rest task ",j," starts at time ",t,", so add it to
    the current set")
        end-do
end-do
!writeln

!-----------------------------------------------------------------------
! Define project_sets from project_matrix data


forall(p in PROJECT) do
        project_set(p) := {}
        forall(j in JOB) do
               if(project_matrix(p,j) = 1) then project_set(p) += {j}; end-if
        end-do
end-do

!       writeln("Define the project sets:")
!       forall(p in PROJECT) writeln("Project ",p," contains tasks ",project_set(p
    ))
!       writeln


!----------------------------------------------------------------------
! Define Ordered Task Set B (of ALL tasks - work, rest)
!writeln("Define the ordered task set of ALL tasks")
declarations
        order_index = 1..(JOBS_TO_PLAN + REST_NUMBER)
        ordered_task_number: array(order_index) of integer
        ordered_task_work, ordered_task_rest: array(order_index) of integer
        work_resource, rest_resource: array(EMP, order_index) of mpvar
        work_resource_zero, rest_resource_zero: array(EMP) of real
        work_unproc, rest_unproc: set of integer
        work_proc, rest_proc: set of integer
        b: integer
end-declarations

initializations from DATAFILE
        work_resource_zero rest_resource_zero
end-initializations


forall(j in JOB) work_unproc += {j}
forall(j in REST) rest_unproc += {j}
!       writeln("List of unprocessed tasks (in terms of ordering):")
!       writeln("Work tasks: ",work_unproc)
!       writeln("Rest tasks: ",rest_unproc)
!       writeln

b := 0
forall(i in 1..(JOBS_TO_PLAN + REST_NUMBER)) do
        min_time := DAYS_TO_PLAN
        forall(j in work_unproc) do
                if(start_time(j) < min_time) then min_time := start_time(j); end-if
        end-do
        forall(j in rest_unproc) do
```

```
                if(rest_start_time(j) < min_time) then min_time := rest_start_time(
                    j); end-if
        end-do
!       writeln("Earliest start time is ",min_time)

        work_proc := {}
        forall(j in work_unproc) do
                if(start_time(j) = min_time) then
                        b := b+1
!                       writeln(b," - WORK")
                        ordered_task_number(b) := j
                        ordered_task_work(b) := 1
                        ordered_task_rest(b) := 0
                        work_proc += {j}
!                       writeln("\t Work task ",j," starts at time ",min_time," -
    assign this as task number ",b)
                end-if
        end-do
        work_unproc -= work_proc
!       writeln("\t\t The following work tasks have now been processed: ",
    work_proc)
!       writeln("\t\t ... leaving the following still to be processed: ",
    work_unproc)

        rest_proc := {}
        forall(j in rest_unproc) do
                if(rest_start_time(j) = min_time) then
                        b := b + 1
!                       writeln(b," - REST")
                        ordered_task_number(b) := j
                        ordered_task_work(b) := 0
                        ordered_task_rest(b) := 1
                        rest_proc += {j}
!                       writeln("\t Rest task ",j," starts at time ",min_time," -
    assign this as task number ",b)
                end-if
        end-do
        rest_unproc -= rest_proc
!       writeln("\t\t The following rest tasks have now been processed: ",
    rest_proc)
!       writeln("\t\t ... leaving the following still to be processed: ",
    rest_unproc)
```

```
end-do


!------------------------------------------------------------------------
! Now that all quantities have been defined, we can state all the contraints:
declarations
       TOTAL_COST: linctr
       job_cover_constr: array(JOB) of linctr
       overlap_constr: array(EMP,beta_index) of linctr
       experience_constr: array(PROJECT) of linctr
       work_resource_constr_1, rest_resrouce_constr_1: array(EMP) of linctr
       work_resource_constr_b, rest_resource_constr_b: array(EMP, order_index) of
            linctr
       work_resource_bound_constr, rest_resource_bound_constr: array(EMP,
           order_index) of linctr
       undertime_constr, overtime_constr: array(GUARANTEED) of linctr
       new_sched_constr: array(EMP, JOB) of linctr
       AG_new_sched_constr: array(JOB) of linctr
end-declarations



! Objective:
TOTAL_COST := sum(i in EMP, j in JOB)(change_cost(i,j) * change(i,j)) + sum(j in
    JOB)(AG_change_cost(j) * AG_change(j)) + sum(i in GUARANTEED)((over_rate(i) *
     overtime(i)) + (under_rate(i) * undertime(i)) - current_excess(i))

! Other constraints:
forall(j in JOB) job_cover_constr(j) := sum(i in EMP)(new_sched(i,j)) +
    AG_new_sched(j) = 1

forall(i in EMP, a in beta_index) overlap_constr(i,a) := sum(j in
    overlap_set_work(a))(new_sched(i,j)) + sum(j in overlap_set_rest(a))(
    rest_new_sched(i,j)) <= 1

forall(k in PROJECT) experience_constr(k) := sum(i in EMP, j in project_set(k))(
    experience(i,j) * new_sched(i,j)) + sum(j in project_set(k))(AG_experience(j)
     * AG_new_sched(j)) >= min_experience(k)

forall(i in EMP) do
! Revised version of inner part of this loop:
!---------------------------------------------------------------
       forall(o in order_index) do
```

```
if(o = 1) then
       if(ordered_task_work(1) = 1) then
              work_resource_constr_b(i,o) := 7*work_resource_zero(
                  i) + (new_sched(i,ordered_task_number(1)) *
                  work_content_work_arc(ordered_task_number(1))) <=
                   work_resource(i,1)
              rest_resource_constr_b(i,o) := rest_resource_zero(i)
                   + (new_sched(i,ordered_task_number(1)) *
                  rest_content_work_arc(ordered_task_number(1))) <=
                   rest_resource(i,1)
       end-if
       if(ordered_task_rest(1) = 1) then
              work_resource_constr_b(i,o) := 7*work_resource_zero(
                  i) + (rest_new_sched(i,ordered_task_number(1)) *
                  work_content_rest_arc(ordered_task_number(1))) <=
                   work_resource(i,1)
              rest_resource_constr_b(i,o) := rest_resource_zero(i)
                   + (rest_new_sched(i,ordered_task_number(1)) *
                  rest_content_rest_arc(ordered_task_number(1))) <=
                   rest_resource(i,1)
       end-if
else
       if(ordered_task_work(o) = 1) then
              work_resource_constr_b(i,o) := work_resource(i,(o-1)
                  ) + (new_sched(i,ordered_task_number(o)) *
                  work_content_work_arc(ordered_task_number(o))) <=
                   work_resource(i,o)
              rest_resource_constr_b(i,o) := rest_resource(i,(o-1)
                  ) + (new_sched(i,ordered_task_number(o)) *
                  rest_content_work_arc(ordered_task_number(o))) <=
                   rest_resource(i,o)
       end-if
       if(ordered_task_rest(o) = 1) then
              work_resource_constr_b(i,o) := work_resource(i,(o-1)
                  ) + (rest_new_sched(i,ordered_task_number(o)) *
                  work_content_rest_arc(ordered_task_number(o))) <=
                   work_resource(i,o)
              rest_resource_constr_b(i,o) := rest_resource(i,(o-1)
                  ) + (rest_new_sched(i,ordered_task_number(o)) *
                  rest_content_rest_arc(ordered_task_number(o))) <=
                   rest_resource(i,o)
       end-if
end-if
```

```
                work_resource_bound_constr(i,o) := work_resource(i,o) <= 7*max_work
                    (i)
                rest_resource_bound_constr(i,o) := rest_resource(i,o) <= 1
        end-do
end-do


forall(i in GUARANTEED) undertime_constr(i) := undertime(i) >= g_days(i) - (
    exp_worktime(i) + sum(j in JOB)(duration(j) * new_sched(i,j)))

forall(i in GUARANTEED) overtime_constr(i) := overtime(i) >= (exp_worktime(i) +
    sum(j in JOB)(duration(j) * new_sched(i,j))) - g_days(i)

forall(i in EMP, j in JOB | eligible(i,j) = 1) do
        if(initial(i,j) = 1) then new_sched_constr(i,j) := new_sched(i,j) =
            initial(i,j) - change(i,j); end-if
        if(initial(i,j) = 0) then new_sched_constr(i,j) := new_sched(i,j) =
            initial(i,j) + change(i,j); end-if
        change(i,j) is_binary
end-do
forall(j in JOB | AG_eligible(j) = 1) do
        if(AG_initial(j) = 1) then AG_new_sched_constr(j) := AG_new_sched(j) =
            AG_initial(j) - AG_change(j); end-if
        if(AG_initial(j) = 0) then AG_new_sched_constr(j) := AG_new_sched(j) =
            AG_initial(j) + AG_change(j); end-if
        AG_change(j) is_binary
end-do



split_time := gettime        ! Get the time so that set up time and actual
    running time can be calculated separately



!-----------------------------------------------------------------------
! Now solve the problem and print the results:
fopen(LOGFILE, F_APPEND)
        writeln
        writeln("Instance: ",DATAFILE)
        writeln("---------------------------------------------")
        writeln
        setparam("XPRS_verbose",true)
        minimize(TOTAL_COST)
        writeln
        writeln("---------------------------------------------")
        writeln
fclose(F_APPEND)

prog_endtime := gettime

prog_setup_time := split_time - prog_starttime
prog_runtime := prog_endtime - split_time
total_time := prog_endtime - prog_starttime

status::([XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH])["Optimum found","
    Unfinished","Infeasible","Unbounded","Failed"]



! Calculate some stats about the current instance
a := 0              ! number of tasks covered by agency in initial schedule
u := 0              ! number of un-covered tasks in the initial schedule
changes_to_reg := 0         ! number of changes to regular employees in the (
    best) solution
changes_to_AG := 0          ! number of changes to agency employees in the (best
    ) solution
number_of_AG := 0           ! number of times agency employees are utilised in
    the (best) solution

forall(j in JOB) do
        x := 0
        forall(i in EMP) do
                if(initial(i,j) = 1) then
                        x := x + 1
                end-if
                changes_to_reg := changes_to_reg + round(getsol(change(i,j)))
        end-do
        if(AG_initial(j) = 1) then
                x := x + 1
                a := a + 1
        end-if
        changes_to_AG := changes_to_AG+ round(getsol(AG_change(j)))
        number_of_AG := number_of_AG+ round(getsol(AG_new_sched(j)))

        if(x = 0) then
                u := u + 1
        end-if

end-do


fopen(SUMMARYFILE, F_APPEND)
        write(InstanceName)
        write("\t",prog_setup_time,"\t",prog_runtime,"\t",total_time)
        write("\t",status(getprobstat))
        write("\t",JOBS_TO_PLAN,"\t",u,"\t",a)
        write("\t",getobjval,"\t",getparam("XPRS_bestbound"))
        write("\t",changes_to_reg,"\t",changes_to_AG,"\t",number_of_AG)
        write("\n")
fclose(F_APPEND)




end-model
```

E.1.3 Change-minimization algorithm

Here we give the code used to solve the Task-Based problem using the Change-Minimization algorithm described in section 5.5.1.2. This was implemented using the FICO Xpress software.

```
model ModelName
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver

! Model for the Subsea 7 problem, using formulation as set out in 2nd Year Review
     document
! This program aims to solve the problem directly from this formulation

! NOTE - This version aims to minimise the number of changes rather than the cost

parameters
        DATE = "09-10-14"
        PARAMETERFILE = "batch-input-parameters.dat"
end-parameters

declarations
        InstanceName: string
end-declarations
```

```
initializations from PARAMETERFILE
       InstanceName
end-initializations


DATAFILE := InstanceName+"\\Task-Based - Captains - "+InstanceName+".txt"
LOGFILE := InstanceName+"\\Logfile - TB min changes and reduce cost - "+
    InstanceName+" - "+DATE+".txt"
SUMMARYFILE := "Results - TB min changes and reduce cost - "+DATE+".txt"


! Set parameters relating to running time and acceptable optimality gap:
setparam("XPRS_verbose",true)
setparam("XPRS_maxtime", -30)
!setparam("XPRS_miprelstop",0.05)
! Set parameters for numbers of Cover and Gomory cuts to apply:
setparam("XPRS_covercuts",30)
setparam("XPRS_gomcuts",10)


prog_starttime := gettime           ! get the time so that at the end, running
    time can be calculated

declarations
       EMP: set of string                 ! Employee names / numbers
       GUARANTEED:   set of string ! Set of employees on guaranteed days
           contracts
       DAYS_TO_PLAN: integer
       JOBS_TO_PLAN: integer
       PROJECT_NUMBER: integer
       status: array({XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH}) of string !
           Used to indicate the solution status of the problem
end-declarations

initializations from DATAFILE
       EMP GUARANTEED DAYS_TO_PLAN JOBS_TO_PLAN PROJECT_NUMBER
end-initializations


if(PROJECT_NUMBER = 0) then THEORETIC_PROJECTS := 1
else THEORETIC_PROJECTS := PROJECT_NUMBER
end-if
```

```
declarations
       JOB = 1..JOBS_TO_PLAN
       PROJECT = 1..THEORETIC_PROJECTS
       eligible: array(EMP, JOB) of real   ! = 1 if employee can carry out a job,
            0 otherwise
       AG_eligible: array(JOB) of real     ! eligibility of agency crew
       change_cost: array(EMP, JOB) of real ! The cost of an employee carrying
           out a job
       AG_change_cost: array(JOB) of real         ! Cost details for agency crew
       start_time, duration: array(JOB) of real
       rest_duration_list: set of real
       REST_NUMBER: integer
       initial: array(EMP, JOB) of real    ! = 1 if employee is currently
           allocated to a task, 0 otherwise
       AG_initial: array(JOB) of real
       under_rate, over_rate, current_excess, g_days, exp_worktime: array(
           GUARANTEED) of real
       project_matrix: array(PROJECT, JOB) of integer
       project_set: array(PROJECT) of set of integer
       experience: array(EMP, JOB) of real
       AG_experience: array(JOB) of real
       min_experience: array(PROJECT) of real
       min_rest: array(EMP) of real
       max_work: array(EMP) of real

       undertimevar, overtimevar: array(GUARANTEED) of mpvar
       undertime, overtime: array(GUARANTEED) of real

       change: dynamic array(EMP, JOB) of mpvar
       AG_change: dynamic array(JOB) of mpvar
       new_sched: dynamic array(EMP, JOB) of mpvar
       AG_new_sched: dynamic array(JOB) of mpvar

       best_change: dynamic array(EMP, JOB) of integer
       best_AG_change: dynamic array(JOB) of integer
       best_new_sched: dynamic array(EMP, JOB) of integer
       best_AG_new_sched: dynamic array(JOB) of integer
end-declarations

initializations from DATAFILE
       eligible AG_eligible change_cost AG_change_cost start_time duration
           initial AG_initial
       under_rate over_rate current_excess g_days exp_worktime
       project_matrix experience AG_experience min_experience min_rest max_work
end-initializations


!-----------------------------------------------------------------------
! Define the change and new_sched (ie y and z) variables ONLY for the
! tasks for which each crew member is eligible
!writeln("Define 'change' and 'new sched' variables:")
forall(j in JOB) do
       forall(i in EMP) do
              if eligible(i,j) = 1 then
!                     writeln("Create variables for task ",j," for ",i)
                     create(change(i,j))
                     create(new_sched(i,j))
                     create(best_change(i,j))
                     create(best_new_sched(i,j))
              end-if
       end-do
       if AG_eligible(j) = 1 then
!             writeln("Create variables for task ",j," for agency")
              create(AG_change(j))
              create(AG_new_sched(j))
              create(best_AG_change(j))
              create(best_AG_new_sched(j))
       end-if

!      writeln
end-do

!-----------------------------------------------------------------------
! Define 'rest' tasks such that they start directly after each work task has
    ended
! (subject to them finishing before the time-window ends)
!writeln("Create rest tasks:")
rest_duration_list := {}
forall(i in EMP) rest_duration_list += {7*min_rest(i)}
!forall(i in EMP) writeln(i," requires a minimum rest period of length ",min_rest
    (i))
!writeln

declarations
       rest_start_list: array(rest_duration_list) of set of real
```

```
end-declarations

REST_NUMBER := 0

forall(j in JOB, r in rest_duration_list) do
	if start_time(j) + duration(j) < (DAYS_TO_PLAN - r + 1) then
		rest_start_list(r) += {start_time(j) + duration(j)}
!		writeln("Task ",j," ends at time ",start_time(j) + duration(j),",
    and we can add a rest task of length ",r," starting at this time")
	end-if
end-do
!writeln
!writeln("The 'rest start list' contains the following:")
!forall(r in rest_duration_list) writeln("Of duration ",r,", starting at: ",
    rest_start_list(r))

forall(r in rest_duration_list) REST_NUMBER := REST_NUMBER + getsize(
    rest_start_list(r))
!writeln("... and so REST_NUMBER = ",REST_NUMBER)
!writeln

declarations
	REST = 1..REST_NUMBER
	rest_start_time, rest_duration: array(REST) of real
	rest_new_sched: dynamic array(EMP, REST) of mpvar
	work_content_work_arc, rest_content_work_arc: array(JOB) of real
	work_content_rest_arc, rest_content_rest_arc: array(REST) of real
end-declarations

index := 0
forall(r in rest_duration_list) do
	forall(s in rest_start_list(r)) do
		index := index + 1
		rest_start_time(index) := s
		rest_duration(index) := r
!		writeln("Rest task ",index," starts at time ",rest_start_time(index
    )," and is of length ",rest_duration(index))
	end-do
end-do
!writeln
```

```
! Note that employees can only be assigned a rest task of length equal to their
    required minimum rest period
forall(r in REST, i in EMP) do
       if(rest_duration(r) = 7*min_rest(i)) then
              create(rest_new_sched(i,r))
!             writeln("Rest task: ",r,"\tDuration: ",rest_duration(r),"\tEmployee
    : ",i,"\tMinRest: ",min_rest(i),"\t=> Create variable")
       end-if
end-do


!-----------------------------------------------------------------------
! Can now define work and rest content for work and rest arcs
!writeln("Define work and rest contents:")
forall(j in JOB) do
       work_content_work_arc(j) := duration(j)
       rest_content_work_arc(j) := 1
!      rest_content_work_arc(j) := (duration(j))/14
!      writeln("For task ",j,", work content = ",work_content_work_arc(j)," and
    rest content = ",rest_content_work_arc(j))
end-do


forall(r in REST) do
       work_content_rest_arc(r) := -DAYS_TO_PLAN
       rest_content_rest_arc(r) := -1
!      writeln("For rest task ",r,", work content = ",work_content_rest_arc(r),"
    and rest content = ",rest_content_rest_arc(r))
end-do
!writeln

!-----------------------------------------------------------------------
! Next we will generate the sets C_beta of overlapping tasks
!writeln("Define the sets of overlapping tasks:")
declarations
       beta: integer
       time_points: set of real
       time_points_ordered: list of real
       NUMBER_OF_POINTS: integer
       min_time: real
       beta_index: range
       overlap_set_work, overlap_set_rest: dynamic array(beta_index) of set of
           integer
       current_set_work, current_set_rest: set of integer
end-declarations

time_points := {}
forall(j in JOB) do
       time_points += {start_time(j)}
       time_points += {start_time(j) + duration(j)}
end-do
forall(j in REST) do
       time_points += {rest_start_time(j)}
       time_points += {rest_start_time(j) + rest_duration(j)}
end-do
!writeln("The set of time points is as follows:",time_points)

time_points_ordered := []
NUMBER_OF_POINTS := getsize(time_points)
forall(i in 1..NUMBER_OF_POINTS) do
       min_time := DAYS_TO_PLAN
       forall(p in time_points) do
              if(p < min_time) then min_time := p; end-if
       end-do
       time_points_ordered += [min_time]
       time_points -= {min_time}
end-do
!writeln("Giving the following ordered set:",time_points_ordered)
!writeln

!writeln("Running algorithm:")
current_set_work := {}
current_set_rest := {}
beta := 1
constraint_required := FALSE
add_constraint := FALSE
forall(t in time_points_ordered) do
!      writeln("\t dealing with time point ",t)
       ! check to see if another constraint is required
       forall(j in JOB | start_time(j) + duration(j) = t) add_constraint := TRUE
!      forall(j in JOB | start_time(j) + duration(j) = t) writeln("\t\t job ",j,"
     ends at time ",t)
       forall(j in REST | rest_start_time(j) + rest_duration(j) = t)
           add_constraint := TRUE
!      forall(j in REST | rest_start_time(j) + rest_duration(j) = t) writeln("\t\
    t rest task ",j," ends at time ",t)
```

```
        ! if so, we create an 'overlap set' and add the current contents to it
        if(constraint_required = TRUE AND add_constraint = TRUE) then
                create(overlap_set_work(beta))
                create(overlap_set_rest(beta))
                overlap_set_work(beta) := current_set_work
                overlap_set_rest(beta) := current_set_rest
!               writeln("\t\t... so we create the overlap set for beta = ",beta)
!               writeln("\t\t\t containing working tasks ",overlap_set_work(beta),"
     and rest tasks ",overlap_set_rest(beta))
                beta := beta + 1
                add_constraint := FALSE
                constraint_required := FALSE
        end-if

        ! remove these tasks from the current set
        forall(j in JOB | start_time(j) + duration(j) = t) current_set_work -= {j}
        forall(j in REST | rest_start_time(j) + rest_duration(j) = t)
            current_set_rest -= {j}

        ! if there are new tasks starting at this point, add them to the current
            set and
        ! indicate that a contraint will now be required again
        forall(j in JOB | start_time(j) = t) do
                constraint_required := TRUE
                current_set_work += {j}
!               writeln("\t\t Task ",j," starts at time ",t,", so add it to the
    current set")
        end-do
        forall(j in REST | rest_start_time(j) = t) do
                constraint_required := TRUE
                current_set_rest += {j}
!               writeln("\t\t Rest task ",j," starts at time ",t,", so add it to
    the current set")
        end-do
end-do
!writeln

!------------------------------------------------------------------------
! Define project_sets from project_matrix data

forall(p in PROJECT) do
        project_set(p) := {}
        forall(j in JOB) do
```

```
                if(project_matrix(p,j) = 1) then project_set(p) += {j}; end-if
        end-do
end-do

!       writeln("Define the project sets:")
!       forall(p in PROJECT) writeln("Project ",p," contains tasks ",project_set(p
    ))
!       writeln


!------------------------------------------------------------------------
! Define Ordered Task Set B (of ALL tasks - work, rest)
!writeln("Define the ordered task set of ALL tasks")
declarations
        order_index = 1..(JOBS_TO_PLAN + REST_NUMBER)
        ordered_task_number: array(order_index) of integer
        ordered_task_work, ordered_task_rest: array(order_index) of integer
        work_resource, rest_resource: array(EMP, order_index) of mpvar
        work_resource_zero, rest_resource_zero: array(EMP) of real
        work_unproc, rest_unproc: set of integer
        work_proc, rest_proc: set of integer
        b: integer
end-declarations

initializations from DATAFILE
        work_resource_zero rest_resource_zero
end-initializations


forall(j in JOB) work_unproc += {j}
forall(j in REST) rest_unproc += {j}
!       writeln("List of unprocessed tasks (in terms of ordering):")
!       writeln("Work tasks: ",work_unproc)
!       writeln("Rest tasks: ",rest_unproc)
!       writeln

b := 0
forall(i in 1..(JOBS_TO_PLAN + REST_NUMBER)) do
        min_time := DAYS_TO_PLAN
        forall(j in work_unproc) do
                if(start_time(j) < min_time) then min_time := start_time(j); end-if
        end-do
        forall(j in rest_unproc) do
```

```
                if(rest_start_time(j) < min_time) then min_time := rest_start_time(
                    j); end-if
        end-do
!       writeln("Earliest start time is ",min_time)

        work_proc := {}
        forall(j in work_unproc) do
                if(start_time(j) = min_time) then
                        b := b+1
!                       writeln(b," - WORK")
                        ordered_task_number(b) := j
                        ordered_task_work(b) := 1
                        ordered_task_rest(b) := 0
                        work_proc += {j}
!                       writeln("\t Work task ",j," starts at time ",min_time," -
    assign this as task number ",b)
                end-if
        end-do
        work_unproc -= work_proc
!       writeln("\t\t The following work tasks have now been processed: ",
    work_proc)
!       writeln("\t\t ... leaving the following still to be processed: ",
    work_unproc)

        rest_proc := {}
        forall(j in rest_unproc) do
                if(rest_start_time(j) = min_time) then
                        b := b + 1
!                       writeln(b," - REST")
                        ordered_task_number(b) := j
                        ordered_task_work(b) := 0
                        ordered_task_rest(b) := 1
                        rest_proc += {j}
!                       writeln("\t Rest task ",j," starts at time ",min_time," -
    assign this as task number ",b)
                end-if
        end-do
        rest_unproc -= rest_proc
!       writeln("\t\t The following rest tasks have now been processed: ",
    rest_proc)
!       writeln("\t\t ... leaving the following still to be processed: ",
    rest_unproc)
```

```
end-do


!-----------------------------------------------------------------------
! Now that all quantities have been defined, we can state all the contraints:
declarations
!       TOTAL_COST: linctr
        TOTAL_COST: real
        PREV_COST: real
        COST_LIMIT: real
        cost_constr: linctr

        NO_CHANGES_BEST: real
        CHANGE_LIMIT: real

        iteration: integer
        terminate: boolean
        percentage: real
        integral: boolean

        Big_M: real
        iter_time_limit: integer
!       UTreq, OTreq: array(GUARANTEED) of mpvar
        OTind: array(GUARANTEED) of mpvar
        undertime_second_constr, overtime_second_constr: array(GUARANTEED) of
            linctr
        undertime_third_constr, overtime_third_constr: array(GUARANTEED) of linctr

        NO_CHANGES: linctr
        job_cover_constr: array(JOB) of linctr
        overlap_constr: array(EMP,beta_index) of linctr
        experience_constr: array(PROJECT) of linctr
        work_resource_constr_1, rest_resrouce_constr_1: array(EMP) of linctr
        work_resource_constr_b, rest_resource_constr_b: array(EMP, order_index) of
             linctr
        work_resource_bound_constr, rest_resource_bound_constr: array(EMP,
            order_index) of linctr
        undertime_constr, overtime_constr: array(GUARANTEED) of linctr
        new_sched_constr: array(EMP, JOB) of linctr
        AG_new_sched_constr: array(JOB) of linctr
end-declarations
```

```
Big_M := 366
iter_time_limit := 30

! Objective:
!TOTAL_COST := sum(i in EMP, j in JOB)(change_cost(i,j) * change(i,j)) + sum(j in
     JOB)(AG_change_cost(j) * AG_change(j)) + sum(i in GUARANTEED)((over_rate(i)
    * overtime(i)) + (under_rate(i) * undertime(i)) - current_excess(i))
NO_CHANGES := sum(i in EMP, j in JOB)(change(i,j)) + sum(j in JOB)(AG_change(j))

! Other constraints:
forall(j in JOB) job_cover_constr(j) := sum(i in EMP)(new_sched(i,j)) +
    AG_new_sched(j) = 1

forall(i in EMP, a in beta_index) overlap_constr(i,a) := sum(j in
    overlap_set_work(a))(new_sched(i,j)) + sum(j in overlap_set_rest(a))(
    rest_new_sched(i,j)) <= 1

forall(k in PROJECT) experience_constr(k) := sum(i in EMP, j in project_set(k))(
    experience(i,j) * new_sched(i,j)) + sum(j in project_set(k))(AG_experience(j)
     * AG_new_sched(j)) >= min_experience(k)

forall(i in EMP) do
! Revised version of inner part of this loop:
!----------------------------------------------------------------
       forall(o in order_index) do
              if(o = 1) then
                     if(ordered_task_work(1) = 1) then
                            work_resource_constr_b(i,o) := 7*work_resource_zero(
                                i) + (new_sched(i,ordered_task_number(1)) *
                                work_content_work_arc(ordered_task_number(1))) <=
                                 work_resource(i,1)
                            rest_resource_constr_b(i,o) := rest_resource_zero(i)
                                 + (new_sched(i,ordered_task_number(1)) *
                                rest_content_work_arc(ordered_task_number(1))) <=
                                 rest_resource(i,1)
                     end-if
                     if(ordered_task_rest(1) = 1) then
                            work_resource_constr_b(i,o) := 7*work_resource_zero(
                                i) + (rest_new_sched(i,ordered_task_number(1)) *
                                work_content_rest_arc(ordered_task_number(1))) <=
                                 work_resource(i,1)
                            rest_resource_constr_b(i,o) := rest_resource_zero(i)
                                 + (rest_new_sched(i,ordered_task_number(1)) *
                              rest_content_rest_arc(ordered_task_number(1))) <=
                               rest_resource(i,1)
                        end-if
                else
                        if(ordered_task_work(o) = 1) then
                                work_resource_constr_b(i,o) := work_resource(i,(o-1)
                                    ) + (new_sched(i,ordered_task_number(o)) *
                                    work_content_work_arc(ordered_task_number(o))) <=
                                     work_resource(i,o)
                                rest_resource_constr_b(i,o) := rest_resource(i,(o-1)
                                    ) + (new_sched(i,ordered_task_number(o)) *
                                    rest_content_work_arc(ordered_task_number(o))) <=
                                     rest_resource(i,o)
                        end-if
                        if(ordered_task_rest(o) = 1) then
                                work_resource_constr_b(i,o) := work_resource(i,(o-1)
                                    ) + (rest_new_sched(i,ordered_task_number(o)) *
                                    work_content_rest_arc(ordered_task_number(o))) <=
                                     work_resource(i,o)
                                rest_resource_constr_b(i,o) := rest_resource(i,(o-1)
                                    ) + (rest_new_sched(i,ordered_task_number(o)) *
                                    rest_content_rest_arc(ordered_task_number(o))) <=
                                     rest_resource(i,o)
                        end-if
                end-if
                work_resource_bound_constr(i,o) := work_resource(i,o) <= 7*max_work
                    (i)
                rest_resource_bound_constr(i,o) := rest_resource(i,o) <= 1
        end-do
end-do


forall(i in GUARANTEED) do
        undertime_constr(i) := undertimevar(i) >= g_days(i) - (exp_worktime(i) +
            sum(j in JOB)(duration(j) * new_sched(i,j)))
        undertime_second_constr(i) := undertimevar(i) <= g_days(i) - (exp_worktime
            (i) + sum(j in JOB)(duration(j) * new_sched(i,j))) + Big_M*(1-OTind(i)
            )
        undertime_third_constr(i) := undertimevar(i) <= Big_M*OTind(i)
!       undertime_second_constr(i) := undertimevar(i) <= g_days(i) - (exp_worktime
    (i) + sum(j in JOB)(duration(j) * new_sched(i,j))) + Big_M*UTreq(i)
!       undertime_third_constr(i) := undertimevar(i) <= Big_M*(1-UTreq(i))
!       UTreq(i) is_binary
```

```
       overtime_constr(i) := overtimevar(i) >= (exp_worktime(i) + sum(j in JOB)(
           duration(j) * new_sched(i,j))) - g_days(i)
       overtime_second_constr(i) := overtimevar(i) <= (exp_worktime(i) + sum(j in
            JOB)(duration(j) * new_sched(i,j))) - g_days(i) + Big_M*OTind(i)
       overtime_third_constr(i) := - Big_M*(1-OTind(i))
!      overtime_second_constr(i) := overtimevar(i) <= (exp_worktime(i) + sum(j in
     JOB)(duration(j) * new_sched(i,j))) - g_days(i) + Big_M*OTreq(i)
!      overtime_third_constr(i) := - Big_M*(1-OTreq(i))
!      OTreq(i) is_binary

       OTind(i) is_binary
end-do

forall(i in EMP, j in JOB | eligible(i,j) = 1) do
       if(initial(i,j) = 1) then new_sched_constr(i,j) := new_sched(i,j) =
           initial(i,j) - change(i,j); end-if
       if(initial(i,j) = 0) then new_sched_constr(i,j) := new_sched(i,j) =
           initial(i,j) + change(i,j); end-if
       change(i,j) is_binary
end-do
forall(j in JOB | AG_eligible(j) = 1) do
       if(AG_initial(j) = 1) then AG_new_sched_constr(j) := AG_new_sched(j) =
           AG_initial(j) - AG_change(j); end-if
       if(AG_initial(j) = 0) then AG_new_sched_constr(j) := AG_new_sched(j) =
           AG_initial(j) + AG_change(j); end-if
       AG_change(j) is_binary
end-do


!-----------------------------------------------------------------------
! Calculate some stats about the current instance

a := 0               ! number of tasks covered by agency in initial schedule
u := 0               ! number of un-covered tasks in the initial schedule

forall(j in JOB) do
       x := 0
       forall(i in EMP) do
              if(initial(i,j) = 1) then
                     x := x + 1
              end-if
       end-do
```

```
        if(AG_initial(j) = 1) then
                x := x + 1
                a := a + 1
        end-if

        if(x = 0) then
                u := u + 1
        end-if

end-do

!------------------------------------------------------------------------

split_time := gettime        ! Get the time so that set up time and actual
    running time can be calculated separately
prog_setup_time := split_time - prog_starttime


fopen(SUMMARYFILE, F_APPEND)
        write(InstanceName)
        write("\t",JOBS_TO_PLAN,"\t",u,"\t",a)
        write("\t",prog_setup_time)
fclose(F_APPEND)


!------------------------------------------------------------------------
!------------------------------------------------------------------------
!------------------------------------------------------------------------

! Now solve the problem and print the results:
fopen(LOGFILE, F_APPEND)
        writeln
        writeln("Instance: ",DATAFILE)
        writeln("---------------------------------------------")
        writeln
        setparam("XPRS_maxtime", -iter_time_limit)
        minimize(NO_CHANGES)
        writeln
        writeln("---------------------------------------------")
        writeln

        status::([XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH])["Optimum found","
            Unfinished","Infeasible","Unbounded","Failed"]
        integral := true
        forall(i in EMP, j in JOB | exists(change(i,j)) ) do
                if(isintegral(change(i,j)) = false) then integral := false; end-if
                if(isintegral(new_sched(i,j)) = false) then integral := false; end-
                    if
        end-do
        forall(j in JOB | exists(AG_change(j)) ) do
                if(isintegral(AG_change(j)) = false) then integral := false; end-if
                if(isintegral(AG_new_sched(j)) = false) then integral := false; end
                    -if
        end-do

        writeln(status(getprobstat))
        if(integral = false) then writeln("No integer solution found"); end-if
        writeln

fclose(F_APPEND)

iter_endtime := gettime
iter_runtime := iter_endtime - split_time
total_time := iter_endtime - prog_starttime


fopen(SUMMARYFILE, F_APPEND)
        write("\t",iter_runtime,"\t",total_time)
        write("\t",status(getprobstat))
fclose(F_APPEND)



if(getprobstat = XPRS_OPT OR (getprobstat = XPRS_UNF and integral = true)) then

        TOTAL_COST := sum(i in EMP, j in JOB)(change_cost(i,j) * getsol(change(i,j
            ))) + sum(j in JOB)(AG_change_cost(j) * getsol(AG_change(j))) + sum(i
            in GUARANTEED)((over_rate(i) * getsol(overtimevar(i))) + (under_rate(i
            ) * getsol(undertimevar(i))) - current_excess(i))

        NO_CHANGES_BEST := getobjval
!       CHANGE_LIMIT := 2*getobjval

        forall(i in EMP, j in JOB | exists(change(i,j)) ) do
                if(getsol(change(i,j)) > 0.7) then best_change(i,j) := 1
                else best_change(i,j) := 0; end-if
                if(getsol(new_sched(i,j)) > 0.7) then best_new_sched(i,j) := 1
                else best_new_sched(i,j) := 0; end-if
        end-do
        forall(j in JOB | exists(AG_change(j)) ) do
                if(getsol(AG_change(j)) > 0.7) then best_AG_change(j) := 1
                else best_AG_change(j) := 0; end-if
                if(getsol(AG_new_sched(j)) > 0.7) then best_AG_new_sched(j) := 1
                else best_AG_new_sched(j) := 0; end-if
        end-do


        fopen(SUMMARYFILE, F_APPEND)
                write("\t",getobjval,"\t",getparam("XPRS_bestbound"))
                write("\t",TOTAL_COST)
        fclose(F_APPEND)

else
        fopen(SUMMARYFILE, F_APPEND)
                write("\t\t\t")
        fclose(F_APPEND)

end-if



!-----------------------------------------------------------------------

iteration := 1
terminate := false
percentage := 0.1
while(terminate = false) do
        if(getprobstat = XPRS_OPT OR (getprobstat = XPRS_UNF and integral = true))
             then

                fopen(LOGFILE, F_APPEND)

                        writeln("\tThe total number of changes is: ",getobjval)
                        writeln("\tand the total cost of this solution is ",
                            TOTAL_COST)
                        writeln

                        PREV_COST := TOTAL_COST
                        if(PREV_COST > 0) then
                               COST_LIMIT := (1-percentage)*PREV_COST
                        elif(PREV_COST < 0) then
                               COST_LIMIT := (1+percentage)*PREV_COST
                        else
                               COST_LIMIT := -10
                        end-if

                        writeln("Set cost limit to ",COST_LIMIT)
                        cost_constr := sum(i in EMP, j in JOB)(change_cost(i,j) *
                            change(i,j)) + sum(j in JOB)(AG_change_cost(j) *
                            AG_change(j)) + sum(i in GUARANTEED)((over_rate(i) *
                            overtimevar(i)) + (under_rate(i) * undertimevar(i)) -
                            current_excess(i)) <= COST_LIMIT
                        writeln("... and re-solve")
                        writeln

                        writeln("---------------------------------------------")
                        writeln
                        setparam("XPRS_maxtime", -iter_time_limit)
                        minimize(NO_CHANGES)
                        writeln
                        writeln("---------------------------------------------")
                        writeln

                        integral := true
                        forall(i in EMP, j in JOB | exists(change(i,j)) ) do
                               if(isintegral(change(i,j)) = false) then integral :=
                                    false; end-if
                               if(isintegral(new_sched(i,j)) = false) then integral
                                    := false; end-if
                        end-do
                        forall(j in JOB | exists(AG_change(j)) ) do
                               if(isintegral(AG_change(j)) = false) then integral
                                   := false; end-if
                               if(isintegral(AG_new_sched(j)) = false) then
                                   integral := false; end-if
                        end-do
                        writeln(status(getprobstat))
                        if(integral = false) then writeln("No integer solution found
                            "); end-if
                        writeln
                fclose(F_APPEND)
```

```
iter_starttime := iter_endtime
iter_endtime := gettime
iter_runtime := iter_endtime - iter_starttime
total_time := iter_endtime - prog_starttime


fopen(SUMMARYFILE, F_APPEND)
       write("\t",iter_runtime,"\t",total_time)
       write("\t",status(getprobstat))
fclose(F_APPEND)


if(getprobstat = XPRS_OPT OR (getprobstat = XPRS_UNF and integral =
     true)) then
       TOTAL_COST := sum(i in EMP, j in JOB)(change_cost(i,j) *
           getsol(change(i,j))) + sum(j in JOB)(AG_change_cost(j) *
            getsol(AG_change(j))) + sum(i in GUARANTEED)((over_rate
           (i) * getsol(overtimevar(i))) + (under_rate(i) * getsol(
           undertimevar(i))) - current_excess(i))

       NO_CHANGES_BEST := getobjval

       forall(i in EMP, j in JOB | exists(change(i,j)) ) do
              if(getsol(change(i,j)) > 0.7) then best_change(i,j)
                  := 1
              else best_change(i,j) := 0; end-if
              if(getsol(new_sched(i,j)) > 0.7) then best_new_sched
                  (i,j) := 1
              else best_new_sched(i,j) := 0; end-if
       end-do
       forall(j in JOB | exists(AG_change(j)) ) do
              if(getsol(AG_change(j)) > 0.7) then best_AG_change(j
                  ) := 1
              else best_AG_change(j) := 0; end-if
              if(getsol(AG_new_sched(j)) > 0.7) then
                  best_AG_new_sched(j) := 1
              else best_AG_new_sched(j) := 0; end-if
       end-do

       fopen(SUMMARYFILE, F_APPEND)
              write("\t",getobjval,"\t",getparam("XPRS_bestbound")
                  )
```

```
                        write("\t",TOTAL_COST)
                fclose(F_APPEND)

        else
                fopen(SUMMARYFILE, F_APPEND)
                        write("\t\t\t")
                fclose(F_APPEND)

        end-if

        iteration := iteration + 1

else
        if(iteration = 1 or percentage = 0.05) then
                terminate := true
        else
                fopen(LOGFILE, F_APPEND)
                        writeln
                        writeln("No solution found")

                        percentage := 0.05
                        if(PREV_COST > 0) then
                                COST_LIMIT := (1-percentage)*PREV_COST
                        elif(PREV_COST < 0) then
                                COST_LIMIT := (1+percentage)*PREV_COST
                        else
                                COST_LIMIT := -1
                        end-if

                        writeln("Set cost limit to ",COST_LIMIT)
                        cost_constr := sum(i in EMP, j in JOB)(change_cost(i
                            ,j) * change(i,j)) + sum(j in JOB)(AG_change_cost
                            (j) * AG_change(j)) + sum(i in GUARANTEED)((
                            over_rate(i) * overtimevar(i)) + (under_rate(i) *
                             undertimevar(i)) - current_excess(i)) <=
                            COST_LIMIT
                        writeln("... and re-solve")
                        writeln

                        writeln
                            ("--------------------------------------------")
                        writeln
                        setparam("XPRS_maxtime", -iter_time_limit)
                        minimize(NO_CHANGES)
                        writeln
                        writeln
                            ("--------------------------------------------")

                        integral := true
                        forall(i in EMP, j in JOB | exists(change(i,j)) ) do
                               if(isintegral(change(i,j)) = false) then
                                   integral := false; end-if
                               if(isintegral(new_sched(i,j)) = false) then
                                   integral := false; end-if
                        end-do
                        forall(j in JOB | exists(AG_change(j)) ) do
                               if(isintegral(AG_change(j)) = false) then
                                   integral := false; end-if
                               if(isintegral(AG_new_sched(j)) = false) then
                                   integral := false; end-if
                        end-do
                        writeln(status(getprobstat))
                        if(integral = false) then writeln("No integer
                            solution found"); end-if
                        writeln

                fclose(F_APPEND)

                 iter_starttime := iter_endtime
                 iter_endtime := gettime
                 iter_runtime := iter_endtime - iter_starttime
                 total_time := iter_endtime - prog_starttime


                 fopen(SUMMARYFILE, F_APPEND)
                        write("\t",iter_runtime,"\t",total_time)
                        write("\t",status(getprobstat))
                 fclose(F_APPEND)


                 if(getprobstat = XPRS_OPT OR (getprobstat = XPRS_UNF and
                     integral = true)) then
                        TOTAL_COST := sum(i in EMP, j in JOB)(change_cost(i,
                            j) * getsol(change(i,j))) + sum(j in JOB)(
                            AG_change_cost(j) * getsol(AG_change(j))) + sum(i
                             in GUARANTEED)((over_rate(i) * getsol(
                            overtimevar(i))) + (under_rate(i) * getsol(
                            undertimevar(i))) - current_excess(i))

                        NO_CHANGES_BEST := getobjval

                        forall(i in EMP, j in JOB | exists(change(i,j)) ) do
                                if(getsol(change(i,j)) > 0.7) then
                                   best_change(i,j) := 1
                                else best_change(i,j) := 0; end-if
                                if(getsol(new_sched(i,j)) > 0.7) then
                                   best_new_sched(i,j) := 1
                                else best_new_sched(i,j) := 0; end-if
                        end-do
                        forall(j in JOB | exists(AG_change(j)) ) do
                                if(getsol(AG_change(j)) > 0.7) then
                                   best_AG_change(j) := 1
                                else best_AG_change(j) := 0; end-if
                                if(getsol(AG_new_sched(j)) > 0.7) then
                                   best_AG_new_sched(j) := 1
                                else best_AG_new_sched(j) := 0; end-if
                        end-do

                        fopen(SUMMARYFILE, F_APPEND)
                                write("\t",getobjval,"\t",getparam("
                                   XPRS_bestbound"))
                                write("\t",TOTAL_COST)
                        fclose(F_APPEND)

                 else
                        fopen(SUMMARYFILE, F_APPEND)
                                write("\t\t\t")
                        fclose(F_APPEND)
                 end-if

          end-if
  end-if

  current_time := gettime

  fopen(LOGFILE, F_APPEND)
          writeln
          writeln("Iterations:   ",iteration)
          writeln("Total run time: ",current_time - prog_starttime)
```

```
                writeln
                if(current_time - prog_starttime >= 120) then
                        terminate := true
                        writeln("Running time > 120 seconds --> terminate")
                        writeln
                elif(current_time - prog_starttime > 90) then
                        iter_time_limit := floor(120 - (current_time -
                            prog_starttime))+1
                        writeln("Time limit approaching - set next iteration time
                            limit to ",iter_time_limit," seconds.")
                        writeln
                end-if
!               if(integral = true and getparam("XPRS_bestbound") >= CHANGE_LIMIT)
    then
!                       terminate := true
!                       writeln("Number of changes has doubled from initial value
    --> terminate")
!                       writeln
!               end-if
        fclose(F_APPEND)

end-do


!-------------------------------------------------------------------------

! Calculate some stats about the final solution

changes_to_reg := 0          ! number of changes to regular employees in the (
    best) solution
changes_to_AG := 0           ! number of changes to agency employees in the (best
    ) solution
number_of_AG := 0            ! number of times agency employees are utilised in
    the (best) solution

forall(j in JOB) do
        forall(i in EMP) do
                changes_to_reg := changes_to_reg + best_change(i,j)
        end-do
        changes_to_AG := changes_to_AG + best_AG_change(j)
        number_of_AG := number_of_AG + best_AG_new_sched(j)
end-do
```

```
! Calculate costs
forall(i in GUARANTEED) do
       undertime(i) := g_days(i) - (exp_worktime(i) + sum(j in JOB)(duration(j) *
            best_new_sched(i,j)))
       overtime(i) := (exp_worktime(i) + sum(j in JOB)(duration(j) *
           best_new_sched(i,j))) - g_days(i)
       if(undertime(i) < 0) then undertime(i) := 0; end-if
       if(overtime(i) < 0) then overtime(i) := 0; end-if
end-do

TOTAL_COST := sum(i in EMP, j in JOB)(change_cost(i,j) * best_change(i,j)) + sum(
    j in JOB)(AG_change_cost(j) * best_AG_change(j)) + sum(i in GUARANTEED)((
    over_rate(i) * overtime(i)) + (under_rate(i) * undertime(i)) - current_excess
    (i))


current_time := gettime

total_time := current_time - prog_starttime

fopen(SUMMARYFILE, F_APPEND)
       write("\t",TOTAL_COST)
       write("\t",changes_to_reg,"\t",changes_to_AG,"\t",number_of_AG)
       write("\t",iteration)
       write("\t",total_time)
       write("\n")
fclose(F_APPEND)

end-model
```

E.2 For the Time-Windows formulation

Here we give the code used for generating the Time-Windows datasets and for testing the various solution approaches for the problem.

E.2.1 Generating datasets

Here we give the code used to generate the Time-Windows datasets, as described in section 6.2. This was implemented using the FICO Xpress software.

```
model ModelName
```

```
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver

! use the following when generating multiple data sets:
!parameters
!       INSTANCELIST = "Time-windows - Captains - Instance List.txt"
!       DATAFILE = "SS7 real data - Captains.txt"
!end-parameters
!
!declarations
!       NO_OF_INSTANCES: integer
!       OUTFILE: string
!end-declarations
!
!initialisations from INSTANCELIST
!       NO_OF_INSTANCES
!end-initialisations
!
!declarations
!       INSTANCES = 1..NO_OF_INSTANCES
!       instance_name: array(INSTANCES) of string
!       ill_given_ill: array(INSTANCES) of real
!       ill_given_fit: array(INSTANCES) of real
!       use_TR: array(INSTANCES) of boolean
!       DF_near: array(INSTANCES) of real
!       DF_long: array(INSTANCES) of real
!       AG_pen: array(INSTANCES) of real
!end-declarations
!
!initialisations from INSTANCELIST
!       instance_name
!       ill_given_ill
!       ill_given_fit
!       use_TR
!       DF_near
!       DF_long
!       AG_pen
!end-initialisations
!
!
!
!declarations
!       ! Parameters to be varied for various data sets:
!       ill_given_prev_fit: real
!       ill_given_prev_ill: real
!
!       use_time_reduction: boolean
!
!       disruption_factor_near: real
!       disruption_factor_long: real
!       agency_penalty: real
!
!end-declarations
!------------------------------------------------------


! NOTE - Above section used when generating multiple data sets
!               Alternatively, use section as given below:

parameters
        OUTFILE = "Time-windows - Captains - Trial runs.txt"
        DATAFILE = "SS7 real data - Captains.txt"

        ! Parameters to be varied for various data sets:
        ill_given_prev_fit = 0.004
        ill_given_prev_ill = 0.5

        use_time_reduction = TRUE

        disruption_factor_near = 1
        disruption_factor_long = 1
        agency_penalty = 1

end-parameters


! Notes on Parameters:
!-------------------------------------------------------------------------
! Probabilities of being ill or fit to work given previous day condition:
! Firstly, assume that the probability of being ill on a given day is 0.8% (=
   0.008)
! Then, can use the following pairs of data:
!       Prob(ill | ill on prev day)         Prob(ill | fit on prev day)
!                    0.8                                              0.0016
!                    0.75                                      0.002
!                    0.7                                              0.0024
!                    0.65                                      0.0028
!                 0.6                                                   0.0032
!                 0.55                                          0.0036
!                 0.5                                                   0.004
          --> currently selected
!                 0.45                                          0.0044
!                 0.4                                                   0.0048
!                 0.35                                          0.0052
!                 0.3                                                   0.0056
!                 0.25                                          0.0061
!                 0.2                                                   0.0065
!---------------------------------------------------------------------
! Time reduction can be used to reduce the above probabilities for times further
    in the future than 4 weeks.
! For a given day index d > 28, the time reduction is calculated as
!           time_reduction := (d-28)/((7*no_of_weeks)-28)
!
! The probabilities are then multiplied by (1 - time_reduction)
! It may be useful to vary the use of the time reduction to see if there is a
    significant impact on results
!---------------------------------------------------------------------
! Disruption factors and agency penalty are used to penalise the making of
    changes in the schedule
!  -->  Disruption factors are multipliers applied to the cost of changing the
    assignment if the cost is
!           positive, and used as a divisor if the cost is negative. They are
    defined differently for tasks which
!           start within the first four weeks (assumed to be penalised more
    harshly if any penalty is applied),
!           and those which start after this time. There is no disruption
    factor for assignments made in the final
!           week of the horizon.
!  -->  Similarly, a penalty is defined as a multiplier for positive or a divisor
    for negative costs of
!           changing the assignment of Agency crew, over and above the actual
    costs involved.
!---------------------------------------------------------------------


!------------------------------------------------------------------------------

! Declarations for data to be read in:
declarations
       no_of_crew: integer
```

```
        no_of_vessels: integer
        no_of_weeks: integer
end-declarations

initialisations from DATAFILE
        no_of_crew no_of_vessels no_of_weeks
end-initialisations

declarations
        crew = 1..no_of_crew
        crew_label: array(crew) of string
        contract_type: array(crew) of string
        legal_entity: array(crew) of string
        crew_continent: array(crew) of string
        crew_nation: array(crew) of string

        vessel = 1..no_of_vessels
        vessel_name: array(vessel) of string
        vessel_label: array(vessel) of string
        required: array(vessel) of integer
        vessel_location: array(vessel) of string
end-declarations

initialisations from DATAFILE
        crew_label vessel_name
        contract_type legal_entity crew_continent crew_nation
        vessel_label required vessel_location
end-initialisations


procedure MAIN_PROG

        !-----------------------------------------------------------------
        ! Declarations for data to be calulated / generated for this instance:
        declarations
                no_of_tasks: integer

                vessel_max_work: array(vessel) of integer

                vessel_roles: array(vessel) of set of integer
                role_tasks: integer

                task_no, time_count: integer
                rand_no: real
                rand_int: integer

        end-declarations


        !------------------------------------------------------------------------
        ! Calculations:

        ! First need to calculate the number of roles required in total
        role_no := 0
        forall(v in vessel | required(v) > 0) do
                vessel_roles(v) := {}
                forall(r in 1..required(v)) do
                        role_no := role_no + 1
                        vessel_roles(v) += {role_no};
                        ! writeln("Role ",role_no,"\tis on board vessel ",v)
                end-do
        end-do

        declarations
                roles = 1..role_no
                role_on_vessel: array(roles) of integer
        end-declarations

        ! to indicate on board which vessel a role takes place
        forall(v in vessel | required(v) > 0) do
                forall(r in vessel_roles(v)) do
                        role_on_vessel(r) := v
                end-do
        end-do


        !------------------------------------------------------------------------
        ! Generate regular assignments:
        declarations
                Norway_emps, Singapore_emps, Other_emps: integer
                Remaining_emps: set of integer
                Norway_cover, Singapore_cover, Other_cover: real
                Min_cover, Max_cover: integer

                active_roles: integer
                Norway_roles, Singapore_roles, Other_roles: integer
                Norway_set, Singapore_set, Other_set, Remaining_set: set of integer
                Set_int, Cover_int: integer

                regular_vessel: array(crew) of integer
                no_of_regulars_vessel: array(vessel) of integer

                regular_role: array(crew) of integer
                no_of_regulars_role: array(roles) of integer

                assigned_crew: array(roles) of integer
                assigned_role: integer

                extra, spare, number: integer
                extra_roles, spare_roles: integer

                Norway_groups, Singapore_groups, Other_groups: integer

        end-declarations



        ! Number of employees on each contract:
        Norway_emps := 0
        Singapore_emps := 0
        Other_emps := 0

        forall(c in crew) do
                if(legal_entity(c) = "Subsea7 Contr (Norway) AS") then Norway_emps
                    := Norway_emps +1
                elif(legal_entity(c) = "Subsea 7 (Sing) PTE-GEC") then
                    Singapore_emps := Singapore_emps +1
                else Other_emps := Other_emps +1
                end-if
        end-do

        ! writeln
        ! writeln("Number on Norway contracts: \t",Norway_emps)
        ! writeln("Number on Singapore contracts:\t",Singapore_emps)
        ! writeln("Number on Other contracts: \t",Other_emps)
        ! writeln("Total number of employees: \t",no_of_crew)
        ! writeln
        ! number of roles which can be covered by each contract type:
        Norway_cover := (Norway_emps / 3);
        ! writeln("Norway cover:\t\t",Norway_cover)
        Singapore_cover := (Singapore_emps * 2/3);
        ! writeln("Singapore cover:\t",Singapore_cover)
        Other_cover := (Other_emps / 2);
        ! writeln("Other cover:\t\t",Other_cover); writeln
        Min_cover := integer(Norway_cover) + integer(Singapore_cover) + integer(
            Other_cover)
        Max_cover := Min_cover
        if(Norway_cover > integer(Norway_cover)) then Max_cover := Max_cover +1;
            end-if
        if(Singapore_cover > integer(Singapore_cover)) then Max_cover := Max_cover
             +1; end-if
        if(Other_cover > integer(Other_cover)) then Max_cover := Max_cover +1; end
            -if



        ! Number of roles which could be assigned to each contract type:
        Norway_roles := integer(role_no * (Norway_cover / (Norway_cover +
            Singapore_cover + Other_cover)));
        ! writeln("There are ",Norway_roles,"\tNorway-operated roles")
        Singapore_roles := integer(role_no * (Singapore_cover / (Norway_cover +
            Singapore_cover + Other_cover)));
        ! writeln("There are ",Singapore_roles,"\tSingapore-operated roles")
        Other_roles := integer(role_no * (Other_cover / (Norway_cover +
            Singapore_cover + Other_cover)));
        ! writeln("There are ",Other_roles,"\tOther-operated roles")
        spare_roles := role_no - (Norway_roles + Singapore_roles + Other_roles);
        ! writeln("There are ",spare_roles,"\troles yet to be determined");
            writeln

        ! Divide roles between the different contract types:
        Remaining_set := {}
        forall(r in roles) do
                Remaining_set += {r};
                ! writeln("Add role ",r," to the set of remaining tasks")
        end-do
        extra_roles := spare_roles;
        ! writeln

        while(getsize(Norway_set) < Norway_roles) do
```

```
        rand_int := integer(random*role_no + 1)

        if(rand_int in Remaining_set) then
                Norway_set += {rand_int};
                ! writeln("Add role ",rand_int," to the set of Norway-
                    operated tasks")
                Remaining_set -= {rand_int};
                ! writeln("\tRemove role ",rand_int," from the set of
                    remaining tasks")

                if((getsize(Norway_set) - 1) + required(role_on_vessel(
                    rand_int)) <= Norway_roles + extra_roles) then
                        forall(r in vessel_roles(role_on_vessel(rand_int)) |
                             r in Remaining_set) do
                                Norway_set += {r};
                                ! writeln("Add role ",r," to the set of
                                    Norway-operated tasks")
                                Remaining_set -= {r};
                                ! writeln("\tRemove role ",r," from the set
                                    of remaining tasks")
                        end-do
                end-if
        end-if
end-do
! writeln; writeln("\t\tRemaining set now contains:\t",Remaining_set)
extra_roles := extra_roles + Norway_roles - getsize(Norway_set);
! writeln; writeln("\t\tRoles still to be allocated:\t",extra_roles);
    writeln

while(getsize(Singapore_set) < Singapore_roles) do
        rand_int := integer(random*role_no + 1)

        if(rand_int in Remaining_set) then
                Singapore_set += {rand_int};
                ! writeln("Add role ",rand_int," to the set of Singapore-
                    operated tasks")
                Remaining_set -= {rand_int};
                ! writeln("\tRemove role ",rand_int," from the set of
                    remaining tasks")

                if((getsize(Singapore_set) - 1) + required(role_on_vessel(
                    rand_int)) <= Singapore_roles + extra_roles) then
                    forall(r in vessel_roles(role_on_vessel(rand_int)) |
                         r in Remaining_set) do
                           Singapore_set += {r};
                           ! writeln("Add role ",r," to the set of
                               Singapore-operated tasks")
                           Remaining_set -= {r};
                           ! writeln("\tRemove role ",r," from the set
                               of remaining tasks")
                    end-do
             end-if
      end-if
end-do
! writeln; writeln("\t\tRemaining set now contains:\t",Remaining_set)
extra_roles := extra_roles + Singapore_roles - getsize(Singapore_set);
! writeln; writeln("\t\tRoles still to be allocated:\t",extra_roles);
    writeln

while(getsize(Other_set) < Other_roles) do
      rand_int := integer(random*role_no + 1)

      if(rand_int in Remaining_set) then
             Other_set += {rand_int};
             ! writeln("Add role ",rand_int," to the set of Other-
                 operated tasks")
             Remaining_set -= {rand_int};
             ! writeln("\tRemove role ",rand_int," from the set of
                 remaining tasks")

             if((getsize(Other_set) - 1) + required(role_on_vessel(
                 rand_int)) <= Other_roles + extra_roles) then
                    forall(r in vessel_roles(role_on_vessel(rand_int)) |
                         r in Remaining_set) do
                           Other_set += {r};
                           ! writeln("Add role ",r," to the set of Other
                               -operated tasks")
                           Remaining_set -= {r};
                           ! writeln("\tRemove role ",r," from the set
                               of remaining tasks")
                    end-do
             end-if
      end-if
end-do
! writeln; writeln("\t\tRemaining set now contains:\t",Remaining_set)
```

```
extra_roles := extra_roles + Other_roles - getsize(Other_set);
! writeln; writeln("\t\tRoles still to be allocated:\t",extra_roles);
    writeln

if(extra_roles > 0) then
       forall(r in Remaining_set) do
              rand_no := random;
              ! writeln("Try to allocate role ",r,"..."); writeln("\
                  tRandom number = ",rand_no)

              if(rand_no < (Norway_cover / (Norway_cover + Singapore_cover
                   + Other_cover))) then
                     Norway_set += {r};
                     ! writeln("\tAdd role ",r," to the set of Norway-
                         operated tasks"); writeln("\tRemove role ",r,"
                         from the set of remaining tasks")
                     Norway_roles := Norway_roles + 1
              elif(rand_no < 1 - (Other_cover / (Norway_cover +
                  Singapore_cover + Other_cover))) then
                     Singapore_set += {r};
                     ! writeln("\tAdd role ",r," to the set of Singapore-
                         operated tasks"); writeln("\tRemove role ",r,"
                         from the set of remaining tasks")
                     Singapore_roles := Singapore_roles + 1
              else
                     Other_set += {r};
                     ! writeln("\tAdd role ",r," to the set of Other-
                         operated tasks"); writeln("\tRemove role ",r,"
                         from the set of remaining tasks")
                     Other_roles := Other_roles + 1
              end-if
       end-do
       Remaining_set := {}
end-if
! writeln; writeln("----------------------------------"); writeln

! Identify how many working 'groups' there need to be for each contract
    type
! For the 'Norway' and 'Other' cases, there will be as many groups as
    tasks to be covered
!... while for the 'Singapore' case there can be as many groups as cover
    can be given for
```

```
Norway_groups := getsize(Norway_set);
! writeln("Will have ",Norway_groups,"\tNorway-operated groups")

Set_int := integer(getsize(Singapore_set)/2)
if(getsize(Singapore_set)/2 > Set_int) then
       Singapore_groups := Set_int +1
       !;     writeln(getsize(Singapore_set)/2," > ",Set_int," =>
           Singapore_groups := ",Set_int," +1")
else
       Singapore_groups := Set_int
       !;     writeln(getsize(Singapore_set)/2," <= ",Set_int," =>
           Singapore_groups := ",Set_int)
end-if
if(Singapore_cover/2 > Singapore_groups*2) then
       Singapore_groups := Singapore_groups*2
elif(Singapore_cover/2 > Singapore_groups) then
       Cover_int := integer(Singapore_cover/2)
       if(Singapore_cover/2 > Cover_int) then
              Singapore_groups := Cover_int +1
       else
              Singapore_groups := Cover_int
       end-if
end-if
! writeln("Will have ",Singapore_groups,"\tSingapore-operated groups\t(No
    of tasks in set: ",getsize(Singapore_set),"; No of tasks can cover: ",
    Singapore_cover,")")
Other_groups := getsize(Other_set);
! writeln("Will have ",Other_groups,"\tOther-operated groups"); writeln


! Define the 'groups' and assign the roles and employees into these groups
declarations
       group = 1..(Norway_groups + Singapore_groups + Other_groups)
       crew_in_group: array(group) of set of integer
       roles_in_group: array(group) of set of integer
       group_type: array(group) of string
end-declarations


Remaining_set := Norway_set;
! writeln; writeln("\t\tNorway tasks to be allocted to groups: (",
    Norway_roles,")",Remaining_set)
```

```
if((3*Norway_groups) > Norway_emps) then extra := (3*Norway_groups) -
    Norway_emps
else extra := 0
end-if
Remaining_emps := {}
forall(c in crew) do
        if(legal_entity(c) = "Subsea7 Contr (Norway) AS") then
            Remaining_emps += {c}
        end-if
end-do
! writeln("\t\tNorway employees to be allocted to groups: (",Norway_emps
    ,")",Remaining_emps); writeln("\t\t plus ",extra," AGECNY crew to be
    allocated"); writeln

forall(g in 1..Norway_groups) do
        group_type(g) := "Norway"

        added := FALSE
        while(added = FALSE) do
                rand_int := integer(random*role_no +1)
                if(rand_int in Remaining_set) then
                        roles_in_group(g) += {rand_int};
                        ! writeln("Add role ",rand_int," to group ",g," (
                            Norway group)")
                        Remaining_set -= {rand_int};
                        ! writeln("\tRemove role ",rand_int," from the set
                            of roles to be allocated")
                        added := TRUE
                end-if
        end-do

        number := 0
        while(number < 3) do
                rand_no := random
                if(rand_no < (extra / (extra + getsize(Remaining_emps))))
                    then
                        number := number +1;
                        ! writeln("* Add an AGENCY employee to group ",g," (
                            Norway group)")
                        extra := extra - 1
                else
                        added := FALSE
                        while(added = FALSE) do
```

```
                              rand_int := integer(random*no_of_crew +1)
                              if(rand_int in Remaining_emps) then
                                     crew_in_group(g) += {rand_int};
                                     ! writeln("Add employee ",rand_int,"
                                         to group ",g," (Norway group)")
                                     Remaining_emps -= {rand_int};
                                     ! writeln("\tRemove employee ",
                                         rand_int," from the set of crew to
                                          be allocated")
                                     added := TRUE
                              end-if
                       end-do
                       number := number + 1
                end-if
       end-do
end-do


Remaining_set := Singapore_set;
! writeln; writeln("\t\tSingapore tasks to be allocted to groups: (",
    Singapore_roles,")",Remaining_set)
if((2*Singapore_groups) > Singapore_roles) then extra_roles := (2*
    Singapore_groups) - Singapore_roles
else extra_roles := 0
end-if
if((3*Singapore_groups) > Singapore_emps) then extra := (3*
    Singapore_groups) - Singapore_emps
else extra := 0
end-if
Remaining_emps := {};
! writeln("\t\t plus ",extra_roles," DUMMY roles to be allocated")
forall(c in crew) do
       if(legal_entity(c) = "Subsea 7 (Sing) PTE-GEC") then Remaining_emps
            += {c}
       end-if
end-do
! writeln("\t\tSingapore employees to be allocted to groups: (",
    Singapore_emps,")",Remaining_emps); writeln("\t\t plus ",extra,"
    AGECNY crew to be allocated"); writeln

forall(g in (Norway_groups +1)..(Norway_groups + Singapore_groups)) do
       group_type(g) := "Singapore"
```

```
number := 0
while(number < 2) do
       rand_no := random
       if(rand_no < (extra_roles / (extra_roles + getsize(
           Remaining_set)))) then
              number := number +1;
              ! writeln("* Add a DUMMY role to group ",g," (
                  Singapore group)")
              extra_roles := extra_roles - 1
       else
              added := FALSE
              while(added = FALSE) do
                     rand_int := integer(random*role_no +1)
                     if(rand_int in Remaining_set) then
                            roles_in_group(g) += {rand_int};
                            ! writeln("Add role ",rand_int," to
                                group ",g," (Singapore group)")
                            Remaining_set -= {rand_int};
                            ! writeln("\tRemove role ",rand_int,"
                                from the set of roles to be
                                allocated")

                            second_added := FALSE
                            if(number = 0 and required(
                                role_on_vessel(rand_int)) > 1)
                                then
                                   forall(r in vessel_roles(
                                       role_on_vessel(rand_int)) |
                                        r <> rand_int) do
                                          if(second_added = FALSE
                                              and r in
                                              Remaining_set) then
                                                 roles_in_group(g
                                                     ) += {r};
                                                 ! writeln("Add
                                                     role ",r," to
                                                      group ",g,"
                                                     (Singapore
                                                     group)")
                                                 Remaining_set -=
                                                      {r};
                                                 ! writeln("\
                                                     tRemove role
```

```
                              ",r," from
                              the set of
                              roles to be
                              allocated")
                            second_added :=
                              TRUE
                        end-if
                    end-do
                end-if

                if(second_added = TRUE) then number :=
                   number +2
                else number := number +1
                end-if
                added := TRUE
            end-if

        end-do
    end-if
end-do

number := 0
while(number < 3) do
    rand_no := random
    if(rand_no < (extra / (extra + getsize(Remaining_emps))))
      then
        number := number +1;
        ! writeln("* Add an AGENCY employee to group ",g," (
          Singapore group)")
        extra := extra - 1
    else
        added := FALSE
        while(added = FALSE) do
            rand_int := integer(random*no_of_crew +1)
            if(rand_int in Remaining_emps) then
                crew_in_group(g) += {rand_int};
                ! writeln("Add employee ",rand_int,"
                  to group ",g," (Singapore group)")
                Remaining_emps -= {rand_int};
                ! writeln("\tRemove employee ",
                  rand_int," from the set of crew to
                   be allocated")
                added := TRUE
```

```
                                end-if
                        end-do
                        number := number +1
                end-if
        end-do
end-do


Remaining_set := Other_set;
! writeln; writeln("\t\tOther tasks to be allocted to groups: (",
    Other_roles,")",Remaining_set)
if((2*Other_groups) > Other_emps) then extra := (2*Other_groups) -
    Other_emps
else extra := 0
end-if
Remaining_emps := {}
forall(c in crew) do
        if(legal_entity(c) <> "Subsea7 Contr (Norway) AS" and legal_entity(
            c) <> "Subsea 7 (Sing) PTE-GEC") then
                Remaining_emps += {c}
        end-if
end-do
! writeln("\t\tOther employees to be allocted to groups: (",Other_emps,")
    ",Remaining_emps); writeln("\t\t plus ",extra," AGECNY crew to be
    allocated"); writeln

forall(g in (Norway_groups + Singapore_groups +1)..(Norway_groups +
    Singapore_groups + Other_groups)) do
        group_type(g) := "Other"

        added := FALSE
        while(added = FALSE) do
                rand_int := integer(random*role_no +1)
                if(rand_int in Remaining_set) then
                        roles_in_group(g) += {rand_int};
                        ! writeln("Add role ",rand_int," to group ",g," (
                            Other group)")
                        Remaining_set -= {rand_int};
                        ! writeln("\tRemove role ",rand_int," from the set
                            of roles to be allocated")
                        added := TRUE
                end-if
        end-do
```

```
       number := 0
       while(number < 2) do
              rand_no := random
              if(rand_no < (extra / (extra + getsize(Remaining_emps))))
                  then
                     number := number +1;
                     ! writeln("* Add an AGENCY employee to group ",g," (
                         Other group)")
                     extra := extra - 1
              else
                     added := FALSE
                     while(added = FALSE) do
                            rand_int := integer(random*no_of_crew +1)
                            if(rand_int in Remaining_emps) then
                                   crew_in_group(g) += {rand_int};
                                   ! writeln("Add employee ",rand_int,"
                                       to group ",g," (Other group)")
                                   Remaining_emps -= {rand_int};
                                   ! writeln("\tRemove employee ",
                                       rand_int," from the set of crew to
                                        be allocated")
                                   added := TRUE
                            end-if
                     end-do
                     number := number +1
              end-if
       end-do
end-do

! writeln
! writeln("Group summary:")
! forall(g in group) do
!      write("Group ")
!      if(g < 10) then write(" "); end-if
!      write(g," (",group_type(g),")")
!      write("\tRoles:\t",roles_in_group(g))
!      write("\t\tCrew: \t",crew_in_group(g),"\n")
! end-do
! writeln; writeln("----------------------------------"); writeln


!------------------------------------------------------------------------
```

```
        ! Next, determine the appropriate crew-change times, and initial
            assignments to the roles

        declarations
                weeks = 1..no_of_weeks

!               tasks = 1..no_of_tasks
!               task_vessel: array(tasks) of integer
!               task_role: array(tasks) of integer
!               task_start, task_duration: array(tasks) of integer

                shipcrew: array(1..3) of integer
                allocated: integer
                initial: array(crew, roles, weeks) of integer
                ag_initial: array(roles, weeks) of integer

                board, depart: array(crew, vessel, weeks) of integer
                ag_board, ag_depart: array(roles, weeks) of integer

                previous: integer
                next_change: integer
                more_previous: integer
                back_count: integer

                start_in_role: array(roles) of integer
                work_resource_zero: array(crew) of integer
                rest_resource_zero: array(crew) of integer
                ag_work_zero: array(roles) of integer

                group_max_work, group_min_rest, emp_min_rest: integer
                overall_max_work: integer
                start_duration: array(roles) of integer
        end-declarations


        forall(c in crew) do
                forall(r in roles, w in weeks) initial(c,r,w) := 0
                work_resource_zero(c) := 0
                rest_resource_zero(c) := 0
        end-do
        forall(r in roles) do
                forall(w in weeks) ag_initial(r,w) := 0
                start_duration(r) := 0
```

```
end-do

overall_max_work := 0
forall(g in group) do
       ! First, need to know duty lengths:
       ! writeln; writeln("Allocate assignments for Group ",g,":")
       if(group_type(g) = "Norway") then
              group_max_work := 2
              group_min_rest := 4
              emp_min_rest := 4
       elif(group_type(g) = "Singapore") then
              group_max_work := 10
              group_min_rest := 5
              emp_min_rest := 5
       elif(group_type(g) = "Other") then
              forall(r in roles_in_group(g)) do
                     if(vessel_location(role_on_vessel(r)) = "Europe")
                         then
                            group_max_work := 4
                            group_min_rest := 4
                     else
                            group_max_work := 5
                            group_min_rest := 5
                     end-if
              end-do
              emp_min_rest := 4
       else
              group_max_work := 0
              group_min_rest := 0
       end-if
       ! writeln("\tMax work: ",group_max_work,"\tMin rest: ",
           group_min_rest); writeln

       if(group_max_work > overall_max_work) then
              overall_max_work := group_max_work
       end-if


       ! Can now determine the starting duration for each of the roles
       index := 0
       previous := 0
       forall(r in roles_in_group(g)) do
              index := index + 1
```

```
        if(index = 1) then
                start_duration(r) := integer(group_max_work * random
                    )
                previous := start_duration(r);
                ! writeln("\tStart duration for role ",r," is ",
                    start_duration(r)," weeks")

        else
                if(previous < (group_max_work / 2)) then
                        start_duration(r) := previous + integer(
                            group_max_work / 2);
                        ! writeln("\tStart duration for role ",r," is
                             ",start_duration(r)," weeks")
                else
                        start_duration(r) := previous - integer(
                            group_max_work / 2);
                        ! writeln("\tStart duration for role ",r," is
                             ",start_duration(r)," weeks")
                end-if
        end-if
end-do
! writeln

! Determine the order in which the crew in each 'group' will work
    their tasks
forall(i in 1..3) shipcrew(i) := 0
allocated := 0
if(group_type(g) = "Other") then
        if(getsize(crew_in_group(g)) > 0) then
                forall(c in crew_in_group(g)) do
                        if(allocated = 0) then
                                rand_no := random
                                if(rand_no < 0.5) then
                                        shipcrew(1) := c;
                                        ! writeln("\tEmployee ",c," is
                                            crew number 1")
                                        allocated := allocated + 1
                                else
                                        shipcrew(2) := c;
                                        ! writeln("\tEmployee ",c," is
                                            crew number 2")
                                        allocated := allocated + 1
                                end-if
                        else
                                if(shipcrew(1) = 0) then
                                        shipcrew(1) := c;
                                        ! writeln("\tEmployee ",c," is
                                            crew number 1")
                                        allocated := allocated + 1
                                else
                                        shipcrew(2) := c;
                                        ! writeln("\tEmployee ",c," is
                                            crew number 2")
                                        allocated := allocated + 1
                                end-if
                        end-if
                end-do
        end-if

else
        if(getsize(crew_in_group(g)) > 0) then
                forall(c in crew_in_group(g)) do
                        if(allocated = 0) then
                                rand_no := random
                                if(rand_no < (1/3)) then
                                        shipcrew(1) := c;
                                        ! writeln("\tEmployee ",c," is
                                            crew number 1")
                                        allocated := allocated + 1
                                else
                                        if(rand_no < (2/3)) then
                                                shipcrew(2) := c;
                                                ! writeln("\tEmployee ",
                                                    c," is crew number
                                                    2")
                                                allocated := allocated +
                                                     1
                                        else
                                                shipcrew(3) := c;
                                                ! writeln("\tEmployee ",
                                                    c," is crew number
                                                    3")
                                                allocated := allocated +
                                                     1
                                        end-if
                               end-if
                       else
                               if(allocated = 1) then
                                       rand_no := random
                                         if(rand_no < 0.5) then
                                                if(shipcrew(1) = 0) then
                                                       shipcrew(1) := c
                                                           ;
                                                       ! writeln("\
                                                           tEmployee ",c
                                                           ," is crew
                                                           number 1")
                                                       allocated :=
                                                           allocated + 1
                                                else
                                                       shipcrew(2) := c
                                                           ;
                                                       ! writeln("\
                                                           tEmployee ",c
                                                           ," is crew
                                                           number 2")
                                                       allocated :=
                                                           allocated + 1
                                                end-if
                                         else
                                                if(shipcrew(3) = 0) then
                                                       shipcrew(3) := c
                                                           ;
                                                       ! writeln("\
                                                           tEmployee ",c
                                                           ," is crew
                                                           number 3")
                                                       allocated :=
                                                           allocated + 1
                                                else
                                                       shipcrew(2) := c
                                                           ;
                                                       ! writeln("\
                                                           tEmployee ",c
                                                           ," is crew
                                                           number 2")
                                                       allocated :=
                                                           allocated + 1
```

```
                                                end-if
                                        end-if
                                else
                                        forall(i in 1..3 | shipcrew(i)
                                            = 0) shipcrew(i) := c;
                                        ! forall(i in 1..3 | shipcrew(i
                                            ) = c) writeln("\tEmployee
                                            ",c," is crew number ",i
                                            ,"*")
                                end-if
                        end-if
                end-do
        end-if
end-if
! writeln


! ... and now define the initial (un-disrupted) solution:
previous := 0
index := 0

forall(r in roles_in_group(g)) do
        index := index + 1
        start_in_role(r) := 0

        if(index = 1) then
                previous := previous +1
        else
                forall(j in roles_in_group(g) | r <> j) do
                        if(start_duration(j) > start_duration(r))
                            then
                                previous := 2
                        else
                                previous := 3
                        end-if
                end-do
        end-if


        ! Calcualte ALL relevant 'rest_resource_zero' values...
        more_previous := previous
        if(start_duration(r) = 0) then
                if(shipcrew(more_previous) > 0) then
                        rest_resource_zero(shipcrew(more_previous))
                            := emp_min_rest;
                        ! writeln("\t(Employee ",shipcrew(
                            more_previous)," requires ",
                            rest_resource_zero(shipcrew(more_previous
                            ))," periods of rest as at time zero)")
                end-if
        end-if

        back_count := group_max_work - start_duration(r)
        while(back_count < emp_min_rest) do
                if(more_previous > 1) then
                        more_previous := more_previous - 1
                else
                        if(group_type(g) = "Other") then
                                more_previous := 2
                        else
                                more_previous := 3
                        end-if
                end-if

                if(shipcrew(more_previous) > 0) then
                        rest_resource_zero(shipcrew(more_previous))
                            := emp_min_rest - back_count;
                        ! writeln("\t(Employee ",shipcrew(
                            more_previous)," requires ",
                            rest_resource_zero(shipcrew(more_previous
                            ))," periods of rest as at time zero)")
                end-if

                back_count := back_count + group_max_work
        end-do


        ! Assign the relevant 'work_resource_zero' values
        if(shipcrew(previous) > 0) then
                work_resource_zero(shipcrew(previous)) := (
                    group_max_work - start_duration(r))
        else
                ag_work_zero(r) := (group_max_work - start_duration(
                    r))
        end-if
```

```
! ... and now make the initial assignments:
start_in_role(r) := shipcrew(previous);
! writeln("\tEmployee ",shipcrew(previous)," is in role ",r
    ," at the start of the planning period")

next_change := start_duration(r) + 1;
! writeln("\t\tNext crew change will take place ahead of
    time window ",next_change)

time_count := 1

while(time_count < no_of_weeks) do
       if(time_count < next_change) then
              if(shipcrew(previous) > 0) then
                     initial(shipcrew(previous), r,
                         time_count) := 1;
                     ! writeln("\t\tEmployee ",shipcrew(
                         previous)," works in role ",r," in
                          time window ",time_count)
              else
                     ag_initial(r, time_count) := 1;
                     ! writeln("\t\tAn AGENCY employee
                         works in role ",r," in time window
                          ",time_count)
              end-if

       else
              if(shipcrew(previous) > 0) then
                     depart(shipcrew(previous),
                         role_on_vessel(r), time_count) :=
                         1;
                     ! writeln("\t\tEmployee ",shipcrew(
                         previous)," leaves vessel ",
                         role_on_vessel(r)," ahead of time
                         window ",time_count)
              else
                     ag_depart(r, time_count) := 1;
                     ! writeln("\t\tAn AGENCY employee
                         leaves role ",r," ahead of time
                         window ",time_count)
              end-if
```

```
        if(group_type(g) = "Other") then
            if(previous = 2) then
                previous := 1
            else
                previous := previous +1
            end-if
        else
            if(previous = 3) then
                previous := 1
            else
                previous := previous +1
            end-if
        end-if

        if(shipcrew(previous) > 0) then
            board(shipcrew(previous),
                role_on_vessel(r), time_count) :=
                1;
            ! writeln("\tEmployee ",shipcrew(
                previous)," boards vessel ",
                role_on_vessel(r)," ahead of time
                window ",time_count)
            initial(shipcrew(previous), r,
                time_count) := 1;
            ! writeln("\t\tEmployee ",shipcrew(
                previous)," works in role ",r," in
                 time window ",time_count)
        else
            ag_board(r, time_count) := 1;
            ! writeln("\tAn AGENCY employee boards
                 role ",r," ahead of time window
                ",time_count)
            ag_initial(r, time_count) := 1;
            ! writeln("\t\tAn AGENCY employee
                works in role ",r," in time window
                 ",time_count)
        end-if

        next_change := next_change + group_max_work;
        ! writeln("\t\tNext crew change will take
            place ahead of time window ",next_change)
    end-if
```

```
                        time_count := time_count + 1
                end-do
                        ! writeln
        end-do
end-do
! writeln; writeln("---------------------------------"); writeln


!------------------------------------------------------------------------
! Generate employee availabilities:
declarations
        days = 0..(7*no_of_weeks)
        available_day: array(crew,days) of integer
        available_week: array(crew,weeks) of integer
!       ill_given_prev_fit: real
!       ill_given_prev_ill: real
        time_reduction: real
end-declarations

! write("\t ")
! forall(w in weeks) do
!       write("|  Week ")
!       if(w < 10) then write(" "); end-if
!       write(w,"  ")
! end-do
! write("\n")
! write("\t 0")
! forall(w in weeks) do
!       forall(i in 1..7) do
!               if(i = 1) then write("|")
!               else write(" ")
!               end-if
!               write(i)
!       end-do
! end-do
! write("\n")

forall(c in crew) do
! write(c,"\t")
        rand_no := random
        if(rand_no < 0.008) then
                available_day(c,0) := 0;
                ! write(" X")
```

```
else
       available_day(c,0) := 1;
       ! write(" a")
end-if

forall(d in days | d > 0) do

       time_reduction := 0
       if(use_time_reduction = TRUE) then
              if(d > 28) then
                     time_reduction := (d-28)/((7*no_of_weeks)-28)
              end-if
       end-if

       rand_no := random
       if(available_day(c,d-1) = 1) then
              if(rand_no < ill_given_prev_fit*(1-time_reduction))
                  then
                     available_day(c,d) := 0;
                     ! write(" X")
              else
                     available_day(c,d) := 1;
                     ! write(" a")
              end-if
       else
              if(rand_no < ill_given_prev_ill*(1-time_reduction))
                  then
                     available_day(c,d) := 0;
                     ! write(" X")
              else
                     available_day(c,d) := 1;
                     ! write(" a")
              end-if
       end-if
end-do
! write("\n")

forall(w in weeks) do
       available_week(c,w) := 1
       forall(d in 1..7) do
              if(available_day(c, (7*(w-1) + d)) = 0) then
                     available_week(c,w) := 0
              end-if
```

```
                end-do
        end-do

end-do


!------------------------------------------------------------------------
! Generate costs and fixed-contract terms:

declarations
        work_change_cost: array(crew, roles, weeks) of real
        board_change_cost, depart_change_cost: array(crew, vessel, weeks)
            of real

        ag_work_change_cost: array(roles, weeks) of real
        ag_board_change_cost, ag_depart_change_cost: array(roles, weeks) of
             real

        delta = 1..(overall_max_work)
        ext_change_cost: array(delta, crew, roles, weeks) of real


        disruption_factor: real


        under_rate, over_rate: array(crew) of real
        current_undertime, current_overtime, exp_worktime: array(crew) of
            real
        current_worktime: array(crew) of real
        g_weeks: real
end-declarations


forall(w in weeks) do

        forall(v in vessel, c in crew) do

                board_change_cost(c,v,w) := 0
                if(board(c,v,w) = 0) then
                        if(crew_continent(c) = "European") then
                                if(vessel_location(v) = "Europe") then
                                    board_change_cost(c,v,w) := 3
                                elif(vessel_location(v) = "Africa" or
                                    vessel_location(v) = "USA") then
                                    board_change_cost(c,v,w) := 6
                                else board_change_cost(c,v,w) := 8
                                end-if
                        elif(crew_continent(c) = "North American") then
                                if(vessel_location(v) = "USA") then
                                    board_change_cost(c,v,w) := 4
                                elif(vessel_location(v) = "Brazil" or
                                    vessel_location(v) = "Europe") then
                                    board_change_cost(c,v,w) := 6
                                else board_change_cost(c,v,w) := 9
                                end-if
                        elif(crew_continent(c) = "Asian" or crew_continent(c
                            ) = "Australasian") then
                                if(vessel_location(v) = "AsiaPac") then
                                    board_change_cost(c,v,w) := 4
                                else board_change_cost(c,v,w) := 9
                                end-if
                        else board_change_cost(c,v,w) := 9
                        end-if

                else
                        if(crew_continent(c) = "European") then
                                if(vessel_location(v) = "Europe") then
                                    board_change_cost(c,v,w) := -2
                                elif(vessel_location(v) = "Africa" or
                                    vessel_location(v) = "USA") then
                                    board_change_cost(c,v,w) := -5
                                else board_change_cost(c,v,w) := -7
                                end-if
                        elif(crew_continent(c) = "North American") then
                                if(vessel_location(v) = "USA") then
                                    board_change_cost(c,v,w) := -3
                                elif(vessel_location(v) = "Brazil" or
                                    vessel_location(v) = "Europe") then
                                    board_change_cost(c,v,w) := -5
                                else board_change_cost(c,v,w) := -8
                                end-if
                        elif(crew_continent(c) = "Asian" or crew_continent(c
                            ) = "Australasian") then
                                if(vessel_location(v) = "AsiaPac") then
                                    board_change_cost(c,v,w) := -3
                                else board_change_cost(c,v,w) := -8
                                end-if
                        else board_change_cost(c,v,w) := -8
                        end-if

                        if(w < 5) then
                                board_change_cost(c,v,w) := board_change_cost
                                    (c,v,w)/2
                        end-if
                end-if


                depart_change_cost(c,v,w) := 0
                if(depart(c,v,w) = 0) then
                        if(crew_continent(c) = "European") then
                                if(vessel_location(v) = "Europe") then
                                    depart_change_cost(c,v,w) := 3
                                elif(vessel_location(v) = "Africa" or
                                    vessel_location(v) = "USA") then
                                    depart_change_cost(c,v,w) := 6
                                else depart_change_cost(c,v,w) := 8
                                end-if
                        elif(crew_continent(c) = "North American") then
                                if(vessel_location(v) = "USA") then
                                    depart_change_cost(c,v,w) := 4
                                elif(vessel_location(v) = "Brazil" or
                                    vessel_location(v) = "Europe") then
                                    depart_change_cost(c,v,w) := 6
                                else depart_change_cost(c,v,w) := 9
                                end-if
                        elif(crew_continent(c) = "Asian" or crew_continent(c
                            ) = "Australasian") then
                                if(vessel_location(v) = "AsiaPac") then
                                    depart_change_cost(c,v,w) := 4
                                else depart_change_cost(c,v,w) := 9
                                end-if
                        else depart_change_cost(c,v,w) := 9
                        end-if

                else
                        if(crew_continent(c) = "European") then
                                if(vessel_location(v) = "Europe") then
                                    depart_change_cost(c,v,w) := -2
```

```
                        elif(vessel_location(v) = "Africa" or
                            vessel_location(v) = "USA") then
                            depart_change_cost(c,v,w) := -5
                        else depart_change_cost(c,v,w) := -7
                        end-if
                elif(crew_continent(c) = "North American") then
                        if(vessel_location(v) = "USA") then
                            depart_change_cost(c,v,w) := -3
                        elif(vessel_location(v) = "Brazil" or
                            vessel_location(v) = "Europe") then
                            depart_change_cost(c,v,w) := -5
                        else depart_change_cost(c,v,w) := -8
                        end-if
                elif(crew_continent(c) = "Asian" or crew_continent(c
                    ) = "Australasian") then
                        if(vessel_location(v) = "AsiaPac") then
                            depart_change_cost(c,v,w) := -3
                        else depart_change_cost(c,v,w) := -8
                        end-if
                else depart_change_cost(c,v,w) := -8
                end-if

                if(w < 5) then
                        depart_change_cost(c,v,w) :=
                            depart_change_cost(c,v,w)/2
                end-if
        end-if

end-do


forall(r in roles) do

        forall(c in crew) do
                work_change_cost(c,r,w) := 0
                if(contract_type(c) = "Permanent") then
                        work_change_cost(c,r,w) := 1

                else
                        if(crew_nation(c) = "NORWAY") then
                                if(initial(c,r,w) = 1) then
                                        work_change_cost(c,r,w) := -90
                                else
                                        work_change_cost(c,r,w) := 100
                                end-if

                        else
                                if(initial(c,r,w) = 1) then
                                        work_change_cost(c,r,w) := -60
                                else
                                        work_change_cost(c,r,w) := 70
                                end-if
                        end-if
                end-if

                if(w < 5) then
                        if(work_change_cost(c,r,w) > 0) then
                                work_change_cost(c,r,w) :=
                                        disruption_factor_near*
                                        work_change_cost(c,r,w)
                        elif(work_change_cost(c,r,w) < 0) then
                                work_change_cost(c,r,w) := (1/
                                        disruption_factor_near)*
                                        work_change_cost(c,r,w)
                        end-if
                elif(w < no_of_weeks) then
                        if(work_change_cost(c,r,w) > 0) then
                                work_change_cost(c,r,w) :=
                                        disruption_factor_long*
                                        work_change_cost(c,r,w)
                        elif(work_change_cost(c,r,w) < 0) then
                                work_change_cost(c,r,w) := (1/
                                        disruption_factor_long)*
                                        work_change_cost(c,r,w)
                        end-if
                end-if


                forall(d in delta) do
                        ext_change_cost(d,c,r,w) := 0

                        if(d = 5 and vessel_location(role_on_vessel(r
                            )) = "Europe") then
                                if(legal_entity(c) <> "Subsea7 Contr (
                                        Norway) AS" and legal_entity(c) <>
                                         "Subsea 7 (Sing) PTE-GEC") then
```

```
                              if(contract_type(c) = "
                                  Permanent") then
                                      ext_change_cost(d,c,r,w)
                                           := 0.5

                              else
                                      if(crew_nation(c) = "
                                          NORWAY") then
                                              ext_change_cost(
                                                  d,c,r,w) :=
                                                  50
                                      else
                                              ext_change_cost(
                                                  d,c,r,w) :=
                                                  35
                                      end-if
                              end-if
                      end-if
              end-if
      end-do
end-do


ag_board_change_cost(r,w) := 0
if(ag_board(r,w) = 0) then
      ag_board_change_cost(r,w) := 2
else
      if(w > 4) then
              ag_board_change_cost(r,w) := -2
      else
              ag_board_change_cost(r,w) := -1
      end-if
end-if

ag_depart_change_cost(r,w) := 0
if(ag_depart(r,w) = 0) then
      ag_depart_change_cost(r,w) := 2
else
      if(w > 4) then
              ag_depart_change_cost(r,w) := -2
      else
              ag_depart_change_cost(r,w) := -1
```

```
                end-if
            end-if

            ag_work_change_cost(r,w) := 0
            if(ag_initial(r,w) = 0) then
                ag_work_change_cost(r,w) := 200
            else
                ag_work_change_cost(r,w) := -180
            end-if

            if(ag_work_change_cost(r,w) > 0) then
                ag_work_change_cost(r,w) := agency_penalty*
                    ag_work_change_cost(r,w)
            elif(ag_work_change_cost(r,w) < 0) then
                ag_work_change_cost(r,w) := (1/agency_penalty)*
                    ag_work_change_cost(r,w)
            end-if

        end-do
end-do


! Fixed contract terms:

forall(c in crew | contract_type(c) = "Permanent") under_rate(c) := 70
forall(c in crew | contract_type(c) = "Permanent") do
        if(legal_entity(c) = "Subsea7 Contr (Norway) AS") then over_rate(c)
            := 500
        elif(legal_entity(c) = "Subsea 7 Norway") then over_rate(c) := 500
        elif(crew_nation(c) = "Norway") then over_rate(c) := 120
        else over_rate(c) := 70
        end-if
end-do


g_weeks := 26
forall(c in crew | contract_type(c) = "Permanent") do
        exp_worktime(c) := 26 - sum(r in roles, w in weeks)(initial(c,r,w))
end-do


forall(c in crew | contract_type(c) = "Permanent") do
```

```
        current_worktime(c) := exp_worktime(c) + sum(r in roles, w in weeks
            )(initial(c,r,w)*available_week(c,w))
end-do


forall(c in crew | contract_type(c) = "Permanent") do
        current_undertime(c) := 0
        current_overtime(c) := 0
        if(current_worktime(c) < 26) then
                current_undertime(c) := 26 - current_worktime(c)
        end-if
        if(current_worktime(c) > 26) then
                current_undertime(c) := current_worktime(c) - 26
        end-if
end-do



!-----------------------------------------------------------------------
! Print dataset into output file:
declarations
        on_vessel: boolean
        ag_number: integer
end-declarations


fopen(OUTFILE, F_OUTPUT)
        writeln("! Dataset generated for time-windows formulation based on
            real data on Subsea 7 Captains.")
        writeln
        writeln("! Parameters used:")
        writeln
        writeln("! -> Probability employee ill given previously fit:\t",
            ill_given_prev_fit)
        writeln("! -> Probability employee ill given previously ill:\t",
            ill_given_prev_ill)
        if(use_time_reduction = TRUE) then writeln("! -> Time-reduction of
            probabilities:\t\t\tON")
        else writeln("! -> Time-reduction of probabilities:\t\t\tOFF")
        end-if
        writeln
        writeln("! -> Disruption factor for near tasks:\t\t\t",
            disruption_factor_near)
```

```
writeln("! -> Disruption factor for more distant tasks:\t\t",
    disruption_factor_long)
writeln("! -> Agency crew penalty:\t\t\t\t",agency_penalty)
writeln
writeln("!---------------------------------------------")
writeln
writeln
writeln("WEEKS_TO_PLAN: ",no_of_weeks)
writeln
writeln
write("REG_EMP : [")
forall(c in crew) write(" '",crew_label(c),"' ")
write("] \n")
writeln
write("GUARANTEED: [")
forall(c in crew | contract_type(c) = "Permanent") write(" '",
    crew_label(c),"'")
write("] \n")
writeln
writeln
writeln("VESSELS: [")
forall(v in vessel | required(v) > 0) write(" '",vessel_name(v),"'
    ")
write("] \n")
writeln
writeln("ROLES: [")
forall(v in vessel | required(v) > 0) do
       index := 0
       write("('",vessel_name(v),"') [")
       forall(r in vessel_roles(v)) do
              index := index + 1
              write(" '",vessel_label(v),"-")
              if(index < 10) then write("0"); end-if
              write(index,"' ")
       end-do
       write("] \n")
end-do
write("] \n")
writeln
writeln
write("under_rate:\t[")
forall(c in crew | contract_type(c) = "Permanent") write(round(
    under_rate(c)),"\t")
```

```
write("]\nover_rate:\t[")
forall(c in crew | contract_type(c) = "Permanent") write(round(
    over_rate(c)),"\t")
write("]\ng_weeks:\t[")
forall(c in crew | contract_type(c) = "Permanent") write(g_weeks,"\
    t")
write("]\nexp_worktime:\t[")
forall(c in crew | contract_type(c) = "Permanent") write(
    exp_worktime(c),"\t")
write("]\n")
writeln
writeln
writeln("required: [")
forall(r in roles) do
        forall(w in weeks) write(" 1")
        write("\n")
end-do
write("]\n")
writeln
writeln
writeln("eligable: [")
forall(c in crew) do
        forall(r in roles) do
                forall(w in weeks) do
                        write(" ",available_week(c,w))
                end-do
        end-do
        write("\n")
end-do
forall(r in roles, w in weeks) write(" 1")
write("] \n")
writeln
writeln
writeln("starting: [")
forall(c in crew) do
        forall(v in vessel | required(v) > 0) do
                on_vessel := FALSE
                forall(r in vessel_roles(v)) do
                        if(start_in_role(r) = c) then
                                on_vessel := TRUE
                        end-if
                end-do
                if(on_vessel = TRUE) then write(" 1")
                else write(" 0")
                end-if
        end-do
        write("\n")
end-do
forall(v in vessel | required(v) > 0) do
        ag_number := 0
        forall(r in vessel_roles(v)) do
                if(start_in_role(r) = 0) then
                        ag_number := ag_number + 1
                end-if
        end-do
        write(" ",ag_number)
end-do
write("] \n")
writeln
write("ag_starting: [")
forall(r in roles) do
        if(start_in_role(r) = 0) then write(" 1")
        else write(" 0")
        end-if
end-do
write("] \n")
writeln
writeln
writeln
write("work_zero: [")
forall(c in crew) write(" ",work_resource_zero(c))
writeln("]")
write("rest_zero: [")
forall(c in crew) write(" ",rest_resource_zero(c))
writeln("]")
writeln
write("max_work: [")
forall(c in crew) do
        if(legal_entity(c) = "Subsea7 Contr (Norway) AS") then write
            (" 2")
        elif(legal_entity(c) = "Subsea 7 (Sing) PTE-GEC") then write
            (" 10")
        else write(" 5")
        end-if
end-do
write("] \n")
write("min_rest: [")
forall(c in crew) do
       if(legal_entity(c) = "Subsea7 Contr (Norway) AS") then write
           (" 4")
       elif(legal_entity(c) = "Subsea 7 (Sing) PTE-GEC") then write
           (" 5")
       else write(" 4")
       end-if
end-do
write("] \n")
writeln
writeln
write("ag_work_zero: [")
forall(r in roles) write(" ",ag_work_zero(r))
write("] \n")
write("ag_max_work: [")
forall(r in roles) do
       forall(g in group) do
              if(r in roles_in_group(g)) then
                     if(group_type(g) = "Norway") then write(" 2")
                     elif(group_type(g) = "Singapore") then write
                         (" 10")
                     else write(" 5")
                     end-if
              end-if
       end-do
end-do
write("] \n")
writeln
writeln
writeln("!-------------------------------------------")
writeln("! Data on current schedule:")
writeln
writeln("cur_allocate: [")
forall(c in crew) do
       forall(r in roles) do
              forall(w in weeks) write(" ",initial(c,r,w)*
                  available_week(c,w))
       end-do
       write("\n")
end-do
forall(r in roles) do
       forall(w in weeks) write(" ",ag_initial(r,w))
```

```
end-do
write("] \n")
writeln
writeln
writeln("cur_board: [")
forall(c in crew) do
       forall(v in vessel | required(v) > 0) do
              forall(w in weeks) write(" ",board(c,v,w))
       end-do
       write("\n")
end-do
write("] \n")
writeln
writeln
writeln("cur_depart: [")
forall(c in crew) do
       forall(v in vessel | required(v) > 0) do
              forall(w in weeks) write(" ",depart(c,v,w))
       end-do
       write("\n")
end-do
write("] \n")
writeln
writeln
writeln("cur_ag_rboard: [")
forall(r in roles) do
       forall(w in weeks) write(" ",ag_board(r,w))
       write("\n")
end-do
write("] \n")
writeln
writeln
writeln("cur_ag_rdepart: [")
forall(r in roles) do
       forall(w in weeks) write(" ",ag_depart(r,w))
       write("\n")
end-do
write("] \n")
writeln
writeln
write("cur_undertime: [")
forall(c in crew | contract_type(c) = "Permanent") write(
    current_undertime(c),"\t")
```

```
write("]\n")
write("cur_overtime: [")
forall(c in crew | contract_type(c) = "Permanent") write(
    current_overtime(c),"\t")
write("]\n")
writeln
writeln
writeln("cur_long_work: [")
forall(d in delta) do
       forall(c in crew) do
              forall(r in roles, w in weeks) write(" 0")
              write("\n")
       end-do
       forall(r in roles, w in weeks) write(" 0")
       write("\n")
       writeln
end-do
write("]\n")
writeln
writeln
writeln("!--------------------------------------------")
writeln("! Cost of changes:")
writeln
writeln("board_chng_cost: [")
forall(c in crew) do
       forall(v in vessel | required(v) > 0) do
              forall(w in weeks) write(board_change_cost(c,v,w),"\
                  t")
       end-do
       write("\n")
end-do
write("] \n")
writeln
writeln("depart_chng_cost: [")
forall(c in crew) do
       forall(v in vessel | required(v) > 0) do
              forall(w in weeks) write(depart_change_cost(c,v,w)
                  ,"\t")
       end-do
       write("\n")
end-do
write("] \n")
writeln
```

```
writeln
writeln("ag_board_chng_cost: [")
forall(r in roles) do
       forall(w in weeks) write(ag_board_change_cost(r,w),"\t")
       write("\n")
end-do
write("] \n")
writeln
writeln("ag_depart_chng_cost: [")
forall(r in roles) do
       forall(w in weeks) write(ag_depart_change_cost(r,w),"\t")
       write("\n")
end-do
write("] \n")
writeln
writeln
writeln("work_chng_cost: [")
forall(c in crew) do
       forall(r in roles) do
              forall(w in weeks) write(work_change_cost(c,r,w),"\t
                  ")
       end-do
       write("\n")
end-do
forall(r in roles) do
       forall(w in weeks) write(ag_work_change_cost(r,w),"\t")
end-do
write("] \n")
writeln
writeln
writeln
writeln("extension_chng_cost: [")
forall(d in delta) do
       forall(c in crew) do
              forall(r in roles) do
                     forall(w in weeks) write(ext_change_cost(d,c,
                         r,w),"\t")
              end-do
              write("\n")
       end-do
       forall(r in roles, w in weeks) write(" 0")
       write("\n")
       writeln
```

```
               end-do
               write("]\n")
               writeln
               writeln



       fclose(F_OUTPUT)

end-procedure


!-----------------------------------------------

!forall(ins in INSTANCES) do
!
!      OUTFILE := instance_name(ins)
!      writeln("Instance: ",instance_name(ins)); writeln
!
!      ! Parameters to be varied for various data sets:
!      ill_given_prev_fit := ill_given_fit(ins)
!      ill_given_prev_ill := ill_given_ill(ins)
!
!      use_time_reduction := use_TR(ins)
!
!      disruption_factor_near := DF_near(ins)
!      disruption_factor_long := DF_long(ins)
!      agency_penalty := AG_pen(ins)
!
       MAIN_PROG
!      writeln; writeln("----------------------------------------------------");
     writeln; writeln
!end-do


end-model
```

E.2.2 Cost-minimization algorithm

Here we give the code used to solve the Time-Windows problem using the Cost-Minimization approach described in section 6.3.1. This was implemented using the FICO Xpress software.

```
model ModelName
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver

! Recovery-type problem version of the time windows formulation, intended to be
    more realistic.
! As before, this program aims to solve the problem directly from this
    formulation.

parameters
       DATE = "27-11-14"
       PARAMETERFILE = "batch-input-parameters.dat"
end-parameters

declarations
       InstanceName: string
end-declarations

initializations from PARAMETERFILE
       InstanceName
end-initializations


DATAFILE := InstanceName+"\\Time-Windows - Captains - "+InstanceName+".txt"
OUTPUTFILE := InstanceName+"\\Logfile - TW-recovery - 2mins - "+InstanceName+" -
    "+DATE+".txt"
SUMMARYFILE := "Results - TW-recovery - 2mins - "+DATE+".txt"
!SOLUTIONFILE := InstanceName+"\\Best direct soln - "+InstanceName+" - "+DATE+".
    txt"


! Forces the program to stop after given length of time
setparam("XPRS_maxtime",-120)  ! Use this for the two-minute time-limit...
! setparam("XPRS_maxtime",-3600) ! ... or this for the one-hour time-limit

declarations
       status: array({XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH}) of string !
           Used to indicate the solution status of the probelm

       REG_EMP: set of string       ! Regular employee names / numbers (ie not
           Agency)
       ALL_EMP: set of string       ! Lables for ALL crew (including Agency)
```

```
        GUARANTEED:   set of string ! Set of employees on guaranteed days
            contracts
        VESSELS: set of string       ! Labels / names of vessels
        WEEKS_TO_PLAN: integer       ! Length of planning horizon in weeks
        ROLES: array(VESSELS) of set of string     ! Labels for the roles which
            will require cover, divided by vessels
        ALL_ROLES: set of string     ! List of all roles

        exp_worktime, g_weeks, under_rate, over_rate: array(GUARANTEED) of real  !
             Data relating to over/undertime payments for employees
end-declarations

initializations from DATAFILE
        REG_EMP GUARANTEED VESSELS WEEKS_TO_PLAN ROLES
        under_rate over_rate g_weeks exp_worktime
end-initializations

ALL_EMP := REG_EMP + {"AGENCY"}
ALL_ROLES := {}
FORALL (v in VESSELS) ALL_ROLES += ROLES(v)

declarations
        TIME = 1..WEEKS_TO_PLAN      ! Time index

        allocate: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar ! Variable for
            allocating employee to role during given time period
        board, depart: array(REG_EMP, VESSELS, TIME) of mpvar    ! =1 if employee
             boards / departs vessel in given time period, 0 otherwise
                ! or takes a non-negative integer value for agency crew
        ag_rboard, ag_rdepart: array(ALL_ROLES, TIME) of mpvar   ! =1 if agency
            crew starts / ends working on a role in given time period, 0 otherwise
        undertime, overtime: array(GUARANTEED) of mpvar                  !
            Variables to calculate the amount of under/overtime carried out by
            employee
        work_total, rest_total: array(REG_EMP, TIME) of mpvar    ! Used to track
            the consecutive working time / rest period requirements of each
            employee
        ag_work_total: array(ALL_ROLES, TIME) of mpvar                   ! Used to
             track the consecutive working time of the agency employees

        board_chng_cost, depart_chng_cost: array(REG_EMP, VESSELS, TIME) of real !
             Costs of CHANGES TO employees boarding / leaving vessel
        ag_board_chng_cost, ag_depart_chng_cost: array(ALL_ROLES, TIME) of real !
            Costs of CHANGES TO agency employees boarding / leaving for a given
            role
        work_chng_cost: array(ALL_EMP, ALL_ROLES, TIME) of real
                          ! (Direct) Costs of CHANGES TO employees working a given
             role at a given time

        required: array(ALL_ROLES, TIME) of integer                      ! =1 if
            role r is required at time t, =0 otherwise
        eligable: array(ALL_EMP, ALL_ROLES, TIME) of integer ! =1 if emp i can
            carry out role r at time t, =0 otherwise
        starting: array(ALL_EMP, VESSELS) of integer             ! =1 if emp i is
             on board vessel k at time 0, =0 otherwise
               ! or takes a non-negative integer value for agency crew
        ag_starting: array(ALL_ROLES) of integer                         ! =1 if
            agency employee is in role r at time 0, =0 otherwise
        work_zero, rest_zero: array(REG_EMP) of integer          ! initial values
             of work_total and rest_total at time zero
        ag_work_zero: array(ALL_ROLES) of integer                        ! initial
             value of work total for agency crew task by task
        max_work, min_rest: array(REG_EMP) of integer            ! legal maximum
            on working time, minimum on resting time
        ag_max_work: array(ALL_ROLES) of integer                         ! maximum
             on working time for agency crew, possibly different for each role
        overall_regular_max_work: integer
        overall_agency_max_work: integer
        overall_max_work: integer

        ! Added for the recovery problem:
        ! - the details of the current roster...
        cur_allocate: array(ALL_EMP, ALL_ROLES, TIME) of integer
        cur_board, cur_depart: array(REG_EMP, VESSELS, TIME) of integer
        cur_ag_rboard, cur_ag_rdepart: array(ALL_ROLES, TIME) of integer
        cur_undertime, cur_overtime: array(GUARANTEED) of real

        ! ... and the 'change' variables...
        chng_allocate: array(ALL_EMP, ALL_ROLES, TIME) of mpvar
        chng_board, chng_depart: array(REG_EMP, VESSELS, TIME) of mpvar
        chng_ag_rboard, chng_ag_rdepart: array(ALL_ROLES, TIME) of mpvar
        chng_undertime, chng_overtime: array(GUARANTEED) of mpvar
end-declarations

initializations from DATAFILE
```

```
        board_chng_cost depart_chng_cost work_chng_cost
        ag_board_chng_cost ag_depart_chng_cost
        required eligable starting ag_starting
        work_zero rest_zero
        max_work min_rest ag_max_work

        cur_allocate cur_board cur_depart cur_ag_rboard cur_ag_rdepart
        cur_undertime cur_overtime
end-initializations

overall_regular_max_work := max(e in REG_EMP) max_work(e)
overall_agency_max_work := max(r in ALL_ROLES) ag_max_work(r)
if(overall_regular_max_work > overall_agency_max_work) then
        overall_max_work := overall_regular_max_work
else
        overall_max_work := overall_agency_max_work
end-if

forall(r in ALL_ROLES, t in TIME) do
        if(required(r,t) > 0) then
                forall(e in ALL_EMP | eligable(e,r,t) = 1) create(allocate(e,r,t))
        end-if
end-do



declarations
        lambda = 1..overall_max_work
                                        ! Index used for number of consecutive
            weeks
        long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of mpvar      ! Used to
             indicate if a special bonus / penalty payment relating to consecutive
             time at sea is required
        extension_chng_cost: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real !
            Cost of CHANGES TO an employee working on board a vessel for longer
            than usual

        !Added for recovery problem - detail of current roster, and change
            variable
        cur_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of integer
        chng_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of mpvar
end-declarations

initializations from DATAFILE
```

```
       extension_chng_cost cur_long_work
end-initializations


!------------------------------------------------------------------------
declarations
       Total_cost: linctr
       All_covered: dynamic array(ALL_ROLES, TIME) of linctr
       No_overlap: array(REG_EMP, TIME) of linctr
       Board_constr: array(REG_EMP, VESSELS, TIME) of linctr
       Depart_constr: array(REG_EMP, VESSELS, TIME) of linctr
       AG_board_vs_depart: array(ALL_ROLES, TIME) of linctr
       Calc_undertime: array(GUARANTEED) of linctr
       Calc_overtime: array(GUARANTEED) of linctr
       Work_count: array(REG_EMP, TIME) of linctr
       Long_work_count: dynamic array(lambda, REG_EMP, ALL_ROLES, TIME) of linctr
       AG_work_count: array(ALL_ROLES, TIME) of linctr
       AG_work_reset: array(ALL_ROLES, TIME) of linctr
       AG_long_work: dynamic array(lambda, ALL_ROLES, TIME) of linctr
       Rest_count: array(REG_EMP, TIME) of linctr
       Rest_reset: array(REG_EMP, TIME) of linctr
       Rest_vs_work: array(REG_EMP, TIME) of linctr

       ! Added for recovery problem - constraints to link change, current and new
            values:
       Update_allocate: array(ALL_EMP, ALL_ROLES, TIME) of linctr
       Update_board: array(REG_EMP, VESSELS, TIME) of linctr
       Update_depart: array(REG_EMP, VESSELS, TIME) of linctr
       Update_ag_rboard: array(ALL_ROLES, TIME) of linctr
       Update_ag_rdepart: array(ALL_ROLES, TIME) of linctr
       Update_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of linctr
       Update_undertime: array(GUARANTEED) of linctr
       Update_overtime: array(GUARANTEED) of linctr
end-declarations


Total_cost := (sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)
    *chng_board(e,v,t)) + (depart_chng_cost(e,v,t)*chng_depart(e,v,t))) +
                             sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r
                                 ,t)*chng_ag_rboard(r,t)) + (ag_depart_chng_cost(r
                                 ,t)*chng_ag_rdepart(r,t))) +
                             sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((
                                 work_chng_cost(e,r,t)*chng_allocate(e,r,t)) + sum
```

```
                                (l in lambda)((extension_chng_cost(l,e,r,t)*
                                chng_long_work(l,e,r,t)))) +
                            sum(e in GUARANTEED)((under_rate(e)*chng_undertime(e
                                ))+ (over_rate(e)*chng_overtime(e))))


forall(r in ALL_ROLES, t in TIME | required(r,t) > 0) do
       create(All_covered(r,t))
       All_covered(r,t) := sum(e in ALL_EMP)(eligable(e,r,t)*allocate(e,r,t)) =
           required(r,t)
end-do

forall(e in REG_EMP, t in TIME) No_overlap(e,t) := sum(r in ALL_ROLES) allocate(e
    ,r,t) <= 1

forall(e in REG_EMP, v in VESSELS) Board_constr(e,v,1) := board(e,v,1) >= sum(r
    in ROLES(v))(allocate(e,r,1)) - starting(e,v)
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) Board_constr(e,v,t) :=
    board(e,v,t) >= sum(r in ROLES(v))(allocate(e,r,t)) - sum(r in ROLES(v))(
    allocate(e,r,(t-1)))

forall(e in REG_EMP, v in VESSELS) Depart_constr(e,v,1) := depart(e,v,1) >=
    starting(e,v) - sum(r in ROLES(v))(allocate(e,r,1))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) Depart_constr(e,v,t) :=
     depart(e,v,t) >= sum(r in ROLES(v))(allocate(e,r,(t-1))) - sum(r in ROLES(v)
    )(allocate(e,r,t))

forall(r in ALL_ROLES) AG_board_vs_depart(r,1) := ag_rboard(r,1) - ag_rdepart(r
    ,1) = allocate("AGENCY",r,1) - ag_starting(r)
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) AG_board_vs_depart(r,t) :=
    ag_rboard(r,t) - ag_rdepart(r,t) = allocate("AGENCY",r,t) - allocate("AGENCY
    ",r,(t-1))

forall(e in GUARANTEED) Calc_undertime(e) := undertime(e) >= g_weeks(e) - (
    exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(allocate(e,r,t)))
forall(e in GUARANTEED) Calc_overtime(e) := overtime(e) >= (exp_worktime(e) + sum
    (r in ALL_ROLES, t in TIME)(allocate(e,r,t)))- g_weeks(e)

forall(e in REG_EMP) Work_count(e,1) := work_total(e,1) >= work_zero(e) + sum(r
    in ALL_ROLES)(allocate(e,r,1)) - max_work(e)*(1-(sum(r in ALL_ROLES)(allocate
    (e,r,1))))
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN) Work_count(e,t) := work_total(e,t) >=
     work_total(e,(t-1)) + sum(r in ALL_ROLES)(allocate(e,r,t)) - max_work(e)
    *(1-(sum(r in ALL_ROLES)(allocate(e,r,t))))

forall(l in lambda, e in REG_EMP, r in ALL_ROLES | exists(long_work(l,e,r,1))) do
        create(Long_work_count(l,e,r,1))
        Long_work_count(l,e,r,1) := max_work(e)*long_work(l,e,r,1) >= work_zero(e)
             - max_work(e)*(1-allocate(e,r,1)) + allocate(e,r,1) - (l-1)
end-do
forall(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN | exists(
    long_work(l,e,r,t))) do
        create(Long_work_count(l,e,r,t))
        Long_work_count(l,e,r,t) := max_work(e)*long_work(l,e,r,t) >= work_total(e
            ,(t-1)) - max_work(e)*(1-allocate(e,r,t)) + allocate(e,r,t) - (l-1)
end-do

forall(r in ALL_ROLES) AG_work_count(r,1) := ag_work_total(r,1) >= ag_work_zero(r
    ) + allocate("AGENCY",r,1) - ag_max_work(r)*ag_rdepart(r,1)
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) AG_work_count(r,t) := ag_work_total
    (r,t) >= ag_work_total(r,(t-1)) + allocate("AGENCY",r,t) - ag_max_work(r)*
    ag_rdepart(r,t)
forall(r in ALL_ROLES, t in TIME) AG_work_reset(r,t) := ag_work_total(r,t) >=
    allocate("AGENCY",r,t)

forall(l in lambda, r in ALL_ROLES, t in TIME | exists(long_work(l,"AGENCY",r,t))
    ) do
        create(AG_long_work(l,r,t))
        AG_long_work(l,r,t) := ag_max_work(r)*long_work(l,"AGENCY",r,t) >=
            ag_work_total(r,t) - (l-1)
end-do

forall(e in REG_EMP) Rest_count(e,1) := rest_total(e,1) >= rest_zero(e) - (1-(sum
    (r in ALL_ROLES)(allocate(e,r,1))))
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN) Rest_count(e,t) := rest_total(e,t) >=
     rest_total(e,(t-1)) - (1-(sum(r in ALL_ROLES)(allocate(e,r,t))))
forall(e in REG_EMP, t in TIME) Rest_reset(e,t) := rest_total(e,t) >= (min_rest(e
    )-1)*(sum(v in VESSELS)(depart(e,v,t)))

forall(e in REG_EMP) Rest_vs_work(e,1) := min_rest(e)*(1-(sum(r in ALL_ROLES)(
    allocate(e,r,1)))) >= rest_zero(e)
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN) Rest_vs_work(e,t) := min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate(e,r,t)))) >= rest_total(e,(t-1))


! Added for recovery problem - linking change to current variables:
```

```
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
       if(exists(allocate(e,r,t)) = true) then
              if(cur_allocate(e,r,t) = 0) then Update_allocate(e,r,t) :=
                  chng_allocate(e,r,t) = allocate(e,r,t)
              else Update_allocate(e,r,t) := chng_allocate(e,r,t) = cur_allocate(
                  e,r,t) - allocate(e,r,t)
              end-if
       else
              Update_allocate(e,r,t) := chng_allocate(e,r,t) = cur_allocate(e,r,t
                  )
       end-if
end-do

forall(e in REG_EMP, v in VESSELS, t in TIME) do
       if(cur_board(e,v,t) = 0) then Update_board(e,v,t) := chng_board(e,v,t) =
           board(e,v,t)
       else Update_board(e,v,t) := chng_board(e,v,t) = cur_board(e,v,t) - board(e
           ,v,t)
       end-if
end-do

forall(e in REG_EMP, v in VESSELS, t in TIME) do
       if(cur_depart(e,v,t) = 0) then Update_depart(e,v,t) := chng_depart(e,v,t)
           = depart(e,v,t)
       else Update_depart(e,v,t) := chng_depart(e,v,t) = cur_depart(e,v,t) -
           depart(e,v,t)
       end-if
end-do

forall(r in ALL_ROLES, t in TIME) do
       if(cur_ag_rboard(r,t) = 0) then Update_ag_rboard(r,t) := chng_ag_rboard(r,
           t) = ag_rboard(r,t)
       else Update_ag_rboard(r,t) := chng_ag_rboard(r,t) = cur_ag_rboard(r,t) -
           ag_rboard(r,t)
       end-if
end-do

forall(r in ALL_ROLES, t in TIME) do
       if(cur_ag_rdepart(r,t) = 0) then Update_ag_rdepart(r,t) := chng_ag_rdepart
           (r,t) = ag_rdepart(r,t)
       else Update_ag_rdepart(r,t) := chng_ag_rdepart(r,t) = cur_ag_rdepart(r,t)
           - ag_rdepart(r,t)
       end-if
```

```
end-do

forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME) do
       if(exists(long_work(l,e,r,t)) = true) then
              if(cur_long_work(l,e,r,t) = 0) then Update_long_work(l,e,r,t) :=
                  chng_long_work(l,e,r,t) = long_work(l,e,r,t)
              else Update_long_work(l,e,r,t) := chng_long_work(l,e,r,t) =
                  cur_long_work(l,e,r,t) - long_work(l,e,r,t)
              end-if
       else
              Update_long_work(l,e,r,t) := chng_long_work(l,e,r,t) =
                  cur_long_work(l,e,r,t)
       end-if
end-do

forall(e in GUARANTEED) Update_undertime(e) := chng_undertime(e) = undertime(e) -
     cur_undertime(e)
forall(e in GUARANTEED) Update_overtime(e) := chng_overtime(e) = overtime(e) -
    cur_overtime(e)


! finally, whether vessels are binary, integer, non-negative, or free:
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(allocate(e,r,t)))
    allocate(e,r,t) is_binary
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) chng_allocate(e,r,t) is_binary

forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(long_work(l,
    e,r,t)) ) long_work(l,e,r,t) is_binary
forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME) chng_long_work(l,e,r
    ,t) is_binary

forall(e in REG_EMP, v in VESSELS, t in TIME) board(e,v,t) is_binary
forall(e in REG_EMP, v in VESSELS, t in TIME) chng_board(e,v,t) is_binary
forall(e in REG_EMP, v in VESSELS, t in TIME) depart(e,v,t) is_binary
forall(e in REG_EMP, v in VESSELS, t in TIME) chng_depart(e,v,t) is_binary

forall(r in ALL_ROLES, t in TIME) ag_rboard(r,t) is_binary
forall(r in ALL_ROLES, t in TIME) chng_ag_rboard(r,t) is_binary
forall(r in ALL_ROLES, t in TIME) chng_ag_rdepart(r,t) is_binary
forall(r in ALL_ROLES, t in TIME) ag_rdepart(r,t) is_binary

forall(e in GUARANTEED) undertime(e) >= 0
forall(e in GUARANTEED) chng_undertime(e) is_free
forall(e in GUARANTEED) overtime(e) >= 0
forall(e in GUARANTEED) chng_overtime(e) is_free

forall(e in REG_EMP, t in TIME) work_total(e,t) >= 0
forall(e in REG_EMP, t in TIME) rest_total(e,t) >= 0


prog_starttime := gettime           ! get the time so that at the end, running
    time can be calculated

fopen(OUTPUTFILE, F_OUTPUT)
       setparam("XPRS_verbose",true)
       minimize(Total_cost)
fclose(F_OUTPUT)

prog_endtime := gettime
prog_runtime := prog_endtime - prog_starttime


!------------------------------------------------------------------
! Calculate some stats about the current instance:
a := 0                              ! number of agency-covered tasks in the
    initial schedule
u := 0                              ! number of un-covered tasks in the initial
    schedule
changes_to_reg := 0         ! number of changes to regular employees in the (
    best) solution
changes_to_AG := 0          ! number of changes to agency employees in the (best
    ) solution
number_of_AG := 0           ! number of times agency employees are utilised in
    the (best) solution
forall(r in ALL_ROLES, t in TIME) do
       x := 0
       forall(e in REG_EMP) do
              if(cur_allocate(e,r,t) = 1) then
                     x := x + 1
              end-if
              changes_to_reg := changes_to_reg + round(getsol(chng_allocate(e,r,t
                  )))
       end-do
       if(cur_allocate("AGENCY",r,t) = 1) then
              x := x + 1
              a := a + 1
```

```
        end-if
        changes_to_AG := changes_to_AG + round(getsol(chng_allocate("AGENCY",r,t))
            )
        number_of_AG := number_of_AG + round(getsol(allocate("AGENCY",r,t)))
        if(x = 0) then u := u + 1
        end-if
end-do


status::([XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH])["Optimum found","
    Unfinished","Infeasible","Unbounded","Failed\t"]


! Print results to the result summary file
fopen(SUMMARYFILE, F_APPEND)
        write(InstanceName)
        write("\t",prog_runtime,"\t",status(getprobstat))
        write("\t",getobjval,"\t",getparam("XPRS_bestbound"))
        write("\t\t",(sum(r in ALL_ROLES, t in TIME) required(r,t)),"\t",u,"\t",a)
        write("\t",changes_to_reg,"\t",changes_to_AG,"\t",number_of_AG)
        write("\n")
fclose(F_APPEND)

! Print solution to a solution file, as it might be useful for comparison when
    looking at heuristics
!fopen(SOLUTIONFILE, F_OUTPUT)
!       writeln("Solution for ",InstanceName)
!       writeln("\tfound using default Xpress solution settings, with maximum time
     1 hour")
!       writeln("\twritten in representation proposed for carrying out heuristics
    .")
!       writeln
!       writeln("Prob status: ",status(getprobstat))
!       writeln("Running time: ",prog_runtime)
!       writeln("Soln value: ",getobjval)
!       writeln("Best bound: ",getparam("XPRS_bestbound"))
!       writeln
!       writeln
!       forall(e in REG_EMP) do
!               write(e,":\t[ ")
!               forall(t in TIME) do
!                       working := FALSE
!                       forall(r in ALL_ROLES) do
!                               if(getsol(allocate(e,r,t)) > 0.9) then
!                                       write("’’",r,"’’")
!                                       working := TRUE
!                               end-if
!                       end-do
!                       if(working = FALSE) then
!                               avail := FALSE
!                               forall(r in ALL_ROLES) do
!                                       if(eligable(e,r,t) = 1) then
!                                               avail := TRUE
!                                       end-if
!                               end-do
!                               if(avail = FALSE) then
!                                       write("’’*unav*’’")
!                               else
!                                       write("’’*rest*’’")
!                               end-if
!                       end-if
!                       if(t < WEEKS_TO_PLAN) then
!                               write(",\t")
!                       end-if
!               end-do
!               write(" ]\n")
!       end-do
!       writeln
!       write("Agency crew: {")
!       forall(r in ALL_ROLES, t in TIME) do
!               if(getsol(allocate("AGENCY",r,t)) > 0.9) then
!                       write(" (’’",r,"’’,",t,") ")
!               end-if
!       end-do
!       write("} \n")
!
!fclose(F_OUTPUT)


end-model
```

E.2.3 Change-minimization algorithm

Here we give the code used to solve the Time-Windows problem using the Change-Minimization algorithm described in section 6.3.2. This was implemented using the FICO Xpress software.

```
model ModelName
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver

! Recovery-type problem version of the time windows formulation, intended to be
    more realistic.
! As before, this program aims to solve the problem directly from this
    formulation.

!-------------------------------------------------------------------------
! Since I have been trying to solve the TB model with the minimal number of
    changes, should also
!       investigate doing similar with the TW model
!-------------------------------------------------------------------------

parameters
       DATE = "28-11-14"
       PARAMETERFILE = "batch-input-parameters.dat"

       overall_limit = 120
       iteration_limit = 30
end-parameters

declarations
       InstanceName: string
end-declarations

initializations from PARAMETERFILE
       InstanceName
end-initializations


DATAFILE := InstanceName+"\\Time-Windows - Captains - "+InstanceName+".txt"
OUTPUTFILE := InstanceName+"\\Logfile - TW Min Change - 2mins - "+InstanceName+"
    - "+DATE+".txt"
SUMMARYFILE := "Results - TW Min Change - 2mins - "+DATE+".txt"
!SOLUTIONFILE := InstanceName+"\\Best direct soln - "+InstanceName+" - "+DATE+".
    txt"


! Forces the program to stop after given length of time
```

```
setparam("XPRS_verbose",true)
!setparam("XPRS_maxtime",-600)
!setparam("XPRS_miprelstop",0.05)
!setparam("XPRS_covercuts",50)
!setparam("XPRS_gomcuts",10)


prog_starttime := gettime           ! get the time so that at the end, running
    time can be calculated

declarations
       status: array({XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH}) of string !
           Used to indicate the solution status of the probelm

       REG_EMP: set of string       ! Regular employee names / numbers (ie not
           Agency)
       ALL_EMP: set of string       ! Lables for ALL crew (including Agency)
       GUARANTEED:   set of string ! Set of employees on guaranteed days
           contracts
       VESSELS: set of string       ! Labels / names of vessels
       WEEKS_TO_PLAN: integer       ! Length of planning horizon in weeks
       ROLES: array(VESSELS) of set of string     ! Labels for the roles which
           will require cover, divided by vessels
       ALL_ROLES: set of string     ! List of all roles

       exp_worktime, g_weeks, under_rate, over_rate: array(GUARANTEED) of real  !
            Data relating to over/undertime payments for employees
end-declarations

initializations from DATAFILE
       REG_EMP GUARANTEED VESSELS WEEKS_TO_PLAN ROLES
       under_rate over_rate g_weeks exp_worktime
end-initializations

ALL_EMP := REG_EMP + {"AGENCY"}
ALL_ROLES := {}
FORALL (v in VESSELS) ALL_ROLES += ROLES(v)

declarations
       TIME = 1..WEEKS_TO_PLAN      ! Time index

       allocate: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar ! Variable for
           allocating employee to role during given time period
```

```
        board, depart: array(REG_EMP, VESSELS, TIME) of mpvar     ! =1 if employee
             boards / departs vessel in given time period, 0 otherwise
               ! or takes a non-negative integer value for agency crew
        ag_rboard, ag_rdepart: array(ALL_ROLES, TIME) of mpvar    ! =1 if agency
            crew starts / ends working on a role in given time period, 0 otherwise
!       undertime, overtime: array(GUARANTEED) of mpvar                 !
    Variables to calculate the amount of under/overtime carried out by employee
        undertimevar, overtimevar: array(GUARANTEED) of mpvar
        undertime, overtime: array(GUARANTEED) of real
        work_total, rest_total: array(REG_EMP, TIME) of mpvar     ! Used to track
            the consecutive working time / rest period requirements of each
            employee
        ag_work_total: array(ALL_ROLES, TIME) of mpvar                  ! Used to
             track the consecutive working time of the agency employees

        board_chng_cost, depart_chng_cost: array(REG_EMP, VESSELS, TIME) of real !
             Costs of CHANGES TO employees boarding / leaving vessel
        ag_board_chng_cost, ag_depart_chng_cost: array(ALL_ROLES, TIME) of real !
            Costs of CHANGES TO agency employees boarding / leaving for a given
            role
        work_chng_cost: array(ALL_EMP, ALL_ROLES, TIME) of real
                          ! (Direct) Costs of CHANGES TO employees working a given
             role at a given time

        required: array(ALL_ROLES, TIME) of integer                     ! =1 if
            role r is required at time t, =0 otherwise
        eligable: array(ALL_EMP, ALL_ROLES, TIME) of integer ! =1 if emp i can
            carry out role r at time t, =0 otherwise
        starting: array(ALL_EMP, VESSELS) of integer              ! =1 if emp i is
             on board vessel k at time 0, =0 otherwise
               ! or takes a non-negative integer value for agency crew
        ag_starting: array(ALL_ROLES) of integer                        ! =1 if
            agency employee is in role r at time 0, =0 otherwise
        work_zero, rest_zero: array(REG_EMP) of integer           ! initial values
             of work_total and rest_total at time zero
        ag_work_zero: array(ALL_ROLES) of integer                       ! initial
             value of work total for agency crew task by task
        max_work, min_rest: array(REG_EMP) of integer             ! legal maximum
            on working time, minimum on resting time
        ag_max_work: array(ALL_ROLES) of integer                        ! maximum
             on working time for agency crew, possibly different for each role
        overall_regular_max_work: integer
        overall_agency_max_work: integer
        overall_max_work: integer

        ! Added for the recovery problem:
        ! - the details of the current roster...
        cur_allocate: array(ALL_EMP, ALL_ROLES, TIME) of integer
        cur_board, cur_depart: array(REG_EMP, VESSELS, TIME) of integer
        cur_ag_rboard, cur_ag_rdepart: array(ALL_ROLES, TIME) of integer
        cur_undertime, cur_overtime: array(GUARANTEED) of real

        ! ... and the 'change' variables...
        chng_allocate: array(ALL_EMP, ALL_ROLES, TIME) of mpvar
        chng_board, chng_depart: array(REG_EMP, VESSELS, TIME) of mpvar
        chng_ag_rboard, chng_ag_rdepart: array(ALL_ROLES, TIME) of mpvar
        chng_undertime, chng_overtime: array(GUARANTEED) of mpvar

        best_change: array(ALL_EMP, ALL_ROLES, TIME) of integer
        best_chng_board, best_chng_depart: array(REG_EMP, VESSELS, TIME) of
            integer
        best_chng_ag_rboard, best_chng_ag_rdepart: array(ALL_ROLES, TIME) of
            integer
        best_new_sched: array(ALL_EMP, ALL_ROLES, TIME) of integer
        best_chng_undertime, best_chng_overtime: array(GUARANTEED) of real
end-declarations

initializations from DATAFILE
        board_chng_cost depart_chng_cost work_chng_cost
        ag_board_chng_cost ag_depart_chng_cost
        required eligable starting ag_starting
        work_zero rest_zero
        max_work min_rest ag_max_work

        cur_allocate cur_board cur_depart cur_ag_rboard cur_ag_rdepart
        cur_undertime cur_overtime
end-initializations

overall_regular_max_work := max(e in REG_EMP) max_work(e)
overall_agency_max_work := max(r in ALL_ROLES) ag_max_work(r)
if(overall_regular_max_work > overall_agency_max_work) then
        overall_max_work := overall_regular_max_work
else
        overall_max_work := overall_agency_max_work
end-if
```

```
forall(r in ALL_ROLES, t in TIME) do
       if(required(r,t) > 0) then
              forall(e in ALL_EMP | eligable(e,r,t) = 1) create(allocate(e,r,t))
       end-if
end-do


declarations
       lambda = 1..overall_max_work
                                  ! Index used for number of consecutive
           weeks
       long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of mpvar      ! Used to
            indicate if a special bonus / penalty payment relating to consecutive
            time at sea is required
       extension_chng_cost: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real !
           Cost of CHANGES TO an employee working on board a vessel for longer
           than usual

       !Added for recovery problem - detail of current roster, and change
           variable
       cur_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of integer
       chng_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of mpvar
       best_chng_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of integer
end-declarations

initializations from DATAFILE
       extension_chng_cost cur_long_work
end-initializations


!-----------------------------------------------------------------------
declarations
       TOTAL_COST: real
       COST_FIRST: real
       PREV_COST: real
       COST_LIMIT: real
       cost_constr: linctr

       NO_CHANGES_BEST: real
       NO_CHANGES_FIRST: real
       CHANGE_LIMIT: real
```

```
        iteration: integer
        terminate: boolean
        percentage: real
        integral: boolean

        solution: boolean
        any_infeas: boolean
        lower_bound: real
        new_limit_ok: boolean

        Big_M: real
        iter_time_limit: integer
!       UTreq, OTreq: array(GUARANTEED) of mpvar
        OTind: array(GUARANTEED) of mpvar
        undertime_second_constr, overtime_second_constr: array(GUARANTEED) of
            linctr
        undertime_third_constr, overtime_third_constr: array(GUARANTEED) of linctr


        !Total_cost: linctr
        NO_CHANGES: linctr
        All_covered: dynamic array(ALL_ROLES, TIME) of linctr
        No_overlap: array(REG_EMP, TIME) of linctr
        Board_constr: array(REG_EMP, VESSELS, TIME) of linctr
        Depart_constr: array(REG_EMP, VESSELS, TIME) of linctr
        AG_board_vs_depart: array(ALL_ROLES, TIME) of linctr
        Calc_undertime: array(GUARANTEED) of linctr
        Calc_overtime: array(GUARANTEED) of linctr
        Work_count: array(REG_EMP, TIME) of linctr
        Long_work_count: dynamic array(lambda, REG_EMP, ALL_ROLES, TIME) of linctr
        AG_work_count: array(ALL_ROLES, TIME) of linctr
        AG_work_reset: array(ALL_ROLES, TIME) of linctr
        AG_long_work: dynamic array(lambda, ALL_ROLES, TIME) of linctr
        Rest_count: array(REG_EMP, TIME) of linctr
        Rest_reset: array(REG_EMP, TIME) of linctr
        Rest_vs_work: array(REG_EMP, TIME) of linctr

        ! Added for recovery problem - constraints to link change, current and new
             values:
        Update_allocate: array(ALL_EMP, ALL_ROLES, TIME) of linctr
        Update_board: array(REG_EMP, VESSELS, TIME) of linctr
        Update_depart: array(REG_EMP, VESSELS, TIME) of linctr
        Update_ag_rboard: array(ALL_ROLES, TIME) of linctr
```

```
       Update_ag_rdepart: array(ALL_ROLES, TIME) of linctr
       Update_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of linctr
       Update_undertime: array(GUARANTEED) of linctr
       Update_overtime: array(GUARANTEED) of linctr
end-declarations

Big_M := 366
iter_time_limit := iteration_limit
lower_bound := 0
any_infeas := false

status::([XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH])["Optimum found","
    Unfinished","Infeasible","Unbounded","Failed"]


!Total_cost := (sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t
    )*chng_board(e,v,t)) + (depart_chng_cost(e,v,t)*chng_depart(e,v,t))) +
!                           sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r
    ,t)*chng_ag_rboard(r,t)) + (ag_depart_chng_cost(r,t)*chng_ag_rdepart(r,t))) +
!                           sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((
    work_chng_cost(e,r,t)*chng_allocate(e,r,t)) + sum(l in lambda)((
    extension_chng_cost(l,e,r,t)*chng_long_work(l,e,r,t)))) +
!                           sum(e in GUARANTEED)((under_rate(e)*chng_undertime(e
    ))+ (over_rate(e)*chng_overtime(e))))

NO_CHANGES := sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(chng_allocate(e,r,t))


forall(r in ALL_ROLES, t in TIME | required(r,t) > 0) do
       create(All_covered(r,t))
       All_covered(r,t) := sum(e in ALL_EMP)(eligable(e,r,t)*allocate(e,r,t)) =
           required(r,t)
end-do

forall(e in REG_EMP, t in TIME) No_overlap(e,t) := sum(r in ALL_ROLES) allocate(e
    ,r,t) <= 1

forall(e in REG_EMP, v in VESSELS) Board_constr(e,v,1) := board(e,v,1) >= sum(r
    in ROLES(v))(allocate(e,r,1)) - starting(e,v)
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) Board_constr(e,v,t) :=
    board(e,v,t) >= sum(r in ROLES(v))(allocate(e,r,t)) - sum(r in ROLES(v))(
    allocate(e,r,(t-1)))
```

```
forall(e in REG_EMP, v in VESSELS) Depart_constr(e,v,1) := depart(e,v,1) >=
    starting(e,v) - sum(r in ROLES(v))(allocate(e,r,1))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) Depart_constr(e,v,t) :=
     depart(e,v,t) >= sum(r in ROLES(v))(allocate(e,r,(t-1))) - sum(r in ROLES(v)
    )(allocate(e,r,t))

forall(r in ALL_ROLES) AG_board_vs_depart(r,1) := ag_rboard(r,1) - ag_rdepart(r
    ,1) = allocate("AGENCY",r,1) - ag_starting(r)
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) AG_board_vs_depart(r,t) :=
    ag_rboard(r,t) - ag_rdepart(r,t) = allocate("AGENCY",r,t) - allocate("AGENCY
    ",r,(t-1))

forall(e in GUARANTEED) do
       Calc_undertime(e) := undertimevar(e) >= g_weeks(e) - (exp_worktime(e) +
           sum(r in ALL_ROLES, t in TIME)(allocate(e,r,t)))
       undertime_second_constr(e) := undertimevar(e) <= g_weeks(e) - (
           exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(allocate(e,r,t))) +
           Big_M*(1-OTind(e))
       undertime_third_constr(e) := undertimevar(e) <= Big_M*OTind(e)

       Calc_overtime(e) := overtimevar(e) >= (exp_worktime(e) + sum(r in
           ALL_ROLES, t in TIME)(allocate(e,r,t)))- g_weeks(e)
       overtime_second_constr(e) := overtimevar(e) <= (exp_worktime(e) + sum(r in
            ALL_ROLES, t in TIME)(allocate(e,r,t)))- g_weeks(e) + Big_M*OTind(e)
       overtime_third_constr(e) := - Big_M*(1-OTind(e))

       OTind(e) is_binary
end-do

forall(e in REG_EMP) Work_count(e,1) := work_total(e,1) >= work_zero(e) + sum(r
    in ALL_ROLES)(allocate(e,r,1)) - max_work(e)*(1-(sum(r in ALL_ROLES)(allocate
    (e,r,1))))
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN) Work_count(e,t) := work_total(e,t) >=
     work_total(e,(t-1)) + sum(r in ALL_ROLES)(allocate(e,r,t)) - max_work(e)
    *(1-(sum(r in ALL_ROLES)(allocate(e,r,t))))

forall(l in lambda, e in REG_EMP, r in ALL_ROLES | exists(long_work(l,e,r,1))) do
       create(Long_work_count(l,e,r,1))
       Long_work_count(l,e,r,1) := max_work(e)*long_work(l,e,r,1) >= work_zero(e)
            - max_work(e)*(1-allocate(e,r,1)) + allocate(e,r,1) - (l-1)
end-do
forall(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN | exists(
    long_work(l,e,r,t))) do
```

```
        create(Long_work_count(l,e,r,t))
        Long_work_count(l,e,r,t) := max_work(e)*long_work(l,e,r,t) >= work_total(e
            ,(t-1)) - max_work(e)*(1-allocate(e,r,t)) + allocate(e,r,t) - (l-1)
end-do

forall(r in ALL_ROLES) AG_work_count(r,1) := ag_work_total(r,1) >= ag_work_zero(r
    ) + allocate("AGENCY",r,1) - ag_max_work(r)*ag_rdepart(r,1)
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) AG_work_count(r,t) := ag_work_total
    (r,t) >= ag_work_total(r,(t-1)) + allocate("AGENCY",r,t) - ag_max_work(r)*
    ag_rdepart(r,t)
forall(r in ALL_ROLES, t in TIME) AG_work_reset(r,t) := ag_work_total(r,t) >=
    allocate("AGENCY",r,t)

forall(l in lambda, r in ALL_ROLES, t in TIME | exists(long_work(l,"AGENCY",r,t))
    ) do
        create(AG_long_work(l,r,t))
        AG_long_work(l,r,t) := ag_max_work(r)*long_work(l,"AGENCY",r,t) >=
            ag_work_total(r,t) - (l-1)
end-do

forall(e in REG_EMP) Rest_count(e,1) := rest_total(e,1) >= rest_zero(e) - (1-(sum
    (r in ALL_ROLES)(allocate(e,r,1))))
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN) Rest_count(e,t) := rest_total(e,t) >=
     rest_total(e,(t-1)) - (1-(sum(r in ALL_ROLES)(allocate(e,r,t))))
forall(e in REG_EMP, t in TIME) Rest_reset(e,t) := rest_total(e,t) >= (min_rest(e
    )-1)*(sum(v in VESSELS)(depart(e,v,t)))

forall(e in REG_EMP) Rest_vs_work(e,1) := min_rest(e)*(1-(sum(r in ALL_ROLES)(
    allocate(e,r,1)))) >= rest_zero(e)
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN) Rest_vs_work(e,t) := min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate(e,r,t)))) >= rest_total(e,(t-1))


! Added for recovery problem - linking change to current variables:
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
        if(exists(allocate(e,r,t)) = true) then
                if(cur_allocate(e,r,t) = 0) then Update_allocate(e,r,t) :=
                    chng_allocate(e,r,t) = allocate(e,r,t)
                else Update_allocate(e,r,t) := chng_allocate(e,r,t) = cur_allocate(
                    e,r,t) - allocate(e,r,t)
                end-if
        else
```

```
               Update_allocate(e,r,t) := chng_allocate(e,r,t) = cur_allocate(e,r,t
                   )
        end-if
end-do

forall(e in REG_EMP, v in VESSELS, t in TIME) do
        if(cur_board(e,v,t) = 0) then Update_board(e,v,t) := chng_board(e,v,t) =
            board(e,v,t)
        else Update_board(e,v,t) := chng_board(e,v,t) = cur_board(e,v,t) - board(e
            ,v,t)
        end-if
end-do

forall(e in REG_EMP, v in VESSELS, t in TIME) do
        if(cur_depart(e,v,t) = 0) then Update_depart(e,v,t) := chng_depart(e,v,t)
            = depart(e,v,t)
        else Update_depart(e,v,t) := chng_depart(e,v,t) = cur_depart(e,v,t) -
            depart(e,v,t)
        end-if
end-do

forall(r in ALL_ROLES, t in TIME) do
        if(cur_ag_rboard(r,t) = 0) then Update_ag_rboard(r,t) := chng_ag_rboard(r,
            t) = ag_rboard(r,t)
        else Update_ag_rboard(r,t) := chng_ag_rboard(r,t) = cur_ag_rboard(r,t) -
            ag_rboard(r,t)
        end-if
end-do

forall(r in ALL_ROLES, t in TIME) do
        if(cur_ag_rdepart(r,t) = 0) then Update_ag_rdepart(r,t) := chng_ag_rdepart
            (r,t) = ag_rdepart(r,t)
        else Update_ag_rdepart(r,t) := chng_ag_rdepart(r,t) = cur_ag_rdepart(r,t)
            - ag_rdepart(r,t)
        end-if
end-do

forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME) do
        if(exists(long_work(l,e,r,t)) = true) then
               if(cur_long_work(l,e,r,t) = 0) then Update_long_work(l,e,r,t) :=
                   chng_long_work(l,e,r,t) = long_work(l,e,r,t)
               else Update_long_work(l,e,r,t) := chng_long_work(l,e,r,t) =
                   cur_long_work(l,e,r,t) - long_work(l,e,r,t)
```

```
              end-if
       else
              Update_long_work(l,e,r,t) := chng_long_work(l,e,r,t) =
                  cur_long_work(l,e,r,t)
       end-if
end-do

forall(e in GUARANTEED) Update_undertime(e) := chng_undertime(e) = undertimevar(e
    ) - cur_undertime(e)
forall(e in GUARANTEED) Update_overtime(e) := chng_overtime(e) = overtimevar(e) -
     cur_overtime(e)



! finally, whether vessels are binary, integer, non-negative, or free:
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(allocate(e,r,t)))
    allocate(e,r,t) is_binary
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) chng_allocate(e,r,t) is_binary

forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(long_work(l,
    e,r,t)) ) long_work(l,e,r,t) is_binary
forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME) chng_long_work(l,e,r
    ,t) is_binary

forall(e in REG_EMP, v in VESSELS, t in TIME) board(e,v,t) is_binary
forall(e in REG_EMP, v in VESSELS, t in TIME) chng_board(e,v,t) is_binary
forall(e in REG_EMP, v in VESSELS, t in TIME) depart(e,v,t) is_binary
forall(e in REG_EMP, v in VESSELS, t in TIME) chng_depart(e,v,t) is_binary

forall(r in ALL_ROLES, t in TIME) ag_rboard(r,t) is_binary
forall(r in ALL_ROLES, t in TIME) chng_ag_rboard(r,t) is_binary
forall(r in ALL_ROLES, t in TIME) chng_ag_rdepart(r,t) is_binary
forall(r in ALL_ROLES, t in TIME) ag_rdepart(r,t) is_binary

forall(e in GUARANTEED) undertimevar(e) >= 0
forall(e in GUARANTEED) chng_undertime(e) is_free
forall(e in GUARANTEED) overtimevar(e) >= 0
forall(e in GUARANTEED) chng_overtime(e) is_free

forall(e in REG_EMP, t in TIME) work_total(e,t) >= 0
forall(e in REG_EMP, t in TIME) rest_total(e,t) >= 0


!-------------------------------------------------------------------------
```

```
!writeln
!writeln("Original schedule has roles carried out as follows:")
!forall(t in TIME) write("\tWeek ",t)
!write("\n")
!forall(r in ALL_ROLES) do
!       write(r)
!       forall(t in TIME) do
!               if(required(r,t) = 0) then
!                       write("\t (n/a)")
!               else
!                       covered := FALSE
!                       forall(e in ALL_EMP) do
!                               if(cur_allocate(e,r,t) = 1) then
!                                       write("\t",e)
!                                       covered := TRUE
!                               end-if
!                       end-do
!                       if(covered = FALSE) then write("\t*TBC*"); end-if
!               end-if
!       end-do
!       write("\n")
!end-do
!writeln
!writeln
!
!writeln("... and crew assigned to roles as follows:")
!forall(t in TIME) write("\tWeek ",t)
!write("\n")
!forall(e in REG_EMP) do
!       write(e)
!       forall(t in TIME) do
!               working := FALSE
!               forall(r in ALL_ROLES) do
!                       if(cur_allocate(e,r,t) = 1) then
!                               write("\t",r)
!                               working := TRUE
!                       end-if
!               end-do
!               if(working = FALSE) then
!                       avail := FALSE
!                       forall(r in ALL_ROLES) do
!                               if(eligable(e,r,t) = 1) then avail := TRUE
!                               end-if
!                       end-do
!                       if(avail = FALSE) then write("\t*unav*")
!                       else write("\t")
!                       end-if
!               end-if
!       end-do
!       write("\n")
!end-do
!writeln
!writeln


!----------------------------------------------------------------------
!----------------------------------------------------------------------



fopen(OUTPUTFILE, F_OUTPUT)
       writeln
       writeln("--------------------------------------------")
       writeln
       setparam("XPRS_maxtime", -iter_time_limit)
       minimize(NO_CHANGES)
       writeln
       writeln("--------------------------------------------")
       writeln
!fclose(F_OUTPUT)

integral := true
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
       if(isintegral(chng_allocate(e,r,t)) = false) then integral := false; end-
           if
       if(isintegral(allocate(e,r,t)) = false) then integral := false; end-if
end-do

writeln(status(getprobstat))
if(integral = false) then writeln("No integer solution found"); end-if
writeln

solution := false
if(getprobstat = XPRS_OPT OR (getprobstat = XPRS_UNF and integral = true)) then

       solution := true
```

```
TOTAL_COST := (sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost
    (e,v,t)*getsol(chng_board(e,v,t))) + (depart_chng_cost(e,v,t)*getsol(
    chng_depart(e,v,t)))) +
                      sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r
                          ,t)*getsol(chng_ag_rboard(r,t))) + (
                          ag_depart_chng_cost(r,t)*getsol(chng_ag_rdepart(r
                          ,t)))) +
                      sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((
                          work_chng_cost(e,r,t)*getsol(chng_allocate(e,r,t)
                          )) + sum(l in lambda)((extension_chng_cost(l,e,r,
                          t)*getsol(chng_long_work(l,e,r,t))))) +
                      sum(e in GUARANTEED)((under_rate(e)*getsol(
                          chng_undertime(e)))+ (over_rate(e)*getsol(
                          chng_overtime(e)))))

COST_FIRST := TOTAL_COST
NO_CHANGES_FIRST := getobjval

NO_CHANGES_BEST := getobjval
CHANGE_LIMIT := 2*getobjval

forall(t in TIME) do
        forall(e in ALL_EMP) do
                forall(r in ALL_ROLES) do
                        if(getsol(chng_allocate(e,r,t)) > 0.7) then
                            best_change(e,r,t) := 1
                        else best_change(e,r,t) := 0; end-if
                        if(getsol(allocate(e,r,t)) > 0.7) then
                            best_new_sched(e,r,t) := 1
                        else best_new_sched(e,r,t) := 0; end-if

                        forall(l in lambda) do
                                if(getsol(chng_long_work(l,e,r,t)) > 0.7)
                                    then best_chng_long_work(l,e,r,t) := 1
                                else best_chng_long_work(l,e,r,t) := 0; end-
                                    if
                        end-do

                        if(e not in REG_EMP) then
                                if(getsol(chng_ag_rboard(r,t)) > 0.7) then
                                    best_chng_ag_rboard(r,t) := 1
                                else best_chng_ag_rboard(r,t) := 0; end-if
```

```
                              if(getsol(chng_ag_rdepart(r,t)) > 0.7) then
                                 best_chng_ag_rdepart(r,t) := 1
                              else best_chng_ag_rdepart(r,t) := 0; end-if
                       end-if
                end-do
                if(e in REG_EMP) then
                       forall(v in VESSELS) do
                              if(getsol(chng_board(e,v,t)) > 0.7) then
                                 best_chng_board(e,v,t) := 1
                              else best_chng_board(e,v,t) := 0; end-if
                              if(getsol(chng_depart(e,v,t)) > 0.7) then
                                 best_chng_depart(e,v,t) := 1
                              else best_chng_depart(e,v,t) := 0; end-if
                       end-do
                end-if
         end-do
  end-do
end-if


current_time := gettime
iteration := 1
writeln
writeln("Iterations:  ",iteration)
writeln("Total run time: ",current_time - prog_starttime)
writeln

terminate := false
percentage := 0.81
while(terminate = false) do
       if(getprobstat = XPRS_OPT OR (getprobstat = XPRS_UNF and integral = true))
            then
              writeln("\tThe total number of changes is: ",getobjval)
              writeln("\tand the total cost of this solution is ",TOTAL_COST)
              writeln

              PREV_COST := TOTAL_COST
              new_limit_ok := false

              while(new_limit_ok = false) do
                     if(PREV_COST > 0) then
                            COST_LIMIT := (1-percentage)*PREV_COST
                     elif(PREV_COST < 0) then
                COST_LIMIT := (1+percentage)*PREV_COST
        else
                COST_LIMIT := -10
        end-if
        if(any_infeas = true and COST_LIMIT < lower_bound) then
            percentage := 2*(percentage/3)
        else new_limit_ok := true
        end-if
end-do

writeln("Set cost limit to ",COST_LIMIT)
cost_constr := (sum(e in REG_EMP, v in VESSELS, t in TIME)((
    board_chng_cost(e,v,t)*chng_board(e,v,t)) + (depart_chng_cost(e
    ,v,t)*chng_depart(e,v,t))) +
                sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r
                    ,t)*chng_ag_rboard(r,t)) + (ag_depart_chng_cost(r
                    ,t)*chng_ag_rdepart(r,t))) +
                sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((
                    work_chng_cost(e,r,t)*chng_allocate(e,r,t)) + sum
                    (l in lambda)((extension_chng_cost(l,e,r,t)*
                    chng_long_work(l,e,r,t)))) +
                sum(e in GUARANTEED)((under_rate(e)*chng_undertime(e
                    ))+ (over_rate(e)*chng_overtime(e)))) <=
                    COST_LIMIT
writeln("... and re-solve")
writeln

writeln("----------------------------------------------")
writeln
setparam("XPRS_maxtime", -iter_time_limit)
minimize(NO_CHANGES)
writeln
writeln("----------------------------------------------")
writeln

integral := true
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
        if(isintegral(chng_allocate(e,r,t)) = false) then integral
            := false; end-if
        if(isintegral(allocate(e,r,t)) = false) then integral :=
            false; end-if
end-do
writeln(status(getprobstat))
```

```
if(integral = false) then writeln("No integer solution found"); end
    -if

if(getprobstat = XPRS_INF) then
        any_infeas := true
        lower_bound := COST_LIMIT
        writeln("New lower bound on solution value = ",lower_bound)
end-if

writeln
if(getprobstat = XPRS_OPT OR (getprobstat = XPRS_UNF and integral =
     true)) then
        TOTAL_COST := (sum(e in REG_EMP, v in VESSELS, t in TIME)((
            board_chng_cost(e,v,t)*getsol(chng_board(e,v,t))) + (
            depart_chng_cost(e,v,t)*getsol(chng_depart(e,v,t)))) +
                                sum(r in ALL_ROLES, t in TIME)((
                                    ag_board_chng_cost(r,t)*getsol(
                                    chng_ag_rboard(r,t))) + (
                                    ag_depart_chng_cost(r,t)*getsol(
                                    chng_ag_rdepart(r,t)))) +
                                sum(e in ALL_EMP, r in ALL_ROLES, t in
                                     TIME)((work_chng_cost(e,r,t)*
                                    getsol(chng_allocate(e,r,t))) +
                                    sum(l in lambda)((
                                    extension_chng_cost(l,e,r,t)*
                                    getsol(chng_long_work(l,e,r,t)))))
                                     +
                                sum(e in GUARANTEED)((under_rate(e)*
                                    getsol(chng_undertime(e)))+ (
                                    over_rate(e)*getsol(chng_overtime(
                                    e)))))

        NO_CHANGES_BEST := getobjval

        forall(t in TIME) do
                forall(e in ALL_EMP) do
                        forall(r in ALL_ROLES) do
                                if(getsol(chng_allocate(e,r,t)) > 0.7)
                                     then best_change(e,r,t) := 1
                                else best_change(e,r,t) := 0; end-if
                                if(getsol(allocate(e,r,t)) > 0.7) then
                                     best_new_sched(e,r,t) := 1
```

```
                    else best_new_sched(e,r,t) := 0; end-
                        if

                    forall(l in lambda) do
                            if(getsol(chng_long_work(l,e,r,
                                t)) > 0.7) then
                                best_chng_long_work(l,e,r,t
                                ) := 1
                            else best_chng_long_work(l,e,r,
                                t) := 0; end-if
                    end-do

                    if(e not in REG_EMP) then
                            if(getsol(chng_ag_rboard(r,t))
                                > 0.7) then
                                best_chng_ag_rboard(r,t) :=
                                 1
                            else best_chng_ag_rboard(r,t)
                                := 0; end-if
                            if(getsol(chng_ag_rdepart(r,t))
                                 > 0.7) then
                                best_chng_ag_rdepart(r,t)
                                := 1
                            else best_chng_ag_rdepart(r,t)
                                := 0; end-if
                    end-if
            end-do
            if(e in REG_EMP) then
                    forall(v in VESSELS) do
                            if(getsol(chng_board(e,v,t)) >
                                0.7) then best_chng_board(e
                                ,v,t) := 1
                            else best_chng_board(e,v,t) :=
                                0; end-if
                            if(getsol(chng_depart(e,v,t)) >
                                 0.7) then best_chng_depart
                                (e,v,t) := 1
                            else best_chng_depart(e,v,t) :=
                                 0; end-if
                    end-do
            end-if
        end-do
end-do
```

```
        end-if

else
        if(iteration = 1 or percentage < 0.06) then
                terminate := true
        else
                writeln
                writeln("No solution found")

                new_limit_ok := false
                percentage := 2*(percentage/3)

                while(new_limit_ok = false) do
                        if(PREV_COST > 0) then
                                COST_LIMIT := (1-percentage)*PREV_COST
                        elif(PREV_COST < 0) then
                                COST_LIMIT := (1+percentage)*PREV_COST
                        else
                                COST_LIMIT := -1
                        end-if
                        if(any_infeas = true and COST_LIMIT < lower_bound)
                            then percentage := 2*(percentage/3)
                        else new_limit_ok := true
                        end-if
                end-do

                writeln("Set cost limit to ",COST_LIMIT)
                cost_constr := (sum(e in REG_EMP, v in VESSELS, t in TIME)((
                    board_chng_cost(e,v,t)*chng_board(e,v,t)) + (
                    depart_chng_cost(e,v,t)*chng_depart(e,v,t))) +
                                sum(r in ALL_ROLES, t in TIME)((
                                    ag_board_chng_cost(r,t)*chng_ag_rboard(r,
                                    t)) + (ag_depart_chng_cost(r,t)*
                                    chng_ag_rdepart(r,t))) +
                                sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)
                                    ((work_chng_cost(e,r,t)*chng_allocate(e,r
                                    ,t)) + sum(l in lambda)((
                                    extension_chng_cost(l,e,r,t)*
                                    chng_long_work(l,e,r,t)))) +
                                sum(e in GUARANTEED)((under_rate(e)*
                                    chng_undertime(e))+ (over_rate(e)*
                                    chng_overtime(e)))) <= COST_LIMIT
                writeln("... and re-solve")
```

```
writeln

writeln("--------------------------------------------")
writeln
setparam("XPRS_maxtime", -iter_time_limit)
minimize(NO_CHANGES)
writeln
writeln("--------------------------------------------")

integral := true
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
       if(isintegral(chng_allocate(e,r,t)) = false) then
           integral := false; end-if
       if(isintegral(allocate(e,r,t)) = false) then
           integral := false; end-if
end-do
writeln(status(getprobstat))
if(integral = false) then writeln("No integer solution found
    "); end-if

if(getprobstat = XPRS_INF) then
       any_infeas := true
       lower_bound := COST_LIMIT
       writeln("New lower bound on solution value = ",
           lower_bound)
end-if

writeln
if(getprobstat = XPRS_OPT OR (getprobstat = XPRS_UNF and
    integral = true)) then
       TOTAL_COST := (sum(e in REG_EMP, v in VESSELS, t in
           TIME)((board_chng_cost(e,v,t)*getsol(chng_board(e
           ,v,t))) + (depart_chng_cost(e,v,t)*getsol(
           chng_depart(e,v,t)))) +
                             sum(r in ALL_ROLES, t in TIME)
                                 ((ag_board_chng_cost(r,t)*
                                 getsol(chng_ag_rboard(r,t))
                                 ) + (ag_depart_chng_cost(r,
                                 t)*getsol(chng_ag_rdepart(r
                                 ,t)))) +
                             sum(e in ALL_EMP, r in
                                 ALL_ROLES, t in TIME)((
                                 work_chng_cost(e,r,t)*
```

```
                                getsol(chng_allocate(e,r,t)
                                )) + sum(l in lambda)((
                                extension_chng_cost(l,e,r,t
                                )*getsol(chng_long_work(l,e
                                ,r,t))))) +
                            sum(e in GUARANTEED)((
                                under_rate(e)*getsol(
                                chng_undertime(e)))+ (
                                over_rate(e)*getsol(
                                chng_overtime(e)))))

NO_CHANGES_BEST := getobjval

forall(t in TIME) do
        forall(e in ALL_EMP) do
                forall(r in ALL_ROLES) do
                        if(getsol(chng_allocate(e,r,t))
                             > 0.7) then best_change(e,
                            r,t) := 1
                        else best_change(e,r,t) := 0;
                            end-if
                        if(getsol(allocate(e,r,t)) >
                            0.7) then best_new_sched(e,
                            r,t) := 1
                        else best_new_sched(e,r,t) :=
                            0; end-if

                        forall(l in lambda) do
                                if(getsol(chng_long_work
                                    (l,e,r,t)) > 0.7)
                                    then
                                    best_chng_long_work(
                                    l,e,r,t) := 1
                                else best_chng_long_work
                                    (l,e,r,t) := 0; end-
                                    if
                        end-do

                        if(e not in REG_EMP) then
                                if(getsol(chng_ag_rboard
                                    (r,t)) > 0.7) then
                                    best_chng_ag_rboard(
                                    r,t) := 1
```

```
                                                   else best_chng_ag_rboard
                                                       (r,t) := 0; end-if
                                                   if(getsol(
                                                       chng_ag_rdepart(r,t)
                                                       ) > 0.7) then
                                                       best_chng_ag_rdepart
                                                       (r,t) := 1
                                                   else
                                                       best_chng_ag_rdepart
                                                       (r,t) := 0; end-if
                                           end-if
                                    end-do
                                    if(e in REG_EMP) then
                                           forall(v in VESSELS) do
                                                   if(getsol(chng_board(e,v
                                                       ,t)) > 0.7) then
                                                       best_chng_board(e,v,
                                                       t) := 1
                                                   else best_chng_board(e,v
                                                       ,t) := 0; end-if
                                                   if(getsol(chng_depart(e,
                                                       v,t)) > 0.7) then
                                                       best_chng_depart(e,v
                                                       ,t) := 1
                                                   else best_chng_depart(e,
                                                       v,t) := 0; end-if
                                           end-do
                                    end-if
                             end-do
                      end-do
              end-if

       end-if
end-if

iteration := iteration + 1

current_time := gettime
writeln
writeln("Iterations:   ",iteration)
writeln("Total run time: ",current_time - prog_starttime)
writeln
if(current_time - prog_starttime >= overall_limit) then
```

```
                terminate := true
                writeln("Running time > ",overall_limit," seconds --> terminate")
                writeln
        elif(current_time - prog_starttime > (overall_limit - iteration_limit))
            then
                iter_time_limit := floor(overall_limit - (current_time -
                    prog_starttime))+1
                writeln("Time limit approaching - set next iteration time limit to
                    ",iter_time_limit," seconds.")
                writeln
        end-if

end-do


!------------------------------------------------------------------------
!------------------------------------------------------------------------

forall(e in GUARANTEED) do
        undertime(e) := g_weeks(e) - (exp_worktime(e) + sum(r in ALL_ROLES, t in
            TIME)(best_new_sched(e,r,t)))
        overtime(e) := (exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(
            best_new_sched(e,r,t)))- g_weeks(e)
        if(undertime(e) < 0) then undertime(e) := 0; end-if
        if(overtime(e) < 0) then overtime(e) := 0; end-if

        best_chng_undertime(e) := undertime(e) - cur_undertime(e)
        best_chng_overtime(e) := overtime(e) - cur_overtime(e)
end-do




TOTAL_COST := (sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)
    *best_chng_board(e,v,t)) + (depart_chng_cost(e,v,t)*best_chng_depart(e,v,t)))
     +
                       sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*
                           best_chng_ag_rboard(r,t)) + (ag_depart_chng_cost(r,t)*
                           best_chng_ag_rdepart(r,t))) +
                       sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((work_chng_cost
                           (e,r,t)*best_change(e,r,t)) + sum(l in lambda)((
                           extension_chng_cost(l,e,r,t)*best_chng_long_work(l,e,r,t
                           )))) +
                       sum(e in GUARANTEED)((under_rate(e)*best_chng_undertime(e))+
                            (over_rate(e)*best_chng_overtime(e))))

if(solution = true) then
        writeln("Final solution contains ",NO_CHANGES_BEST," changes")
        writeln
        writeln("Total cost is: ",TOTAL_COST,", comprising:")
        writeln("\tWrk: ",sum(e in REG_EMP, r in ALL_ROLES, t in TIME)(
            work_chng_cost(e,r,t)*best_change(e,r,t)))
        writeln("\tLWk: ",sum(e in REG_EMP, r in ALL_ROLES, t in TIME, l in lambda
            )(extension_chng_cost(l,e,r,t)*best_chng_long_work(l,e,r,t)))
        writeln("\tAgW: ",sum(r in ALL_ROLES, t in TIME)(work_chng_cost("AGENCY",r
            ,t)*best_change("AGENCY",r,t)))
        writeln("\tAgL: ",sum(r in ALL_ROLES, t in TIME, l in lambda)(
            extension_chng_cost(l,"AGENCY",r,t)*best_chng_long_work(l,"AGENCY",r,t
            )))
        writeln("\tBrd: ",sum(e in REG_EMP, v in VESSELS, t in TIME)(
            board_chng_cost(e,v,t)*best_chng_board(e,v,t)))
        writeln("\tDpt: ",sum(e in REG_EMP, v in VESSELS, t in TIME)(
            depart_chng_cost(e,v,t)*best_chng_depart(e,v,t)))
        writeln("\tAgB: ",sum(r in ALL_ROLES, t in TIME)(ag_board_chng_cost(r,t)*
            best_chng_ag_rboard(r,t)))
        writeln("\tAgD: ",sum(r in ALL_ROLES, t in TIME)(ag_depart_chng_cost(r,t)*
            best_chng_ag_rdepart(r,t)))
        writeln("\tUTm: ",sum(e in GUARANTEED)(under_rate(e)*best_chng_undertime(e
            )))
        writeln("\tOTm: ",sum(e in GUARANTEED)(over_rate(e)*best_chng_overtime(e))
            )
else
        writeln("NO FEASIBLE SOLUTION WAS FOUND")
end-if


!writeln
!writeln("Solution is to make the following changes:")
!
!
!writeln
!forall(e in REG_EMP) do
!      changes := FALSE
!      writeln(e,":")
!      forall(t in TIME) do
!             forall(v in VESSELS) do
!                       if(best_chng_board(e,v,t) > 0.9) then
!                               changes := TRUE
!                               if(cur_board(e,v,t) = 1) then writeln(" - no longer
   boards ",v," in advance of week ",t)
!                               else writeln(" - will now board ",v," in advance of
   week ",t)
!                               end-if
!                       end-if
!                       if(best_chng_depart(e,v,t) > 0.9) then
!                               changes := TRUE
!                               if(cur_depart(e,v,t) = 1) then writeln(" - no longer
    leaves ",v," in advance of week ",t," (i.e. following week ",t-1,")")
!                               else writeln(" - will now leave ",v," in advance of
   week ",t," (i.e. following week ",t-1,")")
!                               end-if
!                       end-if
!               end-do
!               forall(r in ALL_ROLES) do
!                       if(best_change(e,r,t) > 0.9) then
!                               changes := TRUE
!                               if(cur_allocate(e,r,t) = 1) then writeln(" - no
   longer carries out role ",r," during week ",t)
!                               else writeln(" - will now carry out role ",r,"
   during week ",t)
!                               end-if
!                       end-if
!               end-do
!       end-do
!       if(changes = FALSE) then writeln(" - no changes"); end-if
!       writeln
!end-do
!
!writeln("Agency crew:")
!changes := FALSE
!forall(t in TIME) do
!       forall(v in VESSELS) do
!               forall(r in ROLES(v)) do
!                       if(best_chng_ag_rboard(r,t) > 0.9) then
!                               changes := TRUE
!                               if(cur_ag_rboard(r,t) = 1) then writeln(" - no
   longer boards ",v," in advance of week ",t)
!                               else writeln(" - will now board ",v," in advance of
   week ",t)
```

```
!                               end-if
!                       end-if
!                       if(best_chng_ag_rdepart(r,t) > 0.9) then
!                               changes := TRUE
!                               if(cur_ag_rdepart(r,t) = 1) then writeln(" - no
    longer leaves ",v," in advance of week ",t," (i.e. following week ",t-1,")")
!                               else writeln(" - will now leave ",v," in advance of
    week ",t," (i.e. following week ",t-1,")")
!                               end-if
!                       end-if
!               end-do
!       end-do
!       forall(r in ALL_ROLES) do
!               if(best_change("AGENCY",r,t) > 0.9) then
!                       changes := TRUE
!                       if(cur_allocate("AGENCY",r,t) = 1) then writeln(" - no
    longer carries out role ",r," during week ",t)
!                       else writeln(" - will now carry out role ",r," during week
    ",t)
!                       end-if
!               end-if
!       end-do
!end-do
!if(changes = FALSE) then writeln(" - no changes"); end-if
!writeln
!writeln
!
!
!
!writeln("---------------------------------------------")
!writeln
!writeln("Roles are now carried out as follows:")
!forall(t in TIME) write("\tWeek ",t)
!write("\n")
!forall(r in ALL_ROLES) do
!       write(r)
!       forall(t in TIME) do
!               if(required(r,t) = 0) then
!                       write("\t (n/a)")
!               else
!                       forall(e in ALL_EMP) do
!                               if(best_new_sched(e,r,t) = 1) then
!                                       write("\t",e)
!                               end-if
!                       end-do
!               end-if
!       end-do
!       write("\n")
!end-do
!writeln
!writeln
!
!writeln("Crew are now assigned to roles as follows:")
!forall(t in TIME) write("\tWeek ",t)
!write("\n")
!forall(e in REG_EMP) do
!       write(e)
!       forall(t in TIME) do
!               not_work := 1
!               forall(r in ALL_ROLES) do
!                       if(best_new_sched(e,r,t) = 1) then
!                               write("\t",r)
!                               not_work := 0
!                       end-if
!               end-do
!               if(not_work = 1) then
!                       avail := FALSE
!                       forall(r in ALL_ROLES) do
!                               if(eligable(e,r,t) = 1) then avail := TRUE
!                               end-if
!                       end-do
!                       if(avail = FALSE) then write("\t*unav*")
!                       else write("\t(rest)")
!                       end-if
!               end-if
!       end-do
!       write("\n")
!end-do
!writeln("... with any non-covered roles assigned to AGENCY crew when necessary
    .")
!writeln
!writeln

prog_endtime := gettime
prog_runtime := prog_endtime-prog_starttime
writeln
```

```
writeln("Running time:\t",prog_runtime)

fclose(F_OUTPUT)


!-----------------------------------------------------------------------
a := 0 ! number of agency-covered tasks in the initial schedule
u := 0 ! number of un-covered tasks in the initial schedule
changes_to_reg := 0          ! number of changes to regular employees in the (
    best) solution
changes_to_AG := 0           ! number of changes to agency employees in the (best
    ) solution
number_of_AG := 0            ! number of times agency employees are utilised in
    the (best) solution
forall(r in ALL_ROLES, t in TIME) do
       x := 0
       forall(e in REG_EMP) do
              if(cur_allocate(e,r,t) = 1) then
                     x := x + 1
              end-if
              changes_to_reg := changes_to_reg + best_change(e,r,t)
       end-do
       if(cur_allocate("AGENCY",r,t) = 1) then
              x := x + 1
              a := a + 1
       end-if
       changes_to_AG := changes_to_AG + best_change("AGENCY",r,t)
       number_of_AG := number_of_AG + best_new_sched("AGENCY",r,t)
       if(x = 0) then u := u + 1
       end-if
end-do


! Print results to the result summary file
fopen(SUMMARYFILE, F_APPEND)
       write(InstanceName)
       write("\t",prog_runtime,"\t",iteration,"\t")
       if(solution = true) then
              write("\t",NO_CHANGES_BEST,"\t",TOTAL_COST)
              write("\t",NO_CHANGES_FIRST,"\t",COST_FIRST,"\t")
       else
              write("\t n/a \t n/a \t n/a \t n/a \t")
       end-if
```

```
        if(any_infeas = true) then write(lower_bound); else write("n/a"); end-if
        write("\t\t",(sum(r in ALL_ROLES, t in TIME) required(r,t)),"\t",u,"\t",a)
        write("\t",changes_to_reg,"\t",changes_to_AG,"\t",number_of_AG)
        write("\n")
fclose(F_APPEND)


end-model
```

E.2.4 Task-Based Approximation algorithm

Here we give the code used to solve the Time-Windows problem by using the Task-Based formulation as an approximation, as described in section 6.4.1. This was implemented using the FICO Xpress software.

```
model TBApproximation
uses "mmxprs", "mmsystem"    !gain access to the Xpress-Optimizer solver

! Programme to create a Task-Based approximation from a Time-Windows dataset

parameters
        DATE = "25-02-14"
        PARAMETERFILE = "batch-input-parameters-2.dat"
end-parameters

declarations
        InstanceName: string
end-declarations

initializations from PARAMETERFILE
        InstanceName
end-initializations


DATAFILE := InstanceName+"\\Time-Windows - Captains - "+InstanceName+".txt"
LOGFILE := InstanceName+"\\Logfile - TBApprox - "+InstanceName+" - "+DATE+".txt"
SUMMARYFILE := "Results - TBApprox - "+DATE+".txt"
SOLUTIONFILE := InstanceName+"\\TBApprox soln - "+InstanceName+" - "+DATE+".txt"
COLLATEDSOLUTIONS := "Task-Based Approx - Reformatted Solutions\\TBApprox soln -
    "+InstanceName+" - "+DATE+".txt"
```

```
! Set parameters relating to running time and acceptable optimality gap:
setparam("XPRS_maxtime", -120)
!setparam("XPRS_miprelstop",0.05)
! Set parameters for numbers of Cover and Gomory cuts to apply:
!setparam("XPRS_covercuts",1000)
!setparam("XPRS_gomcuts",1000)


prog_starttime := gettime           ! get the time so that at the end, running
    time can be calculated


!------------------------------------------------------------------------
! Declare and read in all Time-Windows Data...

declarations
       REG_EMP: set of string       ! Regular employee names / numbers (ie not
           Agency)
       ALL_EMP: set of string       ! Lables for ALL crew (including Agency)
       GUARANTEED:   set of string ! Set of employees on guaranteed days
           contracts
       VESSELS: set of string       ! Labels / names of vessels
       WEEKS_TO_PLAN: integer       ! Length of planning horizon in weeks
       ROLES: array(VESSELS) of set of string     ! Labels for the roles which
           will require cover, divided by vessels
       ALL_ROLES: set of string     ! List of all roles

       exp_worktime, g_weeks, under_rate, over_rate: array(GUARANTEED) of real  !
            Data relating to over/undertime payments for employees
end-declarations

initializations from DATAFILE
       REG_EMP GUARANTEED VESSELS WEEKS_TO_PLAN ROLES
       under_rate over_rate g_weeks exp_worktime
end-initializations


ALL_EMP := REG_EMP + {"AGENCY"}
ALL_ROLES := {}
FORALL (v in VESSELS) ALL_ROLES += ROLES(v)

declarations
       TIME = 1..WEEKS_TO_PLAN      ! Time index
       board_chng_cost, depart_chng_cost: array(REG_EMP, VESSELS, TIME) of real !
            Costs of CHANGES TO employees boarding / leaving vessel
       ag_board_chng_cost, ag_depart_chng_cost: array(ALL_ROLES, TIME) of real !
           Costs of CHANGES TO agency employees boarding / leaving for a given
           role
       work_chng_cost: array(ALL_EMP, ALL_ROLES, TIME) of real
                          ! (Direct) Costs of CHANGES TO employees working a given
            role at a given time

       required: array(ALL_ROLES, TIME) of integer                       ! =1 if
           role r is required at time t, =0 otherwise
       eligable: array(ALL_EMP, ALL_ROLES, TIME) of integer ! =1 if emp i can
           carry out role r at time t, =0 otherwise
       starting: array(ALL_EMP, VESSELS) of integer              ! =1 if emp i is
            on board vessel k at time 0, =0 otherwise
              ! or takes a non-negative integer value for agency crew
       ag_starting: array(ALL_ROLES) of integer                          ! =1 if
           agency employee is in role r at time 0, =0 otherwise
       work_zero, rest_zero: array(REG_EMP) of integer           ! initial values
            of work_total and rest_total at time zero
       ag_work_zero: array(ALL_ROLES) of integer                         ! initial
            value of work total for agency crew task by task
       max_work, min_rest: array(REG_EMP) of integer             ! legal maximum
           on working time, minimum on resting time
       ag_max_work: array(ALL_ROLES) of integer                          ! maximum
            on working time for agency crew, possibly different for each role
       overall_regular_max_work: integer
       overall_agency_max_work: integer
       overall_max_work: integer

       ! Added for the recovery problem:
       ! - the details of the current roster...
       cur_allocate: array(ALL_EMP, ALL_ROLES, TIME) of integer
       cur_board, cur_depart: array(REG_EMP, VESSELS, TIME) of integer
       cur_ag_rboard, cur_ag_rdepart: array(ALL_ROLES, TIME) of integer
       cur_undertime, cur_overtime: array(GUARANTEED) of real
end-declarations

initializations from DATAFILE
       board_chng_cost depart_chng_cost work_chng_cost
       ag_board_chng_cost ag_depart_chng_cost
       required eligable starting ag_starting
```

```
        work_zero rest_zero ag_work_zero
        max_work min_rest ag_max_work

        cur_allocate cur_board cur_depart cur_ag_rboard cur_ag_rdepart
        cur_undertime cur_overtime
end-initializations


overall_regular_max_work := max(e in REG_EMP) max_work(e)
overall_agency_max_work := max(r in ALL_ROLES) ag_max_work(r)
if(overall_regular_max_work > overall_agency_max_work) then
        overall_max_work := overall_regular_max_work
else
        overall_max_work := overall_agency_max_work
end-if

declarations
        lambda = 1..overall_max_work        ! Index used for number of consecutive
             weeks
        extension_chng_cost: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real !
            Cost of CHANGES TO an employee working on board a vessel for longer
            than usual

        !Added for recovery problem - detail of current roster, and change
            variable
        cur_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of integer
end-declarations

initializations from DATAFILE
        extension_chng_cost cur_long_work
end-initializations


!-------------------------------------------------------------------------
! Declare and calculate first group of Task-Based quantities...

declarations
        EMP: set of string                  ! Employee names / numbers
        DAYS_TO_PLAN: integer
        JOBS_TO_PLAN: integer
        task_length, init_len, add_tasks: array(ALL_ROLES) of integer
end-declarations
```

```
EMP := REG_EMP
DAYS_TO_PLAN := WEEKS_TO_PLAN * 7

! Number of Jobs can be calculated using starting, boarding / departing and
    working data
!       and using the assumption that all 'tasks' for a given role will be the
    same length
JOBS_TO_PLAN := 0
forall(v in VESSELS) do
        forall(r in ROLES(v)) do
                current_emp := ""
                task_length(r) := 0
                init_len(r) := 0
                add_tasks(r) := 0
                forall(e in REG_EMP) do
                        if(starting(e,v) = 1) then
                                if(cur_depart(e,v,1) = 0) then
                                        current_emp := e
                                        init_len(r) := work_zero(e)
                                else
                                        forall(f in REG_EMP) do
                                                if(cur_board(f,v,1) = 1) then
                                                        current_emp := f
                                                        init_len(r) := 0
                                                end-if
                                        end-do
                                        if(current_emp = "") then
                                                if(cur_ag_rboard(r,1) = 1) then
                                                        current_emp := "AGENCY"
                                                        init_len(r) := 0
!                                               else
!                                                       writeln("ERROR \t role ",r)
                                                end-if
                                        end-if
                                end-if
                        end-if
                end-do
                if(current_emp = "") then
                        if(ag_starting(r) = 1) then
                                if(cur_ag_rdepart(r,1) = 0) then
                                        current_emp := "AGENCY"
                                        init_len(r) := ag_work_zero(r)
                                else
                                        forall(f in REG_EMP) do
                                                if(cur_board(f,v,1) = 1) then
                                                        current_emp := f
                                                        init_len(r) := 0
                                                end-if
                                        end-do
                                        if(current_emp = "") then
                                                if(cur_ag_rboard(r,1) = 1) then
                                                        current_emp := "AGENCY"
                                                        init_len(r) := 0
!                                       else
!                                               writeln("ERROR \t role ",r)
                                                end-if
                                        end-if
                                end-if
                        end-if
                end-if

                if(current_emp <> "AGENCY") then
                        if(max_work(current_emp) = 2 or max_work(current_emp) = 10)
                            then
                                task_length(r) := max_work(current_emp)
                        else
                                departed := FALSE
                                task_length(r) := init_len(r)
                                forall(t in TIME) do
                                        if(departed = FALSE) then
                                                if(cur_depart(current_emp,v,t) = 1)
                                                    then
                                                        departed := TRUE
                                                else
                                                        task_length(r) := task_length(r
                                                            ) + 1
                                                end-if
                                        end-if
                                end-do
                        end-if
                else
                        if(ag_max_work(r) = 2 or ag_max_work(r) = 10) then
                                task_length(r) := ag_max_work(r)
                        else
                                departed := FALSE
```

```
                              task_length(r) := init_len(r)
                              forall(t in TIME) do
                                     if(departed = FALSE) then
                                            if(cur_ag_rdepart(r,t) = 1) then
                                                   departed := TRUE
                                            else
                                                   task_length(r) := task_length(r
                                                       ) + 1
                                            end-if
                                     end-if
                              end-do
                       end-if
               end-if

!              writeln
!              writeln("Role: ",r,"\tTask lengths: ",task_length(r),"\tInitial
    length: ",init_len(r))
               ! From this information, can calculate the number of tasks on board
               if(10*integer((WEEKS_TO_PLAN + init_len(r))/task_length(r)) = 10*((
                   WEEKS_TO_PLAN + init_len(r))/task_length(r))) then
!                     writeln("\tCase 1: ",10*integer((WEEKS_TO_PLAN + init_len(r)
    )/task_length(r))," = ",10*((WEEKS_TO_PLAN + init_len(r))/task_length(r)))
                      add_tasks(r) := integer((WEEKS_TO_PLAN + init_len(r))/
                          task_length(r))
               else
!                     writeln("\tCase 2: ",10*integer((WEEKS_TO_PLAN + init_len(r)
    )/task_length(r))," < ",10*((WEEKS_TO_PLAN + init_len(r))/task_length(r)))
                      add_tasks(r) := integer((WEEKS_TO_PLAN + init_len(r))/
                          task_length(r)) +1
               end-if

               JOBS_TO_PLAN := JOBS_TO_PLAN + add_tasks(r)
!              writeln("\t=> ",add_tasks(r)," tasks in total.")
       end-do
end-do
!writeln
!writeln("number of jobs:\t",JOBS_TO_PLAN)
!writeln


!------------------------------------------------------------------------
! Can now declare and calculate remaining Task-Based quantities...
```

```
declarations
       JOB = 1..JOBS_TO_PLAN
       eligible: array(EMP, JOB) of real   ! = 1 if employee can carry out a job,
            0 otherwise
       AG_eligible: array(JOB) of real     ! eligibility of agency crew
       change_cost: array(EMP, JOB) of real ! The cost of an employee carrying
           out a job
       AG_change_cost: array(JOB) of real         ! Cost details for agency crew
       start_time, duration: array(JOB) of real
       rest_duration_list: set of integer
       REST_NUMBER: integer
       initial: array(EMP, JOB) of real    ! = 1 if employee is currently
           allocated to a task, 0 otherwise
       AG_initial: array(JOB) of real
       tb_under_rate, tb_over_rate, current_excess, g_days, tb_exp_worktime:
           array(GUARANTEED) of real
       work_resource_zero, rest_resource_zero: array(EMP) of real
       initial_rest_required: array(EMP) of real

       ! In order to convert back into TimeWindow formulation, need to record:
       role_covered: array(JOB) of string
       pre_assign_cost: array(ALL_EMP) of real
       pre_chng_Utime, pre_chng_Otime: array(GUARANTEED) of real
end-declarations


! Can start by calculating contract conditions, in particular the fixed contracts
!      NOTE: min rest and max work are IDENTICAL, and therefore not altered
forall(e in GUARANTEED) do
       tb_under_rate(e) := (under_rate(e))/7
       tb_over_rate(e) := (over_rate(e))/7
       g_days(e) := 7*g_weeks(e)
       tb_exp_worktime(e) := 7*exp_worktime(e)
       current_excess(e) := (over_rate(e)*cur_overtime(e)) + (under_rate(e)*
           cur_undertime(e))
       pre_chng_Utime(e) := 0
       pre_chng_Otime(e) := 0
end-do


! Next, initial resource values:
forall(e in EMP) do
       work_resource_zero(e) := work_zero(e)
```

```
        if(rest_zero(e) = min_rest(e)) then
                rest_resource_zero(e) := 1
                initial_rest_required(e) := 0
        else
                rest_resource_zero(e) := 0
                initial_rest_required(e) := rest_zero(e)
        end-if
end-do



! Calculate various quantities relating to the 'tasks'
!       ie: start time, duration, eligability, initial allocation, costs
forall(e in ALL_EMP) pre_assign_cost(e) := 0
job_number := 0
forall(v in VESSELS) do
        forall(r in ROLES(v)) do
                timecount := 0

                while(timecount < WEEKS_TO_PLAN) do
                        job_number := job_number +1

                        ! identify starting time and duration
                        start_time(job_number) := 7*timecount
                        role_covered(job_number) := r

                        timeindex := timecount

                        if(timecount = 0) then
                                if((task_length(r) - init_len(r)) < WEEKS_TO_PLAN)
                                    then
                                        duration(job_number) := task_length(r) -
                                            init_len(r)
                                        !;writeln("(Task start: ",timecount,"\tTask
                                            length ",task_length(r)," - Initial
                                            length ",init_len(r)," < Horizon length
                                            ",WEEKS_TO_PLAN)
                                        timecount := timecount + (task_length(r) -
                                            init_len(r))
                                else
                                        duration(job_number) := WEEKS_TO_PLAN
                                        !;writeln("(Task start: ",timecount,"\tTask
                                            length ",task_length(r)," - Initial
```

```
                    length ",init_len(r)," >= Horizon length
                    ",WEEKS_TO_PLAN)
                timecount := timecount + WEEKS_TO_PLAN
        end-if

else
        if(timecount + task_length(r) > WEEKS_TO_PLAN) then
                duration(job_number) := WEEKS_TO_PLAN -
                    timecount
                !;writeln("(Task start: ",timecount,"\tTask
                    length ",task_length(r)," + start time ",
                    timecount," > Horizon length ",
                    WEEKS_TO_PLAN)
                timecount := timecount + (WEEKS_TO_PLAN -
                    timecount)
        else
                duration(job_number) := task_length(r)
                !;writeln("(Task start: ",timecount,"\tTask
                    length ",task_length(r)," + start time ",
                    timecount," <= Horizon length ",
                    WEEKS_TO_PLAN)
                timecount := timecount + task_length(r)
        end-if
end-if
duration(job_number) := 7*duration(job_number)
!;writeln("Task ",job_number,"\tRole: ",r,"\tStart: ",
    start_time(job_number),"\tDuration: ",duration(
    job_number),"\tEnd at: ",start_time(job_number)+duration
    (job_number))
!writeln


! identify each employee’s eligibility to carry out this
    task
! NOTE - assume if an employee is unavailable for ONE part
    of the task,
!       then they are unavailable for ALL parts of the task.
forall(e in EMP) do
        eligible(e,job_number) := 1
        forall(t in (timeindex+1)..timecount) do
                if(eligable(e,r,t) = 0) then
                        eligible(e,job_number) := 0
                end-if
        end-do
end-do
AG_eligible(job_number) := 1
forall(t in (timeindex+1)..timecount) do
        if(eligable("AGENCY",r,t) = 0) then
                AG_eligible(job_number) := 0
        end-if
end-do


! identify which employee (if any) is initially assigned to
    this task
inital_found := false
timesearch := timeindex
if(init_len(r) > 0 and start_time(job_number) = 0) then
        if((sum(e in REG_EMP) starting(e,v)) + ag_starting(r
            ) > 0) then
                forall(e in REG_EMP) do
                        initial(e,job_number) := starting(e,v)
                end-do
                AG_initial(job_number) := starting("AGENCY",v
                    )
                initial_found := true
        end-if
end-if
while(inital_found = false and timesearch < timecount) do
        timesearch := timesearch + 1

        if((sum(e in REG_EMP) cur_board(e,v,timesearch)) +
            cur_ag_rboard(r,timesearch) > 0) then
                forall(e in REG_EMP) do
                        initial(e,job_number) := cur_board(e,v
                            ,timesearch)
                end-do
                AG_initial(job_number) := cur_ag_rboard(r,
                    timesearch)
                initial_found := true
        end-if

        if((sum(e in ALL_EMP) cur_allocate(e,r,timesearch))
            > 0) then
                forall(e in REG_EMP) do
```

```
                                              initial(e,job_number) := cur_allocate(
                                                  e,r,timesearch)
                                      end-do
                                      AG_initial(job_number) := cur_allocate("
                                          AGENCY",r,timesearch)
                                      initial_found := true
                              end-if
                      end-do


                      ! identify the cost of changing an employees assignment to
                          each task
                      !       OR, if an initially assigned employee is ineligable
                          then cancel this assignment and calculate the pre-
                          assigned cost
                      forall(e in EMP) do
                              change_cost(e,job_number) := 0

                              if(initial(e,job_number) = 0 or eligible(e,
                                  job_number) = 1) then
                                      if(start_time(job_number) = 0 and starting(e,
                                          v) = 1) then
                                              change_cost(e,job_number) :=
                                                  change_cost(e,job_number) +
                                                  depart_chng_cost(e,v,1)
                                              work_len := init_len(r)
                                      else
                                              change_cost(e,job_number) :=
                                                  change_cost(e,job_number) +
                                                  board_chng_cost(e,v,(timeindex+1))
                                              work_len := 0
                                      end-if

!                                     writeln("for t = ",timeindex+1," to ",
    timecount,"...")
                                      forall(t in (timeindex+1)..timecount) do
                                              work_len := work_len + 1
                                              if(t = WEEKS_TO_PLAN and initial(e,
                                                  job_number) = 1) then
                                                      change_cost(e,job_number) :=
                                                          change_cost(e,job_number) -
                                                           work_chng_cost(e,r,t)
                                                      change_cost(e,job_number) :=
                                                          change_cost(e,job_number) -
                                                           extension_chng_cost(
                                                          work_len,e,r,t)
                                                      pre_assign_cost(e) :=
                                                          pre_assign_cost(e) +
                                                          work_chng_cost(e,r,t) +
                                                          extension_chng_cost(
                                                          work_len,e,r,t)
!                                             writeln("Pre-assignment cost (
    work): employee ",e,"\tRole ",r,"\tWeek ",t,"\tTask ",job_number,"\tCOST: ",
    work_chng_cost(e,r,t))
!                                             writeln("Pre-assignment cost (
    extd): employee ",e,"\tRole ",r,"\tWeek ",t,"\tTask ",job_number,"\tCOST: ",
    extension_chng_cost(work_len,e,r,t))
                                                      if(e in GUARANTEED) then
                                                             if(cur_undertime(e) +
                                                                 pre_chng_Utime(e) >
                                                                 0) then
                                                                     current_excess(e
                                                                         ) :=
                                                                         current_excess
                                                                         (e) -
                                                                         under_rate(e)
                                                                     pre_chng_Utime(e
                                                                         ) :=
                                                                         pre_chng_Utime
                                                                         (e) - 1
                                                                     pre_assign_cost(
                                                                         e) :=
                                                                         pre_assign_cost
                                                                         (e) -
                                                                         under_rate(e)
!                                                            writeln("Pre-
    assignment cost (NUTm): employee ",e,"\tRole ",r,"\tWeek ",t,"\tTask ",
    job_number,"\tCOST: ", - under_rate(e))
                                                             else
                                                                     current_excess(e
                                                                         ) :=
                                                                         current_excess
                                                                         (e) +
                                                                         over_rate(e)
```

```
                                                               pre_chng_Otime(e
                                                                   ) :=
                                                                   pre_chng_Otime
                                                                   (e) + 1
                                                               pre_assign_cost(
                                                                   e) :=
                                                                   pre_assign_cost
                                                                   (e) +
                                                                   over_rate(e)
!                                                              writeln("Pre-
    assignment cost (OTme): employee ",e,"\tRole ",r,"\tWeek ",t,"\tTask ",
    job_number,"\tCOST: ",over_rate(e))
                                                        end-if
                                                 end-if
                                          else
                                                 change_cost(e,job_number) :=
                                                     change_cost(e,job_number) +
                                                      work_chng_cost(e,r,t)
!                                                writeln("emp: ",e,"\tjob: ",
    job_number,"\ttime: ",t," \twork_len: ",work_len)
                                                 change_cost(e,job_number) :=
                                                     change_cost(e,job_number) +
                                                      extension_chng_cost(
                                                     work_len,e,r,t)
                                          end-if
                                   end-do

                                   if(initial(e,job_number) = 0) then
                                          if(timecount < WEEKS_TO_PLAN) then
                                                 change_cost(e,job_number) :=
                                                     change_cost(e,job_number) +
                                                      depart_chng_cost(e,v,
                                                     timecount+1)
                                          end-if
                                   else
                                          if(timecount < WEEKS_TO_PLAN - 1) then
                                                 change_cost(e,job_number) :=
                                                     change_cost(e,job_number) +
                                                      depart_chng_cost(e,v,
                                                     timecount+1)
                                          elif(timecount = WEEKS_TO_PLAN - 1)
                                              then
```

```
                                                change_cost(e,job_number) :=
                                                    change_cost(e,job_number) -
                                                     depart_chng_cost(e,v,
                                                    timecount+1)
                                                pre_assign_cost(e) :=
                                                    pre_assign_cost(e) +
                                                    depart_chng_cost(e,v,
                                                    timecount+1)
!                                               writeln("Pre-assignment cost (
    dprt): employee ",e,"\tVess ",v,"\tWeek ",timecount+1,"\tTask ",job_number,"\
    tCOST: ",depart_chng_cost(e,v,timecount+1))
                                        end-if
                                end-if

                        else
                                if(start_time(job_number) > 0 or starting(e,v
                                    ) = 0) then
                                        pre_assign_cost(e) := pre_assign_cost(
                                            e) + board_chng_cost(e,v,(
                                            timeindex+1))
!                                       writeln("Pre-assignment cost (Nbrd):
    employee ",e,"\tVess ",v,"\tWeek ",timeindex+1,"\tTask ",job_number,"\tCOST:
    ",board_chng_cost(e,v,(timeindex+1)))
                                elif(start_time(job_number) = 0 and starting(
                                    e,v) = 1) then
                                        pre_assign_cost(e) := pre_assign_cost(
                                            e) + depart_chng_cost(e,v,1)
!                                       writeln("Pre-assignment cost (dprt):
    employee ",e,"\tVess ",v,"\tWeek 1\tTask ",job_number,"\tCOST: ",
    depart_chng_cost(e,v,1))
                                end-if

                                if(start_time(job_number) = 0 and starting(e,
                                    v) = 1) then
                                        work_len := init_len(r)
                                else
                                        work_len := 0
                                end-if

                                forall(t in (timeindex+1)..timecount) do
                                        work_len := work_len + 1
                                        if(eligable(e,r,t) = 1 and t <
                                            WEEKS_TO_PLAN) then
```

```
                                                    pre_assign_cost(e) :=
                                                        pre_assign_cost(e) +
                                                        work_chng_cost(e,r,t)
                                                    pre_assign_cost(e) :=
                                                        pre_assign_cost(e) +
                                                        extension_chng_cost(
                                                        work_len,e,r,t)
!                                                   writeln("Pre-assignment cost (
    Nwrk): employee ",e,"\tRole ",r,"\tWeek ",t,"\tTask ",job_number,"\tCOST: ",
    work_chng_cost(e,r,t))
!                                                   writeln("Pre-assignment cost (
    Next): employee ",e,"\tRole ",r,"\tWeek ",t,"\tTask ",job_number,"\tCOST: ",
    extension_chng_cost(work_len,e,r,t))
                                                    if(e in GUARANTEED) then
                                                           if(cur_overtime(e) +
                                                               pre_chng_Otime(e) >
                                                               0) then
                                                                  current_excess(e
                                                                      ) :=
                                                                      current_excess
                                                                      (e) -
                                                                      over_rate(e)
                                                                  pre_chng_Otime(e
                                                                      ) :=
                                                                      pre_chng_Otime
                                                                      (e) - 1
                                                                  pre_assign_cost(
                                                                      e) :=
                                                                      pre_assign_cost
                                                                      (e) -
                                                                      over_rate(e)
!                                                                 writeln("Pre-
    assignment cost (NOTm): employee ",e,"\tRole ",r,"\tWeek ",t,"\tTask ",
    job_number,"\tCOST: ", - over_rate(e))
                                                           else
                                                                  current_excess(e
                                                                      ) :=
                                                                      current_excess
                                                                      (e) +
                                                                      under_rate(e)
                                                                  pre_chng_Utime(e
                                                                      ) :=
                                                                      pre_chng_Utime
```

```
                                                              (e) + 1
                                                          pre_assign_cost(
                                                              e) :=
                                                              pre_assign_cost
                                                              (e) +
                                                              under_rate(e)
!                                                         writeln("Pre-
    assignment cost (UTme): employee ",e,"\tRole ",r,"\tWeek ",t,"\tTask ",
    job_number,"\tCOST: ",under_rate(e))
                                                 end-if
                                          end-if
                                   end-if
                            end-do

                            if(timecount < WEEKS_TO_PLAN - 1) then
                                   pre_assign_cost(e) := pre_assign_cost(
                                       e) + depart_chng_cost(e,v,
                                       timecount+1)
!                                  writeln("Pre-assignment cost (Ndpt):
    employee ",e,"\tVess ",v,"\tWeek ",timecount+1,"\tTask ",job_number,"\tCOST:
    ",depart_chng_cost(e,v,timecount+1))
                            end-if

                            initial(e,job_number) := 0
                     end-if

              end-do


              AG_change_cost(job_number) := 0

              if(start_time(job_number) = 0 and ag_starting(r) = 1) then
                     work_len := init_len(r)
                     if(init_len(r) = 0) then
                            AG_change_cost(job_number) := AG_change_cost(
                                job_number) + ag_board_chng_cost(r,(
                                timeindex+1))
                     else
                            if(AG_initial(job_number) = 1 or
                                cur_ag_rdepart(r,1) = 1) then
                                   AG_change_cost(job_number) :=
                                       AG_change_cost(job_number) +
                                       ag_depart_chng_cost(r,1)
```

```
                                        else
                                                AG_change_cost(job_number) :=
                                                    AG_change_cost(job_number) -
                                                    ag_depart_chng_cost(r,1)
                                                pre_assign_cost("AGENCY") :=
                                                    pre_assign_cost("AGENCY") +
                                                    ag_depart_chng_cost(r,1)
!                                               writeln("Pre-assignment cost (dprt):
    Agency employee \tRole ",r,"\tWeek 1\tTask ",job_number,"\tCOST: ",
    ag_depart_chng_cost(r,1))
                                        end-if
                                end-if
                        else
                                AG_change_cost(job_number) := AG_change_cost(
                                    job_number) + ag_board_chng_cost(r,(timeindex+1))
                                work_len := 0
                        end-if

                        forall(t in (timeindex+1)..timecount) do
                                work_len := work_len + 1
                                if(t = WEEKS_TO_PLAN and AG_initial(job_number) = 1)
                                     then
                                        AG_change_cost(job_number) := AG_change_cost(
                                            job_number) - work_chng_cost("AGENCY",r,t
                                            )
                                        AG_change_cost(job_number) := AG_change_cost(
                                            job_number) - extension_chng_cost(
                                            work_len,"AGENCY",r,t)
                                        pre_assign_cost("AGENCY") := pre_assign_cost
                                            ("AGENCY") + work_chng_cost("AGENCY",r,t)
                                             + extension_chng_cost(work_len,"AGENCY",
                                            r,t)
!                                       writeln("Pre-assignment cost (work): Agency
    employee \tRole ",r,"\tWeek ",t,"\tTask ",job_number,"\tCOST: ",
    work_chng_cost("AGENCY",r,t))
!                                       writeln("Pre-assignment cost (extd): Agency
    employee \tRole ",r,"\tWeek ",t,"\tTask ",job_number,"\tCOST: ",
    extension_chng_cost(work_len,"AGENCY",r,t))
                                else
                                        AG_change_cost(job_number) := AG_change_cost(
                                            job_number) + work_chng_cost("AGENCY",r,t
                                            )
                                    AG_change_cost(job_number) := AG_change_cost(
                                        job_number) + extension_chng_cost(
                                        work_len,"AGENCY",r,t)
                             end-if
                      end-do

                      if(AG_initial(job_number) = 0) then
                             if(timecount < WEEKS_TO_PLAN) then
                                    AG_change_cost(job_number) := AG_change_cost(
                                        job_number) + ag_depart_chng_cost(r,
                                        timecount+1)
                             end-if
                      else
                             if(timecount < WEEKS_TO_PLAN - 1) then
                                    AG_change_cost(job_number) := AG_change_cost(
                                        job_number) + ag_depart_chng_cost(r,
                                        timecount+1)
                             elif(timecount = WEEKS_TO_PLAN - 1) then
                                    AG_change_cost(job_number) := AG_change_cost(
                                        job_number) - ag_depart_chng_cost(r,
                                        timecount+1)
                                    pre_assign_cost("AGENCY") := pre_assign_cost
                                        ("AGENCY") + ag_depart_chng_cost(r,
                                        timecount+1)
!                                   writeln("Pre-assignment cost (dprt): Agency
    employee \tRole ",r,"\tWeek ",timecount+1,"\tTask ",job_number,"\tCOST: ",
    ag_depart_chng_cost(r,timecount+1))
                             end-if
                      end-if

               end-do

       end-do
end-do
writeln


! Must modify the eligibility values to account for the fact that
!      if an employee starts on a vessel and is then removed from performing the
    subsequent task
!      then they must take a REST prior to performing any other task.
! AND that if an employee is mid-rest period at the start of the planning horizon
!      then they must be ineligable for any tasks overlapping their remaining
    rest period.
forall(e in REG_EMP) do
       forall(v in VESSELS) do
              forall(r in ROLES(v)) do
                     if(starting(e,v) = 1 and init_len(r) > 0) then
                            forall(j in JOB) do
                                   if(role_covered(j) in ROLES(v)) then
                                          if(start_time(j) > 0 and start_time(j)
                                               <= 7*min_rest(e)) then
                                                 eligible(e,j) := 0
                                          end-if
                                   else
                                          if(start_time(j) <= 7*min_rest(e))
                                              then
                                                 eligible(e,j) := 0
                                          end-if
                                   end-if
                            end-do
                     end-if
              end-do
       end-do

       if(initial_rest_required(e) > 0) then
              forall(j in JOB | start_time(j) <= 7*(initial_rest_required(e)-1))
                  do
                     eligible(e,j) := 0
              end-do
       end-if
end-do


split_time_1 := gettime


!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
! Now all the data quantities have been defined, must calculate the other
    secondary information, as
!      per the Task-Based model as implemented before.
```

```
!----------------------------------------------------------------------
! Firstly define 'rest' tasks such that they start directly after each work task
    has ended
! (subject to them finishing before the time-window ends)
rest_duration_list := {}
forall(i in EMP) rest_duration_list += {7*min_rest(i)}

declarations
       rest_start_list: array(rest_duration_list) of set of real
end-declarations

REST_NUMBER := 0

forall(j in JOB, r in rest_duration_list) do
       if start_time(j) + duration(j) < (DAYS_TO_PLAN - r + 1) then
              rest_start_list(r) += {start_time(j) + duration(j)}
       end-if
end-do

forall(r in rest_duration_list) REST_NUMBER := REST_NUMBER + getsize(
    rest_start_list(r))

declarations
       REST = 1..REST_NUMBER
       rest_start_time, rest_duration: array(REST) of real
       work_content_work_arc, rest_content_work_arc: array(JOB) of real
       work_content_rest_arc, rest_content_rest_arc: array(REST) of real
end-declarations

index := 0
forall(r in rest_duration_list) do
       forall(s in rest_start_list(r)) do
              index := index + 1
              rest_start_time(index) := s
              rest_duration(index) := r
       end-do
end-do

!----------------------------------------------------------------------
! Can now define work and rest content for work and rest arcs
forall(j in JOB) do
       work_content_work_arc(j) := duration(j)
       rest_content_work_arc(j) := 1
```

```
end-do


forall(r in REST) do
       work_content_rest_arc(r) := -DAYS_TO_PLAN
       rest_content_rest_arc(r) := -1
end-do

!-----------------------------------------------------------------------
! Next we will generate the sets C_beta of overlapping tasks
declarations
       beta: integer
       time_points: set of real
       time_points_ordered: list of real
       NUMBER_OF_POINTS: integer
       min_time: real
       beta_index: range
       overlap_set_work, overlap_set_rest: dynamic array(beta_index) of set of
           integer
       current_set_work, current_set_rest: set of integer
end-declarations

time_points := {}
forall(j in JOB) do
       time_points += {start_time(j)}
       time_points += {start_time(j) + duration(j)}
end-do
forall(j in REST) do
       time_points += {rest_start_time(j)}
       time_points += {rest_start_time(j) + rest_duration(j)}
end-do

time_points_ordered := []
NUMBER_OF_POINTS := getsize(time_points)
forall(i in 1..NUMBER_OF_POINTS) do
       min_time := DAYS_TO_PLAN
       forall(p in time_points) do
              if(p < min_time) then min_time := p; end-if
       end-do
       time_points_ordered += [min_time]
       time_points -= {min_time}
end-do
```

```
current_set_work := {}
current_set_rest := {}
beta := 1
constraint_required := FALSE
add_constraint := FALSE
forall(t in time_points_ordered) do
        ! check to see if another constraint is required
        forall(j in JOB | start_time(j) + duration(j) = t) add_constraint := TRUE
        forall(j in REST | rest_start_time(j) + rest_duration(j) = t)
            add_constraint := TRUE
        ! if so, we create an 'overlap set' and add the current contents to it
        if(constraint_required = TRUE AND add_constraint = TRUE) then
                create(overlap_set_work(beta))
                create(overlap_set_rest(beta))
                overlap_set_work(beta) := current_set_work
                overlap_set_rest(beta) := current_set_rest
                beta := beta + 1
                add_constraint := FALSE
                constraint_required := FALSE
        end-if

        ! remove these tasks from the current set
        forall(j in JOB | start_time(j) + duration(j) = t) current_set_work -= {j}
        forall(j in REST | rest_start_time(j) + rest_duration(j) = t)
            current_set_rest -= {j}

        ! if there are new tasks starting at this point, add them to the current
            set and
        ! indicate that a contraint will now be required again
        forall(j in JOB | start_time(j) = t) do
                constraint_required := TRUE
                current_set_work += {j}
        end-do
        forall(j in REST | rest_start_time(j) = t) do
                constraint_required := TRUE
                current_set_rest += {j}
        end-do
end-do


!-------------------------------------------------------------------------
! Define Ordered Task Set B (of ALL tasks - work, rest)
!writeln("Define the ordered task set of ALL tasks")
```

```
declarations
       order_index = 1..(JOBS_TO_PLAN + REST_NUMBER)
       ordered_task_number: array(order_index) of integer
       ordered_task_work, ordered_task_rest: array(order_index) of integer
       work_unproc, rest_unproc: set of integer
       work_proc, rest_proc: set of integer
       b: integer
end-declarations


forall(j in JOB) work_unproc += {j}
forall(j in REST) rest_unproc += {j}

b := 0
forall(i in 1..(JOBS_TO_PLAN + REST_NUMBER)) do
       min_time := DAYS_TO_PLAN
       forall(j in work_unproc) do
              if(start_time(j) < min_time) then min_time := start_time(j); end-if
       end-do
       forall(j in rest_unproc) do
              if(rest_start_time(j) < min_time) then min_time := rest_start_time(
                  j); end-if
       end-do

       work_proc := {}
       forall(j in work_unproc) do
              if(start_time(j) = min_time) then
                     b := b+1
                     ordered_task_number(b) := j
                     ordered_task_work(b) := 1
                     ordered_task_rest(b) := 0
                     work_proc += {j}
              end-if
       end-do
       work_unproc -= work_proc

       rest_proc := {}
       forall(j in rest_unproc) do
              if(rest_start_time(j) = min_time) then
                     b := b + 1
                     ordered_task_number(b) := j
                     ordered_task_work(b) := 0
                     ordered_task_rest(b) := 1
```

```
                    rest_proc += {j}
                end-if
        end-do
        rest_unproc -= rest_proc

end-do


!------------------------------------------------------------------------
!------------------------------------------------------------------------
!------------------------------------------------------------------------
! Now that all quantities have been defined, we can state all the contraints:
declarations
        tb_undertime, tb_overtime: array(GUARANTEED) of mpvar
        change: dynamic array(EMP, JOB) of mpvar
        AG_change: dynamic array(JOB) of mpvar
        new_sched: dynamic array(EMP, JOB) of mpvar
        AG_new_sched: dynamic array(JOB) of mpvar
        rest_new_sched: dynamic array(EMP, REST) of mpvar
        work_resource, rest_resource: array(EMP, order_index) of mpvar

        TOTAL_COST: linctr
        job_cover_constr: array(JOB) of linctr
        overlap_constr: array(EMP,beta_index) of linctr
        work_resource_constr_1, rest_resrouce_constr_1: array(EMP) of linctr
        work_resource_constr_b, rest_resource_constr_b: array(EMP, order_index) of
             linctr
        work_resource_bound_constr, rest_resource_bound_constr: array(EMP,
            order_index) of linctr
        undertime_constr, overtime_constr: array(GUARANTEED) of linctr
        new_sched_constr: array(EMP, JOB) of linctr
        AG_new_sched_constr: array(JOB) of linctr

        status: array({XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH}) of string !
            Used to indicate the solution status of the problem
end-declarations


! Must define the change and new_sched (ie y and z) variables ONLY for the
! tasks for which each crew member is eligible or are currently assigned
forall(j in JOB) do
        forall(i in EMP) do
                if (eligible(i,j) = 1) then
```

```
                        create(change(i,j))
                        create(new_sched(i,j))
                end-if
        end-do
        if (AG_eligible(j) = 1) then
                create(AG_change(j))
                create(AG_new_sched(j))
        end-if
end-do


! Must also define the rest 'new sched' variables only if the duration
! is of suitable length for the employee's minimum rest period
forall(r in REST, i in EMP) do
        if(rest_duration(r) = 7*min_rest(i)) then
                create(rest_new_sched(i,r))
!               writeln("Rest task: ",r,"\tDuration: ",rest_duration(r),"\tEmployee
    : ",i,"\tMinRest: ",min_rest(i),"\t=> Create variable")
        end-if
end-do


! Objective:
TOTAL_COST := sum(i in EMP, j in JOB)(change_cost(i,j) * change(i,j)) + sum(j in
    JOB)(AG_change_cost(j) * AG_change(j)) + sum(i in GUARANTEED)((tb_over_rate(i
    ) * tb_overtime(i)) + (tb_under_rate(i) * tb_undertime(i)) - current_excess(i
    )) + sum(e in ALL_EMP) pre_assign_cost(e)

! Other constraints:
forall(j in JOB) job_cover_constr(j) := sum(i in EMP)(new_sched(i,j)) +
    AG_new_sched(j) = 1

forall(i in EMP, a in beta_index) overlap_constr(i,a) := sum(j in
    overlap_set_work(a))(new_sched(i,j)) + sum(j in overlap_set_rest(a))(
    rest_new_sched(i,j)) <= 1

forall(i in EMP) do
! Revised version of inner part of this loop:
        forall(o in order_index) do
                if(o = 1) then
                        if(ordered_task_work(1) = 1) then
                                work_resource_constr_b(i,o) := 7*work_resource_zero(
                                    i) + (new_sched(i,ordered_task_number(1)) *
                                    work_content_work_arc(ordered_task_number(1))) <=
                                     work_resource(i,1)
                                rest_resource_constr_b(i,o) := rest_resource_zero(i)
                                     + (new_sched(i,ordered_task_number(1)) *
                                    rest_content_work_arc(ordered_task_number(1))) <=
                                     rest_resource(i,1)
                        end-if
                        if(ordered_task_rest(1) = 1) then
                                work_resource_constr_b(i,o) := 7*work_resource_zero(
                                    i) + (rest_new_sched(i,ordered_task_number(1)) *
                                    work_content_rest_arc(ordered_task_number(1))) <=
                                     work_resource(i,1)
                                rest_resource_constr_b(i,o) := rest_resource_zero(i)
                                     + (rest_new_sched(i,ordered_task_number(1)) *
                                    rest_content_rest_arc(ordered_task_number(1))) <=
                                     rest_resource(i,1)
                        end-if
                else
                        if(ordered_task_work(o) = 1) then
                                work_resource_constr_b(i,o) := work_resource(i,(o-1)
                                    ) + (new_sched(i,ordered_task_number(o)) *
                                    work_content_work_arc(ordered_task_number(o))) <=
                                     work_resource(i,o)
                                rest_resource_constr_b(i,o) := rest_resource(i,(o-1)
                                    ) + (new_sched(i,ordered_task_number(o)) *
                                    rest_content_work_arc(ordered_task_number(o))) <=
                                     rest_resource(i,o)
                        end-if
                        if(ordered_task_rest(o) = 1) then
                                work_resource_constr_b(i,o) := work_resource(i,(o-1)
                                    ) + (rest_new_sched(i,ordered_task_number(o)) *
                                    work_content_rest_arc(ordered_task_number(o))) <=
                                     work_resource(i,o)
                                rest_resource_constr_b(i,o) := rest_resource(i,(o-1)
                                    ) + (rest_new_sched(i,ordered_task_number(o)) *
                                    rest_content_rest_arc(ordered_task_number(o))) <=
                                     rest_resource(i,o)
                        end-if
                end-if
                work_resource_bound_constr(i,o) := work_resource(i,o) <= 7*max_work
                    (i)
                rest_resource_bound_constr(i,o) := rest_resource(i,o) <= 1
        end-do
```

```
end-do


forall(i in GUARANTEED) undertime_constr(i) := tb_undertime(i) >= g_days(i) - (
    tb_exp_worktime(i) + sum(j in JOB)(duration(j) * new_sched(i,j)))

forall(i in GUARANTEED) overtime_constr(i) := tb_overtime(i) >= (tb_exp_worktime(
    i) + sum(j in JOB)(duration(j) * new_sched(i,j))) - g_days(i)

forall(i in EMP, j in JOB | eligible(i,j) = 1) do
       if(initial(i,j) = 1) then new_sched_constr(i,j) := new_sched(i,j) =
           initial(i,j) - change(i,j); end-if
       if(initial(i,j) = 0) then new_sched_constr(i,j) := new_sched(i,j) =
           initial(i,j) + change(i,j); end-if
       change(i,j) is_binary
end-do
forall(j in JOB | AG_eligible(j) = 1) do
       if(AG_initial(j) = 1) then AG_new_sched_constr(j) := AG_new_sched(j) =
           AG_initial(j) - AG_change(j); end-if
       if(AG_initial(j) = 0) then AG_new_sched_constr(j) := AG_new_sched(j) =
           AG_initial(j) + AG_change(j); end-if
       AG_change(j) is_binary
end-do



!-------------------------------------------------------------------------
! Now solve the problem and print the results:
fopen(LOGFILE, F_OUTPUT)
       setparam("XPRS_verbose",true)
       minimize(TOTAL_COST)
fclose(F_OUTPUT)


status::([XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH])["Optimum found","
    Unfinished","Infeasible","Unbounded","Failed"]

split_time_2 := gettime


!-------------------------------------------------------------------------
!-------------------------------------------------------------------------
```

```
!------------------------------------------------------------------------
! Now convert this solution back to a Time-Windows solution:

declarations
       allocate: array(ALL_EMP, ALL_ROLES, TIME) of integer      ! Variable for
           allocating employee to role during given time period
       board, depart: array(REG_EMP, VESSELS, TIME) of integer   ! =1 if employee
            boards / departs vessel in given time period, 0 otherwise
              ! or takes a non-negative integer value for agency crew
       ag_rboard, ag_rdepart: array(ALL_ROLES, TIME) of integer        ! =1 if
           agency crew starts / ends working on a role in given time period, 0
           otherwise
       undertime, overtime: array(GUARANTEED) of integer                      !
           Variables to calculate the amount of under/overtime carried out by
           employee
       work_total, rest_total: array(REG_EMP, TIME) of real      ! Used to track
           the consecutive working time / rest period requirements of each
           employee
       ag_work_total: array(ALL_ROLES, TIME) of real                   ! Used to
            track the consecutive working time of the agency employees

       chng_allocate: array(ALL_EMP, ALL_ROLES, TIME) of integer
       chng_board, chng_depart: array(REG_EMP, VESSELS, TIME) of integer
       chng_ag_rboard, chng_ag_rdepart: array(ALL_ROLES, TIME) of integer
       chng_undertime, chng_overtime: array(GUARANTEED) of integer

       long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of integer    ! Used to
            indicate if a special bonus / penalty payment relating to consecutive
            time at sea is required
       chng_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of integer

       TW_COST: real
end-declarations



forall(v in VESSELS) do
       forall(r in ROLES(v)) do
              forall(j in JOB | role_covered(j) = r) do

                     tw_start := integer(start_time(j)/7)
                     tw_duration := integer(duration(j)/7)
```

```
                  forall(e in REG_EMP) do
                          if(tw_start = 0 and starting(e,v) = 1) then
                                  if(getsol(new_sched(e,j)) > 0.9) then
                                          work_time := integer(
                                              work_resource_zero(e))
                                          forall(t in (tw_start+1)..(tw_start+
                                              tw_duration)) do
                                                  work_time := work_time + 1
                                                  allocate(e,role_covered(j),t)
                                                      := 1
!                                                 long_work(work_time,e,r,t) := 1
                           ! note, would prefer this description...
                                                  forall(l in 1..work_time) do

                                                              !
                                                          long_work(l,e,r,t) := 1

                                                                              ! but
                                                               this one fits TW
                                                              formulation properly
                                                  end-do


                                                              !
                                          end-do
                                          if(tw_start + tw_duration <
                                              WEEKS_TO_PLAN) then
                                                  depart(e,v,(tw_start+
                                                      tw_duration+1)) := 1
                                          end-if
                                  else
                                          depart(e,v,1) := 1
                                  end-if
                          else
                                  if(getsol(new_sched(e,j)) > 0.9) then
                                          board(e,v, tw_start+1) := 1
                                          work_time := 0
                                          forall(t in (tw_start+1)..(tw_start+
                                              tw_duration)) do
                                                  work_time := work_time + 1
                                                  allocate(e,role_covered(j),t)
                                                      := 1
```

```
!                                         long_work(work_time,e,r,t) := 1
                         ! note, would prefer this description...
                                          forall(l in 1..work_time) do

                                                   !
                                                long_work(l,e,r,t) := 1

                                                               ! but
                                                    this one fits TW
                                                   formulation properly
                                          end-do


                                                   !
                                    end-do
                                    if(tw_start + tw_duration <
                                       WEEKS_TO_PLAN) then
                                          depart(e,v,(tw_start+
                                             tw_duration+1)) := 1
                                    end-if
                              end-if
                        end-if
                  end-do

                  if(tw_start = 0 and ag_starting(r) = 1) then
                        if(getsol(AG_new_sched(j)) > 0.9) then
                              if(ag_work_zero(r) = ag_max_work(r) or
                                 init_len(r) = 0) then
                                    ag_rdepart(r,1) := 1
                                    ag_rboard(r,1) := 1
                                    work_time := 0
                              else
                                    work_time := integer(ag_work_zero(r))
                              end-if
                              forall(t in (tw_start+1)..(tw_start+
                                 tw_duration)) do
                                    work_time := work_time + 1
                                    allocate("AGENCY",role_covered(j),t)
                                       := 1
!                                   long_work(work_time,"AGENCY",r,t) := 1
                    ! note, would prefer this description...
                                    forall(l in 1..work_time) do
                                                               !
```

```
                                        long_work(l,"AGENCY",r,t) := 1

                                            ! but this one fits TW
                                            formulation properly
                                    end-do

                                                        !
                                end-do
                                if(tw_start + tw_duration < WEEKS_TO_PLAN)
                                    then
                                    ag_rdepart(r,(tw_start+tw_duration+1))
                                        := 1
                                end-if
                        else
                                ag_rdepart(r,1) := 1
                        end-if
                else
                        if(getsol(AG_new_sched(j)) > 0.9) then
                                ag_rboard(r,(integer(start_time(j)/7)+1)) :=
                                    1
                                work_time := 0
                                forall(t in (tw_start+1)..(tw_start+
                                    tw_duration)) do
                                    work_time := work_time + 1
                                    allocate("AGENCY",role_covered(j),t)
                                        := 1
!                                   long_work(work_time,"AGENCY",r,t) := 1
                  ! note, would prefer this description...
                                    forall(l in 1..work_time) do
                                                            !
                                        long_work(l,"AGENCY",r,t) := 1

                                            ! but this one fits TW
                                            formulation properly
                                    end-do

                                                        !
                                end-do
                                if(tw_start + tw_duration < WEEKS_TO_PLAN)
                                    then
                                    ag_rdepart(r,(tw_start+tw_duration+1))
                                        := 1
                                end-if
```

```
                              end-if
                        end-if

                  end-do
            end-do
      end-do

      forall(e in GUARANTEED) do
            if(g_weeks(e) > (exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(allocate
                (e,r,t)))) then
                  undertime(e) := integer(g_weeks(e) - (exp_worktime(e) + sum(r in
                      ALL_ROLES, t in TIME)(allocate(e,r,t))))
                  overtime(e) := 0
            else
                  overtime(e) := integer((exp_worktime(e) + sum(r in ALL_ROLES, t in
                      TIME)(allocate(e,r,t)))- g_weeks(e))
                  undertime(e) := 0
            end-if
      end-do



      forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
            if(cur_allocate(e,r,t) = 0) then chng_allocate(e,r,t) := allocate(e,r,t)
            else chng_allocate(e,r,t) := cur_allocate(e,r,t) - allocate(e,r,t)
            end-if
      end-do

      forall(e in REG_EMP, v in VESSELS, t in TIME) do
            if(cur_board(e,v,t) = 0) then chng_board(e,v,t) := board(e,v,t)
            else chng_board(e,v,t) := cur_board(e,v,t) - board(e,v,t)
            end-if
      end-do

      forall(e in REG_EMP, v in VESSELS, t in TIME) do
            if(cur_depart(e,v,t) = 0) then chng_depart(e,v,t) := depart(e,v,t)
            else chng_depart(e,v,t) := cur_depart(e,v,t) - depart(e,v,t)
            end-if
      end-do

      forall(r in ALL_ROLES, t in TIME) do
            if(cur_ag_rboard(r,t) = 0) then chng_ag_rboard(r,t) := ag_rboard(r,t)
            else chng_ag_rboard(r,t) := cur_ag_rboard(r,t) - ag_rboard(r,t)
```

```
        end-if
end-do

forall(r in ALL_ROLES, t in TIME) do
        if(cur_ag_rdepart(r,t) = 0) then chng_ag_rdepart(r,t) := ag_rdepart(r,t)
        else chng_ag_rdepart(r,t) := cur_ag_rdepart(r,t) - ag_rdepart(r,t)
        end-if
end-do

forall(e in GUARANTEED) do
        chng_undertime(e) := integer(undertime(e) - cur_undertime(e))
        chng_overtime(e) := integer(overtime(e) - cur_overtime(e))
end-do

forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME) do
        if(cur_long_work(l,e,r,t) = 0) then chng_long_work(l,e,r,t) := long_work(l
            ,e,r,t)
        else chng_long_work(l,e,r,t) := cur_long_work(l,e,r,t) - long_work(l,e,r,t
            )
        end-if
end-do


TW_COST := (sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*
    chng_board(e,v,t)) + (depart_chng_cost(e,v,t)*chng_depart(e,v,t))) +
                              sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r
                                  ,t)*chng_ag_rboard(r,t)) + (ag_depart_chng_cost(r
                                  ,t)*chng_ag_rdepart(r,t))) +
                              sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((
                                  work_chng_cost(e,r,t)*chng_allocate(e,r,t)) + sum
                                  (l in lambda)((extension_chng_cost(l,e,r,t)*
                                  chng_long_work(l,e,r,t)))) +
                              sum(e in GUARANTEED)((under_rate(e)*chng_undertime(e
                                  ))+ (over_rate(e)*chng_overtime(e))))



!-------------------------------------------------------------------------
! Should now check the Time-Windows version of the solution for Feasibility...

feasible := TRUE
```

```
!writeln("\tChecking Job Cover constraints...")
JCfeas := TRUE
forall(r in ALL_ROLES, t in TIME | required(r,t) > 0) do
       if(sum(e in ALL_EMP)(eligable(e,r,t)*allocate(e,r,t)) <> required(r,t))
           then
              JCfeas := FALSE
!             writeln("\tINFEASIBILITY detected - Job Cover constraint (",r,",",t
    ,")\t",sum(e in ALL_EMP)(eligable(e,r,t)*allocate(e,r,t))," <> ",required(r,t
    ))
       end-if
end-do
if(JCfeas = FALSE) then
       feasible := FALSE
!else
!      writeln("\tJob Cover constraints feasible")
end-if
!writeln


!writeln("\tChecking Overlap constraints...")
OLfeas := TRUE
forall(e in REG_EMP, t in TIME) do
       if(sum(r in ALL_ROLES) allocate(e,r,t) > 1) then
              OLfeas := FALSE
!             writeln("\tINFEASIBILITY detected - Overlap constraint (",e,",",t
    ,")\t",sum(r in ALL_ROLES) allocate(e,r,t)," > 1")
       end-if
end-do
if(OLfeas = FALSE) then
       feasible := FALSE
!else
!      writeln("\tOverlap constraints feasible")
end-if
!writeln


!writeln("\tChecking Boarding constraints...")
Brdfeas := TRUE
forall(e in REG_EMP, v in VESSELS) do
       if(board(e,v,1) < sum(r in ROLES(v))(allocate(e,r,1)) - starting(e,v))
           then
              Brdfeas := FALSE
```

```
!             writeln("\tINFEASIBILITY detected - Boarding constraint (",e,",",v
    ,",1)\t",board(e,v,1)," < ",sum(r in ROLES(v))(allocate(e,r,1)) - starting(e,
    v))
       end-if
       forall(t in 2..WEEKS_TO_PLAN) do
              if(board(e,v,t) < sum(r in ROLES(v))(allocate(e,r,t)) - sum(r in
                  ROLES(v))(allocate(e,r,(t-1)))) then
                     Brdfeas := FALSE
!                    writeln("\tINFEASIBILITY detected - Boarding constraint (",e
    ,",",v,",",t,")\t",board(e,v,t)," < ",sum(r in ROLES(v))(allocate(e,r,t)) -
    sum(r in ROLES(v))(allocate(e,r,(t-1))))
              end-if
       end-do
end-do
if(Brdfeas = FALSE) then
       feasible := FALSE
!else
!      writeln("\tBoarding constraints feasible")
end-if
!writeln


!writeln("\tChecking Departing constraints...")
Dprtfeas := TRUE
forall(e in REG_EMP, v in VESSELS) do
       if(depart(e,v,1) < starting(e,v) - sum(r in ROLES(v))(allocate(e,r,1)))
           then
              Dprtfeas := FALSE
!             writeln("\tINFEASIBILITY detected - Departing constraint (",e,",",v
    ,",1)\t",depart(e,v,1)," < ",starting(e,v) - sum(r in ROLES(v))(allocate(e,r
    ,1)))
       end-if
       forall(t in 2..WEEKS_TO_PLAN) do
              if(depart(e,v,t) < sum(r in ROLES(v))(allocate(e,r,(t-1))) - sum(r
                  in ROLES(v))(allocate(e,r,t))) then
                     Dprtfeas := FALSE
!                    writeln("\tINFEASIBILITY detected - Departing constraint (",
    e,",",v,",",t,")\t",depart(e,v,t)," < ",sum(r in ROLES(v))(allocate(e,r,(t-1)
    )) - sum(r in ROLES(v))(allocate(e,r,t)))
              end-if
       end-do
end-do
if(Dprtfeas = FALSE) then
```

```
       feasible := FALSE
!else
!      writeln("\tDeparting constraints feasible")
end-if
!writeln


!writeln("\tChecking AG board/depart constraints...")
AGBDfeas := TRUE
forall(r in ALL_ROLES) do
       if(ag_rboard(r,1) - ag_rdepart(r,1) <> allocate("AGENCY",r,1) -
           ag_starting(r)) then
              AGBDfeas := FALSE
!             writeln("\tINFEASIBILITY detected - AG board/depart constraint (",r
    ,",1)\t",ag_rboard(r,1) - ag_rdepart(r,1)," <> ",allocate("AGENCY",r,1) -
    ag_starting(r))
       end-if
       forall(t in 2..WEEKS_TO_PLAN) do
              if(ag_rboard(r,t) - ag_rdepart(r,t) <> allocate("AGENCY",r,t) -
                  allocate("AGENCY",r,(t-1))) then
                     AGBDfeas := FALSE
!                    writeln("\tINFEASIBILITY detected - AG board/depart
    constraint (",r,",",t,")\t",ag_rboard(r,t) - ag_rdepart(r,t)," <> ",allocate
    ("AGENCY",r,t) - allocate("AGENCY",r,(t-1)))
              end-if
       end-do
end-do
if(AGBDfeas = FALSE) then
       feasible := FALSE
!else
!      writeln("\tAG board/depart constraints feasible")
end-if
!writeln


!writeln("\tChecking Undertime constraints...")
UTfeas := TRUE
forall(e in GUARANTEED) do
       if(undertime(e) < g_weeks(e) - (exp_worktime(e) + sum(r in ALL_ROLES, t in
            TIME)(allocate(e,r,t)))) then
              UTfeas := FALSE
!             writeln("\tINFEASIBILITY detected - Undertime constraint (",e,")\t
    ",undertime(e)," < ",g_weeks(e) - (exp_worktime(e) + sum(r in ALL_ROLES, t in
     TIME)(allocate(e,r,t))))
        end-if
end-do
if(UTfeas = FALSE) then
       feasible := FALSE
!else
!      writeln("\tUndertime constraints feasible")
end-if
!writeln


!writeln("\tChecking Overtime constraints...")
OTfeas := TRUE
forall(e in GUARANTEED) do
       if(overtime(e) < (exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(
           allocate(e,r,t)))- g_weeks(e)) then
              OTfeas := FALSE
!             writeln("\tINFEASIBILITY detected - Overtime constraint (",e,")\t",
    overtime(e)," < ",(exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(allocate(
    e,r,t)))- g_weeks(e))
       end-if
end-do
if(OTfeas = FALSE) then
       feasible := FALSE
!else
!      writeln("\tOvertime constraints feasible")
end-if
!writeln


!writeln("\tCalculating 'work_total' values...")
forall(e in REG_EMP) do
       work_total(e,1) := work_zero(e) + sum(r in ALL_ROLES)(allocate(e,r,1)) -
           max_work(e)*(1-(sum(r in ALL_ROLES)(allocate(e,r,1))))
       if(work_total(e,1) < 0) then
              work_total(e,1) := 0
       end-if
       forall(t in 2..WEEKS_TO_PLAN) do
              work_total(e,t) := work_total(e,(t-1)) + sum(r in ALL_ROLES)(
                  allocate(e,r,t)) - max_work(e)*(1-(sum(r in ALL_ROLES)(allocate
                  (e,r,t))))
              if(work_total(e,t) < 0) then
                     work_total(e,t) := 0
              end-if
        end-do
end-do
!writeln("\t'work_total' values calculated")
!writeln


!writeln("\tChecking Long Work constraints...")
LWfeas := TRUE
forall(l in lambda, e in REG_EMP, r in ALL_ROLES | exists(long_work(l,e,r,1))) do
!       if(max_work(e)*long_work(l,e,r,1) < work_zero(e) + allocate(e,r,1) - (l-1)
    ) then
        if(max_work(e)*long_work(l,e,r,1) < work_zero(e) - max_work(e)*(1-allocate
            (e,r,1)) + allocate(e,r,1) - (l-1)) then
              LWfeas := FALSE
!             writeln("\tINFEASIBILITY detected - Long Work constraint (",l,",",e
    ,",",r,",1)\t",max_work(e)*long_work(l,e,r,1)," < ",work_zero(e) + allocate(e
    ,r,1) - (l-1))
!             writeln("\tINFEASIBILITY detected - Long Work constraint (",l,",",e
    ,",",r,",1)\t",max_work(e),"*",long_work(l,e,r,1)," < ",work_zero(e)," - ",
    max_work(e),"*",(1-allocate(e,r,1))," + ",allocate(e,r,1)," - (",l,"-1)")
        end-if
        forall(t in 2..WEEKS_TO_PLAN) do
!             if(max_work(e)*long_work(l,e,r,t) < work_total(e,(t-1)) + allocate(
    e,r,t) - (l-1)) then
              if(max_work(e)*long_work(l,e,r,t) < work_total(e,(t-1)) - max_work(
                  e)*(1-allocate(e,r,t)) + allocate(e,r,t) - (l-1)) then
                    LWfeas := FALSE
!                   writeln("\tINFEASIBILITY detected - Long Work constraint (",
    l,",",e,",",r,",",t,")\t",max_work(e)*long_work(l,e,r,t)," < ",work_total(e,(
    t-1)) + allocate(e,r,t) - (l-1))
!                   writeln("\tINFEASIBILITY detected - Long Work constraint (",
    l,",",e,",",r,",",t,")\t",max_work(e),"*",long_work(l,e,r,t)," < ",work_total
    (e,(t-1))," - ",max_work(e),"*",(1-allocate(e,r,t))," + ",allocate(e,r,t)," -
     (",l,"-1)")
              end-if
        end-do
end-do
if(LWfeas = FALSE) then
        feasible := FALSE
!else
!       writeln("\tLong Work constraints feasible")
end-if
```

```
!writeln


!writeln("\tCalculating Agency 'work_total' values...")
forall(r in ALL_ROLES) do
       ag_work_total(r,1) := ag_work_zero(r) + allocate("AGENCY",r,1) -
           ag_max_work(r)*ag_rdepart(r,1)
       if(ag_work_total(r,1) < allocate("AGENCY",r,1)) then
              ag_work_total(r,1) := allocate("AGENCY",r,1)
       end-if
       forall(t in 2..WEEKS_TO_PLAN) do
              ag_work_total(r,t) := ag_work_total(r,(t-1)) + allocate("AGENCY",r,
                  t) - ag_max_work(r)*ag_rdepart(r,t)
              if(ag_work_total(r,t) < allocate("AGENCY",r,t)) then
                     ag_work_total(r,t) := allocate("AGENCY",r,t)
              end-if
       end-do
end-do
!writeln("\tAgency 'work_total' values calculated")
!writeln


!writeln("\tChecking Agency Long Work constraints...")
AGLWfeas := TRUE
forall(l in lambda, r in ALL_ROLES, t in TIME | exists(long_work(l,"AGENCY",r,t))
    ) do
       if(ag_max_work(r)*long_work(l,"AGENCY",r,t) < ag_work_total(r,t) - (l-1))
           then
              AGLWfeas := FALSE
!             writeln("\tINFEASIBILITY detected - Agency Long Work constraint (",
    l,",",r,",",t,")\t",ag_max_work(r)*long_work(l,"AGENCY",r,t)," < ",
    ag_work_total(r,t) - (l-1))
       end-if
end-do
if(AGLWfeas = FALSE) then
       feasible := FALSE
!else
!      writeln("\tAgency Long Work constraints feasible")
end-if
!writeln


!writeln("\tCalculating 'rest_total' values...")
```

```
forall(e in REG_EMP) do
       rest_total(e,1) := rest_zero(e) - (1-(sum(r in ALL_ROLES)(allocate(e,r,1))
           ))                  !;if(e = "C-28") then writeln("rest_total(",e,",1)
           := ",rest_zero(e)," - (1-",(sum(r in ALL_ROLES)(allocate(e,r,1))),")")
           ; end-if
       if(rest_total(e,1) < (min_rest(e)-1)*(sum(v in VESSELS)(depart(e,v,1))))
           then
              rest_total(e,1) := (min_rest(e)-1)*(sum(v in VESSELS)(depart(e,v,1)
                  ))                         !;if(e = "C-28") then writeln("
                  rest_total(",e,",1) < (",min_rest(e),"-1)*",(sum(v in VESSELS)(
                  depart(e,v,1)))," => reset, so rest_total(",e,",1) := (",
                  min_rest(e),"-1)*",(sum(v in VESSELS)(depart(e,v,1)))); end-if
       end-if
       forall(t in 2..WEEKS_TO_PLAN) do
              rest_total(e,t) := rest_total(e,(t-1)) - (1-(sum(r in ALL_ROLES)(
                  allocate(e,r,t))))    !;if(e = "C-28") then writeln("rest_total
                  (",e,",",t,") := ",rest_total(e,(t-1))," - (1-",(sum(r in
                  ALL_ROLES)(allocate(e,r,t))),")"); end-if
              if(rest_total(e,t) < (min_rest(e)-1)*(sum(v in VESSELS)(depart(e,v,
                  t)))) then
                     rest_total(e,t) := (min_rest(e)-1)*(sum(v in VESSELS)(depart
                         (e,v,t)))                 !;if(e = "C-28") then writeln
                         ("rest_total(",e,",",t,") < (",min_rest(e),"-1)*",(sum(v
                          in VESSELS)(depart(e,v,t)))," => reset, so rest_total
                         (",e,",",t,") := (",min_rest(e),"-1)*",(sum(v in VESSELS
                         )(depart(e,v,t)))); end-if
              end-if
       end-do
end-do
!writeln("\t'rest_total' values calculated")
!writeln


!writeln("\tChecking Rest vs Work constraints...")
RvWfeas := TRUE
forall(e in REG_EMP) do
       if(min_rest(e)*(1-(sum(r in ALL_ROLES)(allocate(e,r,1)))) < rest_zero(e))
           then
              RvWfeas := FALSE
!             writeln("\tINFEASIBILITY detected - Rest vs Work constraint (",e
    ,",1)\t",min_rest(e)*(1-(sum(r in ALL_ROLES)(allocate(e,r,1))))," < ",
    rest_zero(e))
       end-if
```

```
        forall(t in 2..WEEKS_TO_PLAN) do
                if(min_rest(e)*(1-(sum(r in ALL_ROLES)(allocate(e,r,t)))) <
                    rest_total(e,(t-1))) then
                        RvWfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Rest vs Work constraint
    (",e,",",t,")\t",min_rest(e)*(1-(sum(r in ALL_ROLES)(allocate(e,r,t)))),"<
    ",rest_total(e,(t-1)))
!                       writeln("\tINFEASIBILITY detected - Rest vs Work constraint
    (",e,",",t,")\t",min_rest(e),"*(1-",(sum(r in ALL_ROLES)(allocate(e,r,t))),")
     < ",rest_total(e,(t-1)))
                end-if
        end-do
end-do
if(RvWfeas = FALSE) then
        feasible := FALSE
!else
!       writeln("\tRest vs Work constraints feasible")
end-if
!writeln


!writeln("\tChecking Variable Linking constraints...")
Linkfeas := TRUE
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
        if(cur_allocate(e,r,t) = 0) then
                if(chng_allocate(e,r,t) <> allocate(e,r,t)) then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of allocate
    variables (unspecified)")
                end-if
        else
                if(chng_allocate(e,r,t) <> cur_allocate(e,r,t) - allocate(e,r,t))
                    then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of allocate
    variables (unspecified)")
                end-if
        end-if
end-do
forall(e in REG_EMP, v in VESSELS, t in TIME) do
        if(cur_board(e,v,t) = 0) then
                if(chng_board(e,v,t) <> board(e,v,t)) then
                        Linkfeas := FALSE
```

```
!                       writeln("\tINFEASIBILITY detected - Link of board variables
    (unspecified)")
                end-if
        else
                if(chng_board(e,v,t) <> cur_board(e,v,t) - board(e,v,t)) then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of board variables
    (unspecified)")
                end-if
        end-if
end-do
forall(e in REG_EMP, v in VESSELS, t in TIME) do
        if(cur_depart(e,v,t) = 0) then
                if(chng_depart(e,v,t) <> depart(e,v,t)) then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of depart variables
     (unspecified)")
                end-if
        else
                if(chng_depart(e,v,t) <> cur_depart(e,v,t) - depart(e,v,t)) then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of depart variables
     (unspecified)")
                end-if
        end-if
end-do
forall(r in ALL_ROLES, t in TIME) do
        if(cur_ag_rboard(r,t) = 0) then
                if(chng_ag_rboard(r,t) <> ag_rboard(r,t)) then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of AG board
    variables (unspecified)")
                end-if
        else
                if(chng_ag_rboard(r,t) <> cur_ag_rboard(r,t) - ag_rboard(r,t)) then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of AG board
    variables (unspecified)")
                end-if
        end-if
end-do
forall(r in ALL_ROLES, t in TIME) do
        if(cur_ag_rdepart(r,t) = 0) then
```

```
                if(chng_ag_rdepart(r,t) <> ag_rdepart(r,t)) then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of AG depart
    variables (unspecified)")
                end-if
        else
                if(chng_ag_rdepart(r,t) <> cur_ag_rdepart(r,t) - ag_rdepart(r,t))
                    then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of AG depart
    variables (unspecified)")
                end-if
        end-if
end-do
forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME) do
        if(cur_long_work(l,e,r,t) = 0) then
                if(chng_long_work(l,e,r,t) <> long_work(l,e,r,t)) then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of long work
    variables (unspecified)")
                end-if
        else
                if(chng_long_work(l,e,r,t) <> cur_long_work(l,e,r,t) - long_work(l,
                    e,r,t)) then
                        Linkfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Link of long work
    variables (unspecified)")
                end-if
        end-if
end-do
forall(e in GUARANTEED) do
        if(chng_undertime(e) <> undertime(e) - cur_undertime(e)) then
                Linkfeas := FALSE
!               writeln("\tINFEASIBILITY detected - Link of undertime variables (
    unspecified)")
        end-if
        if(chng_overtime(e) <> overtime(e) - cur_overtime(e)) then
                Linkfeas := FALSE
!               writeln("\tINFEASIBILITY detected - Link of overtime variables (
    unspecified)")
        end-if
end-do
if(Linkfeas = FALSE) then
```

```
       feasible := FALSE
!else
!      writeln("\tVariable Linking constraints feasible")
end-if
!writeln


!writeln("\tChecking Status of Variables...")
Statfeas := TRUE
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
       if(allocate(e,r,t) <> 0 and allocate(e,r,t) <> 1) then
              Statfeas := FALSE
       end-if
       if(chng_allocate(e,r,t) <> 0 and chng_allocate(e,r,t) <> 1) then
              Statfeas := FALSE
       end-if
end-do
forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME) do
       if(long_work(l,e,r,t) <> 0 and long_work(l,e,r,t) <> 1) then
              Statfeas := FALSE
       end-if
       if(chng_long_work(l,e,r,t) <> 0 and chng_long_work(l,e,r,t) <> 1) then
              Statfeas := FALSE
       end-if
end-do
forall(e in REG_EMP, v in VESSELS, t in TIME) do
       if(board(e,v,t) <> 0 and board(e,v,t) <> 1) then
              Statfeas := FALSE
       end-if
       if(chng_board(e,v,t) <> 0 and chng_board(e,v,t) <> 1) then
              Statfeas := FALSE
       end-if
       if(depart(e,v,t) <> 0 and depart(e,v,t) <> 1) then
              Statfeas := FALSE
       end-if
       if(chng_depart(e,v,t) <> 0 and chng_depart(e,v,t) <> 1) then
              Statfeas := FALSE
       end-if
end-do
forall(r in ALL_ROLES, t in TIME) do
       if(ag_rboard(r,t) <> 0 and ag_rboard(r,t) <> 1) then
              Statfeas := FALSE
       end-if
```

```
        if(chng_ag_rboard(r,t) <> 0 and chng_ag_rboard(r,t) <> 1) then
               Statfeas := FALSE
        end-if
        if(chng_ag_rdepart(r,t) <> 0 and chng_ag_rdepart(r,t) <> 1) then
               Statfeas := FALSE
        end-if
        if(ag_rdepart(r,t) <> 0 and ag_rdepart(r,t) <> 1) then
               Statfeas := FALSE
        end-if
end-do
forall(e in GUARANTEED) do
        if(undertime(e) < 0) then
               Statfeas := FALSE
        end-if
        if(overtime(e) < 0) then
               Statfeas := FALSE
        end-if
end-do
forall(e in REG_EMP, t in TIME) do
        if(work_total(e,t) < 0) then
               Statfeas := FALSE
        end-if
        if(rest_total(e,t) < 0) then
               Statfeas := FALSE
        end-if
end-do
if(Statfeas = FALSE) then
        feasible := FALSE
end-if


if(round(1000*TW_COST) = round(1000*getobjval)) then
        equal_OFs := TRUE
else
        equal_OFs := FALSE
end-if


prog_endtime := gettime


!-------------------------------------------------------------------------
!-------------------------------------------------------------------------
```

```
!-------------------------------------------------------------------------
! Print out details into results files...

! Calculate some stats...
TW_a := 0                                  ! number of agency-covered tasks in
    the initial schedule
TW_u := 0                                  ! number of un-covered tasks in the
    initial schedule
TW_changes_to_reg := 0        ! number of changes to regular employees in the (
    best) solution
TW_changes_to_AG := 0         ! number of changes to agency employees in the (best
    ) solution
TW_number_of_AG := 0          ! number of times agency employees are utilised in
    the (best) solution
forall(r in ALL_ROLES, t in TIME) do
        x := 0
        forall(e in REG_EMP) do
                if(cur_allocate(e,r,t) = 1) then
                        x := x + 1
                end-if
                TW_changes_to_reg := TW_changes_to_reg + chng_allocate(e,r,t)
        end-do
        if(cur_allocate("AGENCY",r,t) = 1) then
                x := x + 1
                TW_a := TW_a + 1
        end-if
        TW_changes_to_AG := TW_changes_to_AG + chng_allocate("AGENCY",r,t)
        TW_number_of_AG := TW_number_of_AG + allocate("AGENCY",r,t)
        if(x = 0) then TW_u := TW_u + 1
        end-if
end-do


TB_a := 0                                  ! number of tasks covered by agency in
     initial schedule
TB_u := 0                                  ! number of un-covered tasks in the
    initial schedule
TB_changes_to_reg := 0        ! number of changes to regular employees in the (
    best) solution
TB_changes_to_AG := 0         ! number of changes to agency employees in the (best
    ) solution
TB_number_of_AG := 0          ! number of times agency employees are utilised in
    the (best) solution
```

```
forall(j in JOB) do
        x := 0
        forall(i in EMP) do
                if(initial(i,j) = 1) then
                        x := x + 1
                end-if
                TB_changes_to_reg := TB_changes_to_reg + round(getsol(change(i,j)))
        end-do
        if(AG_initial(j) = 1) then
                x := x + 1
                TB_a := TB_a + 1
        end-if
        TB_changes_to_AG := TB_changes_to_AG+ round(getsol(AG_change(j)))
        TB_number_of_AG := TB_number_of_AG+ round(getsol(AG_new_sched(j)))

        if(x = 0) then
                TB_u := TB_u + 1
        end-if
end-do


convert_1 := split_time_1 - prog_starttime
soln_time := split_time_2 - split_time_1
convert_2 := prog_endtime - split_time_2
total_time := prog_endtime - prog_starttime


fopen(SUMMARYFILE, F_APPEND)
        write(InstanceName)
        write("\t",convert_1,"\t",soln_time,"\t",convert_2,"\t",total_time)
        write("\t",(sum(r in ALL_ROLES, t in TIME) required(r,t)),"\t",TW_u,"\t",
            TW_a)
        write("\t",(sum(e in ALL_EMP) pre_assign_cost(e)),"\t",JOBS_TO_PLAN,"\t",
            TB_u,"\t",TB_a)
        write("\t",status(getprobstat),"\t ",getobjval,"\t ",getparam("
            XPRS_bestbound"))
        write("\t",TB_changes_to_reg,"\t",TB_changes_to_AG,"\t",TB_number_of_AG)
        write("\t",feasible,"\t",TW_COST,"\t",equal_OFs)
        write("\t",TW_changes_to_reg,"\t",TW_changes_to_AG,"\t",TW_number_of_AG)
        write("\n")
fclose(F_APPEND)
```

```
! Print solution to a solution file, so it can be used as an initial solution for
     the heuristic procudure
fopen(SOLUTIONFILE, F_OUTPUT)
       writeln("!Solution for ",InstanceName)
       writeln("!\tfound using a Task-Based approximation of the the Time-Windows
            problem")
       writeln("!\twritten in representation proposed for carrying out heuristics
           .")
       writeln
       writeln("!TB prob status: ",status(getprobstat))
       writeln("!Feasible for TW? ",feasible)
       writeln("!Total run time: ",total_time)
       writeln("!TB soln value: ",getobjval)
       writeln("!TW soln value: ",TW_COST)
       writeln
       write("!Roles are given in the following order:")
       forall(r in ALL_ROLES) write("\t",r)
       write("\n")
       writeln
       write("! Time: ")
       forall(r in ALL_ROLES) do
              forall(t in TIME) write(t,"\t")
       end-do
       write("\n")
       writeln("taskbased_sol: [")
       forall(e in REG_EMP) do
              forall(r in ALL_ROLES) do
                     forall(t in TIME) do
                            if(allocate(e,r,t) > 0.9) then
                                   write("\t1")
                            else
                                   write("\t0")
                            end-if
                     end-do
              end-do
              write("\n")
       end-do
       forall(r in ALL_ROLES) do
              forall(t in TIME) do
                     if(allocate("AGENCY",r,t) > 0.9) then
                            write("\t1")
                     else
```

```
                        write("\t0")
                    end-if
                end-do
            end-do
            write(" ]\n")
        fclose(F_OUTPUT)


        fopen(COLLATEDSOLUTIONS, F_OUTPUT)
            writeln("!Solution for ",InstanceName)
            writeln("!\tfound using a Task-Based approximation of the the Time-Windows
                 problem")
            writeln("!\twritten in representation proposed for carrying out heuristics
                .")
            writeln
            writeln("!TB prob status: ",status(getprobstat))
            writeln("!Feasible for TW? ",feasible)
            writeln("!Total run time: ",total_time)
            writeln("!TB soln value: ",getobjval)
            writeln("!TW soln value: ",TW_COST)
            writeln
            write("!Roles are given in the following order:")
            forall(r in ALL_ROLES) write(" \t ",r)
            write("\n")
            writeln
            write("! Time: ")
            forall(r in ALL_ROLES) do
                forall(t in TIME) write(t,"\t")
            end-do
            write("\n")
            writeln("taskbased_sol: [")
            forall(e in REG_EMP) do
                forall(r in ALL_ROLES) do
                    forall(t in TIME) do
                        if(allocate(e,r,t) > 0.9) then
                            write("\t1")
                        else
                            write("\t0")
                        end-if
                    end-do
                end-do
                write("\n")
            end-do
        forall(r in ALL_ROLES) do
                forall(t in TIME) do
                        if(allocate("AGENCY",r,t) > 0.9) then
                                write("\t1")
                        else
                                write("\t0")
                        end-if
                end-do
        end-do
        write(" ]\n")
fclose(F_OUTPUT)

end-model
```

E.2.5 Heuristic algorithm

Here we give the code used to improve existing solutions for the Time-Windows problem using the Heuristic algorithm described in 6.5.5. This represents only the formal test version of the algorithm, which was translated into C++ by fellow PhD student Seda Sucu based on the heuristic code developed in FICO Xpress. The preliminary versions and development process are discussed in section 6.5.3; however for simplicity we do not give details of the preliminary Xpress implementations of the Heuristic algorithm here.

The C++ code is divided into several sub-programme files. We first give the main programme of the algorithm, followed by the sub-programme files, and finally the 'header' files which are required by these.

E.2.5.1 Main programme

```
#include "Load_Data.h"
#include "trial.h"
#include "finish.h"
#include "initialise.h"
#include "feasibility_checking.h"
#include "cost.h"
#include "transferring_data.h"
#include "sort_listing.h"
#include "find_usables.h"
#include "tabu.h"
#include "swap_change_update.h"
#include "comparing.h"
#include "rand_kick.h"
```

```
#include "swapping.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <string>
#include <cstdlib>
#include <ctime>
#include <math.h>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <iterator>
#include <vector>
#include <random>
#include <sstream>
using namespace std;
int same_best[51];
int required[13][25];
int eligible_gecici[325][49];
int eligible[13][25][49];
int starting[25][49];
int cur_allocate[13][25][49];
int current_alloc[325][49];
int current_board[325][48];
int cur_board[13][25][48];
int current_depart[325][48];
int cur_depart[13][25][48];
int cur_ag_rboard[13][25];
int cur_ag_rdepart[13][25];
float board_cost[325][48];
float board_chng_cost[13][25][48];
float depart_cost[325][48];
float depart_chng_cost[13][25][48];
float ag_board_chng_cost[13][25];
float ag_depart_chng_cost[13][25];
float workchange_cost[325][49];
float work_chng_cost[13][25][49];
int initial_solution[326][49];
int taskbased_sol[13][25][49];
int contract[45][4];
int under_rate[45];
int over_rate[45];
int g_weeks[45];
int exp_worktime[45];
int crew[48][4];
int work_zero[48];
int rest_zero[48];
int min_rest[48];
int max_work[48];
int ag_crew[25][3];
int ag_work_zero[25];
int ag_max_work[25];
int ag_starting[25];
float ext_chg_cost[325][490];
float ext_chg_cost_new[13][25][490];
float extension_chng_cost[13][25][49][10];
int guaranteed_workers[45][2];
int cur_undertime[45];
int cur_overtime[45];
int list_sol[8][13][48];
int best_sols[51][13][48];
int swaps_examined[48][48];
int ag_list_sol[8][13][25];
int ag_best_sols[51][13][25];
float total_cost[8];
float emp_cost[8][48];
int allocate_sol[8][13][25][49];
int cur_long_work[13][25][49][10];
int current_long_work[325][490];
int cur_long_work_new[13][25][490];
int chng_long_work[13][25][49][10];;
int long_work_sol[8][13][25][49][10];
int board_sol[8][13][25][48];
int depart_sol[8][13][25][48];
int undertime_sol[8][45];
int overtime_sol[8][45];
int chng_undertime[45];
int chng_overtime[45];
int chng_allocate[13][25][49];
int chng_board[13][25][48];
int chng_depart[13][25][48];
float ag_cost[8][13][25];
int ag_crewchange[8][13][25];
int ag_rboard_sol[8][13][25];
int ag_rdepart_sol[8][13][25];
int poss_ag_rboard[13];
int poss_ag_rdepart[13];
int poss_chng_ag_rboard[13];
int poss_chng_ag_rdepart[13];
int poss_ag_long_work[13][10];
int poss_chng_ag_long_work[13][10];
int chng_ag_rboard [13][25];
int chng_ag_rdepart [13][25];
int work_total[13][48];
int ag_work_total[13][25];
int rest_total[13][49];
int last_changed[48];
int order_number[48];
int current_list[13];
int roles[25];
int reserve_list_bkwd[13][25];
int reserve_list_fwd[13][25];
int tabu_sol[13][48];
int ag_tabu_sol[13][25];
int swappable_emp[48];
int problem_set[43];

int main()
{
string ext = ".txt";
string filename1, filename2, filename3, filename4, filename5;
int set_no, accept_rule, kick_rule, order_setting, no_enter,
    initial_solution_type, fraction_settings;
double short_list_fraction;
problem_set[  0      ]=     16     ;
problem_set[  1      ]=     18     ;
problem_set[  2      ]=     27     ;
problem_set[  3      ]=     32     ;
problem_set[  4      ]=     37     ;
problem_set[  5      ]=     46     ;
problem_set[  6      ]=     55     ;
problem_set[  7      ]=     63     ;
problem_set[  8      ]=     65     ;
problem_set[  9      ]=     71     ;
problem_set[  10     ]=     74     ;
problem_set[  11     ]=     79     ;
problem_set[  12     ]=     80     ;
problem_set[  13     ]=     85     ;
```

```
problem_set[  14     ]=     89     ;
problem_set[  15     ]=     91     ;
problem_set[  16     ]=     100    ;
problem_set[  17     ]=     103    ;
problem_set[  18     ]=     109    ;
problem_set[  19     ]=     112    ;
problem_set[  20     ]=     126    ;
problem_set[  21     ]=     129    ;
problem_set[  22     ]=     138    ;
problem_set[  23     ]=     147    ;
problem_set[  24     ]=     148    ;
problem_set[  25     ]=     149    ;
problem_set[  26     ]=     157    ;
problem_set[  27     ]=     159    ;
problem_set[  28     ]=     170    ;
problem_set[  29     ]=     174    ;
problem_set[  30     ]=     178    ;
problem_set[  31     ]=     182    ;
problem_set[  32     ]=     187    ;
problem_set[  33     ]=     188    ;
problem_set[  34     ]=     197    ;
problem_set[  35     ]=     208    ;
problem_set[  36     ]=     216    ;
problem_set[  37     ]=     218    ;
problem_set[  38     ]=     219    ;
problem_set[  39     ]=     223    ;
problem_set[  40     ]=     226    ;
problem_set[  41     ]=     227    ;
problem_set[  42     ]=     229    ;
problem_set[  43     ]=     237    ;
int wrong_data;
initial_solution_type=0;
accept_rule=0;
kick_rule=0;
order_setting=0;
fraction_settings=0;
stringstream sk;
sk<<1;
string st3 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Results\\All_Sets\\
    Summary_of_results_All_Sets";
filename3= st3 + sk.str() + ext;
ofstream results(filename3);
string st5 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Results\\One_Min\\
    Summary_of_results_All_Sets_After_1_Min";
filename5=st5 + sk.str() + ext;
ofstream results_1min(filename5);
results<< "Set Number"<<"    "<<"CPU Time"<<"      "<<"Iteration"<<"     "<<"
    Initial Cost"<<"  "<<"accept_rule"<<"   "<<"kick_rule"<<"     "<<"order_setting
    "<<" "<<" initial_solution_type"<<" "<<"fraction_settings"<<" "<<"no_bkwd"<<"
     "<<"no_fwd"<<" "<< "no_swap"<<"    "<<"no_cand"<<" "<<"no_kick"<<" "<<"
    no_infeas"<<"  "<<"no_tabu"<<" "<<"total_cost(best) "<<"  "<<"total_cost(
    current) "<<" "<<"best_sol_time"<<" "<<"number_best"<<"  "<<"changes_to_reg
    "<<" "<<"changes_to_AG "<<" "<<"number_of_AG"<<'\n';
results<<'\n';
results_1min<< "Set Number"<<" "<<"CPU Time"<<"    "<<"Iteration"<<"     "<<"
    Initial Cost"<<"  "<<"accept_rule"<<"   "<<"kick_rule"<<"     "<<"order_setting
    "<<" "<<" initial_solution_type"<<" "<<"fraction_settings"<<" "<<"no_bkwd"<<"
     "<<"no_fwd"<<" "<< "no_swap"<<"    "<<"no_cand"<<" "<<"no_kick"<<" "<<"
    no_infeas"<<"  "<<"no_tabu"<<" "<<"total_cost(best) "<<"  "<<"total_cost(
    current) "<<" "<<"best_sol_time"<<" "<<"number_best"<<"  "<<"changes_to_reg
    "<<" "<<"changes_to_AG "<<" "<<"number_of_AG"<<'\n';
results_1min<<'\n';
for (fraction_settings=1; fraction_settings<2; fraction_settings++) {
if(fraction_settings==0) {
short_list_fraction=(1.0/3.0);
}
else if(fraction_settings==1) {
short_list_fraction=1;
}
for(accept_rule=1;accept_rule<2; accept_rule++) {
for(kick_rule=2; kick_rule<3;kick_rule++) {
for(order_setting=0;order_setting<1; order_setting++) {
//ofstream results(filename3);
//results<<"Set Number"<<" "<<"Initial Cost"<<" "<<"accept_rule"<<" "<<"kick_rule
    "<<" "<<"order_setting"<<" "<<" initial_solution_type"<<" "<<"
    fraction_settings"<<" "<<"no_bkwd"<<" "<<"no_fwd"<<" "<< "no_swap"<<" "<<"
    no_cand"<<" "<<"no_kick"<<" "<<"no_infeas"<<" "<<"no_tabu"<<" "<<"total_cost(
    best) "<<" "<<"total_cost(current) "<<" "<<"best_sol_time"<<" "<<"number_best
    "<<" "<<"changes_to_reg"<<" "<<"changes_to_AG "<<'\n';
for(wrong_data=0; wrong_data<=43; wrong_data++) {
set_no=problem_set[wrong_data];
stringstream ss;
stringstream sm;
ss << set_no;
sm<< (fraction_settings*12)+(accept_rule*6)+(kick_rule*2)+(order_setting+1);
if(set_no<=9) {
```

```
string st1 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Time_Windows\\Time-
    Windows - Captains - Set R00";
filename1= st1 + ss.str()+ ext;
string st2 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Task_Based\\TBApprox
    soln - Set R00";
filename2= st2 + ss.str() + ext;
string st4 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Results\\Set_by_Set\\
    Cost_at_each_iteration_Set R00";
string st1a = "_Version";
filename4= st4 + ss.str() + st1a +sm.str() + ext;
}
if(set_no>=10 && set_no<=99) {
string st1 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Time_Windows\\Time-
    Windows - Captains - Set R0";
filename1= st1 + ss.str() + ext;
string st2 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Task_Based\\TBApprox
    soln - Set R0";
filename2= st2 + ss.str() + ext;
string st4 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Results\\Set_by_Set\\
    Cost_at_each_iteration_Set R0";
string st1a = "_Version";
filename4= st4 + ss.str() + st1a +sm.str() + ext;
}
if(set_no>=100 && set_no<=240) {
string st1 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Time_Windows\\Time-
    Windows - Captains - Set R";
filename1= st1 + ss.str() + ext;
string st2 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Task_Based\\TBApprox
    soln - Set R";
filename2= st2 + ss.str() + ext;
string st4 = "C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\Results\\Set_by_Set\\
    Cost_at_each_iteration_Set R";
string st1a = "_Version";
filename4= st4 + ss.str() + st1a +sm.str() + ext;
}
int week_last, role_last, emp_last;
int final_count, changes_to_reg , changes_to_AG, number_of_AG;
int rol_10, tabu, add_to_emp,prev_work,in_reserve_bkwd,in_reserve_fwd,
    swap_block_latest, new_block, block_start, block_end, block_len, block_found,
     task_extend, emp_extend,vessel_extend,emp_13, emp_usable,candidate_exist,
    update_done, change_no,min_emp, min_no,order_rule,Statfeas, Linkfeas,RvWfeas,
    AGLWfeas,LWfeas,AGBDfeas,UTfeas, OTfeas, Dprtfeas,Brdfeas,feasible;
int swap_task_extend, all_rest, swap_vessel, swap_new_block, weeks_10,
    swap_allowed, too_early, swap_emp, swap_task, swap_block_start,
    swap_block_end, swap_block_len, swap_find_time, swap_block_found,ag_swappable
    , kick_task, random_length, random_time, kick_emp, random_task, random_emp,
    kick_end, kick_start, kick_feas,sum_start_rand, new_best, emp_count,
    role_count,summation_1, conflict_found_fwd,summation,extend_len_fwd,
    to_check_tabu, length_count, rest_count, conflict_found_bkwd, extend_len_bkwd
    , do_extend_bkwd, max_bkwd_extend, max_fwd_extend, do_extend_fwd, do_swap,
    link_to_vessel,weeks_14, rol_14, swap_block_earliest, OLfeas, JCfeas,
    transfer_sol_from, transfer_sol_to,iteration, overall_regular_max_work,
    overall_agency_max_work,overall_max_work, lambda, reg_emp, all_emp,
    guaranteed, vessels, weeks_to_plan, all_roles, best_index, last_kick_time,
    added_t, to_calculate,consec_work, feas_crewchange, no_fwd,no_bkwd, no_swap,
    no_cand, no_kick, no_tabu, no_infeas, terminate, no_nonreduce,best_sol_time,
    number_best;
int no_AGLWfeas,no_LWfeas,no_UTfeas,no_OTfeas,no_JCfeas,no_OLfeas,no_Statfeas,
    no_Linkfeas,no_RvWfeas, no_Brdfeas,no_Dprtfeas,no_AGBDfeas,count_20, boyut_20
    , emp_10;
//end of declaration
float initial_cost, extend_cost_fwd, extend_cost_bkwd, swapping_cost,
    candidate_cost, random ;
double time;
//declaring vectors
vector<int> emps_changed;
vector<int> ag_roles_changed;
vector<int> evaluate_crewchange;
vector<int> definite_crewchange;
vector<int> possible_crewchange;
vector<int>ordered_list;
vector<int>short_ordered_list;
vector<int>added_set;
vector<int>candidate_emps;
vector<int>emps_to_update;
//end of declaration
FILE *initial;
//FILE *results;
//FILE *results1;
//FILE *results2;
initial=fopen("C:\\Users\\xmb13210\\Desktop\\Problem_Sets\\initial_values.txt","r
    ");
fscanf(initial, "%d %d %d %d %d %d", &reg_emp, &all_emp, &guaranteed, &vessels, &
    weeks_to_plan, &all_roles);
deneme(filename1,ag_max_work,ag_work_zero,under_rate, over_rate, g_weeks,
    exp_worktime, required, eligible, starting, ag_starting, work_zero, rest_zero
    , min_rest, max_work, cur_allocate);
deneme2(filename1,cur_board, cur_depart, cur_ag_rboard, cur_ag_rdepart);
deneme3(filename1,cur_undertime, cur_overtime, current_long_work,
    cur_long_work_new,cur_long_work);
deneme4(filename1,board_cost, board_chng_cost, depart_cost, depart_chng_cost,
    ag_board_chng_cost, ag_depart_chng_cost);
deneme5(filename1,workchange_cost, work_chng_cost,ext_chg_cost, ext_chg_cost_new,
    extension_chng_cost);
deneme6(filename2,initial_solution,taskbased_sol);
ofstream results_alter(filename4);
results_alter<< "Set Number: "<<set_no<<"\n";
results_alter<<"accept_rule: "<<accept_rule<<'\n';
results_alter<<"kick_rule: "<<kick_rule<<'\n';
results_alter<<"order_setting: "<<order_setting<<'\n';
results_alter<<"initial_solution_type: "<<initial_solution_type<<'\n';
results_alter<<"fraction_settings: "<<short_list_fraction<<'\n';
results_alter<<"Iteration"<<" "<<"Current Cost"<<" "<< "Best Cost"<< '\n';
for (int i=0; i<25;i++) {
roles[i]=i;
}
clock_t start, end;
start = clock();
no_AGLWfeas=0;
no_LWfeas=0;
no_UTfeas=0;
no_OTfeas=0;
no_JCfeas=0;
no_OLfeas=0;
no_Statfeas=0;
no_Linkfeas=0;
no_RvWfeas=0;
no_Brdfeas=0;
no_Dprtfeas=0;
no_AGBDfeas=0;
random_time=0;
random_length=0;
kick_task=0;
random_task=0;
kick_emp=0;
random_emp=0;
random=0;
kick_start=0;
kick_end=0;
```

```
kick_feas=0;
sum_start_rand=0;
new_best=0;
emp_count=0;
role_count=0;
overall_regular_max_work=-1;
overall_agency_max_work=-1;
overall_max_work=-1;
lambda=-1;
best_index=51;
added_t=0;
feas_crewchange=1;
iteration=0;
feasible=1;
JCfeas=1;
OLfeas=1;
Brdfeas=1;
Dprtfeas=1;
UTfeas=1;
OTfeas=1;
AGBDfeas=1;
AGLWfeas=1;
LWfeas=1;
RvWfeas=1;
Linkfeas=1;
Statfeas=1;
no_fwd=0;
no_bkwd=0;
no_swap=0;
no_cand=0;
no_kick=0;
no_tabu=0;
no_infeas=0;
terminate=0;
iteration=0;
no_nonreduce=0;
best_sol_time=0;
number_best=1;
last_kick_time=0;
min_emp=0;
min_no=0;
change_no=0;
update_done=0;
```

```
candidate_exist=0;
vessel_extend=-1;
swap_block_earliest=-1;
swap_block_latest=-1;
task_extend=-1;
block_start=-1;
block_end=-1;
block_len=0;
emp_extend=-1;
new_block=0;
do_extend_bkwd=0;
max_bkwd_extend=0;
max_fwd_extend=0;
do_extend_fwd=0;
do_swap=0;
extend_cost_fwd=0;
extend_cost_bkwd=0;
swapping_cost=0;
extend_len_bkwd=0;
extend_len_fwd=0;
to_check_tabu=0;
prev_work=0;
add_to_emp=0;
tabu=0;
summation=0;
summation_1=0;
//accept_rule=0;
initialise_first(overall_regular_max_work, max_work, overall_agency_max_work,
    ag_max_work, overall_max_work,lambda, reg_emp, all_roles); //calculating
    lambda
initialise_second(weeks_to_plan, list_sol, best_index, best_sols, added_t,
    swaps_examined, emps_changed, reg_emp, all_roles, taskbased_sol); //
    emps_changed
initialise_third(weeks_to_plan, ag_list_sol, best_index, ag_best_sols,
    ag_roles_changed, reg_emp, all_roles, taskbased_sol, required);//
    ag_roles_changed
to_calculate=0;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
```

```
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
initial_cost=total_cost[0];
transfer_sol_from=0;
transfer_sol_to=1;
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
initialise_four(weeks_to_plan, ag_best_sols, reg_emp, all_roles, best_sols,
    list_sol, ag_list_sol, last_changed);
no_enter=0;
while(terminate==0) {
iteration=iteration+1;
if(order_setting==0) {
if(no_nonreduce >= 1) {
order_rule=0;
}
else if((iteration - last_kick_time > 1) || (iteration > 1 && last_kick_time== 0)
    ) {
order_rule= 1;
}
else {
order_rule=2;
} }
else {
order_rule=2;
}
sort_employee_list(short_list_fraction, change_no, last_changed, min_emp,min_no,
    order_rule, order_number, reg_emp, ordered_list, short_ordered_list,
    added_set, iteration);
find_usable_1(update_done,candidate_exist,candidate_emps);
for(emp_usable=0;emp_usable<short_ordered_list.size() ;emp_usable++)
//for(emp_usable=0;emp_usable<48 ;emp_usable++)
{
```

```
emp_13=short_ordered_list[emp_usable];
//emp_13=emp_usable;
find_usable_2( weeks_to_plan,vessel_extend,emp_13,swap_block_earliest,update_done
    , emp_extend, task_extend, block_start,block_end, block_len, block_found,
    vessels, starting, work_zero, current_list, list_sol);
for(weeks_14=0;weeks_14<weeks_to_plan;weeks_14++) {
if(update_done==0) {
rol_14=current_list[weeks_14]; // AL - Have changed this from "rol_14=
    current_list[emp_13]"
new_block=0;
if(block_found==0 && rol_14!=-1)    // AL - Have changed this from "... &&
    current_list[weeks_14]!=-1"
{
find_usable_3(roles,vessels, weeks_14, rol_14, current_list, new_block,
    block_found, emp_extend,emp_13, task_extend, block_start, swap_block_earliest
    ,rest_zero, vessel_extend);
}
else if(block_found==1 && rol_14!=-1 && rol_14!=task_extend) {
find_usable_4(link_to_vessel, vessel_extend, rol_14, roles, task_extend);
if(link_to_vessel!=1 )//error
{
find_usable_5(block_end, weeks_14,swap_block_latest, block_len, block_start);
if( task_extend!=-1) {
evaluate_block_new1(required, rest_zero, block_end, task_extend, emp_extend,
    eligible, weeks_to_plan, extend_cost_fwd, extend_cost_bkwd, swapping_cost,
    do_extend_bkwd, do_extend_fwd, do_swap, block_len, max_work, max_bkwd_extend,
    max_fwd_extend);
if(max_bkwd_extend > 0) {
extend_len_bkwd=max_bkwd_extend;
while(do_extend_bkwd==0 && extend_len_bkwd>0) {
evaluate_block_new2( all_roles, reg_emp, weeks_to_plan, max_work,
    conflict_found_bkwd, reserve_list_bkwd, in_reserve_bkwd, block_end,
    task_extend, work_zero, length_count, emp_extend, rest_count, extend_len_bkwd
    , emps_changed, ag_roles_changed, list_sol, ag_list_sol, min_rest);
evaluate_block_new3(to_check_tabu,all_roles, reg_emp, weeks_to_plan, max_work,
    prev_work, add_to_emp, conflict_found_bkwd, reserve_list_bkwd,
    in_reserve_bkwd, block_end, task_extend, work_zero, length_count, emp_extend,
     rest_count, extend_len_bkwd, emps_changed, ag_roles_changed, list_sol,
    ag_list_sol, eligible, rest_zero, min_rest);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
```

```
}
else {
to_calculate=3;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new4(extend_cost_bkwd, total_cost, do_extend_bkwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[3]<total_cost[accept_rule])  {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new5( emps_changed,candidate_emps, extend_cost_bkwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_bkwd==0) {
extend_len_bkwd=extend_len_bkwd-1;
```

```
} } }
if(do_extend_bkwd==0) {
evaluate_block_new6(extend_len_fwd, max_fwd_extend, block_start, rest_zero,
    emp_extend, summation, vessels, roles, task_extend, starting, min_rest,
    eligible, required);
if(max_fwd_extend > 0) {
while(do_extend_fwd==0 && extend_len_fwd>0) {
evaluate_block_new7(extend_len_fwd, emps_changed, ag_roles_changed, reg_emp,
    weeks_to_plan, list_sol, all_roles, ag_list_sol, reserve_list_fwd, emp_extend
    ,rest_count, in_reserve_fwd,task_extend, min_rest,conflict_found_fwd,
    block_start);
evaluate_block_new8(rest_zero,starting,eligible, max_work,length_count,
    to_check_tabu,summation_1,summation, prev_work,add_to_emp,extend_len_fwd,
    emps_changed,ag_roles_changed, reg_emp, weeks_to_plan, list_sol, all_roles,
    ag_list_sol, reserve_list_fwd, emp_extend, rest_count,in_reserve_fwd,
    task_extend, min_rest,conflict_found_fwd,block_start);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=4;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new9(extend_cost_fwd, total_cost, do_extend_fwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[4]<total_cost[accept_rule])  {
```

```
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new10( emps_changed,candidate_emps, extend_cost_fwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_fwd==0) {
extend_len_fwd=extend_len_fwd-1;
} } } }
if(do_extend_bkwd==0 && do_extend_fwd==0 ) {
if(block_start>=0) {
evaluate_swap1(ag_swappable, block_start, block_end, eligible, task_extend);
if(ag_swappable==1) {
for(rol_10=0;rol_10< all_roles;rol_10++) {
if(rol_10!=task_extend) {
evaluate_swap2( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_find_time,
    swap_block_found);
for(weeks_10=0;weeks_10<weeks_to_plan;weeks_10++) {
if(do_swap==0) {
evaluate_swap3( weeks_10, rol_10,swap_block_start, swap_task, swap_emp,
    swap_allowed, eligible , min_rest, starting, emp_extend, all_roles, summation
    , summation_1, too_early, swap_new_block, swap_block_found, ag_list_sol,
    swap_block_latest, swap_block_earliest);
if(swap_block_found ==1 && ag_list_sol[0][weeks_10][rol_10]==1) {
if(ag_crewchange[0][weeks_10][rol_10]==1) {
swap_block_end=weeks_10-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap4( weeks_10, swap_block_found, swap_block_len, swap_block_end,
    swap_allowed, eligible, min_rest, too_early, emp_extend, swap_block_earliest,
     starting, summation, summation_1, all_roles, swap_block_latest, do_swap,
    swap_task,swap_block_start, rol_10);
}
else {
evaluate_swap5(swap_allowed, emp_extend, rol_10, weeks_10, swap_block_latest,
    swap_block_found, eligible);
} }
else if(swap_block_found==1 && ag_list_sol[0][weeks_10][rol_10]==0) //25june
{
swap_block_end=weeks_10-1;
swap_block_len=(swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend])
{       swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan,
    list_sol, all_roles,ag_list_sol, emp_extend, block_start, swap_block_start,
    min_rest, swap_block_end, swap_task, block_end, swap_emp,task_extend,
    to_check_tabu );
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
```

```
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap6( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_block_found );
}
evaluate_swap7( swap_new_block, swap_block_found);
if((weeks_10==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap8( weeks_10, rol_10,swap_block_end, swap_block_start, swap_block_len
    , swap_allowed, eligible,emp_extend);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
```

```
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }     } } } } } } }
if(do_swap==0) {
evaluate_swap9( swap_emp, swap_task, swap_block_start, block_start,
    swap_block_end, block_end );
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
```

```
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
```

```
//part 2 ev_swap
evaluate_swap10( min_rest, work_zero, starting, all_roles, summation, summation_1
    , eligible, task_extend, block_end, rest_zero, block_start, max_work,
    block_len, reg_emp, swappable_emp, emp_extend);
// void evaluate_swap10(int min_rest_1[48], int work_zero_1[48],int starting_1
    [25][49],int& weeks_10x,int all_roles_1,int summation_1, int summation_1x,int
     eligible_1[13][25][48],int& task_extend_1,int& block_end_1,int rest_zero_1
    [48],int& block_start_1,int max_work_1[48],int& block_len_1, int reg_emp_1,
    int swappable_emp_1[48], int& emp_extend_1 )
//part 2 ev_swap
boyut_20=ordered_list.size();
for(count_20=0;count_20<boyut_20; count_20++)
//  for(emp_10=0;emp_10<reg_emp; emp_10++)
{
emp_10=ordered_list[count_20];
if(swappable_emp[emp_10]==1 && swaps_examined[emp_extend][emp_10]!=1)  // AL -
    Have changed this from "if(swappable_emp[emp_10]==0 &&..."
{
evaluate_swap11(swap_find_time, all_rest,swap_allowed, too_early,
    swap_block_found, swap_block_len, do_swap, swap_emp, swap_vessel, swap_task,
    swap_block_start, swap_block_end);
while(do_swap==0 && swap_find_time<weeks_to_plan) {
evaluate_swap12(swap_vessel,roles, swap_block_start, swap_emp, swap_task,
    swap_allowed, eligible, min_rest, emp_extend, starting, all_roles, summation,
     summation_1,too_early, swap_block_found,swap_new_block, all_rest,rol_10,
    emp_10, list_sol, swap_block_earliest,swap_find_time, swap_block_latest);
if(swap_block_found==1 && rol_10!=-1) {
if(rol_10 !=swap_task) {
link_to_vessel=0;
evaluate_swap13( vessel_extend, roles,swap_task, swap_allowed, eligible, rol_10,
    emp_extend, swap_vessel, link_to_vessel, swap_find_time, swap_block_latest,
    swap_block_found);
if(link_to_vessel==-1) {
swap_block_end= swap_find_time-1;
swap_block_len = (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed ==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
```

```
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap14( swap_block_found, swap_block_len, swap_block_end, swap_vessel,
    swap_allowed,eligible , min_rest, starting, emp_extend, roles, all_roles,
    summation, summation_1, too_early, swap_find_time, swap_block_latest, do_swap
    , swap_task_extend, rol_10, swap_block_start, swap_block_earliest);
} }
else {
evaluate_swap15(rol_10, swap_find_time, swap_block_latest, swap_block_found,
    swap_allowed, eligible, emp_extend);
} }
else if(swap_block_found==1 && rol_10==-1) {
swap_block_end=swap_find_time-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }  } }
evaluate_swap16( swap_emp, swap_task, swap_vessel, swap_block_found, do_swap,
    too_early, swap_allowed, swap_block_start, swap_block_end, swap_block_len);
}
evaluate_swap17( swap_new_block, swap_block_found);
if((swap_find_time==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap18(swap_block_start, swap_block_len, swap_block_end, swap_find_time,
     swap_allowed, eligible, swap_find_time, rol_10);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }
swap_find_time++;
}
if(do_swap == 0 && all_rest==1) {
evaluate_swap19( swap_emp, emp_10, swap_task, swap_block_start, block_start,
    swap_block_end, block_end);
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
```

```
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }     } }
evaluate_block_new13(do_swap, no_swap, do_extend_bkwd, transfer_sol_from, no_bkwd
    , do_extend_fwd, no_fwd);
if(do_extend_bkwd==1 || do_extend_fwd==1 || do_swap ==1) {
check_tabu_2(transfer_sol_to, reg_emp,all_roles, weeks_to_plan, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
```

```
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
update_done=1;
compare_to_best(number_best, weeks_to_plan,all_roles, reg_emp, role_count ,
    total_cost, new_best, same_best, emp_count, list_sol, best_sols, ag_list_sol,
     ag_best_sols);
compare_to_best_2(best_sol_time,iteration, ag_best_sols,best_sols, number_best,
    transfer_sol_from, transfer_sol_to, total_cost, all_emp, reg_emp, all_roles,
    weeks_to_plan, lambda, allocate_sol, long_work_sol, emp_cost, list_sol,
    board_sol, depart_sol, undertime_sol, overtime_sol,ag_cost, ag_list_sol,
    ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
} }
else {
if(block_len< max_work[emp_13]) {
if(vessel_extend!=-1) {
task_extend=vessel_extend;
evaluate_block_new1(required, rest_zero, block_end, task_extend, emp_extend,
    eligible, weeks_to_plan, extend_cost_fwd, extend_cost_bkwd, swapping_cost,
    do_extend_bkwd, do_extend_fwd, do_swap, block_len, max_work, max_bkwd_extend,
    max_fwd_extend);
if(max_bkwd_extend > 0) {
extend_len_bkwd=max_bkwd_extend;
while(do_extend_bkwd==0 && extend_len_bkwd>0) {
evaluate_block_new2( all_roles, reg_emp, weeks_to_plan, max_work,
    conflict_found_bkwd, reserve_list_bkwd, in_reserve_bkwd, block_end,
    task_extend, work_zero, length_count, emp_extend, rest_count, extend_len_bkwd
    , emps_changed, ag_roles_changed, list_sol, ag_list_sol, min_rest);
evaluate_block_new3(to_check_tabu,all_roles, reg_emp, weeks_to_plan, max_work,
    prev_work, add_to_emp, conflict_found_bkwd, reserve_list_bkwd,
    in_reserve_bkwd, block_end, task_extend, work_zero, length_count, emp_extend,
     rest_count, extend_len_bkwd, emps_changed, ag_roles_changed, list_sol,
    ag_list_sol, eligible, rest_zero, min_rest);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=3;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
```

```
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
```

```
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new4(extend_cost_bkwd, total_cost, do_extend_bkwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[3]<total_cost[accept_rule])  {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new5( emps_changed,candidate_emps, extend_cost_bkwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_bkwd==0) {
extend_len_bkwd=extend_len_bkwd-1;
} } }
if(do_extend_bkwd==0) {
evaluate_block_new6(extend_len_fwd, max_fwd_extend, block_start, rest_zero,
    emp_extend, summation, vessels, roles, task_extend, starting, min_rest,
    eligible, required);
```

```
if(max_fwd_extend > 0) {
while(do_extend_fwd==0 && extend_len_fwd>0) {
evaluate_block_new7(extend_len_fwd, emps_changed, ag_roles_changed, reg_emp,
    weeks_to_plan, list_sol, all_roles, ag_list_sol, reserve_list_fwd, emp_extend
    ,rest_count, in_reserve_fwd,task_extend, min_rest,conflict_found_fwd,
    block_start);
evaluate_block_new8(rest_zero,starting,eligible, max_work,length_count,
    to_check_tabu,summation_1,summation, prev_work,add_to_emp,extend_len_fwd,
    emps_changed,ag_roles_changed, reg_emp, weeks_to_plan, list_sol, all_roles,
    ag_list_sol, reserve_list_fwd, emp_extend, rest_count,in_reserve_fwd,
    task_extend, min_rest,conflict_found_fwd,block_start);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=4;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new9(extend_cost_fwd, total_cost, do_extend_fwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[4]<total_cost[accept_rule])   {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new10( emps_changed,candidate_emps, extend_cost_fwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_fwd==0) {
extend_len_fwd=extend_len_fwd-1;
} } } }
if(do_extend_bkwd==0 && do_extend_fwd==0 ) {
if(block_start>=0) {
evaluate_swap1(ag_swappable, block_start, block_end, eligible, task_extend);
if(ag_swappable==1) {
for(rol_10=0;rol_10< all_roles;rol_10++) {
if(rol_10!=task_extend) {
evaluate_swap2( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_find_time,
    swap_block_found);
for(weeks_10=0;weeks_10<weeks_to_plan;weeks_10++) {
if(do_swap==0) {
evaluate_swap3( weeks_10, rol_10,swap_block_start, swap_task, swap_emp,
    swap_allowed, eligible , min_rest, starting, emp_extend, all_roles, summation
    , summation_1, too_early, swap_new_block, swap_block_found, ag_list_sol,
    swap_block_latest, swap_block_earliest);
if(swap_block_found ==1 && ag_list_sol[0][weeks_10][rol_10]==1) {
if(ag_crewchange[0][weeks_10][rol_10]==1) {
swap_block_end=weeks_10-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
```

```
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap4( weeks_10, swap_block_found, swap_block_len, swap_block_end,
    swap_allowed, eligible, min_rest, too_early, emp_extend, swap_block_earliest,
     starting, summation, summation_1, all_roles, swap_block_latest, do_swap,
    swap_task,swap_block_start, rol_10);
}
else {
evaluate_swap5(swap_allowed, emp_extend, rol_10, weeks_10, swap_block_latest,
    swap_block_found, eligible);
} }
else if(swap_block_found==1 && ag_list_sol[0][weeks_10][rol_10]!=1) {
swap_block_end=weeks_10-1;
swap_block_len=(swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend])
{       swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan,
    list_sol, all_roles,ag_list_sol, emp_extend, block_start, swap_block_start,
    min_rest, swap_block_end, swap_task, block_end, swap_emp,task_extend,
    to_check_tabu );
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
```

```
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap6( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_block_found );
}
evaluate_swap7( swap_new_block, swap_block_found);
if((weeks_10==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap8( weeks_10, rol_10,swap_block_end, swap_block_start, swap_block_len
    , swap_allowed, eligible,emp_extend);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
```

```
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } } } } } } }
if(do_swap==0) {
evaluate_swap9( swap_emp, swap_task, swap_block_start, block_start,
    swap_block_end, block_end );
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
```

```
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
//part 2 ev_swap
evaluate_swap10( min_rest, work_zero, starting, all_roles, summation, summation_1
    , eligible, task_extend, block_end, rest_zero, block_start, max_work,
    block_len, reg_emp, swappable_emp, emp_extend);
// void evaluate_swap10(int min_rest_1[48], int work_zero_1[48],int starting_1
    [25][49],int& weeks_10x,int all_roles_1,int summation_1, int summation_1x,int
```

```
     eligible_1[13][25][48],int& task_extend_1,int& block_end_1,int rest_zero_1
    [48],int& block_start_1,int max_work_1[48],int& block_len_1, int reg_emp_1,
    int swappable_emp_1[48], int& emp_extend_1 )
//part 2 ev_swap
boyut_20=ordered_list.size();
//for(emp_10=0;emp_10<48; emp_10++)
for(count_20=0;count_20<boyut_20; count_20++) {
emp_10=ordered_list[count_20];
if(swappable_emp[emp_10]==1 && swaps_examined[emp_extend][emp_10]!=1)  // AL -
    Have changed this from "if(swappable_emp[emp_10]==0 &&..."
{
evaluate_swap11(swap_find_time, all_rest,swap_allowed, too_early,
    swap_block_found, swap_block_len, do_swap, swap_emp, swap_vessel, swap_task,
    swap_block_start, swap_block_end);
while(do_swap==0 && swap_find_time<weeks_to_plan) {
evaluate_swap12(swap_vessel,roles, swap_block_start, swap_emp, swap_task,
    swap_allowed, eligible, min_rest, emp_extend, starting, all_roles, summation,
     summation_1,too_early, swap_block_found,swap_new_block, all_rest,rol_10,
    emp_10, list_sol, swap_block_earliest,swap_find_time, swap_block_latest);
if(swap_block_found==1 && rol_10!=-1) {
if(rol_10 !=swap_task) {
link_to_vessel=0;
evaluate_swap13( vessel_extend, roles,swap_task, swap_allowed, eligible, rol_10,
    emp_extend, swap_vessel, link_to_vessel, swap_find_time, swap_block_latest,
    swap_block_found);
if(link_to_vessel==-1) {
swap_block_end= swap_find_time-1;
swap_block_len = (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed ==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
```

```
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap14( swap_block_found, swap_block_len, swap_block_end, swap_vessel,
    swap_allowed,eligible , min_rest, starting, emp_extend, roles, all_roles,
    summation, summation_1, too_early, swap_find_time, swap_block_latest, do_swap
    , swap_task_extend, rol_10, swap_block_start, swap_block_earliest);
} }
else {
evaluate_swap15(rol_10, swap_find_time, swap_block_latest, swap_block_found,
    swap_allowed, eligible, emp_extend);
} }
```

```
else if(swap_block_found==1 && rol_10==-1) {
swap_block_end=swap_find_time-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap16( swap_emp, swap_task, swap_vessel, swap_block_found, do_swap,
    too_early, swap_allowed, swap_block_start, swap_block_end, swap_block_len);
}
evaluate_swap17( swap_new_block, swap_block_found);
if((swap_find_time==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap18(swap_block_start, swap_block_len, swap_block_end, swap_find_time,
     swap_allowed, eligible, swap_find_time, rol_10);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
```

```
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }
swap_find_time++;
}
if(do_swap == 0 && all_rest==1) {
evaluate_swap19( swap_emp, emp_10, swap_task, swap_block_start, block_start,
    swap_block_end, block_end);
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }     } }
evaluate_block_new13(do_swap, no_swap,do_extend_bkwd, transfer_sol_from, no_bkwd,
     do_extend_fwd, no_fwd);
if(do_extend_bkwd==1 || do_extend_fwd==1|| do_swap ==1) {
check_tabu_2(transfer_sol_to, reg_emp,all_roles, weeks_to_plan, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
update_done=1;
compare_to_best(number_best, weeks_to_plan,all_roles, reg_emp, role_count ,
    total_cost, new_best, same_best, emp_count, list_sol, best_sols, ag_list_sol,
     ag_best_sols);
compare_to_best_2(best_sol_time,iteration, ag_best_sols,best_sols, number_best,
    transfer_sol_from, transfer_sol_to, total_cost, all_emp, reg_emp, all_roles,
    weeks_to_plan, lambda, allocate_sol, long_work_sol, emp_cost, list_sol,
    board_sol, depart_sol, undertime_sol, overtime_sol,ag_cost, ag_list_sol,
    ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
} } } }
//11 mayis
evaluate_block_new12(task_extend, rol_14, block_start, weeks_14,
    swap_block_earliest, all_roles, roles, vessel_extend,block_end,block_len);
} }
else if(block_found==1 && rol_14 ==-1) {
find_usable_5(block_end, weeks_14,swap_block_latest, block_len, block_start);
if(task_extend!=-1)  // AL - Have changed this from "if(task_extend==-1)"
{
evaluate_block_new1(required, rest_zero, block_end, task_extend, emp_extend,
    eligible, weeks_to_plan, extend_cost_fwd, extend_cost_bkwd, swapping_cost,
    do_extend_bkwd, do_extend_fwd, do_swap, block_len, max_work, max_bkwd_extend,
    max_fwd_extend);
if(max_bkwd_extend > 0) {
extend_len_bkwd=max_bkwd_extend;
while(do_extend_bkwd==0 && extend_len_bkwd>0) {
evaluate_block_new2( all_roles, reg_emp, weeks_to_plan, max_work,
    conflict_found_bkwd, reserve_list_bkwd, in_reserve_bkwd, block_end,
    task_extend, work_zero, length_count, emp_extend, rest_count, extend_len_bkwd
    , emps_changed, ag_roles_changed, list_sol, ag_list_sol, min_rest);
evaluate_block_new3(to_check_tabu,all_roles, reg_emp, weeks_to_plan, max_work,
    prev_work, add_to_emp, conflict_found_bkwd, reserve_list_bkwd,
    in_reserve_bkwd, block_end, task_extend, work_zero, length_count, emp_extend,
     rest_count, extend_len_bkwd, emps_changed, ag_roles_changed, list_sol,
    ag_list_sol, eligible, rest_zero, min_rest);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=3;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new4(extend_cost_bkwd, total_cost, do_extend_bkwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[3]<total_cost[accept_rule])  {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new5( emps_changed,candidate_emps, extend_cost_bkwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_bkwd==0) {
extend_len_bkwd=extend_len_bkwd-1;
} } }
if(do_extend_bkwd==0) {
evaluate_block_new6(extend_len_fwd, max_fwd_extend, block_start, rest_zero,
    emp_extend, summation, vessels, roles, task_extend, starting, min_rest,
    eligible, required);
if(max_fwd_extend > 0) {
while(do_extend_fwd==0 && extend_len_fwd>0) {
evaluate_block_new7(extend_len_fwd, emps_changed, ag_roles_changed, reg_emp,
    weeks_to_plan, list_sol, all_roles, ag_list_sol, reserve_list_fwd, emp_extend
    ,rest_count, in_reserve_fwd,task_extend, min_rest,conflict_found_fwd,
    block_start);
evaluate_block_new8(rest_zero,starting,eligible, max_work,length_count,
    to_check_tabu,summation_1,summation, prev_work,add_to_emp,extend_len_fwd,
    emps_changed,ag_roles_changed, reg_emp, weeks_to_plan, list_sol, all_roles,
    ag_list_sol, reserve_list_fwd, emp_extend, rest_count,in_reserve_fwd,
    task_extend, min_rest,conflict_found_fwd,block_start);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=4;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
```

```
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new9(extend_cost_fwd, total_cost, do_extend_fwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[4]<total_cost[accept_rule])   {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new10( emps_changed,candidate_emps, extend_cost_fwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_fwd==0) {
extend_len_fwd=extend_len_fwd-1;
} } } }
if(do_extend_bkwd==0 && do_extend_fwd==0 ) {
if(block_start>=0) {
evaluate_swap1(ag_swappable, block_start, block_end, eligible, task_extend);
if(ag_swappable==1) {
for(rol_10=0;rol_10< all_roles;rol_10++) {
if(rol_10!=task_extend) {
evaluate_swap2( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_find_time,
    swap_block_found);
for(weeks_10=0;weeks_10<weeks_to_plan;weeks_10++) {
if(do_swap==0) {
evaluate_swap3( weeks_10, rol_10,swap_block_start, swap_task, swap_emp,
    swap_allowed, eligible , min_rest, starting, emp_extend, all_roles, summation
    , summation_1, too_early, swap_new_block, swap_block_found, ag_list_sol,
    swap_block_latest, swap_block_earliest);
if(swap_block_found ==1 && ag_list_sol[0][weeks_10][rol_10]==1) {
if(ag_crewchange[0][weeks_10][rol_10]==1) {
swap_block_end=weeks_10-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
```

```
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap4( weeks_10, swap_block_found, swap_block_len, swap_block_end,
    swap_allowed, eligible, min_rest, too_early, emp_extend, swap_block_earliest,
     starting, summation, summation_1, all_roles, swap_block_latest, do_swap,
    swap_task,swap_block_start, rol_10);
}
```

```
else {
evaluate_swap5(swap_allowed, emp_extend, rol_10, weeks_10, swap_block_latest,
    swap_block_found, eligible);
} }
else if(swap_block_found==1 && ag_list_sol[0][weeks_10][rol_10]!=1) {
swap_block_end=weeks_10-1;
swap_block_len=(swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend])
{       swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan,
    list_sol, all_roles,ag_list_sol, emp_extend, block_start, swap_block_start,
    min_rest, swap_block_end, swap_task, block_end, swap_emp,task_extend,
    to_check_tabu );
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
```

```
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap6( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_block_found );
}
evaluate_swap7( swap_new_block, swap_block_found);
if((weeks_10==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap8( weeks_10, rol_10,swap_block_end, swap_block_start, swap_block_len
    , swap_allowed, eligible,emp_extend);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } } } } } } }
if(do_swap==0) {
evaluate_swap9( swap_emp, swap_task, swap_block_start, block_start,
    swap_block_end, block_end );
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
```

```
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
//part 2 ev_swap
evaluate_swap10( min_rest, work_zero, starting, all_roles, summation, summation_1
    , eligible, task_extend, block_end, rest_zero, block_start, max_work,
    block_len, reg_emp, swappable_emp, emp_extend);
// void evaluate_swap10(int min_rest_1[48], int work_zero_1[48],int starting_1
    [25][49],int& weeks_10x,int all_roles_1,int summation_1, int summation_1x,int
     eligible_1[13][25][48],int& task_extend_1,int& block_end_1,int rest_zero_1
    [48],int& block_start_1,int max_work_1[48],int& block_len_1, int reg_emp_1,
    int swappable_emp_1[48], int& emp_extend_1 )
```

```
//part 2 ev_swap
boyut_20=ordered_list.size();
//for(emp_10=0;emp_10<48; emp_10++)
for(count_20=0;count_20<boyut_20; count_20++) {
emp_10=ordered_list[count_20];
if(swappable_emp[emp_10]==1 && swaps_examined[emp_extend][emp_10]!=1)  // AL -
    Have changed this from "if(swappable_emp[emp_10]==0 &&..."
{
evaluate_swap11(swap_find_time, all_rest,swap_allowed, too_early,
    swap_block_found, swap_block_len, do_swap, swap_emp, swap_vessel, swap_task,
    swap_block_start, swap_block_end);
while(do_swap==0 && swap_find_time<weeks_to_plan) {
evaluate_swap12(swap_vessel,roles, swap_block_start, swap_emp, swap_task,
    swap_allowed, eligible, min_rest, emp_extend, starting, all_roles, summation,
     summation_1,too_early, swap_block_found,swap_new_block, all_rest,rol_10,
    emp_10, list_sol, swap_block_earliest,swap_find_time, swap_block_latest);
if(swap_block_found==1 && rol_10!=-1) {
if(rol_10 !=swap_task) {
link_to_vessel=0;
evaluate_swap13( vessel_extend, roles,swap_task, swap_allowed, eligible, rol_10,
    emp_extend, swap_vessel, link_to_vessel, swap_find_time, swap_block_latest,
    swap_block_found);
if(link_to_vessel==-1) {
swap_block_end= swap_find_time-1;
swap_block_len = (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed ==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap14( swap_block_found, swap_block_len, swap_block_end, swap_vessel,
    swap_allowed,eligible , min_rest, starting, emp_extend, roles, all_roles,
    summation, summation_1, too_early, swap_find_time, swap_block_latest, do_swap
    , swap_task_extend, rol_10, swap_block_start, swap_block_earliest);
} }
else {
evaluate_swap15(rol_10, swap_find_time, swap_block_latest, swap_block_found,
    swap_allowed, eligible, emp_extend);
} }
else if(swap_block_found==1 && rol_10==-1) {
swap_block_end=swap_find_time-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
```

```
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
```

```
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
```

```
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap16( swap_emp, swap_task, swap_vessel, swap_block_found, do_swap,
    too_early, swap_allowed, swap_block_start, swap_block_end, swap_block_len);
}
evaluate_swap17( swap_new_block, swap_block_found);
if((swap_find_time==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap18(swap_block_start, swap_block_len, swap_block_end, swap_find_time,
     swap_allowed, eligible, swap_find_time, rol_10);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }
swap_find_time++;
}
if(do_swap == 0 && all_rest==1) {
evaluate_swap19( swap_emp, emp_10, swap_task, swap_block_start, block_start,
    swap_block_end, block_end);
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
```

```
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }     } }
evaluate_block_new13(do_swap, no_swap,do_extend_bkwd, transfer_sol_from, no_bkwd,
     do_extend_fwd, no_fwd);
if(do_extend_bkwd==1 || do_extend_fwd==1 || do_swap==1) {
check_tabu_2(transfer_sol_to, reg_emp,all_roles, weeks_to_plan, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
update_done=1;
compare_to_best(number_best, weeks_to_plan,all_roles, reg_emp, role_count ,
    total_cost, new_best, same_best, emp_count, list_sol, best_sols, ag_list_sol,
     ag_best_sols);
compare_to_best_2(best_sol_time,iteration, ag_best_sols,best_sols, number_best,
    transfer_sol_from, transfer_sol_to, total_cost, all_emp, reg_emp, all_roles,
    weeks_to_plan, lambda, allocate_sol, long_work_sol, emp_cost, list_sol,
    board_sol, depart_sol, undertime_sol, overtime_sol,ag_cost, ag_list_sol,
    ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
} }
else {
if(block_len <max_work[emp_13]) {
if(vessel_extend!=-1) {
task_extend=roles[vessel_extend];
evaluate_block_new1(required, rest_zero, block_end, task_extend, emp_extend,
    eligible, weeks_to_plan, extend_cost_fwd, extend_cost_bkwd, swapping_cost,
    do_extend_bkwd, do_extend_fwd, do_swap, block_len, max_work, max_bkwd_extend,
    max_fwd_extend);
if(max_bkwd_extend > 0) {
extend_len_bkwd=max_bkwd_extend;
while(do_extend_bkwd==0 && extend_len_bkwd>0) {
evaluate_block_new2( all_roles, reg_emp, weeks_to_plan, max_work,
    conflict_found_bkwd, reserve_list_bkwd, in_reserve_bkwd, block_end,
    task_extend, work_zero, length_count, emp_extend, rest_count, extend_len_bkwd
    , emps_changed, ag_roles_changed, list_sol, ag_list_sol, min_rest);
evaluate_block_new3(to_check_tabu,all_roles, reg_emp, weeks_to_plan, max_work,
    prev_work, add_to_emp, conflict_found_bkwd, reserve_list_bkwd,
    in_reserve_bkwd, block_end, task_extend, work_zero, length_count, emp_extend,
     rest_count, extend_len_bkwd, emps_changed, ag_roles_changed, list_sol,
    ag_list_sol, eligible, rest_zero, min_rest);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=3;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
```

```
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new4(extend_cost_bkwd, total_cost, do_extend_bkwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[3]<total_cost[accept_rule])   {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new5( emps_changed,candidate_emps, extend_cost_bkwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_bkwd==0) {
extend_len_bkwd=extend_len_bkwd-1;
} } }
if(do_extend_bkwd==0) {
evaluate_block_new6(extend_len_fwd, max_fwd_extend, block_start, rest_zero,
    emp_extend, summation, vessels, roles, task_extend, starting, min_rest,
    eligible, required);
if(max_fwd_extend > 0) {
while(do_extend_fwd==0 && extend_len_fwd>0) {
evaluate_block_new7(extend_len_fwd, emps_changed, ag_roles_changed, reg_emp,
    weeks_to_plan, list_sol, all_roles, ag_list_sol, reserve_list_fwd, emp_extend
    ,rest_count, in_reserve_fwd,task_extend, min_rest,conflict_found_fwd,
    block_start);
evaluate_block_new8(rest_zero,starting,eligible, max_work,length_count,
    to_check_tabu,summation_1,summation, prev_work,add_to_emp,extend_len_fwd,
    emps_changed,ag_roles_changed, reg_emp, weeks_to_plan, list_sol, all_roles,
    ag_list_sol, reserve_list_fwd, emp_extend, rest_count,in_reserve_fwd,
    task_extend, min_rest,conflict_found_fwd,block_start);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=4;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new9(extend_cost_fwd, total_cost, do_extend_fwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[4]<total_cost[accept_rule])  {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new10( emps_changed,candidate_emps, extend_cost_fwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_fwd==0) {
extend_len_fwd=extend_len_fwd-1;
} } } }
if(do_extend_bkwd==0 && do_extend_fwd==0 ) {
if(block_start>=0) {
evaluate_swap1(ag_swappable, block_start, block_end, eligible, task_extend);
if(ag_swappable==1) {
for(rol_10=0;rol_10< all_roles;rol_10++) {
if(rol_10!=task_extend) {
evaluate_swap2( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_find_time,
    swap_block_found);
for(weeks_10=0;weeks_10<weeks_to_plan;weeks_10++) {
if(do_swap==0) {
evaluate_swap3( weeks_10, rol_10,swap_block_start, swap_task, swap_emp,
    swap_allowed, eligible , min_rest, starting, emp_extend, all_roles, summation
    , summation_1, too_early, swap_new_block, swap_block_found, ag_list_sol,
    swap_block_latest, swap_block_earliest);
if(swap_block_found ==1 && ag_list_sol[0][weeks_10][rol_10]==1) {
if(ag_crewchange[0][weeks_10][rol_10]==1) {
swap_block_end=weeks_10-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap4( weeks_10, swap_block_found, swap_block_len, swap_block_end,
    swap_allowed, eligible, min_rest, too_early, emp_extend, swap_block_earliest,
     starting, summation, summation_1, all_roles, swap_block_latest, do_swap,
    swap_task,swap_block_start, rol_10);
}
else {
evaluate_swap5(swap_allowed, emp_extend, rol_10, weeks_10, swap_block_latest,
    swap_block_found, eligible);
} }
else if(swap_block_found==1 && ag_list_sol[0][weeks_10][rol_10]!=1)
{
swap_block_end=weeks_10-1;
```

```
swap_block_len=(swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend])
{      swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan,
    list_sol, all_roles,ag_list_sol, emp_extend, block_start, swap_block_start,
    min_rest, swap_block_end, swap_task, block_end, swap_emp,task_extend,
    to_check_tabu );
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
```

```
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap6( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_block_found );
}
evaluate_swap7( swap_new_block, swap_block_found);
if((weeks_10==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap8( weeks_10, rol_10,swap_block_end, swap_block_start, swap_block_len
    , swap_allowed, eligible,emp_extend);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } } } } } }
if(do_swap==0) {
evaluate_swap9( swap_emp, swap_task, swap_block_start, block_start,
    swap_block_end, block_end );
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
```

```
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
//part 2 ev_swap
evaluate_swap10( min_rest, work_zero, starting, all_roles, summation, summation_1
    , eligible, task_extend, block_end, rest_zero, block_start, max_work,
    block_len, reg_emp, swappable_emp, emp_extend);
// void evaluate_swap10(int min_rest_1[48], int work_zero_1[48],int starting_1
    [25][49],int& weeks_10x,int all_roles_1,int summation_1, int summation_1x,int
     eligible_1[13][25][48],int& task_extend_1,int& block_end_1,int rest_zero_1
    [48],int& block_start_1,int max_work_1[48],int& block_len_1, int reg_emp_1,
    int swappable_emp_1[48], int& emp_extend_1 )
//part 2 ev_swap
boyut_20=ordered_list.size();
//for(emp_10=0;emp_10<48; emp_10++)
for(count_20=0;count_20<boyut_20; count_20++) {
emp_10=ordered_list[count_20];
if(swappable_emp[emp_10]==1 && swaps_examined[emp_extend][emp_10]!=1)  // AL -
```

```
    Have changed this from "if(swappable_emp[emp_10]==0 &&..."
{
evaluate_swap11(swap_find_time, all_rest,swap_allowed, too_early,
    swap_block_found, swap_block_len, do_swap, swap_emp, swap_vessel, swap_task,
    swap_block_start, swap_block_end);
while(do_swap==0 && swap_find_time<weeks_to_plan) {
evaluate_swap12(swap_vessel,roles, swap_block_start, swap_emp, swap_task,
    swap_allowed, eligible, min_rest, emp_extend, starting, all_roles, summation,
     summation_1,too_early, swap_block_found,swap_new_block, all_rest,rol_10,
    emp_10, list_sol, swap_block_earliest,swap_find_time, swap_block_latest);
if(swap_block_found==1 && rol_10!=-1) {
if(rol_10 !=swap_task) {
link_to_vessel=0;
evaluate_swap13( vessel_extend, roles,swap_task, swap_allowed, eligible, rol_10,
    emp_extend, swap_vessel, link_to_vessel, swap_find_time, swap_block_latest,
    swap_block_found);
if(link_to_vessel==-1) {
swap_block_end= swap_find_time-1;
swap_block_len = (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed ==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap14( swap_block_found, swap_block_len, swap_block_end, swap_vessel,
    swap_allowed,eligible , min_rest, starting, emp_extend, roles, all_roles,
    summation, summation_1, too_early, swap_find_time, swap_block_latest, do_swap
    , swap_task_extend, rol_10, swap_block_start, swap_block_earliest);
} }
else {
evaluate_swap15(rol_10, swap_find_time, swap_block_latest, swap_block_found,
    swap_allowed, eligible, emp_extend);
} }
else if(swap_block_found==1 && rol_10==-1) {
swap_block_end=swap_find_time-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
```

```
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
```

```
evaluate_swap16( swap_emp, swap_task, swap_vessel, swap_block_found, do_swap,
    too_early, swap_allowed, swap_block_start, swap_block_end, swap_block_len);
}
evaluate_swap17( swap_new_block, swap_block_found);
if((swap_find_time==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap18(swap_block_start, swap_block_len, swap_block_end, swap_find_time,
     swap_allowed, eligible, swap_find_time, rol_10);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }
swap_find_time++;
}
if(do_swap == 0 && all_rest==1) {
evaluate_swap19( swap_emp, emp_10, swap_task, swap_block_start, block_start,
    swap_block_end, block_end);
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
```

```
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } } } }
evaluate_block_new13(do_swap, no_swap,do_extend_bkwd, transfer_sol_from, no_bkwd,
     do_extend_fwd, no_fwd);
if(do_extend_bkwd==1 || do_extend_fwd==1 || do_swap ==1) {
check_tabu_2(transfer_sol_to, reg_emp,all_roles, weeks_to_plan, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
update_done=1;
compare_to_best(number_best, weeks_to_plan,all_roles, reg_emp, role_count ,
    total_cost, new_best, same_best, emp_count, list_sol, best_sols, ag_list_sol,
     ag_best_sols);
compare_to_best_2(best_sol_time,iteration, ag_best_sols,best_sols, number_best,
    transfer_sol_from, transfer_sol_to, total_cost, all_emp, reg_emp, all_roles,
    weeks_to_plan, lambda, allocate_sol, long_work_sol, emp_cost, list_sol,
    board_sol, depart_sol, undertime_sol, overtime_sol,ag_cost, ag_list_sol,
    ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
} } } }
find_usables_6(block_found, block_start, block_end, block_len, emp_extend,
    task_extend, vessel_extend);
```

```
}
find_usables_7(new_block,block_found);
if((weeks_14==(weeks_to_plan-1)) && block_found==1) {
find_usables_8(block_end, weeks_14, swap_block_latest, block_len, block_start);
if(task_extend!=-1) {
evaluate_block_new1(required, rest_zero, block_end, task_extend, emp_extend,
    eligible, weeks_to_plan, extend_cost_fwd, extend_cost_bkwd, swapping_cost,
    do_extend_bkwd, do_extend_fwd, do_swap, block_len, max_work, max_bkwd_extend,
    max_fwd_extend);
if(max_bkwd_extend > 0) {
extend_len_bkwd=max_bkwd_extend;
while(do_extend_bkwd==0 && extend_len_bkwd>0) {
evaluate_block_new2( all_roles, reg_emp, weeks_to_plan, max_work,
    conflict_found_bkwd, reserve_list_bkwd, in_reserve_bkwd, block_end,
    task_extend, work_zero, length_count, emp_extend, rest_count, extend_len_bkwd
    , emps_changed, ag_roles_changed, list_sol, ag_list_sol, min_rest);
evaluate_block_new3(to_check_tabu,all_roles, reg_emp, weeks_to_plan, max_work,
    prev_work, add_to_emp, conflict_found_bkwd, reserve_list_bkwd,
    in_reserve_bkwd, block_end, task_extend, work_zero, length_count, emp_extend,
     rest_count, extend_len_bkwd, emps_changed, ag_roles_changed, list_sol,
    ag_list_sol, eligible, rest_zero, min_rest);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=3;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new4(extend_cost_bkwd, total_cost, do_extend_bkwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[3]<total_cost[accept_rule])  {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new5( emps_changed,candidate_emps, extend_cost_bkwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_bkwd==0) {
extend_len_bkwd=extend_len_bkwd-1;
} } }
if(do_extend_bkwd==0) {
evaluate_block_new6(extend_len_fwd, max_fwd_extend, block_start, rest_zero,
    emp_extend, summation, vessels, roles, task_extend, starting, min_rest,
    eligible, required);
if(max_fwd_extend > 0) {
while(do_extend_fwd==0 && extend_len_fwd>0) {
evaluate_block_new7(extend_len_fwd, emps_changed, ag_roles_changed, reg_emp,
    weeks_to_plan, list_sol, all_roles, ag_list_sol, reserve_list_fwd, emp_extend
    ,rest_count, in_reserve_fwd,task_extend, min_rest,conflict_found_fwd,
    block_start);
evaluate_block_new8(rest_zero,starting,eligible, max_work,length_count,
    to_check_tabu,summation_1,summation, prev_work,add_to_emp,extend_len_fwd,
    emps_changed,ag_roles_changed, reg_emp, weeks_to_plan, list_sol, all_roles,
    ag_list_sol, reserve_list_fwd, emp_extend, rest_count,in_reserve_fwd,
    task_extend, min_rest,conflict_found_fwd,block_start);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
```

```
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=4;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new9(extend_cost_fwd, total_cost, do_extend_fwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[4]<total_cost[accept_rule])  {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new10( emps_changed,candidate_emps, extend_cost_fwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
```

```
if(do_extend_fwd==0) {
extend_len_fwd=extend_len_fwd-1;
} } } }
if(do_extend_bkwd==0 && do_extend_fwd==0 ) {
if(block_start>=0) {
evaluate_swap1(ag_swappable, block_start, block_end, eligible, task_extend);
if(ag_swappable==1) {
for(rol_10=0;rol_10< all_roles;rol_10++) {
if(rol_10!=task_extend) {
evaluate_swap2( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_find_time,
    swap_block_found);
for(weeks_10=0;weeks_10<weeks_to_plan;weeks_10++) {
if(do_swap==0) {
evaluate_swap3( weeks_10, rol_10,swap_block_start, swap_task, swap_emp,
    swap_allowed, eligible , min_rest, starting, emp_extend, all_roles, summation
    , summation_1, too_early, swap_new_block, swap_block_found, ag_list_sol,
    swap_block_latest, swap_block_earliest);
if(swap_block_found ==1 && ag_list_sol[0][weeks_10][rol_10]==1) {
if(ag_crewchange[0][weeks_10][rol_10]==1) {
swap_block_end=weeks_10-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
```

```
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap4( weeks_10, swap_block_found, swap_block_len, swap_block_end,
    swap_allowed, eligible, min_rest, too_early, emp_extend, swap_block_earliest,
     starting, summation, summation_1, all_roles, swap_block_latest, do_swap,
    swap_task,swap_block_start, rol_10);
}
else {
evaluate_swap5(swap_allowed, emp_extend, rol_10, weeks_10, swap_block_latest,
    swap_block_found, eligible);
} }
else if(swap_block_found==1 && ag_list_sol[0][weeks_10][rol_10]!=1) {
swap_block_end=weeks_10-1;
swap_block_len=(swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend])
{      swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan,
    list_sol, all_roles,ag_list_sol, emp_extend, block_start, swap_block_start,
    min_rest, swap_block_end, swap_task, block_end, swap_emp,task_extend,
    to_check_tabu );
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
```

```
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap6( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_block_found );
}
evaluate_swap7( swap_new_block, swap_block_found);
if((weeks_10==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap8( weeks_10, rol_10,swap_block_end, swap_block_start, swap_block_len
    , swap_allowed, eligible,emp_extend);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
```

```
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } } } } } } }
if(do_swap==0) {
evaluate_swap9( swap_emp, swap_task, swap_block_start, block_start,
    swap_block_end, block_end );
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
//part 2 ev_swap
evaluate_swap10( min_rest, work_zero, starting, all_roles, summation, summation_1
    , eligible, task_extend, block_end, rest_zero, block_start, max_work,
    block_len, reg_emp, swappable_emp, emp_extend);
// void evaluate_swap10(int min_rest_1[48], int work_zero_1[48],int starting_1
    [25][49],int& weeks_10x,int all_roles_1,int summation_1, int summation_1x,int
     eligible_1[13][25][48],int& task_extend_1,int& block_end_1,int rest_zero_1
    [48],int& block_start_1,int max_work_1[48],int& block_len_1, int reg_emp_1,
    int swappable_emp_1[48], int& emp_extend_1 )
//part 2 ev_swap
boyut_20=ordered_list.size();
// for(emp_10=0;emp_10<reg_emp; emp_10++)
for(count_20=0;count_20<boyut_20; count_20++) {
emp_10=ordered_list[count_20];
if(swappable_emp[emp_10]==1 && swaps_examined[emp_extend][emp_10]!=1)  // AL -
    Have changed this from "if(swappable_emp[emp_10]==0 &&..."
{
evaluate_swap11(swap_find_time, all_rest,swap_allowed, too_early,
    swap_block_found, swap_block_len, do_swap, swap_emp, swap_vessel, swap_task,
    swap_block_start, swap_block_end);
while(do_swap==0 && swap_find_time<weeks_to_plan) {
evaluate_swap12(swap_vessel,roles, swap_block_start, swap_emp, swap_task,
    swap_allowed, eligible, min_rest, emp_extend, starting, all_roles, summation,
     summation_1,too_early, swap_block_found,swap_new_block, all_rest,rol_10,
    emp_10, list_sol, swap_block_earliest,swap_find_time, swap_block_latest);
if(swap_block_found==1 && rol_10!=-1) {
if(rol_10 !=swap_task) {
link_to_vessel=0;
evaluate_swap13( vessel_extend, roles,swap_task, swap_allowed, eligible, rol_10,
    emp_extend, swap_vessel, link_to_vessel, swap_find_time, swap_block_latest,
    swap_block_found);
if(link_to_vessel==-1) {
swap_block_end= swap_find_time-1;
swap_block_len = (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed ==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap14( swap_block_found, swap_block_len, swap_block_end, swap_vessel,
    swap_allowed,eligible , min_rest, starting, emp_extend, roles, all_roles,
    summation, summation_1, too_early, swap_find_time, swap_block_latest, do_swap
    , swap_task_extend, rol_10, swap_block_start, swap_block_earliest);
}
else {
evaluate_swap15(rol_10, swap_find_time, swap_block_latest, swap_block_found,
    swap_allowed, eligible, emp_extend);
} }
else if(swap_block_found==1 && rol_10==-1) {
swap_block_end=swap_find_time-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
```

```
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap16( swap_emp, swap_task, swap_vessel, swap_block_found, do_swap,
    too_early, swap_allowed, swap_block_start, swap_block_end, swap_block_len);
}
evaluate_swap17( swap_new_block, swap_block_found);
```

```
if((swap_find_time==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap18(swap_block_start, swap_block_len, swap_block_end, swap_find_time,
     swap_allowed, eligible, swap_find_time, rol_10);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }
swap_find_time++;
}
if(do_swap == 0 && all_rest==1) {
evaluate_swap19( swap_emp, emp_10, swap_task, swap_block_start, block_start,
    swap_block_end, block_end);
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
```

```
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } } } }
evaluate_block_new13(do_swap, no_swap,do_extend_bkwd, transfer_sol_from, no_bkwd,
     do_extend_fwd, no_fwd);
if(do_extend_bkwd==1 || do_extend_fwd==1 || do_swap ==1) {
check_tabu_2(transfer_sol_to, reg_emp,all_roles, weeks_to_plan, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
update_done=1;
compare_to_best(number_best, weeks_to_plan,all_roles, reg_emp, role_count ,
    total_cost, new_best, same_best, emp_count, list_sol, best_sols, ag_list_sol,
     ag_best_sols);
compare_to_best_2(best_sol_time,iteration, ag_best_sols,best_sols, number_best,
    transfer_sol_from, transfer_sol_to, total_cost, all_emp, reg_emp, all_roles,
    weeks_to_plan, lambda, allocate_sol, long_work_sol, emp_cost, list_sol,
    board_sol, depart_sol, undertime_sol, overtime_sol,ag_cost, ag_list_sol,
    ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
} }
else {
if(block_len < max_work[emp_13]) {
if(vessel_extend!=-1) {
task_extend=roles[vessel_extend];
evaluate_block_new1(required, rest_zero, block_end, task_extend, emp_extend,
    eligible, weeks_to_plan, extend_cost_fwd, extend_cost_bkwd, swapping_cost,
    do_extend_bkwd, do_extend_fwd, do_swap, block_len, max_work, max_bkwd_extend,
    max_fwd_extend);
if(max_bkwd_extend > 0) {
extend_len_bkwd=max_bkwd_extend;
while(do_extend_bkwd==0 && extend_len_bkwd>0) {
evaluate_block_new2( all_roles, reg_emp, weeks_to_plan, max_work,
    conflict_found_bkwd, reserve_list_bkwd, in_reserve_bkwd, block_end,
    task_extend, work_zero, length_count, emp_extend, rest_count, extend_len_bkwd
    , emps_changed, ag_roles_changed, list_sol, ag_list_sol, min_rest);
evaluate_block_new3(to_check_tabu,all_roles, reg_emp, weeks_to_plan, max_work,
    prev_work, add_to_emp, conflict_found_bkwd, reserve_list_bkwd,
    in_reserve_bkwd, block_end, task_extend, work_zero, length_count, emp_extend,
     rest_count, extend_len_bkwd, emps_changed, ag_roles_changed, list_sol,
    ag_list_sol, eligible, rest_zero, min_rest);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=3;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
```

```
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new4(extend_cost_bkwd, total_cost, do_extend_bkwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[3]<total_cost[accept_rule])  {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new5( emps_changed,candidate_emps, extend_cost_bkwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_bkwd==0) {
extend_len_bkwd=extend_len_bkwd-1;
} } }
if(do_extend_bkwd==0) {
evaluate_block_new6(extend_len_fwd, max_fwd_extend, block_start, rest_zero,
    emp_extend, summation, vessels, roles, task_extend, starting, min_rest,
    eligible, required);
if(max_fwd_extend > 0) {
while(do_extend_fwd==0 && extend_len_fwd>0) {
evaluate_block_new7(extend_len_fwd, emps_changed, ag_roles_changed, reg_emp,
    weeks_to_plan, list_sol, all_roles, ag_list_sol, reserve_list_fwd, emp_extend
    ,rest_count, in_reserve_fwd,task_extend, min_rest,conflict_found_fwd,
    block_start);
evaluate_block_new8(rest_zero,starting,eligible, max_work,length_count,
    to_check_tabu,summation_1,summation, prev_work,add_to_emp,extend_len_fwd,
    emps_changed,ag_roles_changed, reg_emp, weeks_to_plan, list_sol, all_roles,
    ag_list_sol, reserve_list_fwd, emp_extend, rest_count,in_reserve_fwd,
    task_extend, min_rest,conflict_found_fwd,block_start);
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu=no_tabu+1;
}
else {
to_calculate=4;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
```

```
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
```

```
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
evaluate_block_new9(extend_cost_fwd, total_cost, do_extend_fwd, no_nonreduce,
    emps_to_update, accept_rule, emps_changed);
if(total_cost[4]<total_cost[accept_rule])   {
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
else {
evaluate_block_new10( emps_changed,candidate_emps, extend_cost_fwd,
    candidate_cost, candidate_exist, total_cost,transfer_sol_from,
    transfer_sol_to, all_emp, reg_emp, all_roles, weeks_to_plan, lambda,
    allocate_sol, long_work_sol, emp_cost, list_sol, board_sol, depart_sol,
    undertime_sol, overtime_sol, ag_cost, ag_list_sol, ag_crewchange,
    ag_rboard_sol, ag_rdepart_sol);
} }
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }
if(do_extend_fwd==0) {
extend_len_fwd=extend_len_fwd-1;
} } } }
if(do_extend_bkwd==0 && do_extend_fwd==0 ) {
if(block_start>=0) {
evaluate_swap1(ag_swappable, block_start, block_end, eligible, task_extend);
if(ag_swappable==1) {
```

```
for(rol_10=0;rol_10< all_roles;rol_10++) {
if(rol_10!=task_extend) {
evaluate_swap2( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_find_time,
    swap_block_found);
for(weeks_10=0;weeks_10<weeks_to_plan;weeks_10++) {
if(do_swap==0) {
evaluate_swap3( weeks_10, rol_10,swap_block_start, swap_task, swap_emp,
    swap_allowed, eligible , min_rest, starting, emp_extend, all_roles, summation
    , summation_1, too_early, swap_new_block, swap_block_found, ag_list_sol,
    swap_block_latest, swap_block_earliest);
if(swap_block_found ==1 && ag_list_sol[0][weeks_10][rol_10]==1) {
if(ag_crewchange[0][weeks_10][rol_10]==1) {
swap_block_end=weeks_10-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } }
evaluate_swap4( weeks_10, swap_block_found, swap_block_len, swap_block_end,
    swap_allowed, eligible, min_rest, too_early, emp_extend, swap_block_earliest,
     starting, summation, summation_1, all_roles, swap_block_latest, do_swap,
    swap_task,swap_block_start, rol_10);
}
else {
evaluate_swap5(swap_allowed, emp_extend, rol_10, weeks_10, swap_block_latest,
    swap_block_found, eligible);
} }
else if(swap_block_found==1 && ag_list_sol[0][weeks_10][rol_10]!=1) {
swap_block_end=weeks_10-1;
swap_block_len=(swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend])
{       swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan,
    list_sol, all_roles,ag_list_sol, emp_extend, block_start, swap_block_start,
    min_rest, swap_block_end, swap_task, block_end, swap_emp,task_extend,
    to_check_tabu );
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
```

```
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap6( swap_allowed, too_early, do_swap, swap_emp, swap_task,
    swap_block_start, swap_block_end, swap_block_len, swap_block_found );
}
```

```
evaluate_swap7( swap_new_block, swap_block_found);
if((weeks_10==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap8( weeks_10, rol_10,swap_block_end, swap_block_start, swap_block_len
    , swap_allowed, eligible,emp_extend);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
```

```
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} }    } } } } } } }
if(do_swap==0) {
evaluate_swap9( swap_emp, swap_task, swap_block_start, block_start,
    swap_block_end, block_end );
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
```

```
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
//part 2 ev_swap
evaluate_swap10( min_rest, work_zero, starting, all_roles, summation, summation_1
    , eligible, task_extend, block_end, rest_zero, block_start, max_work,
    block_len, reg_emp, swappable_emp, emp_extend);
// void evaluate_swap10(int min_rest_1[48], int work_zero_1[48],int starting_1
    [25][49],int& weeks_10x,int all_roles_1,int summation_1, int summation_1x,int
     eligible_1[13][25][48],int& task_extend_1,int& block_end_1,int rest_zero_1
    [48],int& block_start_1,int max_work_1[48],int& block_len_1, int reg_emp_1,
    int swappable_emp_1[48], int& emp_extend_1 )
//part 2 ev_swap
boyut_20=ordered_list.size();
for(count_20=0;count_20<boyut_20; count_20++)
//for(emp_10=0;emp_10<48; emp_10++)
{
emp_10=ordered_list[count_20];
if(swappable_emp[emp_10]==1 && swaps_examined[emp_extend][emp_10]!=1)  // AL -
    Have changed this from "if(swappable_emp[emp_10]==0 &&..."
{
evaluate_swap11(swap_find_time, all_rest,swap_allowed, too_early,
    swap_block_found, swap_block_len, do_swap, swap_emp, swap_vessel, swap_task,
    swap_block_start, swap_block_end);
while(do_swap==0 && swap_find_time<weeks_to_plan) {
evaluate_swap12(swap_vessel,roles, swap_block_start, swap_emp, swap_task,
    swap_allowed, eligible, min_rest, emp_extend, starting, all_roles, summation,
     summation_1,too_early, swap_block_found,swap_new_block, all_rest,rol_10,
    emp_10, list_sol, swap_block_earliest,swap_find_time, swap_block_latest);
```

```
if(swap_block_found==1 && rol_10!=-1) {
if(rol_10 !=swap_task) {
link_to_vessel=0;
evaluate_swap13( vessel_extend, roles,swap_task, swap_allowed, eligible, rol_10,
    emp_extend, swap_vessel, link_to_vessel, swap_find_time, swap_block_latest,
    swap_block_found);
if(link_to_vessel==-1) {
swap_block_end= swap_find_time-1;
swap_block_len = (swap_block_end - swap_block_start) +1;
if(too_early ==0 && swap_allowed ==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap14( swap_block_found, swap_block_len, swap_block_end, swap_vessel,
    swap_allowed,eligible , min_rest, starting, emp_extend, roles, all_roles,
    summation, summation_1, too_early, swap_find_time, swap_block_latest, do_swap
    , swap_task_extend, rol_10, swap_block_start, swap_block_earliest);
} }
else {
evaluate_swap15(rol_10, swap_find_time, swap_block_latest, swap_block_found,
    swap_allowed, eligible, emp_extend);
} }
else if(swap_block_found==1 && rol_10==-1) {
swap_block_end=swap_find_time-1;
swap_block_len= (swap_block_end - swap_block_start) +1;
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } }
evaluate_swap16( swap_emp, swap_task, swap_vessel, swap_block_found, do_swap,
    too_early, swap_allowed, swap_block_start, swap_block_end, swap_block_len);
}
evaluate_swap17( swap_new_block, swap_block_found);
if((swap_find_time==(weeks_to_plan-1)) && (swap_block_found==1)) {
evaluate_swap18(swap_block_start, swap_block_len, swap_block_end, swap_find_time,
     swap_allowed, eligible, swap_find_time, rol_10);
if(too_early==0 && swap_allowed==1) {
if(swap_block_len <= max_work[emp_extend]) {
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }
swap_find_time++;
}
if(do_swap == 0 && all_rest==1) {
evaluate_swap19( swap_emp, emp_10, swap_task, swap_block_start, block_start,
    swap_block_end, block_end);
swap_calc1( ag_roles_changed, emps_changed,reg_emp, weeks_to_plan, list_sol,
    all_roles,ag_list_sol, emp_extend, block_start, swap_block_start, min_rest,
    swap_block_end, swap_task, block_end, swap_emp,task_extend,to_check_tabu);
//swap calc part 1 finish
//swap calc part 2
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu ==1 )  // AL - Have changed this from "if(tabu ==0 ) "
{
no_tabu= no_tabu + 1;
}
else {
to_calculate=5;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
```

```
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
swap_calc3(emps_changed,no_nonreduce, swapping_cost, total_cost, accept_rule,
    do_swap, emps_to_update, swaps_examined, reg_emp, last_kick_time, iteration,
    last_changed);
swap_calc4( candidate_emps, swap_emp, emp_extend, candidate_cost, swapping_cost,
    total_cost, accept_rule, candidate_exist, transfer_sol_from, transfer_sol_to,
     all_emp, reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol,
    long_work_sol, emp_cost, list_sol, board_sol, depart_sol, undertime_sol,
    overtime_sol, ag_cost, ag_list_sol,ag_crewchange, ag_rboard_sol,
    ag_rdepart_sol);
}
else {
no_infeas=no_infeas+1;
infeasibility(Statfeas,no_Statfeas,Linkfeas,no_Linkfeas,no_RvWfeas,RvWfeas,
    AGLWfeas,LWfeas,no_AGLWfeas,no_LWfeas,UTfeas,OTfeas,no_UTfeas,no_OTfeas,
    JCfeas,no_JCfeas,OLfeas, no_OLfeas,Brdfeas,Dprtfeas,AGBDfeas,no_Brdfeas,
    no_Dprtfeas,no_AGBDfeas);
} } } } }     } }
evaluate_block_new13(do_swap, no_swap,do_extend_bkwd, transfer_sol_from, no_bkwd,
     do_extend_fwd, no_fwd);
if(do_extend_bkwd==1 || do_extend_fwd==1 || do_swap==1) {
check_tabu_2(transfer_sol_to, reg_emp,all_roles, weeks_to_plan, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
update_done=1;
compare_to_best(number_best, weeks_to_plan,all_roles, reg_emp, role_count ,
    total_cost, new_best, same_best, emp_count, list_sol, best_sols, ag_list_sol,
     ag_best_sols);
compare_to_best_2(best_sol_time,iteration, ag_best_sols,best_sols, number_best,
    transfer_sol_from, transfer_sol_to, total_cost, all_emp, reg_emp, all_roles,
    weeks_to_plan, lambda, allocate_sol, long_work_sol, emp_cost, list_sol,
    board_sol, depart_sol, undertime_sol, overtime_sol,ag_cost, ag_list_sol,
    ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
} } } } }     } }
find_usables_9(update_done, emp_13,reg_emp, swaps_examined);
}
find_usables_10(tabu_sol, ag_tabu_sol, no_nonreduce, candidate_cost,
    transfer_sol_from, transfer_sol_to, candidate_exist, update_done,
    weeks_to_plan, reg_emp, all_roles, list_sol, ag_list_sol);
if(update_done==0 && candidate_exist==1) {
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
update_done=1;
no_cand=no_cand + 1;
```

```
compare_to_best(number_best, weeks_to_plan,all_roles, reg_emp, role_count ,
    total_cost, new_best, same_best, emp_count, list_sol, best_sols, ag_list_sol,
     ag_best_sols);
compare_to_best_2(best_sol_time,iteration, ag_best_sols,best_sols, number_best,
    transfer_sol_from, transfer_sol_to, total_cost, all_emp, reg_emp, all_roles,
    weeks_to_plan, lambda, allocate_sol, long_work_sol, emp_cost, list_sol,
    board_sol, depart_sol, undertime_sol, overtime_sol,ag_cost, ag_list_sol,
    ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
emps_to_update.clear();
emps_to_update.reserve(candidate_emps.size());
copy(candidate_emps.begin(),candidate_emps.end(),back_inserter(emps_to_update));
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
}
end = clock();
time=(double)(end-start)/CLOCKS_PER_SEC;
finishing(update_done, terminate, iteration,time);
if(kick_rule==1) {
if((iteration - best_sol_time)>=4 && terminate == 0)
//if((iteration - best_sol_time)>=8 && terminate == 0)
{
if(last_kick_time==0 || ((last_kick_time > 0) && (iteration - last_kick_time) >=
    20) || (total_cost[0]==total_cost[1] && no_nonreduce >= 1) || no_nonreduce >=
     4)
//if(last_kick_time==0 || ((last_kick_time > 0) && (iteration - last_kick_time)
    >= 40) || (total_cost[0]==total_cost[1] && no_nonreduce >= 1) || no_nonreduce
     >= 8)
{
update_done=0;
while(update_done==0) {
kicking( emps_changed, ag_roles_changed, eligible, work_zero, min_rest, starting,
     all_roles, sum_start_rand,rest_zero, kick_feas, kick_end, kick_start, random
    ,random_emp, reg_emp,kick_emp,random_task, all_roles, kick_task, max_work,
    random_length, random_time, weeks_to_plan);
if(kick_feas==1) {
kicking2( min_rest, starting, rest_count, kick_emp, reg_emp, all_roles,
    weeks_to_plan, ag_list_sol, kick_start, kick_end,kick_task, emps_changed,
    ag_roles_changed, list_sol);
to_check_tabu= 7;
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu = no_tabu + 1;
```

```
}
else {
to_calculate=7;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
kicking3( all_roles, reg_emp, weeks_to_plan, tabu_sol, list_sol, ag_tabu_sol,
    ag_list_sol);
transfer_sol_from=7;
transfer_sol_to=0;
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
last_kick_time=iteration;
update_done=1;
no_nonreduce=0;
compare_to_best(number_best, weeks_to_plan,all_roles, reg_emp, role_count ,
    total_cost, new_best, same_best, emp_count, list_sol, best_sols, ag_list_sol,
     ag_best_sols);
compare_to_best_2(best_sol_time,iteration, ag_best_sols,best_sols, number_best,
    transfer_sol_from, transfer_sol_to, total_cost, all_emp, reg_emp, all_roles,
    weeks_to_plan, lambda, allocate_sol, long_work_sol, emp_cost, list_sol,
    board_sol, depart_sol, undertime_sol, overtime_sol,ag_cost, ag_list_sol,
    ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
emps_to_update.clear();      // AL - Have added this line, to ensure
    emps_to_update takes ONLY the values contained in emps_changed
emps_to_update.reserve(emps_changed.size());
copy(emps_changed.begin(),emps_changed.end(),back_inserter(emps_to_update));
```

```
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
} } } }
no_kick=no_kick + 1;
//writeln("Running time so far: ",current_time - prog_starttime)
} } }
else if(kick_rule==2) {
//if((iteration - best_sol_time)>=4 && terminate == 0)
if((iteration - best_sol_time)>=8 && terminate == 0) {
//if(last_kick_time==0 || ((last_kick_time > 0) && (iteration - last_kick_time)
    >= 20) || (total_cost[0]==total_cost[1] && no_nonreduce >= 1) || no_nonreduce
     >= 4)
if(last_kick_time==0 || ((last_kick_time > 0) && (iteration - last_kick_time) >=
    40) || (total_cost[0]==total_cost[1] && no_nonreduce >= 1) || no_nonreduce >=
     8)
{
update_done=0;
while(update_done==0) {
kicking( emps_changed, ag_roles_changed, eligible, work_zero, min_rest, starting,
     all_roles, sum_start_rand,rest_zero, kick_feas, kick_end, kick_start, random
    ,random_emp, reg_emp,kick_emp,random_task, all_roles, kick_task, max_work,
    random_length, random_time, weeks_to_plan);
if(kick_feas==1) {
kicking2( min_rest, starting, rest_count, kick_emp, reg_emp, all_roles,
    weeks_to_plan, ag_list_sol, kick_start, kick_end,kick_task, emps_changed,
    ag_roles_changed, list_sol);
to_check_tabu= 7;
check_tabu(tabu, weeks_to_plan, reg_emp, all_roles, to_check_tabu, list_sol,
    ag_list_sol, tabu_sol, ag_tabu_sol);
if(tabu==1) {
no_tabu = no_tabu + 1;
}
else {
to_calculate=7;
calc_cost1(all_roles,reg_emp, weeks_to_plan, list_sol, ag_list_sol, to_calculate,
     total_cost);
calc_cost2(total_cost, starting, reg_emp, weeks_to_plan, all_roles,emps_changed,
    emp_cost, allocate_sol, long_work_sol, lambda, board_sol, depart_sol,
    undertime_sol, overtime_sol, consec_work,work_zero, list_sol, g_weeks,
    exp_worktime);
calc_cost3( reg_emp, weeks_to_plan, all_roles,emps_changed, emp_cost,
    allocate_sol, long_work_sol, lambda, board_sol, depart_sol, undertime_sol,
    overtime_sol, chng_undertime, chng_overtime, cur_undertime, cur_overtime,
    under_rate, over_rate, cur_allocate, chng_allocate, cur_board, chng_board,
    board_chng_cost, cur_depart, chng_depart, depart_chng_cost, cur_long_work,
    chng_long_work, extension_chng_cost,work_chng_cost);
calc_cost4( work_chng_cost, ag_max_work, ag_work_zero, consec_work,
    feas_crewchange, to_calculate, iteration, weeks_to_plan, all_roles,
    evaluate_crewchange, ag_roles_changed, definite_crewchange,
    possible_crewchange, ag_cost, ag_crewchange, allocate_sol, ag_rboard_sol,
    ag_rdepart_sol, long_work_sol, lambda, ag_starting, ag_list_sol,
    cur_ag_rboard, poss_chng_ag_rboard, poss_ag_rboard, cur_ag_rdepart,
    poss_chng_ag_rdepart, poss_ag_rdepart, ag_board_chng_cost,
    ag_depart_chng_cost, poss_ag_long_work, cur_long_work, poss_chng_ag_long_work
    , cur_allocate, chng_allocate, extension_chng_cost, chng_ag_rboard,
    chng_ag_rdepart, chng_long_work);
calc_cost5(reg_emp,weeks_to_plan, all_roles,emp_cost, total_cost, ag_cost,
    transfer_sol_from, transfer_sol_to, to_calculate);
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
JC_feas(feasible,JCfeas, all_emp, all_roles, weeks_to_plan, eligible,allocate_sol
    ,required);
Overlap_feas(emps_changed, feasible,OLfeas,all_emp, all_roles,weeks_to_plan,
    allocate_sol);
BoardFeas(emps_changed, feasible, Brdfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, board_sol, starting);
DepartFeas(emps_changed, feasible, Dprtfeas, all_emp, all_roles, weeks_to_plan,
    allocate_sol, depart_sol, starting);
AgBoardDepartFeas(ag_roles_changed, feasible, AGBDfeas, all_roles, weeks_to_plan,
     allocate_sol, ag_rboard_sol, ag_rdepart_sol,ag_starting);
UnderOverFeas(emps_changed, feasible, UTfeas, OTfeas, all_emp, all_roles,
    weeks_to_plan, undertime_sol, overtime_sol, g_weeks, exp_worktime,
    allocate_sol);
LongWorkFeas(ag_roles_changed,emps_changed,AGLWfeas,LWfeas, lambda, feasible,
    work_total, work_zero, reg_emp, all_roles, weeks_to_plan, allocate_sol,
    max_work, long_work_sol, ag_work_total, ag_work_zero, ag_max_work,
    ag_rdepart_sol);
RestFeas(emps_changed,feasible,RvWfeas, reg_emp, all_roles, weeks_to_plan,
    depart_sol, rest_total, rest_zero,allocate_sol, min_rest);
LinkFeasiblity(lambda, ag_roles_changed,emps_changed, feasible, Linkfeas, reg_emp
    , all_roles, weeks_to_plan, cur_allocate, chng_allocate, allocate_sol,
    cur_board, chng_board, board_sol, cur_depart, chng_depart, depart_sol,
    cur_ag_rboard, ag_rboard_sol, chng_ag_rboard, cur_ag_rdepart, ag_rdepart_sol,
     chng_ag_rdepart, cur_long_work, chng_long_work, long_work_sol,
    chng_undertime, undertime_sol, cur_undertime, chng_overtime, overtime_sol,
    cur_overtime);
StatusFeasibility( lambda,ag_roles_changed,emps_changed,feasible,Statfeas,reg_emp
    ,all_roles, weeks_to_plan, allocate_sol, chng_allocate, chng_long_work,
    long_work_sol, chng_board, board_sol, chng_depart, depart_sol, ag_rboard_sol,
     chng_ag_rboard, ag_rdepart_sol, chng_ag_rdepart, undertime_sol, overtime_sol
    );
if(feasible==1) {
kicking3( all_roles, reg_emp, weeks_to_plan, tabu_sol, list_sol, ag_tabu_sol,
    ag_list_sol);
transfer_sol_from=7;
transfer_sol_to=0;
transfer_solution(transfer_sol_from, transfer_sol_to, total_cost, all_emp,
    reg_emp, all_roles, weeks_to_plan, lambda, allocate_sol, long_work_sol,
    emp_cost, list_sol, board_sol, depart_sol, undertime_sol, overtime_sol,
    ag_cost, ag_list_sol, ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
last_kick_time=iteration;
update_done=1;
no_nonreduce=0;
compare_to_best(number_best, weeks_to_plan,all_roles, reg_emp, role_count ,
    total_cost, new_best, same_best, emp_count, list_sol, best_sols, ag_list_sol,
     ag_best_sols);
compare_to_best_2(best_sol_time,iteration, ag_best_sols,best_sols, number_best,
    transfer_sol_from, transfer_sol_to, total_cost, all_emp, reg_emp, all_roles,
    weeks_to_plan, lambda, allocate_sol, long_work_sol, emp_cost, list_sol,
    board_sol, depart_sol, undertime_sol, overtime_sol,ag_cost, ag_list_sol,
    ag_crewchange, ag_rboard_sol, ag_rdepart_sol);
emps_to_update.clear();      // AL - Have added this line, to ensure
    emps_to_update takes ONLY the values contained in emps_changed
emps_to_update.reserve(emps_changed.size());
copy(emps_changed.begin(),emps_changed.end(),back_inserter(emps_to_update));
update_swaps_and_changes( emps_to_update,swaps_examined,reg_emp, last_kick_time,
    iteration, last_changed);
} } } }
no_kick=no_kick + 1;
//writeln("Running time so far: ",current_time - prog_starttime)
}      } }
results_alter<<iteration<<"  "<<total_cost[0]<<"   "<<total_cost[1] << '\n';
if((time>=60 && time>=61) && no_enter==0) {
no_enter=1;
changes_to_reg = 0;          // number of changes to regular employees in the (
    best) solution
changes_to_AG = 0;           //number of changes to agency employees in the (best
    ) solution
number_of_AG = 0;           // number of times agency employees are utilised in
    the (best) solution
for(int e_last=0;e_last<48; e_last++) {
for(int job_last=0;job_last<25; job_last++) {
for(int week_last=0;week_last<12; week_last++) {
if(list_sol[1][week_last][e_last]==job_last && cur_allocate[week_last][job_last][
    e_last]==0) {
changes_to_reg= changes_to_reg +1;
}
if(list_sol[1][week_last][e_last]!=job_last && cur_allocate[week_last][job_last][
    e_last]==1) {
changes_to_reg=changes_to_reg +1;
} } } }
for(int job_last=0;job_last<25; job_last++) {
for(int week_last=0;week_last<12; week_last++) {
if(ag_list_sol[0][week_last][job_last]==1) {
number_of_AG=number_of_AG +1;
if(cur_allocate[week_last][job_last][48]==0) {
changes_to_AG=changes_to_AG +1;
} }
else {
if(cur_allocate[week_last][job_last][48]==1) {
changes_to_AG=changes_to_AG +1;
} } } }
results_1min<< "Set "<<set_no<<"    "<<time<<"    "<<iteration<<" "<<initial_cost
    <<"  "<< accept_rule<<"    "<<kick_rule<<" "<<order_setting<<" "<<
    initial_solution_type<<"  "<<fraction_settings<<" "<<no_bkwd<<" "<<no_fwd<<"
    "<<no_swap<<" "<<no_cand<<" "<<no_kick<<" "<<no_infeas<<" "<<no_tabu<<" "<<
    total_cost[1]<<" "<<total_cost[0]<<"   "<<best_sol_time<<"   "<<number_best<<"
        "<<changes_to_reg<<"  "<<changes_to_AG<<"   "<<number_of_AG<<'\n';
} }
changes_to_reg = 0;         // number of changes to regular employees in the (
    best) solution
changes_to_AG = 0;          //number of changes to agency employees in the (best
    ) solution
number_of_AG = 0;           // number of times agency employees are utilised in
    the (best) solution
for(int e_last=0;e_last<48; e_last++) {
for(int job_last=0;job_last<25; job_last++) {
for(int week_last=0;week_last<12; week_last++) {
if(list_sol[1][week_last][e_last]==job_last && cur_allocate[week_last][job_last][
    e_last]==0) {
```

```
changes_to_reg= changes_to_reg +1;
}
if(list_sol[1][week_last][e_last]!=job_last && cur_allocate[week_last][job_last][
    e_last]==1) {
changes_to_reg=changes_to_reg +1;
} } } }
for(int job_last=0;job_last<25; job_last++) {
for(int week_last=0;week_last<12; week_last++) {
if(ag_list_sol[0][week_last][job_last]==1) {
number_of_AG=number_of_AG +1;
if(cur_allocate[week_last][job_last][48]==0) {
changes_to_AG=changes_to_AG +1;
} }
else {
if(cur_allocate[week_last][job_last][48]==1) {
changes_to_AG=changes_to_AG +1;
} } } }
results<<"Set "<<set_no<<"   "<<time<<"     "<<iteration<<" "<<initial_cost<<"
    "<< accept_rule<<"    "<<kick_rule<<" "<<order_setting<<" "<<
    initial_solution_type<<"  "<<fraction_settings<<" "<<no_bkwd<<" "<<no_fwd<<"
    "<<no_swap<<" "<<no_cand<<" "<<no_kick<<" "<<no_infeas<<" "<<no_tabu<<" "<<
    total_cost[1]<<" "<<total_cost[0]<<"   "<<best_sol_time<<"   "<<number_best<<"
        "<<changes_to_reg<<"  "<<changes_to_AG<<"   "<<number_of_AG<<'\n';
} } } } }
results.close();
system("pause");
return 0;
}
```

E.2.5.2 ‘Calculate cost’ sub-programmes

```
#include "cost.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "string"
#include <cstdlib>
#include <ctime>
#include <math.h>
#include <iostream>
#include <fstream>
#include <algorithm>
```

```
#include <iterator>
#include <vector>
#include <random>
using namespace std;

void calc_cost1(int all_roles_1, int reg_emp_1, int weeks_to_plan_1, int
    list_sol_1[8][13][48], int ag_list_sol_1[8][13][25], int& to_calculate_1,
    float total_cost_1[8])
{
int emp_4,weeks_4, rol_4;
for(emp_4=0;emp_4<reg_emp_1;emp_4++)
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
list_sol_1[2][weeks_4][emp_4]=list_sol_1[to_calculate_1][weeks_4][emp_4];
}
for(rol_4=0;rol_4<all_roles_1;rol_4++)
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
ag_list_sol_1[2][weeks_4][rol_4]= ag_list_sol_1[to_calculate_1][weeks_4][rol_4];
}
total_cost_1[2]=0;
}

void calc_cost2(float total_cost_1[8],int starting_1[25][49],int reg_emp_1,int
    weeks_to_plan_1, int all_roles_1,vector<int> &emps_changed_1, float
    emp_cost_1[8][48], int allocate_sol_1[8][13][25][49], int long_work_sol_1
    [8][13][25][49][10], int lambda_1,int board_sol_1[8][13][25][48], int
    depart_sol_1[8][13][25][48],int undertime_sol_1[8][45], int overtime_sol_1
    [8][45], int& consec_work_1, int work_zero_1[48],int list_sol_1[8][13][48],
    int g_weeks_1[45], int exp_worktime_1[45] )
{
int size_empschanged, count_4, emp_4, long_2, yes, weeks_4, rol_4, weeks_5, rol_5
    ;
int sum_allocation_vessels[48];
total_cost_1[2]=0;
size_empschanged=emps_changed_1.size();
for(emp_4=0;emp_4<reg_emp_1;emp_4++) {
yes=0;
count_4=0;
for(count_4=0; count_4<size_empschanged; count_4++) {
if(emp_4==emps_changed_1[count_4]) {
yes=1;
count_4=size_empschanged;
} }
if(yes==0) {
```

```
emp_cost_1[2][emp_4]=emp_cost_1[0][emp_4];
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
for(rol_4=0;rol_4<all_roles_1;rol_4++) {
allocate_sol_1[2][weeks_4][rol_4][emp_4]=allocate_sol_1[0][weeks_4][rol_4][emp_4
    ];
for(long_2=0;long_2<lambda_1;long_2++) {
long_work_sol_1[2][weeks_4][rol_4][emp_4][long_2]=long_work_sol_1[0][weeks_4][
    rol_4][emp_4][long_2];
} }
for(rol_4=0;rol_4<all_roles_1;rol_4++) {
board_sol_1[2][weeks_4][rol_4][emp_4]=board_sol_1[0][weeks_4][rol_4][emp_4];
depart_sol_1[2][weeks_4][rol_4][emp_4]=depart_sol_1[0][weeks_4][rol_4][emp_4];
} }
if(emp_4>=3) {
undertime_sol_1[2][emp_4]=undertime_sol_1[0][emp_4];
overtime_sol_1[2][emp_4]=overtime_sol_1[0][emp_4];
} }
else if(yes==1) {
emp_cost_1[2][emp_4]=0;
consec_work_1=work_zero_1[emp_4];
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
for(rol_4=0;rol_4<all_roles_1;rol_4++) {
allocate_sol_1[2][weeks_4][rol_4][emp_4]=0;
for(long_2=0;long_2<lambda_1;long_2++) {
long_work_sol_1[2][weeks_4][rol_4][emp_4][long_2]=0;
}
if(list_sol_1[2][weeks_4][emp_4]==rol_4) {
allocate_sol_1[2][weeks_4][rol_4][emp_4]=1;
consec_work_1=consec_work_1+1;
for(long_2=0;long_2<consec_work_1;long_2++) {
long_work_sol_1[2][weeks_4][rol_4][emp_4][long_2]=1;
} } }
if(list_sol_1[2][weeks_4][emp_4]==-1) {
consec_work_1=0;
} }
for(rol_4=0;rol_4<all_roles_1;rol_4++) {
board_sol_1[2][0][rol_4][emp_4]=0;
depart_sol_1[2][0][rol_4][emp_4]=0;
if(allocate_sol_1[2][0][rol_4][emp_4] < starting_1[rol_4][emp_4]) {
depart_sol_1[2][0][rol_4][emp_4]=1;
}
else if(allocate_sol_1[2][0][rol_4][emp_4] > starting_1[rol_4][emp_4]) {
board_sol_1[2][0][rol_4][emp_4]= 1;
}
for(weeks_4=1;weeks_4<weeks_to_plan_1;weeks_4++) {
board_sol_1[2][weeks_4][rol_4][emp_4]=0;
depart_sol_1[2][weeks_4][rol_4][emp_4]=0;
if(allocate_sol_1[2][weeks_4][rol_4][emp_4]<allocate_sol_1[2][weeks_4-1][rol_4][
    emp_4]) {
depart_sol_1[2][weeks_4][rol_4][emp_4]=1;
}
else if(allocate_sol_1[2][weeks_4][rol_4][emp_4]>allocate_sol_1[2][weeks_4-1][
    rol_4][emp_4]) {
board_sol_1[2][weeks_4][rol_4][emp_4]= 1;
} } }
if(emp_4>=3) {
sum_allocation_vessels[emp_4]=0;
for(weeks_5=0;weeks_5<weeks_to_plan_1;weeks_5++)
for(rol_5=0;rol_5<all_roles_1;rol_5++) {
sum_allocation_vessels[emp_4]=allocate_sol_1[2][weeks_5][rol_5][emp_4]+
    sum_allocation_vessels[emp_4];
}
if(g_weeks_1[emp_4-3]>(exp_worktime_1[emp_4-3] + sum_allocation_vessels[emp_4]))
    {
undertime_sol_1[2][emp_4-3]=g_weeks_1[emp_4-3]- (exp_worktime_1[emp_4-3]+
    sum_allocation_vessels[emp_4]);
overtime_sol_1[2][emp_4-3]=0;
}
else {
overtime_sol_1[2][emp_4-3]= (exp_worktime_1[emp_4-3] + sum_allocation_vessels[
    emp_4])- g_weeks_1[emp_4-3];
undertime_sol_1[2][emp_4-3]= 0;
} } } } }

void calc_cost3(int reg_emp_1,int weeks_to_plan_1, int all_roles_1,vector<int> &
    emps_changed_1, float emp_cost_1[8][48], int allocate_sol_1[8][13][25][49],
    int long_work_sol_1[8][13][25][49][10], int lambda_1,int board_sol_1
    [8][13][25][48], int depart_sol_1[8][13][25][48],int undertime_sol_1[8][45],
    int overtime_sol_1[8][45], int chng_undertime_1[45], int chng_overtime_1[45],
     int cur_undertime_1[45], int cur_overtime_1[45],int under_rate_1[45], int
    over_rate_1[45], int cur_allocate_1[13][25][49], int chng_allocate_1
    [13][25][49], int cur_board_1[13][25][48], int chng_board_1[13][25][48],
    float board_chng_cost_1[13][25][48], int cur_depart_1[13][25][48], int
    chng_depart_1[13][25][48], float depart_chng_cost_1[13][25][48],int
    cur_long_work_1[13][25][49][10] ,int chng_long_work_1[13][25][49][10], float
    extension_chng_cost_1[13][25][49][10],float work_chng_cost_1[13][25][49])
```

```
{
int yes,size_empschanged, count_4, emp_4, long_2, weeks_4, rol_4;
size_empschanged=emps_changed_1.size();
for(emp_4=0;emp_4<reg_emp_1;emp_4++) {
yes=0;
count_4=0;
for(count_4=0; count_4<size_empschanged; count_4++) {
if(emp_4==emps_changed_1[count_4]) {
yes=1;
count_4=size_empschanged;
} }
if(yes==1) {
if (emp_4>=3) {
chng_undertime_1[emp_4-3]= undertime_sol_1[2][emp_4-3]- cur_undertime_1[emp_4-3];
chng_overtime_1[emp_4-3]= overtime_sol_1[2][emp_4-3] - cur_overtime_1[emp_4-3];
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4] + (under_rate_1[emp_4-3]*
    chng_undertime_1[emp_4-3]) + (over_rate_1[emp_4-3]*chng_overtime_1[emp_4-3]);
     // under-over cost
}
for(rol_4=0;rol_4<all_roles_1;rol_4++)
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
if(cur_allocate_1[weeks_4][rol_4][emp_4]==0) {
chng_allocate_1[weeks_4][rol_4][emp_4]=allocate_sol_1[2][weeks_4][rol_4][emp_4];
}
else {
chng_allocate_1[weeks_4][rol_4][emp_4]=cur_allocate_1[weeks_4][rol_4][emp_4]-
    allocate_sol_1[2][weeks_4][rol_4][emp_4];
}
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4]+ (work_chng_cost_1[weeks_4][rol_4][
    emp_4]*chng_allocate_1[weeks_4][rol_4][emp_4]); // work_change
}
for(rol_4=0;rol_4<all_roles_1;rol_4++)
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
if(cur_board_1[weeks_4][rol_4][emp_4]==0) {
chng_board_1[weeks_4][rol_4][emp_4]=board_sol_1[2][weeks_4][rol_4][emp_4];
}
else {
chng_board_1[weeks_4][rol_4][emp_4]= cur_board_1[weeks_4][rol_4][emp_4]-
    board_sol_1[2][weeks_4][rol_4][emp_4];
}
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4]+(board_chng_cost_1[weeks_4][rol_4][
    emp_4]*chng_board_1[weeks_4][rol_4][emp_4]); //board_cost
}
for(rol_4=0;rol_4<all_roles_1;rol_4++)
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
if(cur_depart_1[weeks_4][rol_4][emp_4]==0) {
chng_depart_1[weeks_4][rol_4][emp_4]= depart_sol_1[2][weeks_4][rol_4][emp_4];
}
else {
chng_depart_1[weeks_4][rol_4][emp_4]= cur_depart_1[weeks_4][rol_4][emp_4] -
    depart_sol_1[2][weeks_4][rol_4][emp_4];
}
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4] + (depart_chng_cost_1[weeks_4][rol_4][
    emp_4]*chng_depart_1[weeks_4][rol_4][emp_4]); // depart
}
for(long_2=0;long_2<lambda_1;long_2++) {
for(rol_4=0;rol_4<all_roles_1;rol_4++) {
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
if(cur_long_work_1[weeks_4][rol_4][emp_4][long_2]== 0) {
chng_long_work_1[weeks_4][rol_4][emp_4][long_2]= long_work_sol_1[2][weeks_4][
    rol_4][emp_4][long_2];
}
else {
chng_long_work_1[weeks_4][rol_4][emp_4][long_2]=cur_long_work_1[weeks_4][rol_4][
    emp_4][long_2] -long_work_sol_1[2][weeks_4][rol_4][emp_4][long_2];
}
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4] + (extension_chng_cost_1[weeks_4][
    rol_4][emp_4][long_2]*chng_long_work_1[weeks_4][rol_4][emp_4][long_2]); //
    long_work
} } } } } }

void calc_cost31(int reg_emp_1,int weeks_to_plan_1, int all_roles_1,vector<int> &
    emps_changed_1, float emp_cost_1[8][48], int allocate_sol_1[8][13][25][49],
    int long_work_sol_1[8][13][25][49][10], int lambda_1,int board_sol_1
    [8][13][25][48], int depart_sol_1[8][13][25][48],int undertime_sol_1[8][45],
    int overtime_sol_1[8][45], int chng_undertime_1[45], int chng_overtime_1[45],
     int cur_undertime_1[45], int cur_overtime_1[45],int under_rate_1[45], int
    over_rate_1[45], int cur_allocate_1[13][25][49], int chng_allocate_1
    [13][25][49], int cur_board_1[13][25][48], int chng_board_1[13][25][48],
    float board_chng_cost_1[13][25][48], int cur_depart_1[13][25][48], int
    chng_depart_1[13][25][48], float depart_chng_cost_1[13][25][48],int
    cur_long_work_1[13][25][49][10] ,int chng_long_work_1[13][25][49][10], float
    extension_chng_cost_1[13][25][49][10],float work_chng_cost_1[13][25][49])
{
int size_empschanged, count_4, emp_4, long_2, weeks_4, rol_4;
size_empschanged=emps_changed_1.size();
```

```
for(count_4=0; count_4<size_empschanged; count_4++) {
emp_4=emps_changed_1[count_4];
if (emp_4>=3) {
chng_undertime_1[emp_4-3]= undertime_sol_1[2][emp_4-3]- cur_undertime_1[emp_4-3];
chng_overtime_1[emp_4-3]= overtime_sol_1[2][emp_4-3] - cur_overtime_1[emp_4-3];
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4] + (under_rate_1[emp_4-3]*
    chng_undertime_1[emp_4-3]) + (over_rate_1[emp_4-3]*chng_overtime_1[emp_4-3]);
     // under-over cost
}
for(rol_4=0;rol_4<all_roles_1;rol_4++)
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
if(cur_allocate_1[weeks_4][rol_4][emp_4]==0) {
chng_allocate_1[weeks_4][rol_4][emp_4]=allocate_sol_1[2][weeks_4][rol_4][emp_4];
}
else {
chng_allocate_1[weeks_4][rol_4][emp_4]=cur_allocate_1[weeks_4][rol_4][emp_4]-
    allocate_sol_1[2][weeks_4][rol_4][emp_4];
}
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4]+ (work_chng_cost_1[weeks_4][rol_4][
    emp_4]*chng_allocate_1[weeks_4][rol_4][emp_4]); // work_change
}
for(rol_4=0;rol_4<all_roles_1;rol_4++)
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
if(cur_board_1[weeks_4][rol_4][emp_4]==0) {
chng_board_1[weeks_4][rol_4][emp_4]=board_sol_1[2][weeks_4][rol_4][emp_4];
}
else {
chng_board_1[weeks_4][rol_4][emp_4]= cur_board_1[weeks_4][rol_4][emp_4]-
    board_sol_1[2][weeks_4][rol_4][emp_4];
}
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4]+(board_chng_cost_1[weeks_4][rol_4][
    emp_4]*chng_board_1[weeks_4][rol_4][emp_4]); //board_cost
}
for(rol_4=0;rol_4<all_roles_1;rol_4++)
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
if(cur_depart_1[weeks_4][rol_4][emp_4]==0) {
chng_depart_1[weeks_4][rol_4][emp_4]= depart_sol_1[2][weeks_4][rol_4][emp_4];
}
else {
chng_depart_1[weeks_4][rol_4][emp_4]= cur_depart_1[weeks_4][rol_4][emp_4] -
    depart_sol_1[2][weeks_4][rol_4][emp_4];
}
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4] + (depart_chng_cost_1[weeks_4][rol_4][
    emp_4]*chng_depart_1[weeks_4][rol_4][emp_4]); // depart
}
for(long_2=0;long_2<lambda_1;long_2++) {
for(rol_4=0;rol_4<all_roles_1;rol_4++) {
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
if(cur_long_work_1[weeks_4][rol_4][emp_4][long_2]== 0) {
chng_long_work_1[weeks_4][rol_4][emp_4][long_2]= long_work_sol_1[2][weeks_4][
    rol_4][emp_4][long_2];
}
else {
chng_long_work_1[weeks_4][rol_4][emp_4][long_2]=cur_long_work_1[weeks_4][rol_4][
    emp_4][long_2] -long_work_sol_1[2][weeks_4][rol_4][emp_4][long_2];
}
emp_cost_1[2][emp_4]= emp_cost_1[2][emp_4] + (extension_chng_cost_1[weeks_4][
    rol_4][emp_4][long_2]*chng_long_work_1[weeks_4][rol_4][emp_4][long_2]); //
    long_work
} } } } }

void calc_cost4(float work_chng_cost_1[13][25][49],int ag_max_work_1[25], int
    ag_work_zero_1[25], int &consec_work_1,int &feas_crewchange_1,int &
    to_calculate_1,int &iteration_1, int weeks_to_plan_1, int all_roles_1,vector<
    int> & evaluate_crewchange_1,vector<int> &ag_roles_changed_1, vector<int> &
    definite_crewchange_1, vector<int> &possible_crewchange_1,float ag_cost_1
    [8][13][25], int ag_crewchange_1[8][13][25],int allocate_sol_1
    [8][13][25][49],int ag_rboard_sol_1[8][13][25], int ag_rdepart_sol_1
    [8][13][25], int long_work_sol_1[8][13][25][49][10], int lambda_1, int
    ag_starting_1[25], int ag_list_sol_1[8][13][25], int cur_ag_rboard_1[13][25],
    int poss_chng_ag_rboard_1[13], int poss_ag_rboard_1[13], int cur_ag_rdepart_1
    [13][25], int poss_chng_ag_rdepart_1[13], int poss_ag_rdepart_1[13],float
    ag_board_chng_cost_1[13][25],float ag_depart_chng_cost_1[13][25],int
    poss_ag_long_work_1[13][10], int cur_long_work_1[13][25][49][10], int
    poss_chng_ag_long_work_1[13][10], int cur_allocate_1[13][25][49], int
    chng_allocate_1[13][25][49], float extension_chng_cost_1[13][25][49][10], int
     chng_ag_rboard_1[13][25], int chng_ag_rdepart_1[13][25],int chng_long_work_1
    [13][25][49][10] )
{
int weeks_5, long_2, weeks_4, number_to_run, size_agroleschanged, rol_4, yes,
    count_4, boyut_3, onboard, equal, mn, equal_1, boyut_2, sayi, divide_number,
    tracking_number, equal_2;
double y;
float crewchange_cost, min_crewchange_cost;
size_agroleschanged=ag_roles_changed_1.size();
for(rol_4=0;rol_4<all_roles_1;rol_4++) {
```

```
yes=1;
count_4=0;
for(count_4=0; count_4<size_agroleschanged; count_4++) {
if(rol_4==ag_roles_changed_1[count_4]) {
yes=0;
count_4=size_agroleschanged;
} }
if(yes==1) {
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
ag_cost_1[2][weeks_4][rol_4]=ag_cost_1[0][weeks_4][rol_4];
ag_crewchange_1[2][weeks_4][rol_4]=ag_crewchange_1[0][weeks_4][rol_4];
allocate_sol_1[2][weeks_4][rol_4][48]=allocate_sol_1[0][weeks_4][rol_4][48];
ag_rboard_sol_1[2][weeks_4][rol_4]=ag_rboard_sol_1[0][weeks_4][rol_4];
ag_rdepart_sol_1[2][weeks_4][rol_4]=ag_rdepart_sol_1[0][weeks_4][rol_4];
for(long_2=0;long_2<lambda_1;long_2++) {
long_work_sol_1[2][weeks_4][rol_4][48][long_2]=long_work_sol_1[0][weeks_4][rol_4
    ][48][long_2];
} } }
else if(yes==0) {
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
ag_cost_1[2][weeks_4][rol_4]=0;
}
if(ag_starting_1[rol_4]==1) {
onboard=1;
}
else {
onboard=0;
}
definite_crewchange_1.clear();
possible_crewchange_1.clear();
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
if(ag_list_sol_1[2][weeks_4][rol_4]==1) {
allocate_sol_1[2][weeks_4][rol_4][48]=1;
if(onboard==0) {
definite_crewchange_1.push_back(weeks_4);
}
else {
possible_crewchange_1.push_back(weeks_4);
}
onboard=1;
}
else {
allocate_sol_1[2][weeks_4][rol_4][48]=0;
```

```
if(onboard==1) {
definite_crewchange_1.push_back(weeks_4);
}
onboard=0;
} }
equal=1;
for(mn=0;mn<weeks_to_plan_1;mn++) {
if(ag_list_sol_1[2][mn][rol_4]!=ag_list_sol_1[1][mn][rol_4]) {
equal=0;
} }
if((equal==1) && (iteration_1>0)) // Dikkat 1
{
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
ag_crewchange_1[2][weeks_4][rol_4]= ag_crewchange_1[1][weeks_4][rol_4];
} }
else {
equal_1=1;
for( mn=0;mn<weeks_to_plan_1;mn++) {
if(ag_list_sol_1[2][mn][rol_4]!=ag_list_sol_1[0][mn][rol_4]) {
equal_1=0;
} }
if((to_calculate_1!=0) && (equal_1==1)) {
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
ag_crewchange_1[2][weeks_4][rol_4]=ag_crewchange_1[0][weeks_4][rol_4];
} }
else {
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
ag_crewchange_1[2][weeks_4][rol_4]=0;
}
if(possible_crewchange_1.size()==0) {
boyut_2=definite_crewchange_1.size();
for( weeks_5=0;weeks_5<boyut_2;weeks_5++)
for(weeks_4=0; weeks_4<weeks_to_plan_1; weeks_4++) {
if(weeks_4==definite_crewchange_1[weeks_5]) {
ag_crewchange_1[2][weeks_4][rol_4]=1;
} } }
else {
y=possible_crewchange_1.size();
number_to_run=pow(2.0, y);
for( sayi=1; sayi<=number_to_run; sayi++) {
divide_number=number_to_run;
tracking_number=sayi-1;
feas_crewchange_1=1;
```

```
crewchange_cost=0;
evaluate_crewchange_1.clear();
consec_work_1= ag_work_zero_1[rol_4];
boyut_3=possible_crewchange_1.size();
for(weeks_4=0; weeks_4<weeks_to_plan_1; weeks_4++) {
for( weeks_5=0;weeks_5<boyut_3;weeks_5++) {
if(weeks_4==possible_crewchange_1[weeks_5]) {
divide_number=divide_number/2;
if(tracking_number/divide_number < 1) {
poss_ag_rboard_1[weeks_4]=0;
poss_ag_rdepart_1[weeks_4]=0;
}
else {
evaluate_crewchange_1.push_back(weeks_4);
poss_ag_rboard_1[weeks_4]=1;
poss_ag_rdepart_1[weeks_4]=1;
tracking_number=tracking_number - divide_number;
consec_work_1=0;
}
if(cur_ag_rboard_1[weeks_4][rol_4]== 0) {
poss_chng_ag_rboard_1[weeks_4]= poss_ag_rboard_1[weeks_4];
}
else {
poss_chng_ag_rboard_1[weeks_4]= cur_ag_rboard_1[weeks_4][rol_4] -
    poss_ag_rboard_1[weeks_4];
}
if(cur_ag_rdepart_1[weeks_4][rol_4]==0) {
poss_chng_ag_rdepart_1[weeks_4]= poss_ag_rdepart_1[weeks_4];
}
else {
poss_chng_ag_rdepart_1[weeks_4]= cur_ag_rdepart_1[weeks_4][rol_4]-
    poss_ag_rdepart_1[weeks_4];
}
crewchange_cost= crewchange_cost + (ag_board_chng_cost_1[weeks_4][rol_4]*
    poss_chng_ag_rboard_1[weeks_4]) + (ag_depart_chng_cost_1[weeks_4][rol_4]*
    poss_chng_ag_rdepart_1[weeks_4]);
} }
for(long_2=0;long_2<lambda_1;long_2++) {
poss_ag_long_work_1[weeks_4][long_2]= 0;
}
if(allocate_sol_1[2][weeks_4][rol_4][48] ==1) {
consec_work_1=consec_work_1 +1;
if(consec_work_1 > ag_max_work_1[rol_4]) {
```

```
feas_crewchange_1=0;
}
else {
for(long_2=0;long_2<consec_work_1;long_2++) {
poss_ag_long_work_1[weeks_4][long_2]=1;
} } }
else {
consec_work_1=0;
}
for(long_2=0;long_2<lambda_1;long_2++) {
if(cur_long_work_1[weeks_4][rol_4][48][long_2]==0) {
poss_chng_ag_long_work_1[weeks_4][long_2]= poss_ag_long_work_1[weeks_4][long_2];
}
else {
poss_chng_ag_long_work_1[weeks_4][rol_4]= cur_long_work_1[weeks_4][rol_4][48][
    long_2] - poss_ag_long_work_1[weeks_4][rol_4];
}
crewchange_cost= crewchange_cost + (extension_chng_cost_1[weeks_4][rol_4][48][
    long_2]*poss_chng_ag_long_work_1[weeks_4][long_2]);
//ag_lw_cost=ag_lw_cost+(extension_chng_cost[weeks_4][rol_4][48][long_2]*
    poss_chng_ag_long_work[weeks_4][long_2]);
} }
if(feas_crewchange_1==1) {
equal_2=1;
for(weeks_5=0;weeks_5<weeks_to_plan_1;weeks_5++) {
if(ag_crewchange_1[2][weeks_5][rol_4]==1) {
equal_2=0;
} }
boyut_2=definite_crewchange_1.size();
boyut_3=evaluate_crewchange_1.size();
if(equal_2==1) {
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
for(weeks_5=0;weeks_5<boyut_2;weeks_5++) {
if(weeks_4==definite_crewchange_1[weeks_5]) {
ag_crewchange_1[2][weeks_4][rol_4]=1;
} }
for( weeks_5=0;weeks_5<boyut_3;weeks_5++) {
if(weeks_4==evaluate_crewchange_1[weeks_5]) {
ag_crewchange_1[2][weeks_4][rol_4]=1;
} }
min_crewchange_cost=crewchange_cost;
} }
else {
```

```
if(crewchange_cost < min_crewchange_cost) {
boyut_2=definite_crewchange_1.size();
boyut_3=evaluate_crewchange_1.size();
for(weeks_5=0;weeks_5<weeks_to_plan_1;weeks_5++) {
ag_crewchange_1[2][weeks_5][rol_4]=0;
}
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
for(weeks_5=0;weeks_5<boyut_2;weeks_5++) {
if(weeks_4==definite_crewchange_1[weeks_5]) {
ag_crewchange_1[2][weeks_4][rol_4]=1;
} }
for(weeks_5=0;weeks_5<boyut_3;weeks_5++) {
if(weeks_4==evaluate_crewchange_1[weeks_5]) {
ag_crewchange_1[2][weeks_4][rol_4]=1;
} } }
min_crewchange_cost=crewchange_cost ;
} } } } } } }
consec_work_1=ag_work_zero_1[rol_4];
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
ag_rboard_sol_1[2][weeks_4][rol_4]= 0;
ag_rdepart_sol_1[2][weeks_4][rol_4]= 0;
if(ag_crewchange_1[2][weeks_4][rol_4]==1) {
consec_work_1=0;
if(weeks_4==0) {
if(ag_starting_1[rol_4] == 1) {
ag_rdepart_sol_1[2][weeks_4][rol_4]=1;
} }
else {
if(allocate_sol_1[2][weeks_4-1][rol_4][48]== 1) {
ag_rdepart_sol_1[2][weeks_4][rol_4]=1;
} }
if(allocate_sol_1[2][weeks_4][rol_4][48]== 1) {
ag_rboard_sol_1[2][weeks_4][rol_4]=1;
} }
for(long_2=0;long_2<lambda_1;long_2++) {
long_work_sol_1[2][weeks_4][rol_4][48][long_2]= 0;
}
if(allocate_sol_1[2][weeks_4][rol_4][48]== 1) {
consec_work_1= consec_work_1+1;
for(long_2=0;long_2<consec_work_1;long_2++) {
long_work_sol_1[2][weeks_4][rol_4][48][long_2]=1;
} }
else {
```

```
consec_work_1=0;
}
if(cur_allocate_1[weeks_4][rol_4][48]==0) {
chng_allocate_1[weeks_4][rol_4][48]=allocate_sol_1[2][weeks_4][rol_4][48];
}
else {
chng_allocate_1[weeks_4][rol_4][48]= cur_allocate_1[weeks_4][rol_4][48] -
    allocate_sol_1[2][weeks_4][rol_4][48];
}
ag_cost_1[2][weeks_4][rol_4] =ag_cost_1[2][weeks_4][rol_4] + (work_chng_cost_1[
    weeks_4][rol_4][48]*chng_allocate_1[weeks_4][rol_4][48]);
if(cur_ag_rboard_1[weeks_4][rol_4] == 0) {
chng_ag_rboard_1[weeks_4][rol_4]=ag_rboard_sol_1[2][weeks_4][rol_4];
}
else {
chng_ag_rboard_1[weeks_4][rol_4]= cur_ag_rboard_1[weeks_4][rol_4] -
    ag_rboard_sol_1[2][weeks_4][rol_4];
}
ag_cost_1[2][weeks_4][rol_4]=ag_cost_1[2][weeks_4][rol_4] + (ag_board_chng_cost_1
    [weeks_4][rol_4]*chng_ag_rboard_1[weeks_4][rol_4]);
if(cur_ag_rdepart_1[weeks_4][rol_4]== 0) {
chng_ag_rdepart_1[weeks_4][rol_4]=ag_rdepart_sol_1[2][weeks_4][rol_4];
}
else {
chng_ag_rdepart_1[weeks_4][rol_4]= cur_ag_rdepart_1[weeks_4][rol_4]-
    ag_rdepart_sol_1[2][weeks_4][rol_4];
}
ag_cost_1[2][weeks_4][rol_4] =ag_cost_1[2][weeks_4][rol_4] + (
    ag_depart_chng_cost_1[weeks_4][rol_4]*chng_ag_rdepart_1[weeks_4][rol_4]);
for(long_2=0;long_2<lambda_1;long_2++) {
if(cur_long_work_1[weeks_4][rol_4][48][long_2]==0) {
chng_long_work_1[weeks_4][rol_4][48][long_2]= long_work_sol_1[2][weeks_4][rol_4
    ][48][long_2];
}
else {
chng_long_work_1[weeks_4][rol_4][48][long_2]= cur_long_work_1[weeks_4][rol_4
    ][48][long_2]-long_work_sol_1[2][weeks_4][rol_4][48][long_2];
}
ag_cost_1[2][weeks_4][rol_4]=ag_cost_1[2][weeks_4][rol_4]+ (extension_chng_cost_1
    [weeks_4][rol_4][48][long_2]*chng_long_work_1[weeks_4][rol_4][48][long_2]);
} } } } }

void calc_cost5(int reg_emp_1,int weeks_to_plan_1, int all_roles_1,float
    emp_cost_1[8][48],float total_cost_1[8],float ag_cost_1[8][13][25],int &
    transfer_sol_from_1, int &transfer_sol_to_1, int &to_calculate_1)
{
int emp_4, rol_4, weeks_4;
for(emp_4=0;emp_4<reg_emp_1;emp_4++) {
total_cost_1[2]=total_cost_1[2]+emp_cost_1[2][emp_4];
}
for(rol_4=0;rol_4<all_roles_1;rol_4++)
for(weeks_4=0;weeks_4<weeks_to_plan_1;weeks_4++) {
total_cost_1[2]=total_cost_1[2]+ag_cost_1[2][weeks_4][rol_4];
}
transfer_sol_from_1=2;
transfer_sol_to_1=to_calculate_1;
}
```

E.2.5.3 'Compare to best' sub-programmes

```
#include "comparing.h"

void compare_to_best(int& number_best_1, int weeks_to_plan_1, int all_roles_1,
    int reg_emp_1, int role_count_1 , float total_cost_1[8], int new_best_1,int
    same_best_1[51],int emp_count_1,int list_sol_1[8][13][48], int best_sols_1
    [51][13][48], int ag_list_sol_1[8][13][25], int ag_best_sols_1[51][13][25])
{
int weeks_1,emp_1,rol_1,x;
if(total_cost_1[0]==total_cost_1[1]) {
new_best_1=1;
x=0;
while(new_best_1==1 && x<number_best_1) {
//x=x+1;       // AL - Have moved this to the end of the loop
same_best_1[x]=1;
emp_count_1=0;
while( same_best_1[x]==1 && emp_count_1<reg_emp_1) {
//emp_count=emp_count+1;     // AL - Have moved this to the end of the loop
for(weeks_1=0;weeks_1<weeks_to_plan_1;weeks_1++) {
if(list_sol_1[0][weeks_1][emp_count_1]!=best_sols_1[x][weeks_1][emp_count_1]) {
same_best_1[x]=0;
} }
emp_count_1=emp_count_1+1;          // AL - Have moved this from the beginning of
     the loop
}
// AL - Have changed from "while( same_best[x]==!1 &&..."
role_count_1=0;
while(same_best_1[x]==1 && role_count_1<all_roles_1) {
//role_count=role_count+1;   // AL - Have moved this to the end of the loop
for(weeks_1=0;weeks_1<weeks_to_plan_1;weeks_1++) {
if(ag_list_sol_1[0][weeks_1][role_count_1]!=ag_best_sols_1[x][weeks_1][
    role_count_1]) // AL - Have changed this from "...ag_best_sols[x][weeks_1][
    emp_count])"
{
same_best_1[x]=0;
} }
role_count_1=role_count_1+1; // AL - Have moved this from the beginning of the
    loop
}              // AL - Have changed this from "...&& role_count<reg_emp"
if(same_best_1[x]==1 ) {
new_best_1=0;
}
x=x+1; // AL - Have moved this from the beginning of the loop
}
if(new_best_1==1) {
number_best_1=number_best_1+1;
for(weeks_1=0;weeks_1<weeks_to_plan_1;weeks_1++)
for(emp_1=0;emp_1<reg_emp_1;emp_1++) {
best_sols_1[number_best_1-1][weeks_1][emp_1]=list_sol_1[0][weeks_1][emp_1];    //
     AL - Have changed this from "best_sols[number_best][weeks_1][emp_1]..."
}
for(weeks_1=0;weeks_1<weeks_to_plan_1;weeks_1++)
for(rol_1=0;rol_1<all_roles_1;rol_1++) {
ag_best_sols_1[number_best_1-1][weeks_1][rol_1]=ag_list_sol_1[0][weeks_1][rol_1];
     // AL - Have changed this from "ag_best_sols[number_best][weeks_1][rol_1
    ]..."
} } }          // AL - 'if current cost = best cost' should end here
}
best_comparison.txt
1 of 15 items
best_comparison.txtcost_calculation.txtData_1.txtdeneme.txtevaluate_swap.
    txtfeasibility.txtfinishing.txtinitialising.txtlist_sorting.txtmain.
    txtrandom_kick.txttabu_control.txttransfer.txtupdating.txtusable_blocks.
    txtDisplaying best_comparison.txt.
```

E.2.5.4 'Data' sub-programmes

```
#include "Load_Data.h"
#include <stdio.h>
#include <stdlib.h>
```

```
#include <time.h>
#include <conio.h>
#include "string"
#include <cstdlib>
#include <ctime>
#include <math.h>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <iterator>
#include <vector>
#include <random>
using namespace std;

void LoadRequired(int required_1[13][25])
{
int x, y;
ifstream file1("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\gerekli.txt");
if (!file1) {
cout << "Cannot open file1.\n";
return;
}
for (y = 0; y < 25; y++) {
for (x = 0; x < 13; x++) {
file1 >> required_1[x][y];
} }
file1.close();
}

void LoadEligible(int eligible_gecici_1[325][49], int eligible_1[13][25][49])
{
int x, y;
ifstream file2("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\skill.txt");
if (!file2) {
cout << "Cannot open file2.\n";
return;
}
for (y = 0; y < 49; y++) {
for (x = 0; x <325; x++) {
file2 >> eligible_gecici_1[x][y];
} }
int i,j,k,count;
count=1;
```

```
for(k=0;k<49;k++) {
j=0;
for(i=0;i<13;i++) {
eligible_1[i][j][k]=eligible_gecici_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
eligible_1[i-(13*j)][j][k]=eligible_gecici_1[i][k];
}
j++;
}
while(j<25);
};
file2.close();
}

void LoadStarting(int starting_1[25][49])
{
int x, y;
ifstream file3("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\start.txt");
if (!file3) {
cout << "Cannot open file3.\n";
return;
}
for (y = 0; y < 49; y++) {
for (x = 0; x < 25; x++) {
file3 >> starting_1[x][y];
} }
file3.close();
}

void LoadAlloc(int cur_allocate_1[13][25][49], int current_alloc_1[325][49])
{
int x, y;
ifstream file4("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\curr_allocate.txt");
if (!file4) {
cout << "Cannot open file4.\n";
return;
}
for (y = 0; y < 49; y++) {
for (x = 0; x <325; x++) {
file4 >> current_alloc_1[x][y];
```

```
} }
int i,j,k,count;
count=1;
for(k=0;k<49;k++) {
j=0;
for(i=0;i<13;i++) {
cur_allocate_1[i][j][k]=current_alloc_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
cur_allocate_1[i-(13*j)][j][k]=current_alloc_1[i][k];
}
j++;
}
while(j<25);
};
file4.close();
}

void LoadBoarding(int current_board_1[325][48], int cur_board_1[13][25][48] )
{
int x, y;
ifstream file5("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\curr_board.txt");
if (!file5) {
cout << "Cannot open file5.\n";
return;
}
for (y = 0; y < 48; y++) {
for (x = 0; x <325; x++) {
file5 >> current_board_1[x][y];
} }
int i,j,k,count;
count=1;
for(k=0;k<48;k++) {
j=0;
for(i=0;i<13;i++) {
cur_board_1[i][j][k]=current_board_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
cur_board_1[i-(13*j)][j][k]=current_board_1[i][k];
}
j++;
}
while(j<25);
};
file5.close();
}

void LoadDeparting(int current_depart_1[325][48], int cur_depart_1[13][25][48])
{
int x, y;
ifstream file6("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\curr_depart.txt");
if (!file6) {
cout << "Cannot open file6.\n";
return;
}
for (y = 0; y < 48; y++) {
for (x = 0; x <325; x++) {
file6 >> current_depart_1[x][y];
} }
int i,j,k,count;
count=1;
for(k=0;k<48;k++) {
j=0;
for(i=0;i<13;i++) {
cur_depart_1[i][j][k]=current_depart_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
cur_depart_1[i-(13*j)][j][k]=current_depart_1[i][k];
}
j++;
}
while(j<25);
};
file6.close();
}

void LoadAgBoard(int cur_ag_rboard_1[13][25])
{
int x, y;
ifstream file7("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\curr_ag_rboard.txt")
```

```
    ;
if (!file7) {
cout << "Cannot open file7.\n";
return;
}
for (y = 0; y < 25; y++) {
for (x = 0; x < 13; x++) {
file7 >> cur_ag_rboard_1[x][y];
} }
file7.close();
}

void LoadAgDepart(int cur_ag_rdepart_1[13][25])
{
int x, y;
ifstream file8("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\curr_ag_rdepart.txt
    ");
if (!file8) {
cout << "Cannot open file8.\n";
return;
}
for (y = 0; y < 25; y++) {
for (x = 0; x < 13; x++) {
file8 >> cur_ag_rdepart_1[x][y];
} }
file8.close();
}

void LoadChangeCostBoard(float board_cost_1[325][48], float board_chng_cost_1
    [13][25][48])
{
int x, y;
ifstream file9("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\boarding_change_cost
    .txt");
if (!file9) {
cout << "Cannot open file9.\n";
return;
}
for (y = 0; y < 48; y++) {
for (x = 0; x <325; x++) {
file9 >> board_cost_1[x][y];
} }
int i,j,k,count;
```

```
count=1;
for(k=0;k<48;k++) {
j=0;
for(i=0;i<13;i++) {
board_chng_cost_1[i][j][k]=board_cost_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
board_chng_cost_1[i-(13*j)][j][k]=board_cost_1[i][k];
}
j++;
}
while(j<25);
};
file9.close();
}

void LoadChangeCostDepart(float depart_cost_1[325][48], float depart_chng_cost_1
    [13][25][48])
{
int x, y;
ifstream file10("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\
    departing_change_cost.txt");
if (!file10) {
cout << "Cannot open file10.\n";
return;
}
for (y = 0; y < 48; y++) {
for (x = 0; x <325; x++) {
file10 >> depart_cost_1[x][y];
} }
int i,j,k,count;
count=1;
for(k=0;k<48;k++) {
j=0;
for(i=0;i<13;i++) {
depart_chng_cost_1[i][j][k]=depart_cost_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
depart_chng_cost_1[i-(13*j)][j][k]=depart_cost_1[i][k];
}
j++;
}
while(j<25);
};
file10.close();
}

void LoadAgChangeBoard(float ag_board_chng_cost_1[13][25])
{
int x, y;
ifstream file11("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\
    ag_board_change_cost.txt");
if (!file11) {
cout << "Cannot open file11.\n";
return;
}
for (y = 0; y < 25; y++) {
for (x = 0; x < 13; x++) {
file11 >> ag_board_chng_cost_1[x][y];
} }
file11.close();
}

void LoadAgChangeDepart(float ag_depart_chng_cost_1[13][25])
{
int x, y;
ifstream file12("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\
    ag_depart_change_cost.txt");
if (!file12) {
cout << "Cannot open file12.\n";
return;
}
for (y = 0; y < 25; y++) {
for (x = 0; x < 13; x++) {
file12 >> ag_depart_chng_cost_1[x][y];
} }
file12.close();
}

void LoadWorkChangeCost(float workchange_cost_1[325][49], float work_chng_cost_1
    [13][25][49])
{
```

```
int x, y;
ifstream file13("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\work_change_cost.
    txt");
if (!file13) {
cout << "Cannot open file13.\n";
return;
}
for (y = 0; y < 49; y++) {
for (x = 0; x <325; x++) {
file13 >> workchange_cost_1[x][y];
} }
int i,j,k,count;
count=1;
for(k=0;k<49;k++) {
j=0;
for(i=0;i<13;i++) {
work_chng_cost_1[i][j][k]=workchange_cost_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
work_chng_cost_1[i-(13*j)][j][k]=workchange_cost_1[i][k];
}
j++;
}
while(j<25);
};
file13.close();
}

void LoadInitialSolution(int initial_solution_1[325][49],int taskbased_sol_1
    [13][25][49])
{
int x, y;
ifstream file14("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\taskbasedsltn.txt")
    ;
if (!file14) {
cout << "Cannot open file14.\n";
return;
}
for (y = 0; y < 49; y++) {
for (x = 0; x <325; x++) {
file14 >>initial_solution_1[x][y];
} }
int i,j,k,count;
count=1;
for(k=0;k<49;k++) {
j=0;
for(i=0;i<13;i++) {
taskbased_sol_1[i][j][k]=initial_solution_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
taskbased_sol_1[i-(13*j)][j][k]=initial_solution_1[i][k];
}
j++;
}
while(j<25);
};
file14.close();
}

void LoadContract(int contract_1[45][4], int under_rate_1[45], int over_rate_1
    [45], int g_weeks_1[45], int exp_worktime_1[45])
{
int x, y;
ifstream file15("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\contracted.txt");
if (!file15) {
cout << "Cannot open file15.\n";
return;
}
for (y = 0; y < 4; y++) {
for (x = 0; x < 45; x++) {
file15 >> contract_1[x][y];
} }
for (x = 0; x < 45; x++) {
under_rate_1[x]=contract_1[x][0];
over_rate_1[x]=contract_1[x][1];
g_weeks_1[x]=contract_1[x][2];
exp_worktime_1[x]=contract_1[x][3];
}
file15.close();
}

void LoadConst(int crew_1[48][4], int work_zero_1[48], int rest_zero_1[48], int
    min_rest_1[48], int max_work_1[48])
{
int x, y;
ifstream file16("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\whole_crew.txt");
if (!file16) {
cout << "Cannot open file16.\n";
return;
}
for (y = 0; y < 4; y++) {
for (x = 0; x < 48; x++) {
file16 >> crew_1[x][y];
} }
for (x = 0; x < 48; x++) {
work_zero_1[x]=crew_1[x][0];
rest_zero_1[x]=crew_1[x][1];
min_rest_1[x]=crew_1[x][3];
max_work_1[x]=crew_1[x][2];
}
file16.close();
}

void LoadAgencyCrew(int ag_crew_1[25][3], int ag_work_zero_1[25], int
    ag_max_work_1[25], int ag_starting_1[25])
{
int x, y;
ifstream file17("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\agency.txt");
if (!file17) {
cout << "Cannot open file17.\n";
return;
}
for (y = 0; y < 3; y++) {
for (x = 0; x <25; x++) {
file17 >> ag_crew_1[x][y];;
} }
for (x = 0; x < 25; x++) {
ag_work_zero_1[x]=ag_crew_1[x][0];
ag_max_work_1[x]=ag_crew_1[x][1];
ag_starting_1[x]=ag_crew_1[x][2];
}
file17.close();
}

void LoadExtChgCost(float ext_chg_cost_1[325][490],float ext_chg_cost_new_1
    [13][25][490], float extension_chng_cost_1[13][25][49][10])
{
int x, y;
ifstream file18("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\extension_ch_cost.
    txt");
if (!file18) {
cout << "Cannot open file18.\n";
return;
}
for (y = 0; y < 490; y++) {
for (x = 0; x <325; x++) {
file18 >>ext_chg_cost_1[x][y];
} }
int i,j,k;
for(k=0;k<490;k++) {
j=0;
for(i=0;i<13;i++) {
ext_chg_cost_new_1[i][j][k]=ext_chg_cost_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
ext_chg_cost_new_1[i-(13*j)][j][k]=ext_chg_cost_1[i][k];
}
j++;
}
while(j<25);
};
int l;
for(i=0;i<13;i++)
for(j=0;j<25;j++) {
l=0;
for(k=0;k<49;k++) {
extension_chng_cost_1[i][j][k][l]=ext_chg_cost_new_1[i][j][k];
}
l++;
do {
for(k=49*l;k<49*(l+1);k++) {
extension_chng_cost_1[i][j][k-(49*l)][l]=ext_chg_cost_new_1[i][j][k];
}
l++;
}
while(l<10);
```

```
};
file18.close();
}

void LoadOver_under(int guaranteed_workers_1[45][2], int cur_undertime_1[45], int
     cur_overtime_1[45])
{
int x, y;
ifstream file19("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\current_time.txt");
if (!file19) {
cout << "Cannot open file19.\n";
return;
}
for (y = 0; y < 2; y++) {
for (x = 0; x < 45; x++) {
file19 >> guaranteed_workers_1[x][y];
} }
for (x = 0; x < 45; x++) {
cur_undertime_1[x]=guaranteed_workers_1[x][0];
cur_overtime_1[x]=guaranteed_workers_1[x][1];
}
file19.close();
}

void LoadCurLongWork(int cur_long_work_1[325][490],int cur_long_work_new_1
    [13][25][490], int current_long_work_1[13][25][49][10])
{
int x, y;
ifstream file20("C:\\Users\\xmb13210\\Desktop\\data_sezgisel\\curr_long_work.txt
    ");
if (!file20) {
cout << "Cannot open file20.\n";
return;
}
for (y = 0; y < 490; y++) {
for (x = 0; x <325; x++) {
file20 >>cur_long_work_1[x][y];
} }
int i,j,k;
for(k=0;k<490;k++) {
j=0;
for(i=0;i<13;i++) {
cur_long_work_new_1[i][j][k]=cur_long_work_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
cur_long_work_new_1[i-(13*j)][j][k]=cur_long_work_1[i][k];
}
j++;
}
while(j<25);
};
int l;
for(i=0;i<13;i++)
for(j=0;j<25;j++) {
l=0;
for(k=0;k<49;k++) {
current_long_work_1[i][j][k][l]=cur_long_work_new_1[i][j][k];
}
l++;
do {
for(k=49*l;k<49*(l+1);k++) {
current_long_work_1[i][j][k-(49*l)][l]=cur_long_work_new_1[i][j][k];
}
l++;
}
while(l<10);
};
file20.close();
}
```

E.2.5.5 ‘Deneme’ sub-programmes

```
#include "trial.h"
#include <string>
#include <iostream>
#include <fstream>
using namespace std;

void deneme(const std::string &filename,int ag_max_work_1[25],int ag_work_zero_1
    [25],int under_rate_1[45], int over_rate_1[45], int g_weeks_1[45], int
    exp_worktime_1[45], int required_1[13][25], int eligible_1[13][25][49], int
    starting_1[25][49], int ag_starting_1[25], int work_zero_1[48], int
    rest_zero_1[48], int min_rest_1[48], int max_work_1[48], int cur_allocate_1
    [13][25][49])
```

```
{
int i,x,y;
int eligible_gecici_1[325][49];
int current_alloc_1[325][49];
string a[20];
char b[20];
ifstream input(filename);
if (!input) {
cout << "Cannot open file.\n"<<" "<<filename<<"\n";
return;
}
string line;
for(i=0;i<55;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a[0];
input>>b[0];
for (x = 0; x < 45; x++) {
input>>under_rate_1[x];
}
input>>b[0];
input>>a[0];
input>>b[0];
for (x = 0; x < 45; x++) {
input>>over_rate_1[x];
}
input>>b[0];
input>>a[0];
input>>b[0];
for (x = 0; x < 45; x++) {
input>>g_weeks_1[x];
}
input>>b[0];
input>>a[0];
input>>b[0];
for (x = 0; x < 45; x++) {
input>>exp_worktime_1[x];
}
input>>b[0];
for(i=0;i<4;i++) {
getline( input, line );
//cout<<line<<'\n';
}
for (y = 0; y < 25; y++) {
for (x = 0; x < 13; x++) {
input>> required_1[x][y];
} }
input>>b[0];
for(i=0;i<4;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
for (y = 0; y < 49; y++) {
for (x = 0; x <325; x++) {
input >> eligible_gecici_1[x][y];
} }
int j,k,count;
count=1;
for(k=0;k<49;k++) {
j=0;
for(i=0;i<13;i++) {
eligible_1[i][j][k]=eligible_gecici_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
eligible_1[i-(13*j)][j][k]=eligible_gecici_1[i][k];
}
j++;
}
while(j<25);
};
input>>b[0];
for(i=0;i<4;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
for (y = 0; y < 49; y++) {
for (x = 0; x < 25; x++) {
input>> starting_1[x][y];
} }
input>>b[0];
getline( input, line );
input>>a[0];
input>>b[0];
```

```
for (x = 0; x < 25; x++) {
input>>ag_starting_1[x];
}
input>>b[0];
for(i=0;i<3;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a[0];
input>>b[0];
for (x = 0; x < 48; x++) {
input>> work_zero_1[x];
}
input>>b[0];
input>>a[0];
input>>b[0];
for (x = 0; x < 48; x++) {
input>>rest_zero_1[x];
}
input>>b[0];
getline( input, line );
input>>a[0];
input>>b[0];
for (x = 0; x < 48; x++) {
input>> max_work_1[x];
}
input>>b[0];
input>>a[0];
input>>b[0];
for (x = 0; x < 48; x++) {
input>> min_rest_1[x];
}
input>>b[0];
getline( input, line );
getline( input, line );
input>>a[0];
input>>b[0];
for (x = 0; x < 25; x++) {
input>>ag_work_zero_1[x];
}
input>>b[0];
input>>a[0];
input>>b[0];
```

```
for (x = 0; x < 25; x++) {
input>>ag_max_work_1[x];
}
input>>b[0];
for(i=0;i<6;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a[0];
input>>b[0];
for (y = 0; y < 49; y++) {
for (x = 0; x <325; x++) {
input >> current_alloc_1[x][y];
} }
input>>b[0];
for(i=0;i<3;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
count=1;
for(k=0;k<49;k++) {
j=0;
for(i=0;i<13;i++) {
cur_allocate_1[i][j][k]=current_alloc_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
cur_allocate_1[i-(13*j)][j][k]=current_alloc_1[i][k];
}
j++;
}
while(j<25);
};
input.close();
}

void deneme2(const std::string &filename,int cur_board_1[13][25][48], int
    cur_depart_1[13][25][48], int cur_ag_rboard_1[13][25], int cur_ag_rdepart_1
    [13][25])
{
int i,x,y,j,k,count;
int current_board_1[325][48];
int current_depart_1[325][48];
string a[20];
char b[20];
ifstream input(filename);
if (!input) {
cout << "Cannot open file.\n"<<" "<<filename<<"\n";
return;
}
string line;
for(i=0;i<263;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a[0];
input>>b[0];
for (y = 0; y < 48; y++) {
for (x = 0; x <325; x++) {
input >> current_board_1[x][y];
} }
for(i=0;i<5;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
count=1;
for(k=0;k<48;k++) {
j=0;
for(i=0;i<13;i++) {
cur_board_1[i][j][k]=current_board_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
cur_board_1[i-(13*j)][j][k]=current_board_1[i][k];
}
j++;
}
while(j<25);
};
for (y = 0; y < 48; y++) {
for (x = 0; x <325; x++) {
input >> current_depart_1[x][y];
} }
for(i=0;i<5;i++) {
```

```
getline( input, line );
//   cout<<line<<'\n';
}
count=1;
for(k=0;k<48;k++) {
j=0;
for(i=0;i<13;i++) {
cur_depart_1[i][j][k]=current_depart_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
cur_depart_1[i-(13*j)][j][k]=current_depart_1[i][k];
}
j++;
}
while(j<25);
};
for (y = 0; y < 25; y++) {
for (x = 0; x < 13; x++) {
input>> cur_ag_rboard_1[x][y];
} }
input>>b[0];
input>>a[0];
input>>b[0];
for (y = 0; y < 25; y++) {
for (x = 0; x < 13; x++) {
input>> cur_ag_rdepart_1[x][y];
} }
input.close();
}

void deneme3(const std::string &filename,int cur_undertime_1[45], int
    cur_overtime_1[45], int cur_long_work_1[325][490],int cur_long_work_new_1
    [13][25][490], int current_long_work_1[13][25][49][10])
{
int i,x,y,j,k,count;
string a[20];
char b[20];
ifstream input(filename);
if (!input) {
cout << "Cannot open file.\n"<<" "<<filename<<"\n";
return;
```

```
}
string line;
for(i=0;i<425;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a[0];
input>>b[0];
for (x = 0; x < 45; x++) {
input>> cur_undertime_1[x];
}
input>>b[0];
input>>a[0];
input>>b[0];
for (x = 0; x < 45; x++) {
input>> cur_overtime_1[x];
}
input>>b[0];
for(i=0;i<3;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a[0];
input>>b[0];
for (y = 0; y < 490; y++) {
for (x = 0; x <325; x++) {
input >>cur_long_work_1[x][y];
} }
for(k=0;k<490;k++) {
j=0;
for(i=0;i<13;i++) {
cur_long_work_new_1[i][j][k]=cur_long_work_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
cur_long_work_new_1[i-(13*j)][j][k]=cur_long_work_1[i][k];
}
j++;
}
while(j<25);
};
int l;
```

```
for(i=0;i<13;i++)
for(j=0;j<25;j++) {
l=0;
for(k=0;k<49;k++) {
current_long_work_1[i][j][k][l]=cur_long_work_new_1[i][j][k];
}
l++;
do {
for(k=49*l;k<49*(l+1);k++) {
current_long_work_1[i][j][k-(49*l)][l]=cur_long_work_new_1[i][j][k];
}
l++;
}
while(l<10);
};
input.close();
}

void deneme4(const std::string &filename,float board_cost_1[325][48], float
    board_chng_cost_1[13][25][48], float depart_cost_1[325][48], float
    depart_chng_cost_1[13][25][48], float ag_board_chng_cost_1[13][25], float
    ag_depart_chng_cost_1[13][25])
{
int i,x,y,j,k,count;
string a;
char b;
ifstream input(filename);
if (!input) {
cout << "Cannot open file.\n"<<" "<<filename<<"\n";
return;
}
string line;
for(i=0;i<936;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a;
input>>b;
for (y = 0; y < 48; y++) {
for (x = 0; x <325; x++) {
input >> board_cost_1[x][y];
} }
count=1;
```

```
for(k=0;k<48;k++) {
j=0;
for(i=0;i<13;i++) {
board_chng_cost_1[i][j][k]=board_cost_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
board_chng_cost_1[i-(13*j)][j][k]=board_cost_1[i][k];
}
j++;
}
while(j<25);
};
input>>b;
getline( input, line );
input>>a;
input>>b;
for (y = 0; y < 48; y++) {
for (x = 0; x <325; x++) {
input>> depart_cost_1[x][y];
} }
count=1;
for(k=0;k<48;k++) {
j=0;
for(i=0;i<13;i++) {
depart_chng_cost_1[i][j][k]=depart_cost_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
depart_chng_cost_1[i-(13*j)][j][k]=depart_cost_1[i][k];
}
j++;
}
while(j<25);
};
input>>b;
getline( input, line );
getline( input, line );
input>>a;
input>>b;
for (y = 0; y < 25; y++) {
```

```
for (x = 0; x < 13; x++) {
input>> ag_board_chng_cost_1[x][y];
} }
input>>b;
getline( input, line );
input>>a;
input>>b;
for (y = 0; y < 25; y++) {
for (x = 0; x < 13; x++) {
input>>ag_depart_chng_cost_1[x][y];
} }
input.close();
}

void deneme5(const std::string &filename,float workchange_cost_1[325][49], float
    work_chng_cost_1[13][25][49], float ext_chg_cost_1[325][490],float
    ext_chg_cost_new_1[13][25][490], float extension_chng_cost_1[13][25][49][10])
{
int i,x,y,j,k,count;
string a;
char b;
ifstream input(filename);
if (!input) {
cout << "Cannot open file.\n"<<" "<<filename<<"\n";
return;
}
string line;
for(i=0;i<1095;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a;
input>>b;
for (y = 0; y < 49; y++) {
for (x = 0; x <325; x++) {
input>> workchange_cost_1[x][y];
} }
count=1;
for(k=0;k<49;k++) {
j=0;
for(i=0;i<13;i++) {
work_chng_cost_1[i][j][k]=workchange_cost_1[i][k];
}
```

```
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
work_chng_cost_1[i-(13*j)][j][k]=workchange_cost_1[i][k];
}
j++;
}
while(j<25);
};
input>>b;
for(i=0;i<3;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a;
input>>b;
for (y = 0; y < 490; y++) {
for (x = 0; x <325; x++) {
input>>ext_chg_cost_1[x][y];
} }
for(k=0;k<490;k++) {
j=0;
for(i=0;i<13;i++) {
ext_chg_cost_new_1[i][j][k]=ext_chg_cost_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
ext_chg_cost_new_1[i-(13*j)][j][k]=ext_chg_cost_1[i][k];
}
j++;
}
while(j<25);
};
int l;
for(i=0;i<13;i++)
for(j=0;j<25;j++) {
l=0;
for(k=0;k<49;k++) {
extension_chng_cost_1[i][j][k][l]=ext_chg_cost_new_1[i][j][k];
}
l++;
do {
```

```
for(k=49*l;k<49*(l+1);k++) {
extension_chng_cost_1[i][j][k-(49*l)][l]=ext_chg_cost_new_1[i][j][k];
}
l++;
}
while(l<10);
};
input>>b;
input.close();
}

void deneme6(const std::string &filename,int initial_solution_1[326][49],int
    taskbased_sol_1[13][25][49])
{  int i,x,y,j,k,count;
string a[20];
char b;
ifstream input(filename);
if (!input) {
cout << "Cannot open file.\n"<<" "<<filename<<"\n";
return;
}
string line;
for(i=0;i<13;i++) {
getline( input, line );
//   cout<<line<<'\n';
}
input>>a[0];
input>>b;
for (y = 0; y < 49; y++) {
for (x = 0; x <325; x++) {
input>>initial_solution_1[x][y];
} }
count=1;
for(k=0;k<49;k++) {
j=0;
for(i=0;i<13;i++) {
taskbased_sol_1[i][j][k]=initial_solution_1[i][k];
}
j++;
do {
for(i=13*j;i<13*(j+1);i++) {
taskbased_sol_1[i-(13*j)][j][k]=initial_solution_1[i][k];
}
```

```
j++;
}
while(j<25);
};
input>>b;
input.close();
}
```

E.2.5.6 'Evaluate swap' sub-programmes

```
#include "swapping.h"

void swap_calc1( vector<int>& ag_roles_changed_1, vector<int>& emps_changed_1,int
     reg_emp_1,int weeks_to_plan_1, int list_sol_1[8][13][48],int all_roles_1,int
     ag_list_sol_1[8][13][25], int& emp_extend_1, int& block_start_1,int&
    swap_block_start_1, int min_rest_1[48], int& swap_block_end_1, int&
    swap_task_1, int& block_end_1, int& swap_emp_1,int& task_extend_1, int&
    to_check_tabu_1 )
{
int emp_9, weeks_9, rol_9;
emps_changed_1.clear();
ag_roles_changed_1.clear();
for(emp_9=0;emp_9<reg_emp_1;emp_9++)
for(weeks_9=0;weeks_9<weeks_to_plan_1;weeks_9++) {
list_sol_1[5][weeks_9][emp_9]=list_sol_1[0][weeks_9][emp_9];
}
for(rol_9=0;rol_9<all_roles_1;rol_9++)
for(weeks_9=0;weeks_9<weeks_to_plan_1;weeks_9++) {
ag_list_sol_1[5][weeks_9][rol_9]=ag_list_sol_1[0][weeks_9][rol_9];
}
for(weeks_9=0;weeks_9<weeks_to_plan_1;weeks_9++) {
list_sol_1[5][weeks_9][emp_extend_1]=-1;
}
emps_changed_1.push_back(emp_extend_1); // AL - Have added this new line
for(weeks_9=0;weeks_9<weeks_to_plan_1;weeks_9++) {
if((weeks_9 < block_start_1) && (weeks_9<swap_block_start_1)) {
if((swap_block_start_1 < block_start_1) && (weeks_9>= (swap_block_start_1 -
    min_rest_1[emp_extend_1])) && (list_sol_1[0][weeks_9][emp_extend_1]!=-1)) {
list_sol_1[5][weeks_9][emp_extend_1]=-1;
ag_list_sol_1[5][weeks_9][list_sol_1[0][weeks_9][emp_extend_1]]=1; // AL - Have
    changed this from "ag_list_sol[0][w][list_sol[0][weeks_9][emp_extend]]=1;"
ag_roles_changed_1.push_back(list_sol_1[0][weeks_9][emp_extend_1]);
}
```

```
else {
list_sol_1[5][weeks_9][emp_extend_1]=list_sol_1[0][weeks_9][emp_extend_1];
} }
else if((weeks_9>= swap_block_start_1) && (weeks_9<= swap_block_end_1)) {
list_sol_1[5][weeks_9][emp_extend_1]=swap_task_1;
if(list_sol_1[0][weeks_9][emp_extend_1]!=-1 && (weeks_9 < block_start_1 ||
    weeks_9> block_end_1)) {
ag_list_sol_1[5][weeks_9][list_sol_1[0][weeks_9][emp_extend_1]]=1;
ag_roles_changed_1.push_back(list_sol_1[0][weeks_9][emp_extend_1]);
} }
else if(weeks_9 >= block_start_1 && weeks_9 <= block_end_1) {
list_sol_1[5][weeks_9][emp_extend_1]=-1;
}
else {
if((swap_block_end_1 > block_end_1) && (weeks_9<= (swap_block_end_1 + min_rest_1[
    emp_extend_1])) && list_sol_1[0][weeks_9][emp_extend_1]!=-1) {
list_sol_1[5][weeks_9][emp_extend_1]=-1;
ag_list_sol_1[5][weeks_9][list_sol_1[0][weeks_9][emp_extend_1]]=1;
ag_roles_changed_1.push_back(list_sol_1[0][weeks_9][emp_extend_1]);
}
else {
list_sol_1[5][weeks_9][emp_extend_1]=list_sol_1[0][weeks_9][emp_extend_1];     //
     AL - Have changed this from "...=1;"
} } }
if(swap_emp_1!=48) {
emps_changed_1.push_back(swap_emp_1);
for(weeks_9=0;weeks_9<weeks_to_plan_1;weeks_9++) {
list_sol_1[5][weeks_9][swap_emp_1]=-1;
if(weeks_9< block_start_1 && weeks_9<swap_block_start_1) {
if((block_start_1 < swap_block_start_1 || swap_task_1 == -1) && (weeks_9>= (
    block_start_1 - min_rest_1[swap_emp_1])) && (list_sol_1[0][weeks_9][
    swap_emp_1]!=-1)) // AL - Have changed this from "...&& list_sol[0][weeks_9][
    emp_extend]!=-1)"
{
list_sol_1[5][weeks_9][swap_emp_1]=-1;
ag_list_sol_1[5][weeks_9][list_sol_1[0][weeks_9][swap_emp_1]]=1;
ag_roles_changed_1.push_back(list_sol_1[0][weeks_9][swap_emp_1]);
}
else {
list_sol_1[5][weeks_9][swap_emp_1]=list_sol_1[0][weeks_9][swap_emp_1];  // AL -
    Have changed this from "...=1;"
} }
else if(weeks_9 >= block_start_1 && weeks_9 <= block_end_1) {
```

```
list_sol_1[5][weeks_9][swap_emp_1]=task_extend_1;
if(list_sol_1[0][weeks_9][swap_emp_1]!=-1 && (weeks_9< swap_block_start_1 ||
    weeks_9> swap_block_end_1)) {
ag_list_sol_1[5][weeks_9][list_sol_1[0][weeks_9][swap_emp_1]]=1;
ag_roles_changed_1.push_back(list_sol_1[0][weeks_9][swap_emp_1]);
} }
else if(weeks_9 >= swap_block_start_1 && weeks_9<= swap_block_end_1) {
list_sol_1[5][weeks_9][swap_emp_1]=-1;
}
else {
if((block_end_1> swap_block_end_1|| swap_task_1==-1) && weeks_9 <= (block_end_1 +
     min_rest_1[swap_emp_1]) && list_sol_1[0][weeks_9][swap_emp_1]!=-1)  // AL -
    Have changed this from "...&& list_sol[0][weeks_9][emp_extend]!=-1)"
{
list_sol_1[5][weeks_9][swap_emp_1]= -1;
ag_list_sol_1[5][weeks_9][list_sol_1[0][weeks_9][swap_emp_1]]= 1;       // AL -
    Have changed this from "ag_list_sol[5][w][list_sol[0][weeks_9][emp_extend]]=
    1;"
ag_roles_changed_1.push_back(list_sol_1[0][weeks_9][swap_emp_1]);       // AL -
    Have changed this from "ag_roles_changed.push_back(list_sol[0][weeks_9][
    emp_extend]);"
}
else {
list_sol_1[5][weeks_9][swap_emp_1]=list_sol_1[0][weeks_9][swap_emp_1]; // AL -
    Have changed this from "list_sol[5][weeks_9][emp_extend]=list_sol[0][weeks_9
    ][emp_extend];"
} } } }
else {
for(weeks_9=block_start_1;weeks_9<=block_end_1;weeks_9++)  // AL - Have changed
    this from "...weeks_9<block_end;weeks_9++)"
{
if(weeks_9>=0) {
ag_list_sol_1[5][weeks_9][task_extend_1]=1;//dikkat 4
} }
ag_roles_changed_1.push_back(task_extend_1);
if(swap_task_1!=-1) {
for(weeks_9=swap_block_start_1;weeks_9<=swap_block_end_1;weeks_9++) {
ag_list_sol_1[5][weeks_9][swap_task_1]=0;
ag_roles_changed_1.push_back(swap_task_1);
} } }
to_check_tabu_1=5;
}
```

```
void evaluate_swap1(int& ag_swappable_1, int& block_start_1, int& block_end_1,
    int eligible_1[13][25][49], int& task_extend_1)
{
int weeks_10xy;
ag_swappable_1=1;
for(weeks_10xy=block_start_1;weeks_10xy<block_end_1;weeks_10xy++) {
if(eligible_1[weeks_10xy][task_extend_1][48]< 1) {
ag_swappable_1=0;
} } }

void evaluate_swap2(int& swap_allowed_1,int& too_early_1, int& do_swap_1, int&
    swap_emp_1, int& swap_task_1,int& swap_block_start_1,int& swap_block_end_1,
    int& swap_block_len_1,int& swap_find_time_1, int& swap_block_found_1 )
{
if(do_swap_1==0) {
swap_emp_1=-1;
swap_task_1=-1;
swap_block_start_1=-1;
swap_block_end_1=-1;
swap_block_len_1=0;
swap_find_time_1=-1;
swap_block_found_1=0;
too_early_1=0;
swap_allowed_1=1;
} }

void evaluate_swap6(int& swap_allowed_1,int& too_early_1, int& do_swap_1, int&
    swap_emp_1, int& swap_task_1,int& swap_block_start_1,int& swap_block_end_1,
    int& swap_block_len_1,int& swap_block_found_1 )
{
swap_block_found_1=0;
if(do_swap_1==0) {
too_early_1=0;
swap_allowed_1=1;
swap_block_start_1=-1;
swap_block_end_1=-1;
swap_block_len_1=0;
swap_emp_1=-1;
swap_task_1=-1;
} }

void evaluate_swap3(int& weeks_10x, int& rol_10x,int& swap_block_start_1,int&
    swap_task_1,int& swap_emp_1,int& swap_allowed_1,int eligible_1[13][25][49] ,
    int min_rest_1[48],int starting_1[25][49], int& emp_extend_1, int all_roles_1
    ,int summation_1x, int summation_1y, int& too_early_1,int& swap_new_block_1,
    int& swap_block_found_1, int ag_list_sol_1[8][13][25], int&
    swap_block_latest_1, int& swap_block_earliest_1)
{
int rol_11;
swap_new_block_1=0;
if((swap_block_found_1==0 && ag_list_sol_1[0][weeks_10x][rol_10x]==1) &&
    weeks_10x<= swap_block_latest_1) {
if(weeks_10x< swap_block_earliest_1) {
too_early_1=1;
}
///9 aralik
summation_1x=0;
summation_1y=0;
for(rol_11=0; rol_11<all_roles_1;rol_11++) {
if(rol_11==rol_10x)          // AL - Have added this IF statement...
{
summation_1x=summation_1x+starting_1[rol_11][emp_extend_1];
}
else           // AL - ... and have added this 'else' in order to calculate "
    summation_1" as well
{
summation_1y=summation_1y+starting_1[rol_11][emp_extend_1];
} }
if((summation_1x>0) && (weeks_10x>=1) && (weeks_10x< min_rest_1[emp_extend_1]))
    // AL - Have changed this from "if((summation > 0 && weeks_10>=1) && (
    weeks_10<= min_rest[emp_extend]))"
{
too_early_1=1;
}
if(summation_1y > 0 && weeks_10x< min_rest_1[emp_extend_1])      // AL - Have
    added this statement which uses the calculated "summation_1" quantity
{
too_early_1=1;
}
if(eligible_1[weeks_10x][rol_10x][emp_extend_1] < 1) {
swap_allowed_1=0;
}
swap_new_block_1=1;
swap_emp_1=48;
swap_task_1=rol_10x;
swap_block_start_1=weeks_10x;
```

```
} }

void evaluate_swap4(int& weeks_10x, int& swap_block_found_1,int& swap_block_len_1
    , int& swap_block_end_1,int& swap_allowed_1, int eligible_1[13][25][49],int
    min_rest_1[48], int& too_early_1,int& emp_extend_1, int&
    swap_block_earliest_1,int starting_1[25][49],int summation_1x, int
    summation_1y, int all_roles_1, int& swap_block_latest_1, int& do_swap_1, int&
     swap_task_1, int& swap_block_start_1, int& rol_10x)
{
int rol_11;
if(weeks_10x <= swap_block_latest_1 && do_swap_1==0) {
swap_task_1=rol_10x;
swap_block_start_1=weeks_10x;
summation_1x=0;       // AL - Have added in clearing the values of "summation" and
     "summation_1" as they were likely also caluclated earlier
summation_1y=0;
for(rol_11=0;rol_11<all_roles_1;rol_11++) {
if(rol_11==rol_10x) {
summation_1x=summation_1x+starting_1[rol_11][emp_extend_1];
}
else {
summation_1y=summation_1y+starting_1[rol_11][emp_extend_1];
} }
if(weeks_10x < swap_block_earliest_1) {
too_early_1=1;
//     summation=0;  // AL - Have removed this line - don't think it is needed
    here
}
else if ((summation_1x > 0) && (weeks_10x>= 1) && (weeks_10x < min_rest_1[
    emp_extend_1]))  // AL - Have changed this from "if (summation > 0 &&
    weeks_10>= 2 && weeks_10<= min_rest[emp_extend])"
{
too_early_1=1;
}
else if(summation_1y > 0 && weeks_10x < min_rest_1[emp_extend_1])       // AL -
    Have added this statement which uses the calculated "summation_1" quantity
{
too_early_1=1;
}
else {
too_early_1=0;
}             // AL - The 'else' part of the split 'else / if' statement ends
    here
```

```
if(eligible_1[weeks_10x][rol_10x][emp_extend_1]< 1) {
swap_allowed_1=0;
}
else {
swap_allowed_1=1;
}
swap_block_end_1=-1;
swap_block_len_1=0;
}
else {
swap_block_found_1=0;
} }

void evaluate_swap5(int& swap_allowed_1,int& emp_extend_1,int& rol_10x,int&
    weeks_10x, int& swap_block_latest_1, int& swap_block_found_1, int eligible_1
    [13][25][49])
{
if(weeks_10x> swap_block_latest_1) {
swap_block_found_1=0;
}
else {
if(eligible_1[weeks_10x][rol_10x][emp_extend_1] < 1) {
swap_allowed_1=0;
} } }

void evaluate_swap7(int& swap_new_block_1, int& swap_block_found_1)
{
if(swap_new_block_1==1) {
swap_block_found_1=1;
swap_new_block_1=0;
} }

void evaluate_swap8(int& weeks_10x, int& rol_10x,int& swap_block_end_1, int&
    swap_block_start_1,int& swap_block_len_1,int& swap_allowed_1, int eligible_1
    [13][25][49], int& emp_extend_1)
{
if(eligible_1[weeks_10x][rol_10x][emp_extend_1] < 1) {
swap_allowed_1=0;
}
swap_block_end_1=weeks_10x;
swap_block_len_1=(swap_block_end_1 - swap_block_start_1) +1;
}
```

```
void evaluate_swap9(int& swap_emp_1, int& swap_task_1, int& swap_block_start_1,
    int& block_start_1, int& swap_block_end_1, int& block_end_1 )
{
swap_emp_1=48;
swap_task_1=-1;
swap_block_start_1=block_start_1;
swap_block_end_1=block_end_1;
}

void evaluate_swap10(int min_rest_1[48], int work_zero_1[48],int starting_1
    [25][49],int all_roles_1,int summation_1x, int summation_1y,int eligible_1
    [13][25][49],int& task_extend_1,int& block_end_1,int rest_zero_1[48],int&
    block_start_1,int max_work_1[48],int& block_len_1, int reg_emp_1, int
    swappable_emp_1[48], int& emp_extend_1 )
{
int emp_11, rol_11, weeks_100x;
for(emp_11=0;emp_11<reg_emp_1;emp_11++) {
swappable_emp_1[emp_11]=1;//10 aralik
if(emp_11==emp_extend_1) {
swappable_emp_1[emp_11]=0;
}
if(block_len_1 > max_work_1[emp_11]) {
swappable_emp_1[emp_11]=0;
}
if(block_start_1+1 <=rest_zero_1[emp_11])  // AL - Have changed this from "if(
    block_start <= rest_zero[emp_10])"
{
swappable_emp_1[emp_11]=0;
}
// AL - Have added this quantity to take into account that "block_start" may be <
     0, and so eligible will not be defined for this.
if(block_start_1 >=0)               //     I am assuming this would cause an
    error when the programme is run - if it is able to handle the 'out of range'
    index then
{                            //     no need to make this change.
//revised_start=block_start_1;
for (weeks_100x=block_start_1;weeks_100x<=block_end_1;weeks_100x++) // AL - Have
    changed this from "for (weeks_10=block_start;weeks_10<=block_end;weeks_10++)"
     - see above for reason
{
if(eligible_1[weeks_100x][task_extend_1][emp_11] < 1) {
swappable_emp_1[emp_11]=0;
} } }
```

```
summation_1x=0;
summation_1y=0;
for(rol_11=0;rol_11<all_roles_1;rol_11++) {
if(rol_11!=task_extend_1) {
summation_1x=summation_1x+starting_1[rol_11][emp_11];
}
else {// AL - Have added this 'else' bracket so as to combine the two
    calculations...
summation_1y=summation_1y+starting_1[rol_11][emp_11];
} }
if(summation_1x> 0 && (block_start_1+1 <= min_rest_1[emp_11]))   // AL - Have
    changed this from "...&& block_start <= min_rest..."
{
swappable_emp_1[emp_11]=0;
}
if(summation_1y> 0) {
if(block_start_1>=1 && block_start_1+1 <=min_rest_1[emp_11]) // AL - Have changed
     this from "...&& block_start <= min_rest..."
{
swappable_emp_1[emp_11]=0;
}
if((block_start_1<= 0) && ((block_len_1 + work_zero_1[emp_11]) > max_work_1[
    emp_11]))  // AL - Have changed this from "if(block_start <= 0 &&..."
{
swappable_emp_1[emp_11]=0;
} } } }

void evaluate_swap11(int& swap_find_time_1, int& all_rest_1,int& swap_allowed_1,
    int& too_early_1,int& swap_block_found_1, int& swap_block_len_1, int&
    do_swap_1, int& swap_emp_1, int& swap_vessel_1, int& swap_task_1, int&
    swap_block_start_1, int& swap_block_end_1 )
{
if(do_swap_1==0) {
swap_emp_1=-1;
swap_task_1=-1;
swap_vessel_1=-1;
swap_block_start_1=-1;
swap_block_end_1=-1;
swap_block_len_1=0;
swap_find_time_1=0;
swap_block_found_1=0;
too_early_1=0;
swap_allowed_1=1;
all_rest_1=1;
} }

void evaluate_swap12(int& swap_vessel_1,int roles_1[25],int& swap_block_start_1,
    int& swap_emp_1, int& swap_task_1,int& swap_allowed_1, int eligible_1
    [13][25][49],int min_rest_1[48],int& emp_extend_1, int starting_1[25][49],int
     all_roles_1, int summation_1x, int summation_1y, int& too_early_1,int&
    swap_block_found_1,int& swap_new_block_1, int& all_rest_1,int& rol_10x, int&
    emp_10x, int list_sol_1[8][13][48],int& swap_block_earliest_1, int&
    swap_find_time_1, int& swap_block_latest_1 )
{
int rol_11;
rol_10x=list_sol_1[0][swap_find_time_1][emp_10x];
if((rol_10x!=-1) && (swap_find_time_1 >= swap_block_earliest_1) && (
    swap_find_time_1 <= swap_block_latest_1)) {
all_rest_1=0;
}
swap_new_block_1=0;
//}
if((swap_block_found_1==0) && (rol_10x!=-1) && (swap_find_time_1<=
    swap_block_latest_1)) {
if(swap_find_time_1 <swap_block_earliest_1) {
too_early_1=1;
}
summation_1x=0;
summation_1y=0;       // AL - Have moved this statement in from below...
for(rol_11=0;rol_11<all_roles_1;rol_11++) {
if(rol_11==rol_10x) {
summation_1x=summation_1x+starting_1[rol_11][emp_extend_1];
}
else           // AL - ... and have added this 'else' to combine the "summation_1"
     calculation from below...
{
summation_1y=summation_1y+starting_1[rol_11][emp_extend_1];
}
}
if((summation_1x> 0) && (swap_find_time_1 >=1) && (swap_find_time_1 +1<=
    min_rest_1[emp_extend_1])) // AL - Have changed this from "...swap_find_time
    <= min_rest[emp_extend]) "
{
too_early_1=1;
}
if(summation_1y> 0 && swap_find_time_1 <min_rest_1[emp_extend_1])  // AL - Have
```

```
    changed this from "...&& swap_find_time<= min_rest[emp_extend]) "
{
too_early_1=1;
}
if(eligible_1[swap_find_time_1][rol_10x][emp_extend_1]< 1) {
swap_allowed_1=0;
}
swap_new_block_1=1;
swap_emp_1=emp_10x;
swap_task_1=rol_10x;  // AL - Have changed this from "swap_task=rol_11"
swap_block_start_1=swap_find_time_1;
for(rol_11=0;rol_11<all_roles_1;rol_11++) {
if(rol_10x==roles_1[rol_11]) {
swap_vessel_1=rol_11;
rol_11=all_roles_1;
} } }  }

void evaluate_swap13(int& vessel_extend_1, int roles_1[25],int& swap_task_1, int&
     swap_allowed_1,int eligible_1[13][25][49] ,int& rol_10x, int& emp_extend_1,
    int& swap_vessel_1, int& link_to_vessel_1, int& swap_find_time_1, int&
    swap_block_latest_1, int& swap_block_found_1)
{
if(swap_vessel_1!=-1) {
if(rol_10x==roles_1[vessel_extend_1]) {
link_to_vessel_1=1;
} }
if(link_to_vessel_1==1) {
if(swap_find_time_1 > swap_block_latest_1) {
swap_block_found_1=0;
}
else {
if(eligible_1[swap_find_time_1][rol_10x][emp_extend_1] < 1) {
swap_allowed_1=0;
}
else {
swap_task_1=rol_10x;
} } } }

void evaluate_swap14(int& swap_block_found_1,int& swap_block_len_1,int&
    swap_block_end_1, int& swap_vessel_1,int& swap_allowed_1,int eligible_1
    [13][25][49] ,int min_rest_1[48],int starting_1[25][49], int& emp_extend_1,
    int roles_1[25],int all_roles_1,int summation_1x, int summation_1y,int&
    too_early_1,int& swap_find_time_1, int& swap_block_latest_1, int& do_swap_1,
    int& swap_task_extend_1, int& rol_10x, int& swap_block_start_1, int&
    swap_block_earliest_1)
{
int rol_11;
if(swap_find_time_1 <= swap_block_latest_1 && do_swap_1==0) {
swap_task_extend_1=rol_10x;
swap_block_start_1=swap_find_time_1;
if(swap_find_time_1 < swap_block_earliest_1) {
too_early_1=1;
}
else   // AL - Have had to split the 'else' and 'if' statements to incorporate
    the calculations
{
summation_1x=0;
summation_1y=0;
for(rol_11=0;rol_11<all_roles_1;rol_11++) {
if(rol_10x==roles_1[rol_11]) {
summation_1x=summation_1x+starting_1[rol_11][emp_extend_1];
}
else   // AL - Have combined the two parts of the calculation into the one loop
{
summation_1y=summation_1y+starting_1[rol_11][emp_extend_1];      // AL - Have
    changed this from "...+starting[v][emp_extend]"
} }
if((summation_1x>0) && (swap_find_time_1>=1) && (swap_find_time_1+1<= min_rest_1[
    emp_extend_1])) // AL - Have changed this from "...&& swap_find_time <=
    min_rest[emp_extend]) "
{
too_early_1=1; // AL - Have changed this from "too_early=0;"
}
else if((summation_1y > 0) && (swap_find_time_1+1<=min_rest_1[emp_extend_1]))  //
     AL - Have changed this from "...&& swap_find_time <= min_rest[emp_extend]) "
{
too_early_1=1; // AL - Have changed this from "too_early=0;"
}
else {
too_early_1=0;
} }           // AL - The 'else' part of the split 'else / if' statement ends
    here
if(eligible_1[swap_find_time_1][rol_10x][emp_extend_1]<1) {
swap_allowed_1=0;
}
else {
```

```
swap_allowed_1=1;
}
for(rol_11=0;rol_11<all_roles_1;rol_11++) {
if(rol_10x ==roles_1[rol_11]) {
swap_vessel_1=rol_11;
} }
swap_block_end_1=-1;
swap_block_len_1=0;
}
else {
swap_block_found_1=0;
} }

void evaluate_swap15(int& rol_10x, int& swap_find_time_1, int&
    swap_block_latest_1, int& swap_block_found_1, int& swap_allowed_1, int
    eligible_1[13][25][49], int& emp_extend_1)
{
if(swap_find_time_1 > swap_block_latest_1) {
swap_block_found_1=0;
}
else {
if((swap_allowed_1==1) && (eligible_1[swap_find_time_1][rol_10x][emp_extend_1] <
    1)) {
swap_allowed_1=0;
} } }

void evaluate_swap16(int& swap_emp_1, int& swap_task_1, int& swap_vessel_1, int&
    swap_block_found_1, int& do_swap_1, int& too_early_1, int& swap_allowed_1,
    int& swap_block_start_1, int& swap_block_end_1, int& swap_block_len_1 )
{
swap_block_found_1= 0;
if(do_swap_1==0) {
too_early_1=0;
swap_allowed_1=1;
swap_block_start_1=-1;
swap_block_end_1=-1;
swap_block_len_1=0;
swap_emp_1=-1;
swap_task_1=-1;
swap_vessel_1=-1;
} }

void evaluate_swap17(int& swap_new_block_1, int& swap_block_found_1)
```

```
{
if(swap_new_block_1==1) {
swap_block_found_1=1;
swap_new_block_1=0;
} }

void evaluate_swap18(int& swap_block_start_1, int& swap_block_len_1,int&
    swap_block_end_1, int& swap_find_time_1, int& swap_allowed_1, int eligible_1
    [13][25][49], int& rol_10x, int& emp_extend_1)
{
if(eligible_1[swap_find_time_1][rol_10x][emp_extend_1]< 1) {
swap_allowed_1=0;
}
swap_block_end_1=swap_find_time_1;
swap_block_len_1=(swap_block_end_1 - swap_block_start_1) +1;
}

void evaluate_swap19(int& swap_emp_1, int& emp_10x, int& swap_task_1, int&
    swap_block_start_1, int& block_start_1, int& swap_block_end_1, int&
    block_end_1 )
{
swap_emp_1=emp_10x;
swap_task_1=-1;
swap_block_start_1=block_start_1;
swap_block_end_1=block_end_1;
}
```

E.2.5.7 'Feasibility check' sub-programmes

```
#include "feasibility_checking.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "string"
#include <cstdlib>
#include <ctime>
#include <math.h>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <iterator>
#include <vector>
```

```
#include <random>
using namespace std;

void JC_feas(int &feasible_1, int &JCfeas_1,int all_emp_1,int all_roles_1, int
    weeks_to_plan_1, int eligible_1[13][25][49], int allocate_sol_1
    [8][13][25][49], int required_1[13][25])
{
feasible_1=1;
int rol_3,weeks_3, emp_3;
int sum_product_eligable_allocate[13][25];
//------------------------------------------------
// Job Cover constraints:
JCfeas_1= 1;
for(rol_3=0; rol_3<all_roles_1;rol_3++)
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
sum_product_eligable_allocate[weeks_3][rol_3]=0;
}
for(rol_3=0; rol_3<all_roles_1;rol_3++)
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++)
for(emp_3=0; emp_3<all_emp_1;emp_3++) {
sum_product_eligable_allocate[weeks_3][rol_3]= sum_product_eligable_allocate[
    weeks_3][rol_3]+ (eligible_1[weeks_3][rol_3][emp_3]* allocate_sol_1[2][
    weeks_3][rol_3][emp_3]);
}
for(rol_3=0; rol_3<all_roles_1;rol_3++) {
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
if(required_1[weeks_3][rol_3]>=1) {
if(sum_product_eligable_allocate[weeks_3][rol_3]!= required_1[weeks_3][rol_3]) {
JCfeas_1=0;
} } } }
if(JCfeas_1==0) {
feasible_1=0;
}
}

void Overlap_feas(vector<int> &emps_changed_1,int &feasible_1, int &OLfeas_1,int
    all_emp_1,int all_roles_1, int weeks_to_plan_1, int allocate_sol_1
    [8][13][25][49])
{ int count_emps, weeks_3, rol_3, emp_3, size_emps_changed;
int sum_allocate_roles[13][49];
size_emps_changed=emps_changed_1.size();
if(feasible_1==1) {
OLfeas_1=1;
```

```
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
sum_allocate_roles[weeks_3][emp_3]=0;
for(rol_3=0; rol_3<all_roles_1;rol_3++) {
sum_allocate_roles[weeks_3][emp_3]=sum_allocate_roles[weeks_3][emp_3]+
    allocate_sol_1[2][weeks_3][rol_3][emp_3];//float check here
if(sum_allocate_roles[weeks_3][emp_3]>1) {
OLfeas_1=0;
} } } }
if(OLfeas_1==0) {
feasible_1=0;
} } }

void BoardFeas(vector<int> &emps_changed_1,int &feasible_1, int &Brdfeas_1,int
    all_emp_1,int all_roles_1, int weeks_to_plan_1, int allocate_sol_1
    [8][13][25][49], int board_sol_1[8][13][25][48],int starting_1[25][49])
{
int count_emps, weeks_3, rol_3, emp_3, size_emps_changed;
int sum_allocate[13][25][49];
size_emps_changed=emps_changed_1.size();
//--------------------------------------------------
// Boarding constraints
if(feasible_1==1) {
Brdfeas_1=1;
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++)
for(rol_3=0; rol_3<all_roles_1;rol_3++) {
sum_allocate[weeks_3][rol_3][emp_3]=allocate_sol_1[2][weeks_3][rol_3][emp_3];
}
for(rol_3=0; rol_3<all_roles_1;rol_3++) {
if(board_sol_1[2][0][rol_3][emp_3] < (sum_allocate[0][rol_3][emp_3] - starting_1[
    rol_3][emp_3])) {
Brdfeas_1=0;
}
for(weeks_3=1;weeks_3<weeks_to_plan_1;weeks_3++) {
if(board_sol_1[2][weeks_3][rol_3][emp_3] < (sum_allocate[weeks_3][rol_3][emp_3] -
     sum_allocate[weeks_3-1][rol_3][emp_3])) {
Brdfeas_1=0;
} } } }
if(Brdfeas_1==0) {
feasible_1=0;
} } }

void DepartFeas(vector<int> &emps_changed_1,int &feasible_1, int &Dprtfeas_1,int
    all_emp_1,int all_roles_1, int weeks_to_plan_1, int allocate_sol_1
    [8][13][25][49], int depart_sol_1[8][13][25][48],int starting_1[25][49])
{
// Departing constraints
int count_emps, weeks_3, rol_3, emp_3, size_emps_changed;
int sum_allocate[13][25][49];
size_emps_changed=emps_changed_1.size();
if(feasible_1==1) {
Dprtfeas_1= 1;
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++)
for(rol_3=0; rol_3<all_roles_1;rol_3++) {
sum_allocate[weeks_3][rol_3][emp_3]=allocate_sol_1[2][weeks_3][rol_3][emp_3];
}
for(rol_3=0; rol_3<all_roles_1;rol_3++) {
if(depart_sol_1[2][0][rol_3][emp_3]<(starting_1[rol_3][emp_3]- sum_allocate[0][
    rol_3][emp_3])) {
Dprtfeas_1=0;
}
for(weeks_3=1;weeks_3<weeks_to_plan_1;weeks_3++) {
if(depart_sol_1[2][weeks_3][rol_3][emp_3]<(sum_allocate[weeks_3 - 1][rol_3][emp_3
    ]- sum_allocate[weeks_3][rol_3][emp_3])) {
Dprtfeas_1=0;
} } } }
if(Dprtfeas_1==0) {
feasible_1=0;
} } }

void AgBoardDepartFeas(vector<int> &ag_roles_changed_1,int &feasible_1, int &
    AGBDfeas_1,int all_roles_1, int weeks_to_plan_1, int allocate_sol_1
    [8][13][25][49], int ag_rboard_sol_1[8][13][25],int ag_rdepart_sol_1
    [8][13][25],int ag_starting_1[25])
{
int count_roles, weeks_3, rol_3, size_ag_roles_changed;
size_ag_roles_changed=ag_roles_changed_1.size();
//----------------------------------------------------
//Agency board / depart constraints
if(feasible_1==1) {
AGBDfeas_1=1;
```

```
for(count_roles=0; count_roles<size_ag_roles_changed; count_roles++) {
rol_3=ag_roles_changed_1[count_roles];
if( (ag_rboard_sol_1[2][0][rol_3]-ag_rdepart_sol_1[2][0][rol_3]) != (
    allocate_sol_1[2][0][rol_3][48] - ag_starting_1[rol_3])) {
AGBDfeas_1=0;
}
for(weeks_3=1;weeks_3<weeks_to_plan_1;weeks_3++) {
if( (ag_rboard_sol_1[2][weeks_3][rol_3] - ag_rdepart_sol_1[2][weeks_3][rol_3])!=
    (allocate_sol_1[2][weeks_3][rol_3][48] - allocate_sol_1[2][weeks_3-1][rol_3
    ][48])) {
AGBDfeas_1=0;
} } }
if(AGBDfeas_1==0) {
feasible_1=0;
} } }

void UnderOverFeas(vector<int> &emps_changed_1,int &feasible_1, int &UTfeas_1,
    int &OTfeas_1,int all_emp_1,int all_roles_1, int weeks_to_plan_1,int
    undertime_sol_1[8][45],int overtime_sol_1[8][45],int g_weeks_1[45], int
    exp_worktime_1[45],int allocate_sol_1[8][13][25][49])
{
int count_emps, weeks_3, rol_3, emp_3, size_emps_changed;
int sum_allocate_roles[13][49];
int sum_allocation_vessels[49];
size_emps_changed=emps_changed_1.size();
for(emp_3=0;emp_3<all_emp_1;emp_3++) {
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
sum_allocate_roles[weeks_3][emp_3]=0;
} }
for(emp_3=0;emp_3<all_emp_1;emp_3++) {
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
for(rol_3=0; rol_3<all_roles_1;rol_3++) {
sum_allocate_roles[weeks_3][emp_3]=sum_allocate_roles[weeks_3][emp_3]+
    allocate_sol_1[2][weeks_3][rol_3][emp_3];//float check here
} } }
for(emp_3=0;emp_3<all_emp_1;emp_3++) {
sum_allocation_vessels[emp_3]=0;
}
for(emp_3=0;emp_3<all_emp_1;emp_3++) {
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
sum_allocation_vessels[emp_3]=sum_allocate_roles[weeks_3][emp_3]+
    sum_allocation_vessels[emp_3];
} }
```

```
if(feasible_1==1) {
UTfeas_1=1;
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
if(emp_3>=3) {
if(undertime_sol_1[2][emp_3-3] < ( g_weeks_1[emp_3-3] - (exp_worktime_1[emp_3-3]
    +sum_allocation_vessels[emp_3])) ) {
UTfeas_1=0;
} } }
if(UTfeas_1==0) {
feasible_1=0;
} }
//----------------------------------------------------
// Overtime
if(feasible_1==1) {
OTfeas_1=1;
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
if(emp_3>=3) {
if(overtime_sol_1[2][emp_3-3] < ((exp_worktime_1[emp_3-3] +sum_allocation_vessels
    [emp_3]) - g_weeks_1[emp_3-3])) {
OTfeas_1=0;
} } }
if(OTfeas_1==0) {
feasible_1=0;
} } }

void LongWorkFeas(vector<int> &ag_roles_changed_1,vector<int> &emps_changed_1,int
     &AGLWfeas_1,int &LWfeas_1,int lambda_1, int &feasible_1, int work_total_1
    [13][48], int work_zero_1[48], int reg_emp_1,int all_roles_1, int
    weeks_to_plan_1,int allocate_sol_1[8][13][25][49], int max_work_1[48],int
    long_work_sol_1[8][13][25][49][10],int ag_work_total_1[13][25], int
    ag_work_zero_1[25], int ag_max_work_1[25], int ag_rdepart_sol_1[8][13][25])
{
int count_roles,rol_3, weeks_3, emp_3, size_emps_changed,count_emps, long_1,
    size_ag_roles_changed;
int sum_allocate_roles[13][48];
for(emp_3=0;emp_3<reg_emp_1;emp_3++) {
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
sum_allocate_roles[weeks_3][emp_3]=0;
} }
for(emp_3=0;emp_3<reg_emp_1;emp_3++) {
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
```

```
for(rol_3=0; rol_3<all_roles_1;rol_3++) {
sum_allocate_roles[weeks_3][emp_3]=sum_allocate_roles[weeks_3][emp_3]+
    allocate_sol_1[2][weeks_3][rol_3][emp_3];//float check here
} } }
size_emps_changed=emps_changed_1.size();
// Long work constraints
// - Calculate work resource values
if(feasible_1==1) {
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
work_total_1[0][emp_3]= work_zero_1[emp_3] + sum_allocate_roles[0][emp_3]-
    max_work_1[emp_3]*(1-sum_allocate_roles[0][emp_3]);
if(work_total_1[0][emp_3] < 0) {
work_total_1[0][emp_3]=0;
}
for(weeks_3=1;weeks_3<weeks_to_plan_1;weeks_3++) {
work_total_1[weeks_3][emp_3] = work_total_1[weeks_3-1][emp_3] +
    sum_allocate_roles[weeks_3][emp_3]- (max_work_1[emp_3]*(1-sum_allocate_roles[
    weeks_3][emp_3]));
if(work_total_1[weeks_3][emp_3] < 0) {
work_total_1[weeks_3][emp_3]=0;
} } }
// - Check constraints
LWfeas_1=1;
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for( long_1=0;long_1<lambda_1;long_1++)
for(rol_3=0;rol_3<all_roles_1;rol_3++) {
if(long_work_sol_1[2][0][rol_3][emp_3][long_1]==1) {
if((max_work_1[emp_3]*long_work_sol_1[2][0][rol_3][emp_3][long_1])<( work_zero_1[
    emp_3] -(max_work_1[emp_3]*(1-allocate_sol_1[2][0][rol_3][emp_3]) )+
    allocate_sol_1[2][0][rol_3][emp_3]-(long_1))) {
LWfeas_1=0;
} }
for(weeks_3=1;weeks_3<weeks_to_plan_1;weeks_3++) {
if(long_work_sol_1[2][weeks_3][rol_3][emp_3][long_1]==1) {
if((max_work_1[emp_3]*long_work_sol_1[2][weeks_3][rol_3][emp_3][long_1]) <(
    work_total_1[weeks_3-1][emp_3]-(max_work_1[emp_3]*(1-allocate_sol_1[2][
    weeks_3][rol_3][emp_3])) + allocate_sol_1[2][weeks_3][rol_3][emp_3]-(long_1))
    ) {
LWfeas_1=0;
} } } } }
if(LWfeas_1==0) {
```

```
feasible_1=0;
} }
//----------------------------------------------------
// Agency Long work constraints
// - Calculate agency work resource values
if(feasible_1==1) {
size_ag_roles_changed=ag_roles_changed_1.size();
for(count_roles=0; count_roles<size_ag_roles_changed; count_roles++) {
rol_3=ag_roles_changed_1[count_roles];
ag_work_total_1[0][rol_3]= ag_work_zero_1[rol_3] + allocate_sol_1[2][0][rol_3
    ][48]- (ag_max_work_1[rol_3]*ag_rdepart_sol_1[2][0][rol_3]);
if(ag_work_total_1[0][rol_3] <allocate_sol_1[2][0][rol_3][48]) {
ag_work_total_1[0][rol_3]=allocate_sol_1[2][0][rol_3][48];
}
for(weeks_3=1;weeks_3<weeks_to_plan_1;weeks_3++) {
ag_work_total_1[weeks_3][rol_3]= ag_work_total_1[weeks_3-1][rol_3] +
    allocate_sol_1[2][weeks_3][rol_3][48] - (ag_max_work_1[rol_3]*
    ag_rdepart_sol_1[2][weeks_3][rol_3]);
if(ag_work_total_1[weeks_3][rol_3] <allocate_sol_1[2][weeks_3][rol_3][48]) {
ag_work_total_1[weeks_3][rol_3]= allocate_sol_1[2][weeks_3][rol_3][48];
} } }
// - Check constraints
AGLWfeas_1=1;
for(count_roles=0; count_roles<size_ag_roles_changed; count_roles++) {
rol_3=ag_roles_changed_1[count_roles];
for(long_1=0;long_1<lambda_1;long_1++)
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
if(long_work_sol_1[2][weeks_3][rol_3][48][long_1]==1) {
if((ag_max_work_1[rol_3]*long_work_sol_1[2][weeks_3][rol_3][48][long_1])< (
    ag_work_total_1[weeks_3][rol_3]-(long_1))) {
AGLWfeas_1=0;
} } } }
if(AGLWfeas_1==0) {
feasible_1=0;
} } }

void RestFeas(vector<int> &emps_changed_1,int &feasible_1,int &RvWfeas_1,int
    reg_emp_1,int all_roles_1, int weeks_to_plan_1,int depart_sol_1
    [8][13][25][48],int rest_total_1[13][49], int rest_zero_1[48],int
    allocate_sol_1[8][13][25][49],int min_rest_1[48])
{
int rol_3, weeks_3, emp_3, size_emps_changed,count_emps;
size_emps_changed=emps_changed_1.size();
```

```
int sum_depart[13][48];
int sum_allocate_roles[13][48];
for(emp_3=0;emp_3<reg_emp_1;emp_3++) {
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
sum_allocate_roles[weeks_3][emp_3]=0;
} }
for(emp_3=0;emp_3<reg_emp_1;emp_3++) {
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
for(rol_3=0; rol_3<all_roles_1;rol_3++) {
sum_allocate_roles[weeks_3][emp_3]=sum_allocate_roles[weeks_3][emp_3]+
    allocate_sol_1[2][weeks_3][rol_3][emp_3];//float check here
} } }
if(feasible_1==1) {
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(weeks_3=0;weeks_3<weeks_to_plan_1;weeks_3++) {
sum_depart[weeks_3][emp_3]=0;
for(rol_3=0;rol_3<all_roles_1;rol_3++) {
sum_depart[weeks_3][emp_3]=sum_depart[weeks_3][emp_3]+depart_sol_1[2][weeks_3][
    rol_3][emp_3];
} }
rest_total_1[0][emp_3]= rest_zero_1[emp_3] - (1-sum_allocate_roles[0][emp_3]);
if(rest_total_1[0][emp_3] < ((min_rest_1[emp_3]-1)*sum_depart[0][emp_3])) {
rest_total_1[0][emp_3]= (min_rest_1[emp_3]-1)*sum_depart[0][emp_3];
}
for(weeks_3=1;weeks_3<weeks_to_plan_1;weeks_3++) {
rest_total_1[weeks_3][emp_3]= rest_total_1[weeks_3-1][emp_3] - (1-
    sum_allocate_roles[weeks_3][emp_3]);
if(rest_total_1[weeks_3][emp_3] < ((min_rest_1[emp_3]-1)*sum_depart[weeks_3][
    emp_3])) {
rest_total_1[weeks_3][emp_3]= (min_rest_1[emp_3]-1)*sum_depart[weeks_3][emp_3];
} } }
// - Check constraints
RvWfeas_1=1;
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
if(min_rest_1[emp_3]*(1-sum_allocate_roles[0][emp_3]) < rest_zero_1[emp_3]) {
RvWfeas_1=0;
}
for(weeks_3=1;weeks_3<weeks_to_plan_1;weeks_3++) {
if(min_rest_1[emp_3]*(1-sum_allocate_roles[weeks_3][emp_3]) < rest_total_1[
    weeks_3-1][emp_3]) {
RvWfeas_1=0;
} } }
if(RvWfeas_1==0) {
feasible_1=0;
} } }

void LinkFeasiblity(int lambda_1,vector<int> &ag_roles_changed_1,vector<int> &
    emps_changed_1,int &feasible_1,int &Linkfeas_1,int reg_emp_1,int all_roles_1,
     int weeks_to_plan_1,int cur_allocate_1[13][25][49],int chng_allocate_1
    [13][25][49], int allocate_sol_1[8][13][25][49],int cur_board_1[13][25][48],
    int chng_board_1[13][25][48], int board_sol_1[8][13][25][48],int cur_depart_1
    [13][25][48], int chng_depart_1[13][25][48], int depart_sol_1[8][13][25][48],
    int cur_ag_rboard_1[13][25], int ag_rboard_sol_1[8][13][25], int
    chng_ag_rboard_1[13][25], int cur_ag_rdepart_1[13][25], int ag_rdepart_sol_1
    [8][13][25], int chng_ag_rdepart_1[13][25], int cur_long_work_1
    [13][25][49][10], int chng_long_work_1[13][25][49][10], int long_work_sol_1
    [8][13][25][49][10], int chng_undertime_1[45], int undertime_sol_1[8][45],
    int cur_undertime_1[45], int chng_overtime_1[45], int overtime_sol_1[8][45],
    int cur_overtime_1[45])
{
int count_roles,rol_3, week_3, emp_3, size_emps_changed,count_emps, long_1,
    size_ag_roles_changed;
size_emps_changed=emps_changed_1.size();
if(feasible_1==1) {
Linkfeas_1=1;
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(rol_3=0;rol_3<all_roles_1;rol_3++)
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
if(cur_allocate_1[week_3][rol_3][emp_3]== 0) {
if(chng_allocate_1[week_3][rol_3][emp_3]!=allocate_sol_1[2][week_3][rol_3][emp_3
    ]) {
Linkfeas_1=0;
} }
else {
if(chng_allocate_1[week_3][rol_3][emp_3]!=(cur_allocate_1[week_3][rol_3][emp_3] -
    allocate_sol_1[2][week_3][rol_3][emp_3])) {
Linkfeas_1=0;
} } } }
emp_3=48;
size_ag_roles_changed=ag_roles_changed_1.size();
for(count_roles=0; count_roles<size_ag_roles_changed; count_roles++) {
rol_3=ag_roles_changed_1[count_roles];
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
```

```
if(cur_allocate_1[week_3][rol_3][emp_3]== 0) {
if(chng_allocate_1[week_3][rol_3][emp_3]!=allocate_sol_1[2][week_3][rol_3][emp_3
    ]) {
Linkfeas_1=0;
} }
else {
if(chng_allocate_1[week_3][rol_3][emp_3]!=(cur_allocate_1[week_3][rol_3][emp_3] -
    allocate_sol_1[2][week_3][rol_3][emp_3])) {
Linkfeas_1=0;
} } } }
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(rol_3=0;rol_3<all_roles_1;rol_3++)
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
if(cur_board_1[week_3][rol_3][emp_3] == 0) {
if(chng_board_1[week_3][rol_3][emp_3] != board_sol_1[2][week_3][rol_3][emp_3]) {
Linkfeas_1=0;
} }
else {
if(chng_board_1[week_3][rol_3][emp_3] !=(cur_board_1[week_3][rol_3][emp_3] -
    board_sol_1[2][week_3][rol_3][emp_3])) {
Linkfeas_1=0;
} } } }
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(rol_3=0;rol_3<all_roles_1;rol_3++)
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
if(cur_depart_1[week_3][rol_3][emp_3] == 0) {
if(chng_depart_1[week_3][rol_3][emp_3]!=depart_sol_1[2][week_3][rol_3][emp_3]) {
Linkfeas_1=0;
} }
else {
if(chng_depart_1[week_3][rol_3][emp_3]!=(cur_depart_1[week_3][rol_3][emp_3]-
    depart_sol_1[2][week_3][rol_3][emp_3])) {
Linkfeas_1=0;
} } } }
for(count_roles=0; count_roles<size_ag_roles_changed; count_roles++) {
rol_3=ag_roles_changed_1[count_roles];
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
if(cur_ag_rboard_1[week_3][rol_3] == 0) {
if(chng_ag_rboard_1[week_3][rol_3]!=ag_rboard_sol_1[2][week_3][rol_3]) {
Linkfeas_1=0;
} }
```

```
else {
if(chng_ag_rboard_1[week_3][rol_3]!=(cur_ag_rboard_1[week_3][rol_3] -
    ag_rboard_sol_1[2][week_3][rol_3])) {
Linkfeas_1=0;
} } } }
for(count_roles=0; count_roles<size_ag_roles_changed; count_roles++) {
rol_3=ag_roles_changed_1[count_roles];
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
if(cur_ag_rdepart_1[week_3][rol_3] == 0) {
if(chng_ag_rdepart_1[week_3][rol_3]!=ag_rdepart_sol_1[2][week_3][rol_3]) {
Linkfeas_1=0;
} }
else {
if(chng_ag_rdepart_1[week_3][rol_3]!=(cur_ag_rdepart_1[week_3][rol_3] -
    ag_rdepart_sol_1[2][week_3][rol_3])) {
Linkfeas_1=0;
} } } }
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(long_1=0;long_1<lambda_1;long_1++)
for(week_3=0;week_3<weeks_to_plan_1;week_3++)
for(rol_3=0;rol_3<all_roles_1;rol_3++) {
if(cur_long_work_1[week_3][rol_3][emp_3][long_1] == 0) {
if(chng_long_work_1[week_3][rol_3][emp_3][long_1] != long_work_sol_1[2][week_3][
    rol_3][emp_3][long_1]) {
Linkfeas_1=0;
} }
else {
if(chng_long_work_1[week_3][rol_3][emp_3][long_1]!=(cur_long_work_1[week_3][rol_3
    ][emp_3][long_1] - long_work_sol_1[2][week_3][rol_3][emp_3][long_1])) {
Linkfeas_1=0;
} } } }
emp_3=48;
for(count_roles=0; count_roles<size_ag_roles_changed; count_roles++) {
rol_3=ag_roles_changed_1[count_roles];
for(long_1=0;long_1<lambda_1;long_1++)
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
if(cur_long_work_1[week_3][rol_3][emp_3][long_1] == 0) {
if(chng_long_work_1[week_3][rol_3][emp_3][long_1] != long_work_sol_1[2][week_3][
    rol_3][emp_3][long_1]) {
Linkfeas_1=0;
} }
else {
```

```
if(chng_long_work_1[week_3][rol_3][emp_3][long_1]!=(cur_long_work_1[week_3][rol_3
    ][emp_3][long_1] - long_work_sol_1[2][week_3][rol_3][emp_3][long_1])) {
Linkfeas_1=0;
} } } }
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
if(emp_3>=3) {
if(chng_undertime_1[emp_3-3] !=( undertime_sol_1[2][emp_3-3] - cur_undertime_1[
    emp_3-3])) {
Linkfeas_1=0;
}
if(chng_overtime_1[emp_3-3] !=(overtime_sol_1[2][emp_3-3] - cur_overtime_1[emp_3
    -3])) {
Linkfeas_1=0;
} } }
if(Linkfeas_1== 0) {
feasible_1 = 0;
} } }

void StatusFeasibility(int lambda_1,vector<int> &ag_roles_changed_1,vector<int> &
    emps_changed_1,int &feasible_1,int &Statfeas_1,int reg_emp_1,int all_roles_1,
     int weeks_to_plan_1, int allocate_sol_1[8][13][25][49],int chng_allocate_1
    [13][25][49], int chng_long_work_1[13][25][49][10], int long_work_sol_1
    [8][13][25][49][10], int chng_board_1[13][25][48], int board_sol_1
    [8][13][25][48], int chng_depart_1[13][25][48], int depart_sol_1
    [8][13][25][48], int ag_rboard_sol_1[8][13][25], int chng_ag_rboard_1
    [13][25], int ag_rdepart_sol_1[8][13][25], int chng_ag_rdepart_1[13][25], int
     undertime_sol_1[8][45], int overtime_sol_1[8][45])
{
int count_roles,rol_3, week_3, emp_3, size_emps_changed,count_emps, long_1,
    size_ag_roles_changed;
size_emps_changed=emps_changed_1.size();
//--------------------------------------------------
// Status of variables
if(feasible_1==1) {
Statfeas_1=1;
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(rol_3=0;rol_3<all_roles_1;rol_3++) {
if((allocate_sol_1[2][week_3][rol_3][emp_3]!=0) && (allocate_sol_1[2][week_3][
    rol_3][emp_3]!=1)) {
Statfeas_1=0;
}
if((chng_allocate_1[week_3][rol_3][emp_3]!=0) && (chng_allocate_1[week_3][rol_3][
    emp_3]!=1)) {
Statfeas_1=0;
}
for(long_1=0;long_1<lambda_1;long_1++) {
if((long_work_sol_1[2][week_3][rol_3][emp_3][long_1]!=0) && (long_work_sol_1[2][
    week_3][rol_3][emp_3][long_1]!=1)) {
Statfeas_1=0;
}
if((chng_long_work_1[week_3][rol_3][emp_3][long_1]!=0) && (chng_long_work_1[
    week_3][rol_3][emp_3][long_1]!=1)) {
Statfeas_1=0;
} } } }
emp_3=48;
size_ag_roles_changed=ag_roles_changed_1.size();
for(count_roles=0; count_roles<size_ag_roles_changed; count_roles++) {
rol_3=ag_roles_changed_1[count_roles];
if((allocate_sol_1[2][week_3][rol_3][emp_3]!=0) &&(allocate_sol_1[2][week_3][
    rol_3][emp_3]!=1)) {
Statfeas_1=0;
}
if((chng_allocate_1[week_3][rol_3][emp_3]!=0) && (chng_allocate_1[week_3][rol_3][
    emp_3]!=1)) {
Statfeas_1=0;
}
for(long_1=0;long_1<lambda_1;long_1++) {
if((long_work_sol_1[2][week_3][rol_3][emp_3][long_1]!=0) && (long_work_sol_1[2][
    week_3][rol_3][emp_3][long_1]!=1)) {
Statfeas_1=0;
}
if((chng_long_work_1[week_3][rol_3][emp_3][long_1]!=0) && (chng_long_work_1[
    week_3][rol_3][emp_3][long_1]!=1)) {
Statfeas_1=0;
} } } }
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
for(rol_3=0;rol_3<all_roles_1;rol_3++)
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
if((board_sol_1[2][week_3][rol_3][emp_3]!=0)&&(board_sol_1[2][week_3][rol_3][
    emp_3]!=1)) {
Statfeas_1=0;
}
```

```
if((chng_board_1[week_3][rol_3][emp_3]!=0) && (chng_board_1[week_3][rol_3][emp_3
    ]!=1)) {
Statfeas_1=0;
}
if((depart_sol_1[2][week_3][rol_3][emp_3]!=0) &&(depart_sol_1[2][week_3][rol_3][
    emp_3]!= 1)) {
Statfeas_1=0;
}
if((chng_depart_1[week_3][rol_3][emp_3]!=0) && (chng_depart_1[week_3][rol_3][
    emp_3]!= 1)) {
Statfeas_1=0;
} } }
for(count_roles=0; count_roles<size_ag_roles_changed; count_roles++) {
rol_3=ag_roles_changed_1[count_roles];
for(week_3=0;week_3<weeks_to_plan_1;week_3++) {
if((ag_rboard_sol_1[2][week_3][rol_3]!= 0) && (ag_rboard_sol_1[2][week_3][rol_3
    ]!=1)) {
Statfeas_1=0;
}
if((chng_ag_rboard_1[week_3][rol_3]!=0) && (chng_ag_rboard_1[week_3][rol_3]!= 1))
     {
Statfeas_1=0;
}
if((chng_ag_rdepart_1[week_3][rol_3]!=0) && (chng_ag_rdepart_1[week_3][rol_3]!=1)
    ) {
Statfeas_1=0;
}
if((ag_rdepart_sol_1[2][week_3][rol_3]!=0) && (ag_rdepart_sol_1[2][week_3][rol_3
    ]!=1)) {
Statfeas_1=0;
} } }
for(count_emps=0; count_emps<size_emps_changed; count_emps++) {
emp_3=emps_changed_1[count_emps];
if(emp_3>=3) {
if(undertime_sol_1[2][emp_3-3] < 0) {
Statfeas_1=0;
}
if(overtime_sol_1[2][emp_3-3] < 0) {
Statfeas_1=0;
} } } } }

void infeasibility(int &Statfeas_1, int &no_Statfeas_1,int &Linkfeas_1,int &
    no_Linkfeas_1,int &no_RvWfeas_1 ,int &RvWfeas_1,int &AGLWfeas_1,int &LWfeas_1
    , int &no_AGLWfeas_1,int &no_LWfeas_1, int &UTfeas_1, int &OTfeas_1, int &
    no_UTfeas_1, int &no_OTfeas_1,int &JCfeas_1, int &no_JCfeas_1, int &OLfeas_1,
     int &no_OLfeas_1, int &Brdfeas_1, int &Dprtfeas_1, int &AGBDfeas_1, int &
    no_Brdfeas_1, int &no_Dprtfeas_1, int &no_AGBDfeas_1)
{
if(JCfeas_1==0) {
no_JCfeas_1=no_JCfeas_1+1;
}
else if(OLfeas_1==0) {
no_OLfeas_1=no_OLfeas_1+1;
}
else if(Brdfeas_1==0) {
no_Brdfeas_1=no_Brdfeas_1+1;
}
else if(Dprtfeas_1==0) {
no_Dprtfeas_1=no_Dprtfeas_1+1;
}
else if(AGBDfeas_1==0) {
no_AGBDfeas_1=no_AGBDfeas_1+1;
}
else if(UTfeas_1==0) {
no_UTfeas_1=no_UTfeas_1+1;
}
else if(OTfeas_1==0) {
no_OTfeas_1=no_OTfeas_1+1;
}
else if(AGLWfeas_1==0) {
no_AGLWfeas_1=no_AGLWfeas_1+1;
}
else if(LWfeas_1==0) {
no_LWfeas_1=no_LWfeas_1+1;
}
else if(RvWfeas_1==0) {
no_RvWfeas_1=no_RvWfeas_1+1;
}
else if(Linkfeas_1==0) {
no_Linkfeas_1=no_Linkfeas_1+1;
}
else if(Statfeas_1==0) {
no_Statfeas_1=no_Statfeas_1+1;
}
else {
no_JCfeas_1=no_JCfeas_1;
no_Statfeas_1=no_Statfeas_1;
no_Linkfeas_1=no_Linkfeas_1;
no_RvWfeas_1=no_RvWfeas_1;
no_OLfeas_1=no_OLfeas_1;
no_AGLWfeas_1=no_AGLWfeas_1;
no_OTfeas_1=no_OTfeas_1;
no_UTfeas_1=no_UTfeas_1;
no_Brdfeas_1=no_Brdfeas_1;
} }
```

E.2.5.8 ‘Finishing’ sub-programme

```
#include "finish.h"

void finishing(int &update_done_1, int &terminate_1, int &iteration_1, double&
    time_1)
{
if(update_done_1==0) {
terminate_1=1;
}
else {
//current_time=gettime;
/*if(iteration_1==50)
{
terminate_1=1;
}*/
/*else
{
if(time_1 >= 650) {
terminate_1=1;
} }*/
if(time_1 >= 120) {
terminate_1=1;
} } }
```

E.2.5.9 ‘Initialise’ sub-programme

```
#include "initialise.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "string"
```

```
#include <cstdlib>
#include <ctime>
#include <math.h>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <iterator>
#include <vector>
#include <random>
using namespace std;

void initialise_first(int overall_regular_max_work_1, int max_work_1[48], int
    overall_agency_max_work_1, int ag_max_work_1[25], int overall_max_work_1,int&
     lambda_1, int reg_emp_1, int all_roles_1)
{
int emp_17, rol_17;
for(emp_17=0;emp_17<reg_emp_1;emp_17++) {
if(overall_regular_max_work_1<= max_work_1[emp_17]) {
overall_regular_max_work_1=max_work_1[emp_17];
} }
for(rol_17=0;rol_17<all_roles_1;rol_17++) {
if(overall_agency_max_work_1<=ag_max_work_1[rol_17]) {
overall_agency_max_work_1=ag_max_work_1[rol_17];
} }
if(overall_regular_max_work_1>=overall_agency_max_work_1) {
overall_max_work_1=overall_regular_max_work_1;
}
else {
overall_max_work_1=overall_agency_max_work_1;
}
lambda_1=overall_max_work_1;
}

void initialise_second(int weeks_to_plan_1, int list_sol_1[8][13][48], int
    best_index_1, int best_sols_1[51][13][48], int& added_t_1, int
    swaps_examined_1[48][48], vector<int> &emps_changed_1, int reg_emp_1, int
    all_roles_1, int taskbased_sol_1[13][25][49])
{
int emp_17, weeks_17, emp_18, index_17, rol_17;
emps_changed_1.clear();
for(emp_17=0;emp_17<reg_emp_1;emp_17++) {
for(weeks_17=0; weeks_17<weeks_to_plan_1;weeks_17++) {
list_sol_1[0][weeks_17][emp_17]=-1;
```

```
list_sol_1[1][weeks_17][emp_17]=-1;
for(index_17=0;index_17<best_index_1;index_17++) {
best_sols_1[index_17][weeks_17][emp_17]=-1;
}
added_t_1=0;
for(rol_17=0;rol_17<all_roles_1;rol_17++) {
if(added_t_1==0 && taskbased_sol_1[weeks_17][rol_17][emp_17]==1) {
list_sol_1[0][weeks_17][emp_17]=rol_17;
list_sol_1[1][weeks_17][emp_17]=rol_17;
added_t_1=1;
} }
if(added_t_1==0) {
list_sol_1[0][weeks_17][emp_17]=-1;
list_sol_1[6][weeks_17][emp_17]=-1;
} }
emps_changed_1.push_back(emp_17);
for(emp_18=0;emp_18<reg_emp_1;emp_18++) {
swaps_examined_1[emp_17][emp_18]=0;
} } }

void initialise_third(int weeks_to_plan_1, int ag_list_sol_1[8][13][25], int
    best_index_1, int ag_best_sols_1[51][13][25], vector<int> &ag_roles_changed_1
    , int reg_emp_1, int all_roles_1,int taskbased_sol_1[13][25][49], int
    required_1[13][25])
{
int sum_task_based[13][25];
int emp_17, weeks_17, index_17, rol_17;
ag_roles_changed_1.clear();
for(rol_17=0; rol_17<all_roles_1;rol_17++)
for(weeks_17=0;weeks_17<weeks_to_plan_1;weeks_17++) {
sum_task_based[weeks_17][rol_17]=0;
}
for(rol_17=0; rol_17<all_roles_1;rol_17++)
for(weeks_17=0;weeks_17<weeks_to_plan_1;weeks_17++)
for(emp_17=0; emp_17<reg_emp_1; emp_17++) {
sum_task_based[weeks_17][rol_17]=sum_task_based[weeks_17][rol_17]+taskbased_sol_1
    [weeks_17][rol_17][emp_17];
}
for(rol_17=0; rol_17<all_roles_1;rol_17++) {
for(weeks_17=0;weeks_17<weeks_to_plan_1;weeks_17++) {
ag_list_sol_1[0][weeks_17][rol_17]=0;
ag_list_sol_1[1][weeks_17][rol_17]=0;
for(index_17=0;index_17<best_index_1;index_17++) {
```

```
ag_best_sols_1[index_17][weeks_17][emp_17]=0;
}
if(required_1[weeks_17][rol_17]>=1) {
if(sum_task_based[weeks_17][rol_17]==0) {
ag_list_sol_1[0][weeks_17][rol_17]=1;
ag_list_sol_1[1][weeks_17][rol_17]=1;
} } }
ag_roles_changed_1.push_back(rol_17);
} }

void initialise_four(int weeks_to_plan_1,int ag_best_sols_1[51][13][25], int
    reg_emp_1,int all_roles_1,int best_sols_1[51][13][48], int list_sol_1
    [8][13][48], int ag_list_sol_1[8][13][25],int last_changed_1[48])
{
int emp_17, weeks_17, rol_17;
for(emp_17=0;emp_17<reg_emp_1;emp_17++)
for(weeks_17=0;weeks_17<weeks_to_plan_1;weeks_17++) {
best_sols_1[0][weeks_17][emp_17]=list_sol_1[1][weeks_17][emp_17];
}
for(rol_17=0;rol_17<all_roles_1;rol_17++)
for(weeks_17=0; weeks_17<weeks_to_plan_1;weeks_17++) {
ag_best_sols_1[0][weeks_17][rol_17]=ag_list_sol_1[1][weeks_17][rol_17];
}
for(emp_17=0;emp_17<reg_emp_1;emp_17++) {
last_changed_1[emp_17]=0;
} }
```

E.2.5.10 ‘Random kick’ sub-programmes

```
#include "rand_kick.h"

void kicking(vector<int>& emps_changed_1, vector<int>& ag_roles_changed_1, int
    eligible_1[13][25][49],int work_zero_1[48], int min_rest_1[48], int
    starting_1[25][49], int all_roles_1, int sum_start_rand_1, int rest_zero_1
    [48], int &kick_feas1, int &kick_end_1, int &kick_start_1, float &random_1,
    int &random_emp_1, int reg_emp_1, int &kick_emp_1, int &random_task_1, int
    all_roles1, int &kick_task_1, int max_work1[48], int &random_length_1, int &
    random_time1, int weeks_to_plan_1 )
{
int rol_randomkick1, weeks_randomkick1;
emps_changed_1.clear();
ag_roles_changed_1.clear();
```

```
random_1= ((float) rand() / (RAND_MAX));   // AL - Have removed "+1" from here...
random_emp_1= (reg_emp_1*random_1);
//random_emp_1=6;// AL - ... and from here
kick_emp_1 = random_emp_1;
random_1= ((float) rand() / (RAND_MAX));   // AL - Similarly, "+1" removed from
    here...
random_task_1= all_roles1*random_1;        // AL - ... and here
kick_task_1 = random_task_1;
//kick_task_1=2;// AL - Similar to random_task...
random_1= ((float) rand() / (RAND_MAX));
random_length_1= (max_work1[kick_emp_1]*random_1) + 1;
//random_length_1=2; // AL - Keep "+1" - assume this gives a lenrgh between 1 and
     max_work(e)
random_1= ((float) rand() / (RAND_MAX));   // AL - "+1" removed here...
random_time1= ((weeks_to_plan_1 - (random_length_1-1))*random_1);       // AL -
    ... and here
//random_time1=4;
kick_start_1=random_time1;
kick_end_1=random_time1 + random_length_1 - 1;
kick_feas1=1;
if(kick_start_1 < rest_zero_1[kick_emp_1]) // AL - Have changed this from "<="
{
kick_feas1=0;
}
sum_start_rand_1=0;
for(rol_randomkick1=0; rol_randomkick1<all_roles_1; rol_randomkick1++) {
if(rol_randomkick1!=kick_task_1) {
sum_start_rand_1=sum_start_rand_1+starting_1[rol_randomkick1][kick_emp_1];
} }
if(sum_start_rand_1 > 0 && kick_start_1 <= min_rest_1[kick_emp_1]) // AL - Have
    changed this from "if(starting[rol_randomkick][kick_emp]> 0 ..."
{
kick_feas1=0;
}
if( starting_1[kick_task_1][kick_emp_1]> 0) {
if(kick_start_1 >= 1 && kick_start_1 < min_rest_1[kick_emp_1]) // AL - Have
    changed this from "if(kick_start >= 2 && kick_start <= min_rest[kick_emp])"
{
kick_feas1=0;
}
if(kick_start_1 <= 0 && ((random_length_1 + work_zero_1[kick_emp_1]) > max_work1[
    kick_emp_1]))  // AL - Have changed this from "if(kick_start <= 1..."
{
```

```
kick_feas1=0;
} }                   // AL - Close bracket added here
for( weeks_randomkick1=kick_start_1; weeks_randomkick1<= kick_end_1;
    weeks_randomkick1++)     // AL - Have changed this from "weeks_randomkick<
    kick_end"
{
if(eligible_1[weeks_randomkick1][kick_task_1][kick_emp_1]==0) {
kick_feas1=0;
} } }

void kicking2(int min_rest_1[48],int starting_1[25][49] ,int &rest_count_1, int &
    kick_emp1,int reg_emp1,int all_roles1,int weeks_to_plan1, int ag_list_sol_1
    [8][13][25], int &kick_start_1, int &kick_end_1, int &kick_task_1, vector<int
    >& emps_changed_1, vector<int>& ag_roles_changed_1, int list_sol_1
    [8][13][48])
{
int rol_randomkick1, weeks_randomkick1, emp_randomkick1;
for(rol_randomkick1=0; rol_randomkick1<all_roles1; rol_randomkick1++)
for(weeks_randomkick1=0; weeks_randomkick1<weeks_to_plan1; weeks_randomkick1++)
    // AL - Have changed this from "for( weeks_randomkick=kick_start;
    weeks_randomkick< kick_end; weeks_randomkick++)"
{
ag_list_sol_1[7][weeks_randomkick1][rol_randomkick1]= ag_list_sol_1[0][
    weeks_randomkick1][rol_randomkick1];
}
//for(weeks2_randomkick=kick_start;weeks2_randomkick<=kick_end; weeks2_randomkick
    ++)  // AL - Have removed this line...
for(weeks_randomkick1= kick_start_1; weeks_randomkick1<=kick_end_1;
    weeks_randomkick1++) {
if( ag_list_sol_1[7][weeks_randomkick1][kick_task_1]==1)         // AL - Have
    changed this from "if( ag_list_sol[7][weeks2_randomkick][kick_task]==1)"
{
ag_list_sol_1[7][weeks_randomkick1][kick_task_1]=0;       // AL - Have changed
    this from "ag_list_sol[7][weeks2_randomkick][kick_task]=0;"
ag_roles_changed_1.push_back(kick_task_1);
} }
for(emp_randomkick1=0; emp_randomkick1<reg_emp1; emp_randomkick1++) {
for(weeks_randomkick1=0; weeks_randomkick1<weeks_to_plan1; weeks_randomkick1++) {
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=-1;
}
if(emp_randomkick1!=kick_emp1) {
//kick_count=0;       // AL - Can ignore 'kick_count', and use the count of weeks
    instead
```

```
rest_count_1=0;
if(starting_1[kick_task_1][emp_randomkick1] > 0) {
if(kick_start_1 ==1) {
rest_count_1= min_rest_1[emp_randomkick1];
} }
for(weeks_randomkick1=0; weeks_randomkick1<weeks_to_plan1; weeks_randomkick1++) {
if(weeks_randomkick1==kick_start_1-1 && list_sol_1[0][weeks_randomkick1][
    emp_randomkick1]== kick_task_1) // AL - This replaces the above 'if'
    statement
{
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=list_sol_1[0][weeks_randomkick1
    ][emp_randomkick1];
rest_count_1= min_rest_1[emp_randomkick1];
}
else if(weeks_randomkick1==kick_start_1)          // AL - Have changed this from
    "else if(kick_count ==kick_start)"
{
if( list_sol_1[0][weeks_randomkick1][emp_randomkick1]==kick_task_1) {
list_sol_1[7][weeks_randomkick1][emp_randomkick1]= -1;
emps_changed_1.push_back(emp_randomkick1);
rest_count_1= rest_count_1 - 1;
}
else {
list_sol_1[7][weeks_randomkick1][emp_randomkick1]= list_sol_1[0][
    weeks_randomkick1][emp_randomkick1];
rest_count_1= 0;
} }
else if((weeks_randomkick1>kick_start_1) && (weeks_randomkick1 <= kick_end_1)) //
     AL - Have changed this from "else if(kick_count > kick_start && kick_count
    <= kick_end)"
{
if(list_sol_1[0][weeks_randomkick1][emp_randomkick1]== kick_task_1) {
list_sol_1[7][weeks_randomkick1][emp_randomkick1]= -1;
emps_changed_1.push_back(emp_randomkick1);
rest_count_1= rest_count_1 - 1;
}
else if(list_sol_1[0][weeks_randomkick1][emp_randomkick1]!=-1) {
if(rest_count_1 > 0) {
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=-1;
ag_list_sol_1[7][weeks_randomkick1][list_sol_1[0][weeks_randomkick1][
    emp_randomkick1]]=1; // AL - Have changed this from "ag_list_sol[7][
    kick_count][list_sol[0][weeks_randomkick][emp_randomkick]]=0;"
ag_roles_changed_1.push_back(list_sol_1[0][weeks_randomkick1][emp_randomkick1]);
rest_count_1= rest_count_1 - 1;
}
else {
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=list_sol_1[0][weeks_randomkick1
    ][emp_randomkick1];
} }
else {
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=list_sol_1[0][weeks_randomkick1
    ][emp_randomkick1];
rest_count_1=rest_count_1-1;
} }
else if((weeks_randomkick1 > kick_end_1) && (list_sol_1[0][weeks_randomkick1][
    emp_randomkick1]!=-1) && (rest_count_1> 0))  // AL - Have changed this from "
    else if(kick_count > kick_end &&..."
{
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=-1;
ag_list_sol_1[7][weeks_randomkick1][list_sol_1[0][weeks_randomkick1][
    emp_randomkick1]]=1;     // AL - Have changed this from "ag_list_sol[7][
    kick_count][list_sol[0][weeks_randomkick][emp_randomkick]]=1;"
ag_roles_changed_1.push_back(list_sol_1[0][weeks_randomkick1][emp_randomkick1]);
rest_count_1 = rest_count_1- 1;
}
else {
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=list_sol_1[0][weeks_randomkick1
    ][emp_randomkick1];
} }            // AL - Have added this to end 'do' loop from above
}
else {
emps_changed_1.push_back(emp_randomkick1);
//kick_count=0;              // AL - As above, do not need 'kick_count', as we
    can use 'weeks_randomkick' in loop instead...
//forall(j in list_sol("current",e)) do
for(weeks_randomkick1=0; weeks_randomkick1<weeks_to_plan1; weeks_randomkick1++)
    // AL - ... so have added this loop...
{
//kick_count=kick_count + 1;        // AL - ... and we can remove this line
if(weeks_randomkick1 < (kick_start_1 - min_rest_1[emp_randomkick1]))    // AL -
    Have changed this from "if(kick_count < kick_start - min_rest[emp_randomkick
    ]) "
{
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=list_sol_1[0][weeks_randomkick1
    ][emp_randomkick1];
}
```

```
else if(weeks_randomkick1< kick_start_1)          // AL - Have changed this from
    "else if(kick_count < kick_start) "
{
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=-1;
if(list_sol_1[0][weeks_randomkick1][emp_randomkick1]!=-1) {
ag_list_sol_1[7][weeks_randomkick1][list_sol_1[0][weeks_randomkick1][
    emp_randomkick1]]=1;     // AL - Have changed this from "ag_list_sol[7][
    kick_count][..."
ag_roles_changed_1.push_back(list_sol_1[0][weeks_randomkick1][emp_randomkick1]);
} }
else if(weeks_randomkick1 <= kick_end_1)          // AL - Have changed this from
    "else if(kick_count <= kick_end) "
{
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=kick_task_1;
if((list_sol_1[0][weeks_randomkick1][emp_randomkick1] !=-1) && (list_sol_1[0][
    weeks_randomkick1][emp_randomkick1]!= kick_task_1))
{
ag_list_sol_1[7][weeks_randomkick1][list_sol_1[0][weeks_randomkick1][
    emp_randomkick1]]=1;     // AL - Have changed this from "ag_list_sol[7][
    kick_count][..."
ag_roles_changed_1.push_back(list_sol_1[0][weeks_randomkick1][emp_randomkick1]);
} }
else if(weeks_randomkick1 <= (kick_end_1 + min_rest_1[emp_randomkick1]))       //
     AL - Have changed this from "else if(kick_count <=..."
{
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=-1;
if(list_sol_1[0][weeks_randomkick1][emp_randomkick1]!=-1) {
ag_list_sol_1[7][weeks_randomkick1][list_sol_1[0][weeks_randomkick1][
    emp_randomkick1]]=1;     // AL - Have changed this from "ag_list_sol[7][
    kick_count][..."
ag_roles_changed_1.push_back(list_sol_1[0][weeks_randomkick1][emp_randomkick1]);
} }                                  // AL - Have added a close-bracket here
else {
list_sol_1[7][weeks_randomkick1][emp_randomkick1]=list_sol_1[0][weeks_randomkick1
    ][emp_randomkick1];
} } } } }

void kicking3(int all_roles_1, int reg_emp_1,int weeks_to_plan_1, int tabu_sol_1
    [13][48], int list_sol_1[8][13][48], int ag_tabu_sol_1[13][25], int
    ag_list_sol_1[8][13][25])
{
int emp_randomkick1, weeks_randomkick1, rol_randomkick1;
for(emp_randomkick1=0; emp_randomkick1<reg_emp_1; emp_randomkick1++)
for( weeks_randomkick1=0; weeks_randomkick1< weeks_to_plan_1; weeks_randomkick1
    ++)     // AL - Have changed this from "for( weeks_randomkick=kick_start;
    weeks_randomkick< kick_end; weeks_randomkick++)"
{
tabu_sol_1[weeks_randomkick1][emp_randomkick1]=list_sol_1[0][weeks_randomkick1][
    emp_randomkick1];
}
for( weeks_randomkick1=0; weeks_randomkick1< weeks_to_plan_1; weeks_randomkick1
    ++)     // AL - Have changed this from "for( weeks_randomkick=kick_start;
    weeks_randomkick< kick_end; weeks_randomkick++)"
for(rol_randomkick1=0; rol_randomkick1<all_roles_1; rol_randomkick1++) {
ag_tabu_sol_1[weeks_randomkick1][rol_randomkick1]= ag_list_sol_1[0][
    weeks_randomkick1][rol_randomkick1];
} }
```

E.2.5.11 ‘Sort employee list’ sub-programme

```
#include "sort_listing.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "string"
#include <cstdlib>
#include <ctime>
#include <math.h>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <iterator>
#include <vector>
#include <random>
using namespace std;

void sort_employee_list(float short_list_fraction_1, int& change_no_1, int
    last_changed_1[48],int min_emp_1,int min_no_1, int& order_rule_1, int
    order_number_1[48], int reg_emp_1, vector<int>& ordered_list_1, vector<int>&
    short_ordered_list_1,vector<int>& added_set_1, int& iteration_1)
{
int count_sort;
int emp_sort, emp2_sort, yes_sort, boyut_5;
if(order_rule_1>=3) {
order_rule_1=2;
```

```
}
for (emp_sort=0;emp_sort<reg_emp_1;emp_sort++ ) {
order_number_1[emp_sort]= rand()%(10000-0 + 1) + 0;
//order_number_1[emp_sort]= emp_sort;
}
ordered_list_1.clear();
short_ordered_list_1.clear();
added_set_1.clear();
for (emp_sort=0;emp_sort<reg_emp_1;emp_sort++ ) {
min_no_1= 10000;
if(order_rule_1==0) {
change_no_1=iteration_1;
}
else {
change_no_1=0;
}
min_emp_1=-1;
for (emp2_sort=0;emp2_sort<reg_emp_1;emp2_sort++ ) {
boyut_5=added_set_1.size();
yes_sort=1;
for(count_sort=0; count_sort<boyut_5;count_sort++) {
if(added_set_1[count_sort]==emp2_sort) {
yes_sort=0;
} }
if(yes_sort==1) {
if(((order_rule_1==0) && (last_changed_1[emp2_sort] < change_no_1)) ||((
    order_rule_1==1) && (last_changed_1[emp2_sort]> change_no_1))) {
change_no_1=last_changed_1[emp2_sort];
min_no_1=order_number_1[emp2_sort];
min_emp_1=emp2_sort;
}
else if(((order_rule_1==0) && (last_changed_1[emp2_sort]==change_no_1)) || ((
    order_rule_1==1) && (last_changed_1[emp2_sort]==change_no_1)) || (
    order_rule_1==2)) {
if(order_number_1[emp2_sort] < min_no_1) {
min_no_1=order_number_1[emp2_sort];
min_emp_1=emp2_sort;
} } } }
if(short_ordered_list_1.size() < ((short_list_fraction_1*reg_emp_1) - 1)) {
short_ordered_list_1.push_back(min_emp_1);
}
ordered_list_1.push_back(min_emp_1);
added_set_1.push_back(min_emp_1);
```

```
} }
```

E.2.5.12 'Tabu check' sub-programmes

```
#include "tabu.h"

void check_tabu(int& tabu_1,int weeks_to_plan_1, int reg_emp_1, int all_roles_1,
    int& to_check_tabu_1, int list_sol_1[8][13][48], int ag_list_sol_1
    [8][13][25],int tabu_sol_1[13][48], int ag_tabu_sol_1[13][25])
{
int weeks_2, emp_count, role_count;
tabu_1=1;
emp_count=0;
while(tabu_1==1 && emp_count<reg_emp_1) {
for(weeks_2=0;weeks_2<weeks_to_plan_1;weeks_2++) {
if(list_sol_1[to_check_tabu_1][weeks_2][emp_count]!=tabu_sol_1[weeks_2][emp_count
    ]) {
tabu_1=0;
} }
emp_count=emp_count+1;
}
role_count=0;
while(tabu_1==1 && role_count<all_roles_1) {
for(weeks_2=0;weeks_2<weeks_to_plan_1;weeks_2++) {
if(ag_list_sol_1[to_check_tabu_1][weeks_2][role_count]!=ag_tabu_sol_1[weeks_2][
    role_count]) {
tabu_1=0;
} }
role_count=role_count+1;
} }

void check_tabu_2(int &transfer_sol_to_1,int reg_emp_1, int all_roles_1, int
    weeks_to_plan_1, int list_sol_1[8][13][48], int ag_list_sol_1[8][13][25],int
    tabu_sol_1[13][48], int ag_tabu_sol_1[13][25])
{
int emp_2, rol_2, weeks_2;
for(emp_2=0; emp_2<reg_emp_1; emp_2++)
for(weeks_2=0;weeks_2<weeks_to_plan_1;weeks_2++) {
tabu_sol_1[weeks_2][emp_2]=list_sol_1[0][weeks_2][emp_2];
}
for(rol_2=0; rol_2<all_roles_1;rol_2++)
for(weeks_2=0;weeks_2<weeks_to_plan_1;weeks_2++) {
ag_tabu_sol_1[weeks_2][rol_2]=ag_list_sol_1[0][weeks_2][rol_2];
}
transfer_sol_to_1=0;
}
```

E.2.5.13 'Transfer' sub-programmes

```
#include "transferring_data.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "string"
#include <cstdlib>
#include <ctime>
#include <math.h>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <iterator>
#include <vector>
#include <random>
using namespace std;

void transfer_solution(int& transfer_sol_from_1, int& transfer_sol_to_1, float
    total_cost_1[8], int all_emp_1, int reg_emp_1, int all_roles_1, int
    weeks_to_plan_1, int lambda_1, int allocate_sol_1[8][13][25][49], int
    long_work_sol_1[8][13][25][49][10], float emp_cost_1[8][48], int list_sol_1
    [8][13][48], int board_sol_1[8][13][25][48], int depart_sol_1[8][13][25][48],
     int undertime_sol_1[8][45], int overtime_sol_1[8][45],float ag_cost_1
    [8][13][25],int ag_list_sol_1[8][13][25],int ag_crewchange_1[8][13][25], int
    ag_rboard_sol_1[8][13][25], int ag_rdepart_sol_1[8][13][25])
{
int emp_trnsfr_sol,rol_trnsfr_sol,weeks_trnsfr_sol, l_trnsfr_sol;
if((transfer_sol_to_1<=7 && transfer_sol_to_1>=0) && (transfer_sol_from_1<=7 &&
    transfer_sol_from_1>=0)) {
total_cost_1[transfer_sol_to_1]=total_cost_1[transfer_sol_from_1];
for( emp_trnsfr_sol=0; emp_trnsfr_sol<all_emp_1; emp_trnsfr_sol++)
for(rol_trnsfr_sol=0;rol_trnsfr_sol<all_roles_1;rol_trnsfr_sol++)
for(weeks_trnsfr_sol=0;weeks_trnsfr_sol<weeks_to_plan_1;weeks_trnsfr_sol++) {
allocate_sol_1[transfer_sol_to_1][weeks_trnsfr_sol][rol_trnsfr_sol][
    emp_trnsfr_sol]=allocate_sol_1[transfer_sol_from_1][weeks_trnsfr_sol][
    rol_trnsfr_sol][ emp_trnsfr_sol];
}
```

```
for( emp_trnsfr_sol=0; emp_trnsfr_sol<all_emp_1; emp_trnsfr_sol++)
for(rol_trnsfr_sol=0;rol_trnsfr_sol<all_roles_1;rol_trnsfr_sol++)
for(weeks_trnsfr_sol=0;weeks_trnsfr_sol<weeks_to_plan_1;weeks_trnsfr_sol++)
for(l_trnsfr_sol=0;l_trnsfr_sol<lambda_1;l_trnsfr_sol++) {
long_work_sol_1[transfer_sol_to_1][weeks_trnsfr_sol][rol_trnsfr_sol][
    emp_trnsfr_sol][l_trnsfr_sol]=long_work_sol_1[transfer_sol_from_1][
    weeks_trnsfr_sol][rol_trnsfr_sol][emp_trnsfr_sol][l_trnsfr_sol];
}
for( emp_trnsfr_sol=0; emp_trnsfr_sol<reg_emp_1; emp_trnsfr_sol++) {
emp_cost_1[transfer_sol_to_1][emp_trnsfr_sol]=emp_cost_1[transfer_sol_from_1][
    emp_trnsfr_sol];
}
for( emp_trnsfr_sol=0; emp_trnsfr_sol<reg_emp_1; emp_trnsfr_sol++)
for(weeks_trnsfr_sol=0;weeks_trnsfr_sol<weeks_to_plan_1;weeks_trnsfr_sol++) {
list_sol_1[transfer_sol_to_1][weeks_trnsfr_sol][emp_trnsfr_sol]=list_sol_1[
    transfer_sol_from_1][weeks_trnsfr_sol][emp_trnsfr_sol];
}
for( emp_trnsfr_sol=0; emp_trnsfr_sol<reg_emp_1; emp_trnsfr_sol++)
for(rol_trnsfr_sol=0;rol_trnsfr_sol<all_roles_1;rol_trnsfr_sol++)
for(weeks_trnsfr_sol=0;weeks_trnsfr_sol<weeks_to_plan_1;weeks_trnsfr_sol++) {
board_sol_1[transfer_sol_to_1][weeks_trnsfr_sol][rol_trnsfr_sol][emp_trnsfr_sol]=
    board_sol_1[transfer_sol_from_1][weeks_trnsfr_sol][rol_trnsfr_sol][
    emp_trnsfr_sol];
depart_sol_1[transfer_sol_to_1][weeks_trnsfr_sol][rol_trnsfr_sol][emp_trnsfr_sol
    ]=depart_sol_1[transfer_sol_from_1][weeks_trnsfr_sol][rol_trnsfr_sol][
    emp_trnsfr_sol];
}
for( emp_trnsfr_sol=3; emp_trnsfr_sol<reg_emp_1; emp_trnsfr_sol++) {
undertime_sol_1[transfer_sol_to_1][emp_trnsfr_sol-3]=undertime_sol_1[
    transfer_sol_from_1][emp_trnsfr_sol-3];
overtime_sol_1[transfer_sol_to_1][emp_trnsfr_sol-3]=overtime_sol_1[
    transfer_sol_from_1][emp_trnsfr_sol-3];
}
for(rol_trnsfr_sol=0;rol_trnsfr_sol<all_roles_1;rol_trnsfr_sol++)
for(weeks_trnsfr_sol=0;weeks_trnsfr_sol<weeks_to_plan_1;weeks_trnsfr_sol++) {
ag_cost_1[transfer_sol_to_1][weeks_trnsfr_sol][rol_trnsfr_sol]=ag_cost_1[
    transfer_sol_from_1][weeks_trnsfr_sol][rol_trnsfr_sol];
ag_list_sol_1[transfer_sol_to_1][weeks_trnsfr_sol][rol_trnsfr_sol]=ag_list_sol_1[
    transfer_sol_from_1][weeks_trnsfr_sol][rol_trnsfr_sol];
ag_crewchange_1[transfer_sol_to_1][weeks_trnsfr_sol][rol_trnsfr_sol]=
    ag_crewchange_1[transfer_sol_from_1][weeks_trnsfr_sol][rol_trnsfr_sol];
ag_rboard_sol_1[transfer_sol_to_1][weeks_trnsfr_sol][rol_trnsfr_sol]=
    ag_rboard_sol_1[transfer_sol_from_1][weeks_trnsfr_sol][rol_trnsfr_sol];
ag_rdepart_sol_1[transfer_sol_to_1][weeks_trnsfr_sol][rol_trnsfr_sol]=
    ag_rdepart_sol_1[transfer_sol_from_1][weeks_trnsfr_sol][rol_trnsfr_sol];
} }
transfer_sol_to_1=-1;
transfer_sol_from_1=-1;
}

void evaluate_block_new5(vector<int>& emps_changed_1, vector<int>&
    candidate_emps_1, float& extend_cost_bkwd_1, float &candidate_cost_1, int &
    candidate_exist_1, float total_cost_1[8],int &transfer_sol_from_1, int &
    transfer_sol_to_1, int all_emp_1, int reg_emp_1, int all_roles_1, int
    weeks_to_plan_1, int lambda_1, int allocate_sol_1[8][13][25][49], int
    long_work_sol_1[8][13][25][49][10], float emp_cost_1[8][48], int list_sol_1
    [8][13][48], int board_sol_1[8][13][25][48], int depart_sol_1[8][13][25][48],
     int undertime_sol_1[8][45], int overtime_sol_1[8][45],float ag_cost_1
    [8][13][25],int ag_list_sol_1[8][13][25],int ag_crewchange_1[8][13][25], int
    ag_rboard_sol_1[8][13][25], int ag_rdepart_sol_1[8][13][25])
{
int boyut_100;
if((candidate_exist_1==0) || (total_cost_1[3]<total_cost_1[6])) {
candidate_exist_1=1;
transfer_sol_from_1=3;
transfer_sol_to_1=6;
transfer_solution(transfer_sol_from_1, transfer_sol_to_1,total_cost_1,all_emp_1,
    reg_emp_1, all_roles_1, weeks_to_plan_1,lambda_1,allocate_sol_1,
    long_work_sol_1, emp_cost_1, list_sol_1, board_sol_1, depart_sol_1,
    undertime_sol_1, overtime_sol_1, ag_cost_1, ag_list_sol_1, ag_crewchange_1,
    ag_rboard_sol_1, ag_rdepart_sol_1);
candidate_cost_1=extend_cost_bkwd_1;
for( boyut_100=0; boyut_100<emps_changed_1.size(); boyut_100++) {
candidate_emps_1.push_back(emps_changed_1[boyut_100]);
} } }

void evaluate_block_new10(vector<int>& emps_changed_1, vector<int>&
    candidate_emps_1, float& extend_cost_fwd_1, float &candidate_cost_1, int &
    candidate_exist_1, float total_cost_1[8],int &transfer_sol_from_1, int &
    transfer_sol_to_1, int all_emp_1, int reg_emp_1, int all_roles_1, int
    weeks_to_plan_1, int lambda_1, int allocate_sol_1[8][13][25][49], int
    long_work_sol_1[8][13][25][49][10], float emp_cost_1[8][48], int list_sol_1
    [8][13][48], int board_sol_1[8][13][25][48], int depart_sol_1[8][13][25][48],
     int undertime_sol_1[8][45], int overtime_sol_1[8][45],float ag_cost_1
    [8][13][25],int ag_list_sol_1[8][13][25],int ag_crewchange_1[8][13][25], int
    ag_rboard_sol_1[8][13][25], int ag_rdepart_sol_1[8][13][25])
```

```
{
int boyut_100;
if((candidate_exist_1==0) || (total_cost_1[4]<total_cost_1[6])) {
candidate_exist_1=1;
transfer_sol_from_1=4;
transfer_sol_to_1=6;
transfer_solution(transfer_sol_from_1, transfer_sol_to_1,total_cost_1,all_emp_1,
    reg_emp_1, all_roles_1, weeks_to_plan_1,lambda_1,allocate_sol_1,
    long_work_sol_1, emp_cost_1, list_sol_1, board_sol_1, depart_sol_1,
    undertime_sol_1, overtime_sol_1, ag_cost_1, ag_list_sol_1, ag_crewchange_1,
    ag_rboard_sol_1, ag_rdepart_sol_1);
candidate_cost_1=extend_cost_fwd_1;
for( boyut_100=0; boyut_100<emps_changed_1.size(); boyut_100++) {
candidate_emps_1.push_back(emps_changed_1[boyut_100]);
} } }

void compare_to_best_2(int &best_sol_time_1, int &iteration_1, int ag_best_sols_1
    [51][13][25], int best_sols_1[51][13][48], int &number_best_1, int &
    transfer_sol_from_1, int &transfer_sol_to_1, float total_cost_1[8], int
    all_emp_1, int reg_emp_1, int all_roles_1, int weeks_to_plan_1, int lambda_1,
     int allocate_sol_1[8][13][25][49], int long_work_sol_1[8][13][25][49][10],
    float emp_cost_1[8][48], int list_sol_1[8][13][48], int board_sol_1
    [8][13][25][48], int depart_sol_1[8][13][25][48], int undertime_sol_1[8][45],
     int overtime_sol_1[8][45],float ag_cost_1[8][13][25],int ag_list_sol_1
    [8][13][25],int ag_crewchange_1[8][13][25], int ag_rboard_sol_1[8][13][25],
    int ag_rdepart_sol_1[8][13][25])
{
int weeks_1, emp_1, rol_1;
if(total_cost_1[0]<total_cost_1[1]) {
transfer_sol_from_1=0;
transfer_sol_to_1=1;
transfer_solution(transfer_sol_from_1, transfer_sol_to_1,total_cost_1,all_emp_1,
    reg_emp_1, all_roles_1, weeks_to_plan_1,lambda_1,allocate_sol_1,
    long_work_sol_1, emp_cost_1, list_sol_1, board_sol_1, depart_sol_1,
    undertime_sol_1, overtime_sol_1, ag_cost_1, ag_list_sol_1, ag_crewchange_1,
    ag_rboard_sol_1, ag_rdepart_sol_1);
number_best_1=1;
for(weeks_1=0;weeks_1<weeks_to_plan_1;weeks_1++)
for(emp_1=0;emp_1<reg_emp_1;emp_1++) {
best_sols_1[0][weeks_1][emp_1]=list_sol_1[1][weeks_1][emp_1];    // AL - Have
    changed this from "best_sols[1][weeks_1][emp_1]..."
}
for(weeks_1=0;weeks_1<weeks_to_plan_1;weeks_1++)
for(rol_1=0;rol_1<all_roles_1;rol_1++) {
ag_best_sols_1[0][weeks_1][rol_1]=ag_list_sol_1[1][weeks_1][rol_1];     // AL -
    Have changed this from "ag_best_sols[1][weeks_1][rol_1]..."
}
best_sol_time_1=iteration_1;
} }

void swap_calc4(vector<int>& candidate_emps_1, int& swap_emp_1,int& emp_extend_1,
     float& candidate_cost_1, float& swapping_cost_1,float total_cost_1[8], int&
    accept_rule_1, int& candidate_exist_1, int& transfer_sol_from_1, int&
    transfer_sol_to_1, int all_emp_1, int reg_emp_1, int all_roles_1, int
    weeks_to_plan_1, int& lambda_1, int allocate_sol_1[8][13][25][49], int
    long_work_sol_1[8][13][25][49][10], float emp_cost_1[8][48], int list_sol_1
    [8][13][48], int board_sol_1[8][13][25][48], int depart_sol_1[8][13][25][48],
     int undertime_sol_1[8][45], int overtime_sol_1[8][45],float ag_cost_1
    [8][13][25],int ag_list_sol_1[8][13][25],int ag_crewchange_1[8][13][25], int
    ag_rboard_sol_1[8][13][25], int ag_rdepart_sol_1[8][13][25] )
{
if(total_cost_1[5]>= total_cost_1[accept_rule_1]) {
if(candidate_exist_1 ==0 || total_cost_1[5]< total_cost_1[6]) {
candidate_exist_1=1;
transfer_sol_from_1=5;
transfer_sol_to_1=6;
transfer_solution(transfer_sol_from_1, transfer_sol_to_1, total_cost_1 ,
    all_emp_1, reg_emp_1, all_roles_1, weeks_to_plan_1, lambda_1, allocate_sol_1,
     long_work_sol_1, emp_cost_1, list_sol_1, board_sol_1, depart_sol_1,
    undertime_sol_1, overtime_sol_1, ag_cost_1, ag_list_sol_1 ,ag_crewchange_1,
    ag_rboard_sol_1, ag_rdepart_sol_1);
candidate_cost_1=swapping_cost_1;
candidate_emps_1.clear();                   // AL - Have added this new line...
candidate_emps_1.push_back(emp_extend_1);         // AL - ... so that following
    this line, "emp_extend" is the ONLY entry in the set "candidate_emps"
if(swap_emp_1!=48) {
candidate_emps_1.push_back(swap_emp_1);
} } } }
```

E.2.5.14 ‘Updating’ sub-programmes

```
#include "swap_change_update.h"

void update_swaps_and_changes(vector<int>& emps_to_update_1,int swaps_examined_1
    [48][48], int reg_emp_1, int& last_kick_time_1, int& iteration_1, int
    last_changed_1[48])
```

```
{
int emp_300, emp_301, emp_302, emp_303, in_emps_to_update, boyut_1;
boyut_1=emps_to_update_1.size();
for(emp_300=0; emp_300<boyut_1; emp_300++) {
emp_301=emps_to_update_1[emp_300];
for(emp_302=0; emp_302<reg_emp_1; emp_302++) // AL - These lines equate to
{             //     'swaps_examined(e) := {}'
swaps_examined_1[emp_301][emp_302]=0; //   (where 'e' is now 'emp_301')
}             //
for(emp_302=0; emp_302<reg_emp_1; emp_302++) //
{             //
in_emps_to_update=0;  //
for(emp_303=0; emp_303<boyut_1; emp_303++)        // AL - The lines equate to
{                            //     'forall(f in REG_EMP | f not in
    emps_to_update)'
if(emps_to_update_1[emp_303]==emp_302) //  (where 'f' is now 'emp_302')
{
in_emps_to_update=1;
} }
if(in_emps_to_update==0) {
if(swaps_examined_1[emp_302][emp_301]==1) {
swaps_examined_1[emp_302][emp_301]=0; // AL - equates to 'swaps_examined(f) -= {e
    }'
}             //     (Note, no 'if' required - just ensure it equals zero)
} } }
if(last_kick_time_1==iteration_1) {
for(emp_301=0; emp_301<reg_emp_1; emp_301++) {
last_changed_1[emp_301]=iteration_1;
} }
else {
for(emp_300=0; emp_300<boyut_1; emp_300++) //
{
emp_301=emps_to_update_1[emp_300];
last_changed_1[emp_301] =iteration_1;
} } }

void swap_calc3(vector<int>& emps_changed_1,int& no_nonreduce_1,float&
    swapping_cost_1, float total_cost_1[8], int& accept_rule_1, int& do_swap_1,
    vector<int>& emps_to_update_1,int swaps_examined_1[48][48], int reg_emp_1,
    int& last_kick_time_1, int& iteration_1, int last_changed_1[48])
{
swapping_cost_1=total_cost_1[5] - total_cost_1[0];
if(total_cost_1[5]< total_cost_1[accept_rule_1]) {
```

```
do_swap_1=1;
no_nonreduce_1=0;
emps_to_update_1.clear();           // AL - Have added this line, further to
    conversation on 06/02/15
emps_to_update_1.reserve(emps_changed_1.size());
copy(emps_changed_1.begin(),emps_changed_1.end(), back_inserter(emps_to_update_1)
    );
update_swaps_and_changes( emps_to_update_1,swaps_examined_1, reg_emp_1,
    last_kick_time_1, iteration_1, last_changed_1);
} }
```

E.2.5.15 'Usable blocks' sub-programmes

```
#include "find_usables.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "string"
#include <cstdlib>
#include <ctime>
#include <math.h>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <iterator>
#include <vector>
#include <random>
using namespace std;

void evaluate_block_new1(int required_1[13][25],int rest_zero_1[48], int&
    block_end_1, int& task_extend_1,int& emp_extend_1,int eligible_1[13][25][49],
    int weeks_to_plan_1,float& extend_cost_fwd_1,float& extend_cost_bkwd_1, float
    & swapping_cost_1, int& do_extend_bkwd_1, int& do_extend_fwd_1, int&
    do_swap_1, int& block_len_1, int max_work_1[48],int& max_bkwd_extend_1, int&
    max_fwd_extend_1)
{
int weeks_12,weeks_13;
extend_cost_fwd_1=0;
extend_cost_bkwd_1=0;
swapping_cost_1=0;
do_extend_bkwd_1=0;
do_extend_fwd_1=0;
```

```
do_swap_1=0;
if((max_work_1[emp_extend_1]-block_len_1) > 0) {
max_bkwd_extend_1=max_work_1[emp_extend_1] - block_len_1;
max_fwd_extend_1= max_work_1[emp_extend_1] - block_len_1;
}
else {
max_bkwd_extend_1=0;
max_fwd_extend_1=0;
}
if(max_bkwd_extend_1 > weeks_to_plan_1 - block_end_1-1) {
max_bkwd_extend_1= weeks_to_plan_1 -1-block_end_1;
}
if(max_bkwd_extend_1> 0) {
weeks_13=max_bkwd_extend_1;
for(weeks_12=1;weeks_12<=weeks_13;weeks_12++) {
if(eligible_1[block_end_1+weeks_12][ task_extend_1][emp_extend_1]==0|| required_1
    [block_end_1+weeks_12][task_extend_1]==0) {
if(max_bkwd_extend_1>= weeks_12) {
max_bkwd_extend_1= weeks_12-1;
} } } }
if(block_end_1==-1 && rest_zero_1[emp_extend_1]> 0)       // AL - Have changed
    this from "if(block_end ==0..."
{
max_bkwd_extend_1= 0;
} }

void evaluate_block_new2(int all_roles_1, int reg_emp_1, int weeks_to_plan_1, int
     max_work_1[48], int& conflict_found_bkwd_1, int reserve_list_bkwd_1[13][25],
     int& in_reserve_bkwd_1, int& block_end_1, int& task_extend_1, int
    work_zero_1[48], int& length_count_1,int &emp_extend_1,int& rest_count_1, int
    & extend_len_bkwd_1, vector<int>& emps_changed_1,vector<int>&
    ag_roles_changed_1, int list_sol_1[8][13][48], int ag_list_sol_1[8][13][25],
    int min_rest_1[25])
{
int weeks_6,rol_6,emp_6;
emps_changed_1.clear();
ag_roles_changed_1.clear();
for(emp_6=0;emp_6<reg_emp_1;emp_6++)
for(weeks_6=0;weeks_6<weeks_to_plan_1;weeks_6++) {
list_sol_1[3][weeks_6][emp_6]=list_sol_1[0][weeks_6][emp_6];
}
for(rol_6=0;rol_6<all_roles_1;rol_6++)
for(weeks_6=0;weeks_6<weeks_to_plan_1;weeks_6++) {
```

```
ag_list_sol_1[3][weeks_6][rol_6]=ag_list_sol_1[0][weeks_6][rol_6];
reserve_list_bkwd_1[weeks_6][rol_6]=0; // AL - Have changed this from "=-1", so
    as to make this an array of 0-1 variables
}
for(weeks_6=0;weeks_6<weeks_to_plan_1;weeks_6++) {
list_sol_1[3][weeks_6][emp_extend_1]=-1;
}
emps_changed_1.push_back(emp_extend_1);
rest_count_1=0;             // AL - Have added this line...
length_count_1=work_zero_1[emp_extend_1];                // AL - ... and this
    line...
in_reserve_bkwd_1=0;        // AL - ... and also this line.
for(weeks_6=0;weeks_6<weeks_to_plan_1;weeks_6++) {
if(weeks_6<=block_end_1) {
list_sol_1[3][weeks_6][emp_extend_1]=list_sol_1[0][weeks_6][emp_extend_1];
if(list_sol_1[0][weeks_6][emp_extend_1]==-1) {
length_count_1=0;
}
else {
length_count_1=length_count_1+1;
} }
else if(weeks_6<=(block_end_1+extend_len_bkwd_1)) {
list_sol_1[3][weeks_6][emp_extend_1]=task_extend_1;
length_count_1=length_count_1+1;
if((list_sol_1[0][weeks_6][emp_extend_1]!=task_extend_1) && (list_sol_1[0][
    weeks_6][emp_extend_1]!=-1)) {
reserve_list_bkwd_1[weeks_6][list_sol_1[0][weeks_6][emp_extend_1]]=1;
in_reserve_bkwd_1=in_reserve_bkwd_1+1;
} }
else {
if((list_sol_1[0][weeks_6][emp_extend_1]!=-1) && (rest_count_1<min_rest_1[
    emp_extend_1] || length_count_1>max_work_1[emp_extend_1] )) {
list_sol_1[3][weeks_6][emp_extend_1]=-1;
reserve_list_bkwd_1[weeks_6][list_sol_1[0][weeks_6][emp_extend_1]]=1;   // AL -
    Have added this line
in_reserve_bkwd_1=in_reserve_bkwd_1+1;
length_count_1=0;
rest_count_1=rest_count_1+1;
}
else {
list_sol_1[3][weeks_6][emp_extend_1]=list_sol_1[0][weeks_6][emp_extend_1];
if(list_sol_1[0][weeks_6][emp_extend_1]==-1) {
rest_count_1=rest_count_1+1;
```

```
length_count_1=0;
}
else {
length_count_1=length_count_1+1;
} } } }
conflict_found_bkwd_1=0;
for(weeks_6=1;weeks_6<=extend_len_bkwd_1;weeks_6++) {
if(ag_list_sol_1[3][block_end_1+weeks_6][task_extend_1]==1) {
conflict_found_bkwd_1=conflict_found_bkwd_1+1;
ag_list_sol_1[3][block_end_1+weeks_6][task_extend_1]=0;
ag_roles_changed_1.push_back(task_extend_1);
} }
for(weeks_6=0;weeks_6<weeks_to_plan_1;weeks_6++)
for(rol_6=0;rol_6<all_roles_1;rol_6++) {
if(reserve_list_bkwd_1[weeks_6][rol_6]==1) {
if(ag_list_sol_1[3][weeks_6-1][rol_6]==1) {
ag_list_sol_1[3][weeks_6][rol_6]=1;
ag_roles_changed_1.push_back(rol_6);
reserve_list_bkwd_1[weeks_6][rol_6]=0;
in_reserve_bkwd_1=in_reserve_bkwd_1-1;
} } } }

void evaluate_block_new3(int &to_check_tabu_1, int all_roles_1, int reg_emp_1,
    int weeks_to_plan_1, int max_work_1[48], int& prev_work_1, int& add_to_emp_1,
     int& conflict_found_bkwd_1, int reserve_list_bkwd_1[13][25], int&
    in_reserve_bkwd_1, int& block_end_1, int& task_extend_1, int work_zero_1[48],
     int& length_count_1,int &emp_extend_1,int& rest_count_1, int&
    extend_len_bkwd_1, vector<int>& emps_changed_1,vector<int>&
    ag_roles_changed_1, int list_sol_1[8][13][48], int ag_list_sol_1[8][13][25],
    int eligible_1[13][25][49], int rest_zero_1[25], int min_rest_1[25])
{ int emp_6, weeks_6, rol_6;
for(emp_6=0;emp_6<reg_emp_1;emp_6++) {
if(emp_6!=emp_extend_1) {
if((conflict_found_bkwd_1< extend_len_bkwd_1) || (in_reserve_bkwd_1>0)) {
for(weeks_6=0; weeks_6<weeks_to_plan_1; weeks_6++) {
list_sol_1[3][weeks_6][emp_6]=-1;
}
rest_count_1=0;
length_count_1=work_zero_1[emp_6];
add_to_emp_1=-1;      // AL - Have changed this from "add_to_emp=0;"
if(work_zero_1[emp_6]>0) {
prev_work_1=1;
rest_count_1=0;
```

```
}
else {
prev_work_1=0;
rest_count_1=min_rest_1[emp_6]- rest_zero_1[emp_6];
}
for(weeks_6=0; weeks_6<weeks_to_plan_1; weeks_6++)        // AL - Have removed the
     line "for(rol_6=0;rol_6<all_roles;rol_6++)", and replaced with the 'for all
    weeks' loop from above
{
//if(list_sol[0][weeks_6][emp_6]==rol_6)   // AL - Have removed this. Not sure
    where it came from.
//{
if(weeks_6<=block_end_1) {
list_sol_1[3][weeks_6][emp_6]=list_sol_1[0][weeks_6][emp_6];
if(list_sol_1[0][weeks_6][emp_6]!=-1) {
length_count_1=length_count_1+1;
prev_work_1=1;
rest_count_1=0;
}
else {
length_count_1=0;
prev_work_1=0;
rest_count_1=rest_count_1+1;
} }
else {
if(list_sol_1[0][weeks_6][emp_6]==-1) {
if(add_to_emp_1!=-1) {
list_sol_1[3][weeks_6][emp_6]=add_to_emp_1;
emps_changed_1.push_back(emp_6);
length_count_1=length_count_1+1;
reserve_list_bkwd_1[weeks_6][add_to_emp_1]=0;
in_reserve_bkwd_1=in_reserve_bkwd_1-1;
rest_count_1=0;
prev_work_1=1;
if(weeks_6<weeks_to_plan_1-1) {
if((length_count_1>=max_work_1[emp_6]) || (eligible_1[weeks_6+1][add_to_emp_1][
    emp_6]<1) || (reserve_list_bkwd_1[weeks_6+1][add_to_emp_1]==0)) // AL - Have
    changed this from "...reserve_list_bkwd[weeks_6][add_to_emp!=1])"
{
add_to_emp_1=-1;
} }
else {
add_to_emp_1=-1;
```

```
} }
else {
list_sol_1[3][weeks_6][emp_6]=list_sol_1[0][weeks_6][emp_6];
length_count_1=0;
rest_count_1=rest_count_1+1;
prev_work_1=0;
} }
else {
if((list_sol_1[0][weeks_6][emp_6]==task_extend_1) && (weeks_6<=(block_end_1+
    extend_len_bkwd_1))) // AL - Have changed this from "if(rol_6==task_extend &&
     emp_6<=..."
{
conflict_found_bkwd_1=conflict_found_bkwd_1+1;
list_sol_1[3][weeks_6][emp_6]=-1;
emps_changed_1.push_back(emp_6);
length_count_1=0;
rest_count_1=rest_count_1+1;
prev_work_1=0;
}
else {
if((prev_work_1==0) && (rest_count_1<min_rest_1[emp_6])) {
list_sol_1[3][weeks_6][emp_6]=-1;
ag_list_sol_1[3][weeks_6][list_sol_1[0][weeks_6][emp_6]]=1;      // AL - Have
    changed this from "...[list_sol[3][weeks_6][emp_6]]=1;"
ag_roles_changed_1.push_back(list_sol_1[0][weeks_6][emp_6]); // AL - Have changed
     this from "...(list_sol[3][weeks_6][emp_6]);"
rest_count_1=rest_count_1+1;
}
else {
list_sol_1[3][weeks_6][emp_6]=list_sol_1[0][weeks_6][emp_6]; // AL - Have changed
     this from "list_sol[3][weeks_6][emp_6]=list_sol[3][weeks_6][emp_6];"
length_count_1=length_count_1+1;
rest_count_1=0;
prev_work_1=1;
if(weeks_6<weeks_to_plan_1-1) {
if((length_count_1<max_work_1[emp_6]) && (eligible_1[weeks_6+1][list_sol_1[0][
    weeks_6][emp_6]][emp_6]==1) && (reserve_list_bkwd_1[weeks_6+1][list_sol_1[0][
    weeks_6][emp_6]]==1)) // AL - Have changed this from "if(length_count<=
    max_work[emp_6]-1 && eligible[weeks_6+1][rol_6][emp_6]==1 &&
    reserve_list_bkwd[weeks_6+1][rol_6]==1)"
{
add_to_emp_1=list_sol_1[0][weeks_6][emp_6];       // AL - Have changed this from
     "add_to_emp=rol_6;"
}
else {
add_to_emp_1=-1;
} }
else {
add_to_emp_1=-1;
} } } } } } } } }
for(rol_6=0;rol_6<all_roles_1;rol_6++)
for(weeks_6=0;weeks_6<weeks_to_plan_1;weeks_6++) {
if(reserve_list_bkwd_1[weeks_6][rol_6]==1)        // AL - Have changed this from
    "...!=-1)"
{
ag_list_sol_1[3][weeks_6][rol_6]=1;
ag_roles_changed_1.push_back(rol_6);
} }
to_check_tabu_1=3;
}

void evaluate_block_new4(float& extend_cost_bkwd_1, float total_cost_1[8], int &
    do_extend_bkwd_1, int &no_nonreduce_1, vector<int>&emps_to_update_1, int&
    accept_rule_1, vector<int>& emps_changed_1 )
{
extend_cost_bkwd_1=total_cost_1[3]-total_cost_1[accept_rule_1];
if(total_cost_1[3]<total_cost_1[accept_rule_1])   // AL - Have changed this from
    "if(total_cost[3]<total_cost[0])"
{
do_extend_bkwd_1=1;
no_nonreduce_1=0;
emps_to_update_1.clear();
emps_to_update_1.reserve(emps_changed_1.size());
copy(emps_changed_1.begin(),emps_changed_1.end(),back_inserter(emps_to_update_1))
    ;
} }

void evaluate_block_new6(int &extend_len_fwd_1,int &max_fwd_extend_1, int &
    block_start_1, int rest_zero_1[48], int& emp_extend_1, int summation_1, int
    vessels_1, int roles_1[25], int& task_extend_1, int starting_1[25][49],int
    min_rest_1[48], int eligible_1[13][25][49], int required_1[13][25])
{             int rol_13, weeks_13, weeks_12;
if(max_fwd_extend_1 > block_start_1 - rest_zero_1[emp_extend_1]) {
max_fwd_extend_1=block_start_1- rest_zero_1[emp_extend_1];       // AL - Have
    changed this from "...=block_start - rest_zero[emp_extend] - 1"
}
```

```
summation_1=0;
for(rol_13=0;rol_13<vessels_1;rol_13++) {
if(task_extend_1!=roles_1[rol_13]) {
summation_1=summation_1+ starting_1[rol_13][emp_extend_1];
} }
if(summation_1> 0) {
if(max_fwd_extend_1 > (block_start_1 - min_rest_1[emp_extend_1])) {
max_fwd_extend_1=block_start_1 - min_rest_1[emp_extend_1];
} }
if(max_fwd_extend_1> 0) {
summation_1=0;
for(rol_13=0;rol_13<vessels_1;rol_13++) {
if(task_extend_1==roles_1[rol_13]) {
summation_1=summation_1+starting_1[rol_13][emp_extend_1];
} }
weeks_13=max_fwd_extend_1;
for(weeks_12=1;weeks_12<=weeks_13;weeks_12++) {
if(max_fwd_extend_1 >= weeks_12) {
if( summation_1 > 0 && (block_start_1 - weeks_12)> 0 && (block_start_1 - weeks_12
     < min_rest_1[emp_extend_1])) {
max_fwd_extend_1= weeks_12-1;
}
if(eligible_1[block_start_1-weeks_12][task_extend_1][emp_extend_1]==0 ||
    required_1[block_start_1-weeks_12][task_extend_1]==0) {
max_fwd_extend_1=weeks_12 -1;
} } } }
extend_len_fwd_1=max_fwd_extend_1;
}

void evaluate_block_new7(int &extend_len_fwd_1,vector<int>& emps_changed_1,
    vector<int>& ag_roles_changed_1, int reg_emp_1, int weeks_to_plan_1, int
    list_sol_1[8][13][48], int all_roles_1, int ag_list_sol_1[8][13][25], int
    reserve_list_fwd_1[13][25], int &emp_extend_1, int &rest_count_1,int &
    in_reserve_fwd_1,int &task_extend_1, int min_rest_1[48], int &
    conflict_found_fwd_1, int &block_start_1)
{ int emp_7, rol_7, weeks_7, rol_40;
emps_changed_1.clear();
ag_roles_changed_1.clear();

for(emp_7=0;emp_7<reg_emp_1;emp_7++)
for(weeks_7=0;weeks_7<weeks_to_plan_1;weeks_7++) {
list_sol_1[4][weeks_7][emp_7]=list_sol_1[0][weeks_7][emp_7];
}
```

```
for(rol_7=0;rol_7<all_roles_1;rol_7++)
for(weeks_7=0;weeks_7<weeks_to_plan_1;weeks_7++) {
ag_list_sol_1[4][weeks_7][rol_7]=ag_list_sol_1[0][weeks_7][rol_7];
reserve_list_fwd_1[weeks_7][rol_7]=0;
}
for(weeks_7=0;weeks_7<weeks_to_plan_1;weeks_7++) {
list_sol_1[4][weeks_7][emp_extend_1]=-1;
}
rest_count_1=0;
//length_count=work_zero[emp_extend]; // AL - Have removed this line. Don't see
    that it's needed here
in_reserve_fwd_1=0;                         // AL - Have changed this from "
    in_reserve_bkwd=0"
for(weeks_7=weeks_to_plan_1-1; weeks_7>=0;weeks_7--) {
rol_40 = list_sol_1[0][weeks_7][emp_extend_1];     // AL - Have added this line. "
    rol_40" is essentially "j" from the Xpress programme
if(weeks_7>=block_start_1) {
list_sol_1[4][weeks_7][emp_extend_1]=rol_40; // AL - Have changed this from "
    reverse_new_list[weeks_7][emp_extend]=reverse_current_list[weeks_7][
    emp_extend];"
}
else if(weeks_7>=(block_start_1-extend_len_fwd_1)) // AL - Have changed this from
     "...weeks_7<=block_start..."
{
if(rol_40!=task_extend_1 )          // AL - Have changed this from "if(
    reverse_current_list[weeks_7][emp_extend]!=task_extend )"
{
list_sol_1[4][weeks_7][emp_extend_1]=task_extend_1;       // AL - Have changed
    this from "reverse_new_list[weeks_7][emp_extend]=task_extend;"
if(rol_40!=-1)       // AL - Have changed this from "if(reverse_current_list[
    weeks_7][emp_extend]!=-1)"
{
reserve_list_fwd_1[weeks_7][rol_40]=1; // AL - Have changed from from "
    reserve_list_fwd[weeks_7][reverse_current_list[weeks_7][emp_extend]]=1;"
in_reserve_fwd_1=in_reserve_fwd_1+1;
} } }
else {
if(rol_40!=-1 && rest_count_1 <min_rest_1[emp_extend_1]) // AL - Have changed
    this from "if(reverse_current_list[weeks_7][emp_extend]!=-1 &&..."
{
list_sol_1[4][weeks_7][emp_extend_1]=-1;          // AL - Have changed this from
    "reverse_new_list[weeks_7][emp_extend]=-1;"
reserve_list_fwd_1[weeks_7][rol_40]=1;     // AL - Have changed this from "
```

```
    reserve_list_fwd[weeks_7][reverse_current_list[weeks_7][emp_extend]]=1;"
in_reserve_fwd_1=in_reserve_fwd_1+1;
ag_roles_changed_1.push_back(rol_40);      // AL - Have changed this from "
    ag_roles_changed.push_back(reverse_current_list[weeks_7][emp_extend]);"
rest_count_1=rest_count_1+1;
}
else {
list_sol_1[4][weeks_7][emp_extend_1] = rol_40;     // AL - Have changed this from
    "reverse_new_list[weeks_7][reverse_current_list[weeks_7][emp_extend]]=1;"
if(rol_40==-1)               // AL - Have changed this from "if(
    reverse_current_list[weeks_7][emp_extend]==-1)"
{
rest_count_1=rest_count_1+1;
} } } }
emps_changed_1.push_back(emp_extend_1);
conflict_found_fwd_1=0;
for(weeks_7=1;weeks_7<=extend_len_fwd_1;weeks_7++) {
if(ag_list_sol_1[4][block_start_1-weeks_7][task_extend_1]==1) {
conflict_found_fwd_1=conflict_found_fwd_1+1;
ag_list_sol_1[4][block_start_1-weeks_7][task_extend_1]=0;        // AL - Have
    changed this from "...[block_end+weeks_7]..."
ag_roles_changed_1.push_back(task_extend_1);
} }
for(rol_7=0;rol_7<all_roles_1;rol_7++)
for(weeks_7=0;weeks_7<weeks_to_plan_1;weeks_7++) {
if(reserve_list_fwd_1[weeks_7][rol_7]==1) {
if(ag_list_sol_1[4][weeks_7+1][rol_7]==1) {
ag_list_sol_1[4][weeks_7][rol_7]=1;
ag_roles_changed_1.push_back(rol_7);
reserve_list_fwd_1[weeks_7][rol_7]=0;              // AL - Have changed this from
    "removed.push_back(weeks_7);"
in_reserve_fwd_1=in_reserve_fwd_1-1;
} } } }

void evaluate_block_new8(int rest_zero_1[48], int starting_1[25][49], int
    eligible_1[13][25][49], int max_work_1[48], int &length_count_1,int &
    to_check_tabu_1, int &summation_1x, int &summation_x, int &prev_work_1,int &
    add_to_emp_1, int &extend_len_fwd_1,vector<int>& emps_changed_1, vector<int>&
     ag_roles_changed_1, int reg_emp_1, int weeks_to_plan_1, int list_sol_1
    [8][13][48], int all_roles_1, int ag_list_sol_1[8][13][25], int
    reserve_list_fwd_1[13][25], int &emp_extend_1, int &rest_count_1,int &
    in_reserve_fwd_1,int &task_extend_1, int min_rest_1[48], int &
    conflict_found_fwd_1, int &block_start_1)
```

```
{
int rol_44, emp_7, weeks_7, rol_7;
for(emp_7=0;emp_7<reg_emp_1;emp_7++) {
if(emp_7!=emp_extend_1) {
if((conflict_found_fwd_1< extend_len_fwd_1) || (in_reserve_fwd_1>0)) // AL - Have
     changed this from "if(conflict_found_bkwd< ..."
{
rest_count_1=min_rest_1[emp_7];
length_count_1=0;
add_to_emp_1=-1;             // AL - Have changed this from "add_to_emp=0;" to
    try to be consistent. Other 'strings' equal to "" in Xpress are given as
    '=-1' in the C++ prog.
prev_work_1=0;
for(weeks_7=weeks_to_plan_1-1; weeks_7>= 0; weeks_7--) // AL - Have added this
    start of loop here, rather than above
{
list_sol_1[4][weeks_7][emp_7]=-1;
rol_44 = list_sol_1[0][weeks_7][emp_7]; // AL - As before, no need to use the
    reversed lists here, and "rol_44" has been added to make the notation easier
    to change
if(weeks_7>= block_start_1)                        // AL - Have changed this from
    "if(12-weeks_7>= block_start)"
{
list_sol_1[4][weeks_7][emp_7]=rol_44;      // AL - Have changed this from "
    reverse_new_list[weeks_7][emp_7]=reverse_current_list[weeks_7][emp_7];"
if(rol_44!=-1)        // AL - Have changed this from "if(reverse_current_list[
    weeks_7][emp_7]!=-1)"
{
prev_work_1=1;
length_count_1=length_count_1+1;
}
else {
prev_work_1=0;
length_count_1=0;
} }
else {
if(rol_44==-1)        // AL - Have changed this from "if(reverse_current_list[
    weeks_7][emp_7]==-1)"
{
if(add_to_emp_1!=-1)         // AL - Have changed this from "if(add_to_emp==0)"
{
list_sol_1[4][weeks_7][emp_7]=add_to_emp_1;               // AL - Have changed
    this from "reverse_new_list[weeks_7][add_to_emp]=1;"
```

```
emps_changed_1.push_back(emp_7);                          // AL - Have changed
    this from "emps_changed.push_back(add_to_emp);
length_count_1= length_count_1 + 1;
reserve_list_fwd_1[weeks_7][add_to_emp_1]=0;
in_reserve_fwd_1= in_reserve_fwd_1-1;
prev_work_1=1;               // AL - Have changed this from "prev_work=0;
rest_count_1=0;
if(weeks_7>= 1) {
if((length_count_1 >= max_work_1[emp_7]) || (eligible_1[weeks_7-1][add_to_emp_1][
    emp_7] < 1) || (reserve_list_fwd_1[weeks_7-1][add_to_emp_1]==0) || (weeks_7
    <= rest_zero_1[emp_7]))  // AL - Have changed this from "...reserve_list_fwd[
    weeks_7-1][add_to_emp]==-1 || weeks_7-1 <=..."
{
add_to_emp_1=-1;
}
else {
summation_x=0;
summation_1x=0;                     // AL - Have added this here so that both '
    summations' can be calculated in the same loop...
for(rol_7=0;rol_7<all_roles_1;rol_7++) {
if(rol_7!=add_to_emp_1) {
summation_x=summation_x+starting_1[rol_7][emp_7];
}
else          // AL - ...so have also added this 'else'...
{
summation_1x=summation_1x+starting_1[rol_7][emp_7];       // AL - ... and this
    line.
}      }
if((summation_x > 0 && weeks_7 <= min_rest_1[emp_7])|| ((summation_1x > 0) && (
    weeks_7>= 2) && (weeks_7<= min_rest_1[emp_7])))  // AL - Have changed this
    from "if((summation > 0 && time_count-1 <= min_rest[emp_7])|| (summation_1 >
    0 && 12-weeks_7-1>= 2 && 12-weeks_7-1<= min_rest[emp_7])) "
//
{
add_to_emp_1=-1;
} } }
else {
add_to_emp_1=-1;
} }
else {
list_sol_1[4][weeks_7][emp_7] = rol_44;           // AL - Have changed this from
    "reverse_new_list[weeks_7][emp_7]=reverse_current_list[weeks_7][emp_7];"
length_count_1=0;
```

```
rest_count_1=rest_count_1+ 1;
prev_work_1=0;
} }
else {
if((rol_44==task_extend_1) && (weeks_7>= (block_start_1 - extend_len_fwd_1)))  //
     AL - Have changed this from "if(reverse_current_list[weeks_7][emp_7] ==
    task_extend && 12-weeks_7>= block_start - extend_len_fwd)"
{
conflict_found_fwd_1= conflict_found_fwd_1 + 1;
list_sol_1[4][weeks_7][emp_7]=-1;          // AL - Have changed this from "
    reverse_new_list [weeks_7][emp_7]=-1;"
emps_changed_1.push_back(emp_7);
length_count_1= 0;
rest_count_1= rest_count_1 + 1;
prev_work_1=0;
}
else {
if(prev_work_1==0 && rest_count_1 < min_rest_1[emp_7]) {
list_sol_1[4][weeks_7][emp_7]=-1;          // AL - Have changed this from "
    reverse_new_list [weeks_7][emp_7]=-1;"
ag_list_sol_1[4][weeks_7][rol_44]=1;       // AL - Have changed this from "
    ag_list_sol[4][12-weeks_7][reverse_current_list[weeks_7][emp_7]]=1;"
ag_roles_changed_1.push_back(rol_44);      // AL - Have changed this from "
    ag_roles_changed.push_back(reverse_current_list[weeks_7][emp_7]);"
rest_count_1=rest_count_1 + 1;
}
else {
list_sol_1[4][weeks_7][emp_7]=rol_44;      // AL - Have changed this from "
    reverse_new_list [weeks_7][emp_7]=reverse_current_list[weeks_7][emp_7];"
length_count_1=length_count_1 + 1;
rest_count_1=0;
prev_work_1=1;
if(weeks_7> 0) // AL - Have changed this from "if(12-weeks_7> 1)"
{
if((length_count_1< max_work_1[emp_7]) && (eligible_1[weeks_7-1][rol_44][emp_7] >
     0) && (reserve_list_fwd_1[weeks_7-1][rol_44]==1) && (weeks_7 > rest_zero_1[
    emp_7]) ) // AL - Have changed this from "if(length_count < max_work[emp_7]
    && eligible[12-weeks_7-1][reverse_current_list[weeks_7][emp_7]][emp_7] > 0 &&
     reserve_list_fwd[weeks_7-1][emp_7]==reverse_current_list[weeks_7][emp_7] &&
    12-weeks_7-1 > rest_zero[emp_7])"
{
summation_x=0;
summation_1x=0;       // AL - Have added this so we can calculate both 'summations
```

```
    ' in the one loop...
for(rol_7=0;rol_7<all_roles_1;rol_7++) {
if(rol_7!=add_to_emp_1) {
summation_x=summation_x+starting_1[rol_7][emp_7];
}
else          // AL - ... so have also added this 'else'...
{
summation_1x=summation_1x+starting_1[rol_7][emp_7]; // AL - ... and also this
    command
} }
if((summation_x< 1 || weeks_7 > min_rest_1[emp_7]) && (summation_1x < 1 ||
    weeks_7-1==0 || weeks_7 > min_rest_1[emp_7]))   // AL - Have changed this
    from "if((summation < 1 || 12-weeks_7-1 > min_rest[emp_7]) && (summation_1 <
    1 || 12-weeks_7-1 == 1 || 12-weeks_7-1 > min_rest[emp_7]))"
{
add_to_emp_1= rol_44;        // AL - Have changed this from "add_to_emp = -1;"
}
else {
add_to_emp_1= -1;
} }
else {
add_to_emp_1=-1;
} }
else {
add_to_emp_1=-1;
} } } }
}             // AL - Have added this close bracket to end the 'else' for "if(
    weeks_7>= block_start)"...
}             // AL - ... and this one to end the 'for all weeks 12 to 0' loop
} }  }
for(rol_7=0;rol_7<all_roles_1;rol_7++)
for(weeks_7=0;weeks_7<weeks_to_plan_1;weeks_7++) {
if(reserve_list_fwd_1[weeks_7][rol_7]==1)          // AL - Have changed this from
    "...[weeks_7][rol_7]!=-1)"
{
ag_list_sol_1[4][weeks_7][rol_7]=1;
ag_roles_changed_1.push_back(rol_7);
} }
to_check_tabu_1=4;
}

void evaluate_block_new9(float& extend_cost_fwd_1, float total_cost_1[8], int &
    do_extend_fwd_1, int &no_nonreduce_1, vector<int>&emps_to_update_1, int&
    accept_rule_1, vector<int>& emps_changed_1 )
{
extend_cost_fwd_1=total_cost_1[4]-total_cost_1[0];
if(total_cost_1[4]<total_cost_1[accept_rule_1])          // AL - Have changed
    this from "if(total_cost[3]<total_cost[0])"
{
do_extend_fwd_1=1;
no_nonreduce_1=0;
emps_to_update_1.clear();
emps_to_update_1.reserve(emps_changed_1.size());
copy(emps_changed_1.begin(),emps_changed_1.end(),back_inserter(emps_to_update_1))
    ;
} }

void evaluate_block_new11(float& swapping_cost_1, float total_cost_1[8], int &
    do_swap_1, int &no_nonreduce_1, vector<int>&emps_to_update_1, int&
    accept_rule_1, vector<int>& emps_changed_1 )
{
swapping_cost_1=total_cost_1[4]-total_cost_1[accept_rule_1];
if(total_cost_1[5]<total_cost_1[accept_rule_1])          // AL - Have changed
    this from "if(total_cost[3]<total_cost[0])"
{
do_swap_1=1;
no_nonreduce_1=0;
emps_to_update_1.clear();
emps_to_update_1.reserve(emps_changed_1.size());
copy(emps_changed_1.begin(),emps_changed_1.end(),back_inserter(emps_to_update_1))
    ;
} }

void evaluate_block_lasting(int no_swap_1, int do_swap_1,int ag_list_sol_1
    [8][13][25], int list_sol_1[8][13][48], int tabu_sol_1[13][48], int
    ag_tabu_sol_1[13][25], int all_roles_1, int reg_emp_1, int weeks_to_plan_1,
    int &do_extend_bkwd_1,int &do_extend_fwd_1, int &transfer_sol_from_1, int &
    transfer_sol_to_1, int &no_bkwd_1, int &no_fwd_1 )
{
int weeks_12, emp_12, rol_12;
if(do_extend_bkwd_1==1) {
transfer_sol_from_1=3;
no_bkwd_1=no_bkwd_1+1;
}
else if(do_extend_fwd_1==1) {
transfer_sol_from_1=4;
no_fwd_1=no_fwd_1+1;
}
else if(do_swap_1==1) {
transfer_sol_from_1=5;
no_swap_1=no_swap_1+1;
}
if(do_extend_bkwd_1==1 || do_extend_fwd_1== 1 || do_swap_1 ==1) {
for(weeks_12=0;weeks_12<weeks_to_plan_1;weeks_12++) {
for(emp_12=0;emp_12<reg_emp_1;emp_12++) {
tabu_sol_1[weeks_12][emp_12]=list_sol_1[0][weeks_12][emp_12];
}
for(rol_12=0;rol_12<all_roles_1;rol_12++) {
ag_tabu_sol_1[weeks_12][rol_12]= ag_list_sol_1[0][weeks_12][rol_12];
} }    // AL - Have added this bracket in order to end the 'for all weeks' loop
    here, rather than below
transfer_sol_to_1=0;
} }

void evaluate_block_new12(int &task_extend_1, int &rol_14x, int &block_start_1,
    int &weeks_14x, int &swap_block_earliest_1, int all_roles_1, int roles_1[25],
     int &vessel_extend_1, int &block_end_1, int &block_len_1 )
{
int rol_16;
task_extend_1=rol_14x;       // AL - Have changed this from "task_extend=rol_15"
block_start_1=weeks_14x;
swap_block_earliest_1=weeks_14x;
for(rol_16=0; rol_16<all_roles_1;rol_16++) {
if(roles_1[rol_16]==task_extend_1) {
vessel_extend_1=task_extend_1;
rol_16=all_roles_1;
} }
block_end_1=-1;
block_len_1=0;
}

void evaluate_block_new13(int &do_swap_1, int &no_swap_1,int &do_extend_bkwd_1,
    int &transfer_sol_from_1, int &no_bkwd_1, int &do_extend_fwd_1, int &no_fwd_1
    )
{
if(do_extend_bkwd_1==1) {
transfer_sol_from_1=3;
no_bkwd_1= no_bkwd_1 + 1;
}
```

```
else if(do_extend_fwd_1==1) {
transfer_sol_from_1=4;
no_fwd_1=no_fwd_1 + 1;
}
else if(do_swap_1==1) {
transfer_sol_from_1=5;
no_swap_1=no_swap_1 + 1;
} }

void find_usable_1(int &update_done_1, int &candidate_exist_1,vector<int>&
    candidate_emps_1)
{
update_done_1=0;
candidate_exist_1=0;
candidate_emps_1.clear();
}

void find_usable_2(int weeks_to_plan_1, int &vessel_extend_1, int &emp_13x, int &
    swap_block_earliest_1,int &update_done_1, int &emp_extend_1, int &
    task_extend_1, int &block_start_1, int &block_end_1, int &block_len_1, int &
    block_found_1, int vessels_1, int starting_1[25][49], int work_zero_1[48],int
     current_list_1[13], int list_sol_1[8][13][48])
{
int rol_14, weeks_14;
if(update_done_1==0) {
emp_extend_1=-1;
task_extend_1=-1;
block_start_1=-1;
block_end_1=-1;
block_len_1=0;
block_found_1=0;
for(rol_14=0;rol_14<vessels_1;rol_14++) {
if(starting_1[rol_14][emp_13x]==1) {
block_found_1=1;             // AL - Have changed this from "block_found=0"
emp_extend_1=emp_13x;
block_start_1=0-work_zero_1[emp_13x];
swap_block_earliest_1=0;
vessel_extend_1=rol_14;
rol_14=vessels_1;
}
if(block_found_1==0) {
vessel_extend_1=-1;
} } }
```

```
for(weeks_14=0;weeks_14<weeks_to_plan_1;weeks_14++) {
current_list_1[weeks_14]=list_sol_1[0][weeks_14][emp_13x];
} }

void find_usable_3(int roles_1[25],int vessels_1, int &weeks_14x, int &rol_14x,
    int current_list_1[13], int &new_block_1, int &block_found_1, int &
    emp_extend_1, int &emp_13x, int &task_extend_1, int &block_start_1, int &
    swap_block_earliest_1, int rest_zero_1[48], int &vessel_extend_1)
{
int rol_15;
new_block_1=1;
emp_extend_1=emp_13x;
task_extend_1=rol_14x;
block_start_1=weeks_14x;
if(weeks_14x==0)      // AL - Have changed this from "if(weeks_14==1)"
{
swap_block_earliest_1= 0;                  // AL - Have changed this from "
    swap_block_earliest= 1"
}
else if(weeks_14x <= rest_zero_1[emp_13x]) // AL - Have changed this from "... if
    (weeks_14-1 <=..."
{
swap_block_earliest_1=weeks_14x;
}
else {
swap_block_earliest_1=weeks_14x-1;
}
for(rol_15=0;rol_15<vessels_1;rol_15++) {
if(rol_14x==roles_1[rol_15]) {
vessel_extend_1=rol_15;
rol_15=vessels_1;
} } }

void find_usable_4(int &link_to_vessel_1, int &vessel_extend_1, int &rol_14x, int
     roles_1[25], int &task_extend_1)
{
link_to_vessel_1=0;
if(vessel_extend_1!=-1) {
if(rol_14x==roles_1[vessel_extend_1]) {
link_to_vessel_1=1;
} }
if(link_to_vessel_1==1) {
task_extend_1=rol_14x;
```

```
} }

void find_usable_5(int &block_end_1, int &weeks_14x, int &swap_block_latest_1,
    int &block_len_1, int &block_start_1)
{
block_end_1=weeks_14x-1;
swap_block_latest_1=weeks_14x;
block_len_1=(block_end_1 - block_start_1)+1;
}

void find_usables_6(int &block_found_1, int &block_start_1, int &block_end_1, int
     &block_len_1, int &emp_extend_1, int &task_extend_1, int &vessel_extend_1 )
{
block_found_1=0;
block_start_1=-1;
block_end_1=-1;
block_len_1=0;
emp_extend_1=-1;
task_extend_1=-1;
vessel_extend_1=-1;
}

void find_usables_7(int &new_block_1, int &block_found_1)
{
if(new_block_1==1) {
block_found_1=1;
new_block_1=0;
} }

void find_usables_8(int &block_end_1, int &weeks_14x, int &swap_block_latest_1,
    int &block_len_1, int &block_start_1)
{
block_end_1=weeks_14x;
swap_block_latest_1=weeks_14x;
block_len_1=(block_end_1 - block_start_1) +1;
}

void find_usables_9(int &update_done_1, int &emp_13x, int reg_emp_1, int
    swaps_examined_1[48][48])
{
int emp_14;
if(update_done_1==0) {
for(emp_14=0; emp_14<reg_emp_1;emp_14++) {
```

```
if(emp_14!=emp_13x) {
swaps_examined_1[emp_13x][emp_14]=1;
} } } }

void find_usables_10(int tabu_sol_1[13][48], int ag_tabu_sol_1[13][25], int &
    no_nonreduce_1, float& candidate_cost_1,int& transfer_sol_from_1,int&
    transfer_sol_to_1, int &candidate_exist_1, int &update_done_1, int
    weeks_to_plan_1, int reg_emp_1, int all_roles_1, int list_sol_1[8][13][48],
    int ag_list_sol_1[8][13][25])
{
int weeks_14, rol_14, emp_13;
if(update_done_1==0 && candidate_exist_1==1) {
for(emp_13=0;emp_13<reg_emp_1;emp_13++)
for(weeks_14=0;weeks_14<weeks_to_plan_1;weeks_14++) {
tabu_sol_1[weeks_14][emp_13]=list_sol_1[0][weeks_14][emp_13];
}
for(rol_14=0;rol_14<all_roles_1;rol_14++)
for(weeks_14=0;weeks_14<weeks_to_plan_1;weeks_14++) {
ag_tabu_sol_1[weeks_14][rol_14]=ag_list_sol_1[0][weeks_14][rol_14];
}
if(candidate_cost_1 < 0) {
no_nonreduce_1=0;
}
else {
no_nonreduce_1=no_nonreduce_1+1;
}
transfer_sol_from_1=6;
transfer_sol_to_1=0;
} }
```

E.2.5.16 File ‘comparing.h’

```
void compare_to_best(int& number_best_2, int weeks_to_plan_2, int all_roles_2,
    int reg_emp_2, int role_count_2 , float total_cost_2[8], int new_best_2,int
    same_best_2[51],int emp_count_2,int list_sol_2[8][13][48], int best_sols_2
    [51][13][48], int ag_list_sol_2[8][13][25], int ag_best_sols_2[51][13][25]);
```

E.2.5.17 File ‘cost.h’

```
#include <vector>
using namespace std;
```

```
void calc_cost1(int all_roles_2,int reg_emp_2, int weeks_to_plan_2, int
    list_sol_2[8][13][48], int ag_list_sol_2[8][13][25], int& to_calculate_2,
    float total_cost_2[8]);
void calc_cost2(float total_cost_2[8],int starting_2[25][49],int reg_emp_2,int
    weeks_to_plan_2, int all_roles_2, vector<int> &emps_changed_2, float
    emp_cost_2[8][48], int allocate_sol_2[8][13][25][49], int long_work_sol_2
    [8][13][25][49][10], int lambda_2,int board_sol_2[8][13][25][48], int
    depart_sol_2[8][13][25][48],int undertime_sol_2[8][45], int overtime_sol_2
    [8][45], int& consec_work_2, int work_zero_2[48],int list_sol_2[8][13][48],
    int g_weeks_2[45], int exp_worktime_2[45]);
void calc_cost3(int reg_emp_2,int weeks_to_plan_2, int all_roles_2,vector<int> &
    emps_changed_2, float emp_cost_2[8][48], int allocate_sol_2[8][13][25][49],
    int long_work_sol_2[8][13][25][49][10], int lambda_2,int board_sol_2
    [8][13][25][48], int depart_sol_2[8][13][25][48],int undertime_sol_2[8][45],
    int overtime_sol_2[8][45], int chng_undertime_2[45], int chng_overtime_2[45],
     int cur_undertime_2[45], int cur_overtime_2[45],int under_rate_2[45], int
    over_rate_2[45], int cur_allocate_2[13][25][49], int chng_allocate_2
    [13][25][49], int cur_board_2[13][25][48], int chng_board_2[13][25][48],
    float board_chng_cost_2[13][25][48], int cur_depart_2[13][25][48], int
    chng_depart_2[13][25][48], float depart_chng_cost_2[13][25][48],int
    cur_long_work_2[13][25][49][10] ,int chng_long_work_2[13][25][49][10], float
    extension_chng_cost_2[13][25][49][10], float work_chng_cost_2[13][25][49]);
void calc_cost31(int reg_emp_2,int weeks_to_plan_2, int all_roles_2,vector<int> &
    emps_changed_2, float emp_cost_2[8][48], int allocate_sol_2[8][13][25][49],
    int long_work_sol_2[8][13][25][49][10], int lambda_2,int board_sol_2
    [8][13][25][48], int depart_sol_2[8][13][25][48],int undertime_sol_2[8][45],
    int overtime_sol_2[8][45], int chng_undertime_2[45], int chng_overtime_2[45],
     int cur_undertime_2[45], int cur_overtime_2[45],int under_rate_2[45], int
    over_rate_2[45], int cur_allocate_2[13][25][49], int chng_allocate_2
    [13][25][49], int cur_board_2[13][25][48], int chng_board_2[13][25][48],
    float board_chng_cost_2[13][25][48], int cur_depart_2[13][25][48], int
    chng_depart_2[13][25][48], float depart_chng_cost_2[13][25][48],int
    cur_long_work_2[13][25][49][10] ,int chng_long_work_2[13][25][49][10], float
    extension_chng_cost_2[13][25][49][10], float work_chng_cost_2[13][25][49]);
void calc_cost4(float work_chng_cost_2[13][25][49],int ag_max_work_2[25], int
    ag_work_zero_2[25], int &consec_work_2,int &feas_crewchange_2,int &
    to_calculate_2,int &iteration_2, int weeks_to_plan_2, int all_roles_2, vector
    <int> & evaluate_crewchange_2,vector<int> &ag_roles_changed_2, vector<int> &
    definite_crewchange_2, vector<int> &possible_crewchange_2,float ag_cost_2
    [8][13][25], int ag_crewchange_2[8][13][25],int allocate_sol_2
    [8][13][25][49],int ag_rboard_sol_2[8][13][25], int ag_rdepart_sol_2
    [8][13][25], int long_work_sol_2[8][13][25][49][10], int lambda_2, int
    ag_starting_2[25], int ag_list_sol_2[8][13][25], int cur_ag_rboard_2[13][25],
    int poss_chng_ag_rboard_2[13], int poss_ag_rboard_2[13], int cur_ag_rdepart_2
    [13][25], int poss_chng_ag_rdepart_2[13], int poss_ag_rdepart_2[13],float
    ag_board_chng_cost_2[13][25],float ag_depart_chng_cost_2[13][25],int
    poss_ag_long_work_2[13][10], int cur_long_work_2[13][25][49][10], int
    poss_chng_ag_long_work_2[13][10], int cur_allocate_2[13][25][49], int
    chng_allocate_2[13][25][49], float extension_chng_cost_2[13][25][49][10], int
     chng_ag_rboard_2[13][25], int chng_ag_rdepart_2[13][25],int chng_long_work_2
    [13][25][49][10]);
void calc_cost5(int reg_emp_2,int weeks_to_plan_2, int all_roles_2,float
    emp_cost_2[8][48],float total_cost_2[8],float ag_cost_2[8][13][25],int &
    transfer_sol_from_2, int &transfer_sol_to_2, int &to_calculate_2);
```

E.2.5.18 File ‘feasibility_checking.h’

```
#include <vector>
using namespace std;
void JC_feas(int &feasible_2, int &JCfeas_2,int all_emp_2,int all_roles_2, int
    weeks_to_plan_2, int eligible_2[13][25][49], int allocate_sol_2
    [8][13][25][49], int required_2[13][25]);
void Overlap_feas(vector<int> &emps_changed_2,int &feasible_2, int &OLfeas_2,int
    all_emp_2,int all_roles_2, int weeks_to_plan_2, int allocate_sol_2
    [8][13][25][49]);
void BoardFeas(vector<int> &emps_changed_2,int &feasible_2, int &Brdfeas_2,int
    all_emp_2,int all_roles_2, int weeks_to_plan_2, int allocate_sol_2
    [8][13][25][49], int board_sol_2[8][13][25][48],int starting_2[25][49]);
void DepartFeas(vector<int> &emps_changed_2,int &feasible_2, int &Dprtfeas_2,int
    all_emp_2,int all_roles_2, int weeks_to_plan_2, int allocate_sol_2
    [8][13][25][49], int depart_sol_2[8][13][25][48],int starting_2[25][49]);
void AgBoardDepartFeas(vector<int> &ag_roles_changed_2,int &feasible_2, int &
    AGBDfeas_2,int all_roles_2, int weeks_to_plan_2, int allocate_sol_2
    [8][13][25][49], int ag_rboard_sol_2[8][13][25],int ag_rdepart_sol_2
    [8][13][25],int ag_starting_2[25]);
void UnderOverFeas(vector<int> &emps_changed_2,int &feasible_2, int &UTfeas_2,
    int &OTfeas_2,int all_emp_2,int all_roles_2, int weeks_to_plan_2,int
    undertime_sol_2[8][45],int overtime_sol_2[8][45],int g_weeks_2[45], int
    exp_worktime_2[45], int allocate_sol_2[8][13][25][49]);
void LongWorkFeas(vector<int> &ag_roles_changed_2,vector<int> &emps_changed_2,int
     &AGLWfeas_2,int &LWfeas_2,int lambda_2, int &feasible_2, int work_total_2
    [13][48], int work_zero_2[48], int reg_emp_2,int all_roles_2, int
    weeks_to_plan_2,int allocate_sol_2[8][13][25][49], int max_work_2[48],int
    long_work_sol_2[8][13][25][49][10],int ag_work_total_2[13][25], int
    ag_work_zero[25], int ag_max_work[25], int ag_rdepart_sol[8][13][25]);
```

```
void RestFeas(vector<int> &emps_changed_2,int &feasible_2,int &RvWfeas_2,int
    reg_emp_2,int all_roles_2, int weeks_to_plan_2,int depart_sol_2
    [8][13][25][48],int rest_total_2[13][49], int rest_zero_2[48],int
    allocate_sol_2[8][13][25][49],int min_rest_2[48]);
void LinkFeasiblity(int lambda_2, vector<int> &ag_roles_changed_2,vector<int> &
    emps_changed_2,int &feasible_2,int &Linkfeas_2,int reg_emp_2,int all_roles_2,
     int weeks_to_plan_2,int cur_allocate_2[13][25][49],int chng_allocate_2
    [13][25][49], int allocate_sol_2[8][13][25][49],int cur_board_2[13][25][48],
    int chng_board_2[13][25][48], int board_sol_2[8][13][25][48],int cur_depart_2
    [13][25][48], int chng_depart_2[13][25][48], int depart_sol_2[8][13][25][48],
    int cur_ag_rboard_2[13][25], int ag_rboard_sol_2[8][13][25], int
    chng_ag_rboard_2[13][25], int cur_ag_rdepart_2[13][25], int ag_rdepart_sol_2
    [8][13][25], int chng_ag_rdepart_2[13][25], int cur_long_work_2
    [13][25][49][10], int chng_long_work_2[13][25][49][10], int long_work_sol_2
    [8][13][25][49][10], int chng_undertime_2[45], int undertime_sol_2[8][45],
    int cur_undertime_2[45], int chng_overtime_2[45], int overtime_sol_2[8][45],
    int cur_overtime_2[45]);
void StatusFeasibility(int lambda_2,vector<int> &ag_roles_changed_2,vector<int> &
    emps_changed_2,int &feasible_2,int &Statfeas_2,int reg_emp_2,int all_roles_2,
     int weeks_to_plan_2, int allocate_sol_2[8][13][25][49],int chng_allocate_2
    [13][25][49], int chng_long_work_2[13][25][49][10], int long_work_sol_2
    [8][13][25][49][10], int chng_board_2[13][25][48], int board_sol_2
    [8][13][25][48], int chng_depart_2[13][25][48], int depart_sol_2
    [8][13][25][48], int ag_rboard_sol_2[8][13][25], int chng_ag_rboard_2
    [13][25], int ag_rdepart_sol_2[8][13][25], int chng_ag_rdepart_2[13][25], int
     undertime_sol_2[8][45], int overtime_sol_2[8][45]);
void infeasibility(int &Statfeas_2, int &no_Statfeas_2,int &Linkfeas_2,int &
    no_Linkfeas_2,int &no_RvWfeas_2 ,int &RvWfeas_2,int &AGLWfeas_2,int &LWfeas_2
    , int &no_AGLWfeas_2,int &no_LWfeas_2, int &UTfeas_2, int &OTfeas_2, int &
    no_UTfeas_2, int &no_OTfeas_2,int &JCfeas_2, int &no_JCfeas_2, int &OLfeas_2,
     int &no_OLfeas_2, int &Brdfeas_2, int &Dprtfeas_2, int &AGBDfeas_2, int &
    no_Brdfeas_2, int &no_Dprtfeas_2, int &no_AGBDfeas_2);
```

E.2.5.19 File 'find_usables.h'

```
#include <vector>
using namespace std;
void find_usable_1(int &update_done_2, int &candidate_exist_2,vector<int>&
    candidate_emps_2);
void find_usable_2(int weeks_to_plan_2, int &vessel_extend_2, int &emp_13y, int &
    swap_block_earliest_2,int &update_done_2, int &emp_extend_2, int &
    task_extend_2, int &block_start_2, int &block_end_2, int &block_len_2, int &
    block_found_2, int vessels_2, int starting_2[25][49],int work_zero_2[48],int
    current_list_2[13], int list_sol_2[8][13][48]);
void find_usable_3(int roles_2[25],int vessels_2, int &weeks_14y, int &rol_14y,
    int current_list_2[13], int &new_block_2, int &block_found_2, int &
    emp_extend_2, int &emp_13y, int &task_extend_2, int &block_start_2, int &
    swap_block_earliest_2, int rest_zero_2[48], int &vessel_extend_2);
void find_usable_4(int &link_to_vessel_2, int &vessel_extend_2, int &rol_14y, int
     roles_2[25], int &task_extend_2);
void find_usable_5(int &block_end_2, int &weeks_14y, int &swap_block_latest_2,
    int &block_len_2, int &block_start_2);
void evaluate_block_new1(int required_2[13][25],int rest_zero_2[48], int&
    block_end_2, int &task_extend_2,int& emp_extend_2,int eligible_2[13][25][49],
    int weeks_to_plan_2,float& extend_cost_fwd_2,float& extend_cost_bkwd_2, float
    & swapping_cost_2, int& do_extend_bkwd_2, int& do_extend_fwd_2, int&
    do_swap_2, int& block_len_2, int max_work_2[48],int& max_bkwd_extend_2, int&
    max_fwd_extend_2);
void evaluate_block_new2(int all_roles_2, int reg_emp_2, int weeks_to_plan_2, int
     max_work_2[48], int& conflict_found_bkwd_2, int reserve_list_bkwd_2[13][25],
     int& in_reserve_bkwd_2, int& block_end_2, int& task_extend_2, int
    work_zero_2[48], int& length_count_2,int &emp_extend_2,int& rest_count_2, int
    & extend_len_bkwd_2, vector<int>& emps_changed_2,vector<int>&
    ag_roles_changed_2, int list_sol_2[8][13][48], int ag_list_sol_2[8][13][25],
    int min_rest_2[25]);
void evaluate_block_new3(int &to_check_tabu_2,int all_roles_2, int reg_emp_2, int
     weeks_to_plan_2, int max_work_2[48], int& prev_work_2, int& add_to_emp_2,
    int& conflict_found_bkwd_2, int reserve_list_bkwd_2[13][25], int&
    in_reserve_bkwd_2, int& block_end_2, int& task_extend_2, int work_zero_2[48],
     int& length_count_2,int &emp_extend_2,int& rest_count_2, int&
    extend_len_bkwd_2, vector<int>& emps_changed_2,vector<int>&
    ag_roles_changed_2, int list_sol_2[8][13][48], int ag_list_sol_2[8][13][25],
    int eligible_2[13][25][49], int rest_zero_2[25], int min_rest_2[25]);
void evaluate_block_new4(float& extend_cost_bkwd_2, float total_cost_2[8], int &
    do_extend_bkwd_2, int &no_nonreduce_2, vector<int>&emps_to_update_2, int&
    accept_rule_2, vector<int>& emps_changed_2 );
void update_swaps_and_changes(vector<int>& emps_to_update_2,int swaps_examined_2
    [48][48], int reg_emp_2, int& last_kick_time_2, int& iteration_2, int
    last_changed_2[48]);
void evaluate_block_new6(int& extend_len_fwd_2,int& max_fwd_extend_2, int&
    block_start_2, int rest_zero_2[48], int& emp_extend_2, int summation_2, int
    vessels_2, int roles_2[25], int& task_extend_2, int starting_2[25][49],int
    min_rest_2[48], int eligible_2[13][25][49], int required_2[13][25]);
void evaluate_block_new7(int &extend_len_fwd_2,vector<int>& emps_changed_2,
    vector<int>& ag_roles_changed_2, int reg_emp_2, int weeks_to_plan_2, int
    list_sol_2[8][13][48], int all_roles_2, int ag_list_sol_2[8][13][25], int
    reserve_list_fwd_2[13][25], int &emp_extend_2, int &rest_count_2,int &
    in_reserve_fwd_2,int &task_extend_2, int min_rest_2[48], int &
    conflict_found_fwd_2, int &block_start_2);
void evaluate_block_new8(int rest_zero_2[48],int starting_2[25][49],int
    eligible_2[13][25][49],int max_work_2[48],int &length_count_2,int &
    to_check_tabu_2, int &summation_1y, int &summation_y, int &prev_work_2,int &
    add_to_emp_2, int &extend_len_fwd_2,vector<int>& emps_changed_2, vector<int>&
     ag_roles_changed_2, int reg_emp_2, int weeks_to_plan_2, int list_sol_2
    [8][13][48], int all_roles_2, int ag_list_sol_2[8][13][25], int
    reserve_list_fwd_2[13][25], int &emp_extend_2, int &rest_count_2,int &
    in_reserve_fwd_2,int &task_extend_2, int min_rest_2[48], int &
    conflict_found_fwd_2, int &block_start_2);
void evaluate_block_new9(float& extend_cost_fwd_2, float total_cost_2[8], int &
    do_extend_fwd_2, int &no_nonreduce_2, vector<int>&emps_to_update_2, int&
    accept_rule_2, vector<int>& emps_changed_2 );
void evaluate_block_new12(int &task_extend_2, int &rol_14y, int &block_start_2,
    int &weeks_14y, int &swap_block_earliest_2, int all_roles_2, int roles_2[25],
     int &vessel_extend_2, int &block_end_2, int &block_len_2);
void find_usables_6(int &block_found_2, int &block_start_2, int &block_end_2, int
     &block_len_2, int &emp_extend_2, int &task_extend_2, int &vessel_extend_2);
void find_usables_7(int &new_block_2, int &block_found_2);
void find_usables_8(int &block_end_2, int &weeks_14y, int &swap_block_latest_2,
    int &block_len_2, int &block_start_2);
void find_usables_9(int &update_done_2, int &emp_13y, int reg_emp_2, int
    swaps_examined_2[48][48]);
void find_usables_10(int tabu_sol_2[13][48], int ag_tabu_sol_2[13][25], int &
    no_nonreduce_2, float& candidate_cost_2,int& transfer_sol_from_2,int&
    transfer_sol_to_2, int &candidate_exist_2, int &update_done_2, int
    weeks_to_plan_2, int reg_emp_2, int all_roles_2, int list_sol_2[8][13][48],
    int ag_list_sol_2[8][13][25]);
void evaluate_block_new13(int &do_swap_2, int &no_swap_2,int &do_extend_bkwd_2,
    int &transfer_sol_from_2, int &no_bkwd_2, int &do_extend_fwd_2, int &no_fwd_2
    );
```

E.2.5.20 File ‘finish.h’

```
void finishing(int &update_done_2, int &terminate_2, int &iteration_2, double &
    time_2);
```

E.2.5.21 File ‘initialise.h’

```
#include <vector>
using namespace std;
```

```
void initialise_first(int overall_regular_max_work_2, int max_work_2[48], int
    overall_agency_max_work_2, int ag_max_work_2[25], int overall_max_work_2,int&
     lambda_2, int reg_emp_2, int all_roles_2);
void initialise_second(int weeks_to_plan_2, int list_sol_2[8][13][48], int
    best_index_2, int best_sols_2[51][13][48], int& added_t_2, int
    swaps_examined_2[48][48],vector<int> &emps_changed_2, int reg_emp_2, int
    all_roles_2, int taskbased_sol_2[13][25][49]);
void initialise_third(int weeks_to_plan_2, int ag_list_sol_2[8][13][25], int
    best_index_2, int ag_best_sols_2[51][13][25], vector<int> &ag_roles_changed_2
    , int reg_emp_2, int all_roles_2,int taskbased_sol_2[13][25][49], int
    required_2[13][25]);
void initialise_four(int weeks_to_plan_2,int ag_best_sols_2[51][13][25], int
    reg_emp_2,int all_roles_2,int best_sols_2[51][13][48], int list_sol_2
    [8][13][48], int ag_list_sol_2[8][13][25],int last_changed_2[48]);
```

E.2.5.22 File 'load_data.h'

```
// Load_Data.h
void LoadRequired(int arr_1[13][25]);
void LoadEligible(int arr_1[325][49], int arr_2[13][25][49]);
void LoadStarting(int arr_1[25][49]);
void LoadAlloc(int arr_1[13][25][49], int arr_2[325][49]);
void LoadBoarding(int arr_1[325][48], int arr_2[13][25][48]);
void LoadDeparting(int arr_1[325][48], int arr_2[13][25][48]);
void LoadAgBoard(int arr_1[13][25]);
void LoadAgDepart(int arr_1[13][25]);
void LoadChangeCostBoard(float arr_1[325][48], float arr_2[13][25][48]);
void LoadChangeCostDepart(float arr_1[325][48], float arr_2[13][25][48]);
void LoadAgChangeBoard(float arr_1[13][25]);
void LoadAgChangeDepart(float arr_1[13][25]);
void LoadWorkChangeCost(float arr_1[325][49], float arr_2[13][25][49]);
void LoadInitialSolution(int arr_1[325][49],int arr_2[13][25][49]);
void LoadContract(int arr_1[45][4], int arr_2[45], int arr_3[45], int arr_4[45],
    int arr_5[45]);
void LoadConst(int arr_1[48][4], int arr_2[48], int arr_3[48], int arr_4[48], int
     arr_5[48]);
void LoadAgencyCrew(int arr_1[25][3], int arr_2[25], int arr_3[25], int arr_4
    [25]);
void LoadExtChgCost(float ext_chg_cost_2[325][490],float ext_chg_cost_new_2
    [13][25][490],float extension_chng_cost_2[13][25][49][10]);
void LoadOver_under(int guaranteed_workers_2[45][2], int cur_undertime_2[45], int
     cur_overtime_2[45]);
```

```
void LoadCurLongWork(int cur_long_work_2[325][490],int cur_long_work_new_2
    [13][25][490], int current_long_work_2[13][25][49][10]);
```

E.2.5.23 File 'rand_kick.h'

```
#include <vector>
#include <random>
using namespace std;
void kicking(vector<int>& emps_changed_2, vector<int>& ag_roles_changed_2, int
    eligible_2[13][25][49],int work_zero_2[48], int min_rest_2[48], int
    starting_2[25][49], int all_roles_2, int sum_start_rand_2, int rest_zero_2
    [48], int &kick_feas2, int &kick_end_2, int &kick_start_2, float &random_2,
    int &random_emp_2, int reg_emp_2, int &kick_emp_2, int &random_task_2, int
    all_roles2, int &kick_task_2, int max_work2[48], int &random_length_2, int &
    random_time2, int weeks_to_plan_2);
void kicking2(int min_rest_2[48],int starting_2[25][49] ,int &rest_count_2, int &
    kick_emp2,int reg_emp2,int all_roles2,int weeks_to_plan2, int ag_list_sol_2
    [8][13][25], int &kick_start_2, int &kick_end_2, int &kick_task_2, vector<int
    >& emps_changed_2, vector<int>& ag_roles_changed_2, int list_sol_2
    [8][13][48]);
void kicking3(int all_roles_2, int reg_emp_2,int weeks_to_plan_2, int tabu_sol_2
    [13][48], int list_sol_2[8][13][48], int ag_tabu_sol_2[13][25], int
    ag_list_sol_2[8][13][25] );
```

E.2.5.24 File 'sort_listing.h'

```
#include <vector>
using namespace std;
void sort_employee_list(float short_list_fraction_2, int& change_no_2, int
    last_changed_2[48],int min_emp_2,int min_no_2, int& order_rule_2, int
    order_number_2[48], int reg_emp_2, vector<int>& ordered_list_2, vector<int>&
    short_ordered_list_2,vector<int>& added_set_2, int& iteration_2);
```

E.2.5.25 File 'swap_change_update.h'

```
#include <vector>
#include <iterator>
using namespace std;
void update_swaps_and_changes(vector<int>& emps_to_update_2,int swaps_examined_2
    [48][48], int reg_emp_2, int& last_kick_time_2, int& iteration_2, int
    last_changed_2[48]);
```

```
void swap_calc3(vector<int>& emps_changed_2,int& no_nonreduce_2,float&
    swapping_cost_2, float total_cost_2[8], int& accept_rule_2, int& do_swap_2,
    vector<int>& emps_to_update_2,int swaps_examined_2[48][48], int reg_emp_2,
    int& last_kick_time_2, int& iteration_2, int last_changed_2[48]);
```

E.2.5.26 File 'swapping.h'

```
#include <vector>
using namespace std;
void swap_calc1( vector<int>& ag_roles_changed_2, vector<int>& emps_changed_2,int
     reg_emp_2,int weeks_to_plan_2, int list_sol_2[8][13][48],int all_roles_2,int
     ag_list_sol_2[8][13][25], int& emp_extend_2, int& block_start_2,int&
    swap_block_start_2, int min_rest_2[48], int& swap_block_end_2, int&
    swap_task_2, int& block_end_2, int& swap_emp_2,int& task_extend_2, int&
    to_check_tabu_2);
void evaluate_swap1(int& ag_swappable_2, int& block_start_2, int& block_end_2,
    int eligible_2[13][25][49], int& task_extend_2);
void evaluate_swap2(int& swap_allowed_2,int& too_early_2, int& do_swap_2, int&
    swap_emp_2, int& swap_task_2,int& swap_block_start_2,int& swap_block_end_2,
    int& swap_block_len_2,int& swap_find_time_2, int& swap_block_found_2);
void evaluate_swap6(int& swap_allowed_2,int& too_early_2, int& do_swap_2, int&
    swap_emp_2, int& swap_task_2,int& swap_block_start_2,int& swap_block_end_2,
    int& swap_block_len_2,int& swap_block_found_2 );
void evaluate_swap3(int& weeks_20x, int& rol_20x,int& swap_block_start_2,int&
    swap_task_2,int& swap_emp_2,int& swap_allowed_2,int eligible_2[13][25][49] ,
    int min_rest_2[48],int starting_2[25][49], int& emp_extend_2, int all_roles_2
    ,int summation_2, int summation_2x, int& too_early_2,int& swap_new_block_2,
    int& swap_block_found_2, int ag_list_sol_2[8][13][25], int&
    swap_block_latest_2, int& swap_block_earliest_2);
void evaluate_swap4(int& weeks_20x, int& swap_block_found_2,int& swap_block_len_2
    , int& swap_block_end_2,int& swap_allowed_2, int eligible_2[13][25][49],int
    min_rest_2[48], int& too_early_2,int& emp_extend_2, int&
    swap_block_earliest_2,int starting_2[25][49],int summation_2, int
    summation_2x, int all_roles_2, int& swap_block_latest_2, int& do_swap_2, int&
     swap_task_2, int& swap_block_start_2, int& rol_20x );
void evaluate_swap5(int& swap_allowed_2,int& emp_extend_2,int& rol_20x,int&
    weeks_20x, int& swap_block_latest_2, int& swap_block_found_2, int eligible_2
    [13][25][49]);
void evaluate_swap7(int& swap_new_block_2, int& swap_block_found_2);
void evaluate_swap8(int& weeks_20x, int& rol_20x,int& swap_block_end_2, int&
    swap_block_start_2,int& swap_block_len_2,int& swap_allowed_2, int eligible_2
    [13][25][49], int& emp_extend_2);
```

```
void evaluate_swap9(int& swap_emp_2, int& swap_task_2, int& swap_block_start_2,
    int& block_start_2, int& swap_block_end_2, int& block_end_2 );
void evaluate_swap10(int min_rest_2[48], int work_zero_2[48],int starting_2
    [25][49],int all_roles_2,int summation_2, int summation_2x,int eligible_2
    [13][25][49],int& task_extend_2,int& block_end_2,int rest_zero_2[48],int&
    block_start_2,int max_work_2[48],int& block_len_2, int reg_emp_2, int
    swappable_emp_2[48], int& emp_extend_2 );
void evaluate_swap11(int& swap_find_time_2, int& all_rest_2,int& swap_allowed_2,
    int& too_early_2,int& swap_block_found_2, int& swap_block_len_2, int&
    do_swap_2, int& swap_emp_2, int& swap_vessel_2, int& swap_task_2, int&
    swap_block_start_2, int& swap_block_end_2 );
void evaluate_swap12(int& swap_vessel_2,int roles_2[25],int& swap_block_start_2,
    int& swap_emp_2, int& swap_task_2,int& swap_allowed_2, int eligible_2
    [13][25][49],int min_rest_2[48],int& emp_extend_2, int starting_2[25][49],int
     all_roles_2, int summation_2, int summation_2x, int& too_early_2,int&
    swap_block_found_2,int& swap_new_block_2, int& all_rest_2,int& rol_20x, int&
    emp_20x, int list_sol_2[8][13][48],int& swap_block_earliest_2, int&
    swap_find_time_2, int& swap_block_latest_2 );
void evaluate_swap13(int& vessel_extend_2, int roles_2[25],int& swap_task_2, int&
     swap_allowed_2,int eligible_2[13][25][49] ,int& rol_20x, int& emp_extend_2,
    int& swap_vessel_2, int& link_to_vessel_2, int& swap_find_time_2, int&
    swap_block_latest_2, int& swap_block_found_2);
void evaluate_swap14(int& swap_block_found_2,int& swap_block_len_2,int&
    swap_block_end_2, int& swap_vessel_2,int& swap_allowed_2,int eligible_2
    [13][25][49] ,int min_rest_2[48],int starting_2[25][49], int& emp_extend_2,
    int roles_2[25],int all_roles_2,int summation_2, int summation_2x,int&
    too_early_2,int& swap_find_time_2, int& swap_block_latest_2, int& do_swap_2,
    int& swap_task_extend_2, int& rol_20x, int& swap_block_start_2, int&
    swap_block_earliest_2);
void evaluate_swap15(int& rol_20x, int& swap_find_time_2, int&
    swap_block_latest_2, int& swap_block_found_2, int& swap_allowed_2, int
    eligible_2[13][25][49], int& emp_extend_2);
void evaluate_swap16(int& swap_emp_2, int& swap_task_2, int& swap_vessel_2, int&
    swap_block_found_2, int& do_swap_2, int& too_early_2, int& swap_allowed_2,
    int& swap_block_start_2, int& swap_block_end_2, int& swap_block_len_2 );
void evaluate_swap17(int& swap_new_block_2, int& swap_block_found_2);
void evaluate_swap18(int& swap_block_start_2, int& swap_block_len_2,int&
    swap_block_end_2, int& swap_find_time_2, int& swap_allowed_2, int eligible_2
    [13][25][49], int& rol_20x, int& emp_extend_2);
void evaluate_swap19(int& swap_emp_2, int& emp_20x, int& swap_task_2, int&
    swap_block_start_2, int& block_start_2, int& swap_block_end_2, int&
    block_end_2 );
```

E.2.5.27 File ‘tabu.h’

```
void check_tabu(int& tabu_2,int weeks_to_plan_2, int reg_emp_2, int all_roles_2,
    int& to_check_tabu_2, int list_sol_2[8][13][48], int ag_list_sol_2
    [8][13][25],int tabu_sol_2[13][48], int ag_tabu_sol_2[13][25]);
void check_tabu_2(int &transfer_sol_to_2,int reg_emp_2, int all_roles_2, int
    weeks_to_plan_2, int list_sol_2[8][13][48], int ag_list_sol_2[8][13][25],int
    tabu_sol_2[13][48], int ag_tabu_sol_2[13][25]);
```

E.2.5.28 File ‘transferring_data.h’

```
//transfer
#include <vector>
using namespace std;
void transfer_solution(int& transfer_sol_from_2, int& transfer_sol_to_2, float
    total_cost_2[8], int all_emp_2, int reg_emp_2, int all_roles_2, int
    weeks_to_plan_2, int lambda_2, int allocate_sol_2[8][13][25][49], int
    long_work_sol_2[8][13][25][49][10], float emp_cost_2[8][48], int list_sol_2
    [8][13][48], int board_sol_2[8][13][25][48], int depart_sol_2[8][13][25][48],
     int undertime_sol_2[8][45], int overtime_sol_2[8][45],float ag_cost_2
    [8][13][25],int ag_list_sol_2[8][13][25],int ag_crewchange_2[8][13][25], int
    ag_rboard_sol_2[8][13][25], int ag_rdepart_sol_2[8][13][25]);
void evaluate_block_new5(vector<int>& emps_changed_2, vector<int>&
    candidate_emps_2, float& extend_cost_bkwd_2, float &candidate_cost_2, int &
    candidate_exist_2, float total_cost_2[8],int &transfer_sol_from_2, int &
    transfer_sol_to_2, int all_emp_2, int reg_emp_2, int all_roles_2, int
    weeks_to_plan_2, int lambda_2, int allocate_sol_2[8][13][25][49], int
    long_work_sol_2[8][13][25][49][10], float emp_cost_2[8][48], int list_sol_2
    [8][13][48], int board_sol_2[8][13][25][48], int depart_sol_2[8][13][25][48],
     int undertime_sol_2[8][45], int overtime_sol_2[8][45],float ag_cost_2
    [8][13][25],int ag_list_sol_2[8][13][25],int ag_crewchange_2[8][13][25], int
    ag_rboard_sol_2[8][13][25], int ag_rdepart_sol_2[8][13][25]);
void evaluate_block_new10(vector<int>& emps_changed_2, vector<int>&
    candidate_emps_2, float& extend_cost_fwd_2, float &candidate_cost_2, int &
    candidate_exist_2, float total_cost_2[8],int &transfer_sol_from_2, int &
    transfer_sol_to_2, int all_emp_2, int reg_emp_2, int all_roles_2, int
    weeks_to_plan_2, int lambda_2, int allocate_sol_2[8][13][25][49], int
    long_work_sol_2[8][13][25][49][10], float emp_cost_2[8][48], int list_sol_2
    [8][13][48], int board_sol_2[8][13][25][48], int depart_sol_2[8][13][25][48],
     int undertime_sol_2[8][45], int overtime_sol_2[8][45],float ag_cost_2
    [8][13][25],int ag_list_sol_2[8][13][25],int ag_crewchange_2[8][13][25], int
    ag_rboard_sol_2[8][13][25], int ag_rdepart_sol_2[8][13][25]);
```

```
void compare_to_best_2(int &best_sol_time_2, int &iteration_2, int ag_best_sols_2
    [51][13][25],int best_sols_2[51][13][48], int &number_best_2,int&
    transfer_sol_from_2, int& transfer_sol_to_2, float total_cost_2[8], int
    all_emp_2, int reg_emp_2, int all_roles_2, int weeks_to_plan_2, int lambda_2,
     int allocate_sol_2[8][13][25][49], int long_work_sol_2[8][13][25][49][10],
    float emp_cost_2[8][48], int list_sol_2[8][13][48], int board_sol_2
    [8][13][25][48], int depart_sol_2[8][13][25][48], int undertime_sol_2[8][45],
     int overtime_sol_2[8][45],float ag_cost_2[8][13][25],int ag_list_sol_2
    [8][13][25],int ag_crewchange_2[8][13][25], int ag_rboard_sol_2[8][13][25],
    int ag_rdepart_sol_2[8][13][25]);
void evaluate_block_new14(vector<int>& emps_changed_2, vector<int>&
    candidate_emps_2, float& extend_cost_fwd_2, float &candidate_cost_2, int &
    candidate_exist_2, float total_cost_2[8],int &transfer_sol_from_2, int &
    transfer_sol_to_2, int all_emp_2, int reg_emp_2, int all_roles_2, int
    weeks_to_plan_2, int lambda_2, int allocate_sol_2[8][13][25][49], int
    long_work_sol_2[8][13][25][49][10], float emp_cost_2[8][48], int list_sol_2
    [8][13][48], int board_sol_2[8][13][25][48], int depart_sol_2[8][13][25][48],
     int undertime_sol_2[8][45], int overtime_sol_2[8][45],float ag_cost_2
    [8][13][25],int ag_list_sol_2[8][13][25],int ag_crewchange_2[8][13][25], int
    ag_rboard_sol_2[8][13][25], int ag_rdepart_sol_2[8][13][25]);
void swap_calc4(vector<int>& candidate_emps_2, int& swap_emp_2,int& emp_extend_2,
     float& candidate_cost_2, float& swapping_cost_2,float total_cost_2[8], int&
    accept_rule_2, int& candidate_exist_2, int& transfer_sol_from_2, int&
    transfer_sol_to_2, int all_emp_2, int reg_emp_2, int all_roles_2, int
    weeks_to_plan_2, int& lambda_2, int allocate_sol_2[8][13][25][49], int
    long_work_sol_2[8][13][25][49][10], float emp_cost_2[8][48], int list_sol_2
    [8][13][48], int board_sol_2[8][13][25][48], int depart_sol_2[8][13][25][48],
     int undertime_sol_2[8][45], int overtime_sol_2[8][45],float ag_cost_2
    [8][13][25],int ag_list_sol_2[8][13][25],int ag_crewchange_2[8][13][25], int
    ag_rboard_sol_2[8][13][25], int ag_rdepart_sol_2[8][13][25]);
```

E.2.5.29 File 'trial.h'

```
#include <string>
#include <iostream>
void deneme(const std::string &file_name,int ag_max_work_2[25],int ag_work_zero_2
    [25],int under_rate_2[45], int over_rate_2[45], int g_weeks_2[45], int
    exp_worktime_2[45], int required_2[13][25], int eligible_2[13][25][49], int
    starting_2[25][49], int ag_starting_2[25], int work_zero_2[48], int
    rest_zero_2[48], int min_rest_2[48], int max_work_2[48], int cur_allocate_2
    [13][25][49]);
void deneme2(const std::string &file_name,int cur_board_2[13][25][48], int
    cur_depart_1[13][25][48], int cur_ag_rboard_2[13][25], int cur_ag_rdepart_2
    [13][25]);
void deneme3(const std::string &file_name,int cur_undertime_2[45], int
    cur_overtime_2[45], int cur_long_work_2[325][490],int cur_long_work_new_2
    [13][25][490], int current_long_work_2[13][25][49][10]);
void deneme4(const std::string &file_name,float board_cost_2[325][48], float
    board_chng_cost_2[13][25][48], float depart_cost_2[325][48], float
    depart_chng_cost_2[13][25][48], float ag_board_chng_cost_2[13][25], float
    ag_depart_chng_cost_2[13][25]);
void deneme5(const std::string &file_name,float workchange_cost_2[325][49], float
     work_chng_cost_2[13][25][49], float ext_chg_cost_2[325][490],float
    ext_chg_cost_new_2[13][25][490], float extension_chng_cost_2[13][25][49][10])
    ;
void deneme6(const std::string &file_name,int initial_solution_2[326][49],int
    taskbased_sol_2[13][25][49]);
```

E.2.6 Heuristic initial solution algorithm

Here we give the code used to solve the Time-Windows problem using the Heuristic initial solution algorithm described in section 6.6. This was implemented using the FICO Xpress software.

```
model ModelName
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver



! Recovery-type problem for the time-windows formulation,
!       attempting to find a (good??) initial solution using a heuristic method

parameters
DATE = "21-08-15"
PARAMETERFILE = "batch-input-parameters.dat"
end-parameters

declarations
InstanceName: string
end-declarations

initializations from PARAMETERFILE
InstanceName
end-initializations
```

```
DATAFILE := InstanceName+"\\Time-Windows - Captains - "+InstanceName+".txt"
OUTPUTFILE := InstanceName+"\\Logfile - Heuristic Initial Sol - "+InstanceName+"
    - "+DATE+".txt"
SUMMARYFILE := "Results - Heuristic Initial Sol - "+DATE+".txt"
SOLUTIONFILE := InstanceName+"\\Heuristic Initial soln - "+InstanceName+" - "+
    DATE+".txt"
COLLATEDSOLUTIONS := "Heuristic Initial solutions\\Heuristic Initial soln - "+
    InstanceName+" - "+DATE+".txt"


prog_starttime := gettime


!------------------------------------------------------------------------
!------------------------------------------------------------------------

! declare the basic values
declarations
REG_EMP: set of string       ! Regular employee names / numbers (ie not Agency)
ALL_EMP: set of string       ! Lables for ALL crew (including Agency)
GUARANTEED:   set of string  ! Set of employees on guaranteed days contracts
VESSELS: set of string       ! Labels / names of vessels
WEEKS_TO_PLAN: integer       ! Length of planning horizon in weeks
ROLES: array(VESSELS) of set of string     ! Labels for the roles which will
    require cover, divided by vessels
ALL_ROLES: set of string     ! List of all roles

EMP_NO: integer
ROLE_NO: integer

SOL_TYPE: set of string

end-declarations

initializations from DATAFILE
REG_EMP GUARANTEED VESSELS WEEKS_TO_PLAN ROLES
end-initializations

SOL_TYPE := {"initial", "current", "fixed", "eval", "trial"}

! calculate the set of all employees, and declare the rest of the Time-Windows
    variables
ALL_EMP := REG_EMP + {"AGENCY"}
```

```
ALL_ROLES := {}
FORALL (v in VESSELS) ALL_ROLES += ROLES(v)

EMP_NO := getsize(REG_EMP)
ROLE_NO := getsize(ALL_ROLES)

declarations
emp_count, role_count: integer
number_vacant: integer

TIME = 1..WEEKS_TO_PLAN      ! Time index

board_chng_cost, depart_chng_cost: array(REG_EMP, VESSELS, TIME) of real ! Costs
    of CHANGES TO employees boarding / leaving vessel
ag_board_chng_cost, ag_depart_chng_cost: array(ALL_ROLES, TIME) of real  ! Costs
    of CHANGES TO agency employees boarding / leaving for a given role
work_chng_cost: array(ALL_EMP, ALL_ROLES, TIME) of real
           ! (Direct) Costs of CHANGES TO employees working a given role at a
    given time

required: array(ALL_ROLES, TIME) of integer                     ! =1 if role r
    is required at time t, =0 otherwise
eligable: array(ALL_EMP, ALL_ROLES, TIME) of integer ! =1 if emp i can carry out
    role r at time t, =0 otherwise
starting: array(ALL_EMP, VESSELS) of integer             ! =1 if emp i is on
    board vessel k at time 0, =0 otherwise
! or takes a non-negative integer value for agency crew
ag_starting: array(ALL_ROLES) of integer                        ! =1 if agency
    employee is in role r at time 0, =0 otherwise
work_zero, rest_zero: array(REG_EMP) of integer          ! initial values of
    work_total and rest_total at time zero
ag_work_zero: array(ALL_ROLES) of integer                       ! initial value
    of work total for agency crew task by task
max_work, min_rest: array(REG_EMP) of integer            ! legal maximum on
    working time, minimum on resting time
ag_max_work: array(ALL_ROLES) of integer                        ! maximum on
    working time for agency crew, possibly different for each role

exp_worktime, g_weeks, under_rate, over_rate: array(GUARANTEED) of real  ! Data
    relating to over/undertime payments for employees

! Added for the recovery problem - the details of the current roster...
cur_allocate: array(ALL_EMP, ALL_ROLES, TIME) of integer
```

```
cur_board, cur_depart: array(REG_EMP, VESSELS, TIME) of integer
cur_ag_rboard, cur_ag_rdepart: array(ALL_ROLES, TIME) of integer
cur_undertime, cur_overtime: array(GUARANTEED) of real
end-declarations

initializations from DATAFILE
board_chng_cost depart_chng_cost work_chng_cost
ag_board_chng_cost ag_depart_chng_cost
required eligable starting ag_starting
work_zero rest_zero
max_work min_rest ag_max_work ag_work_zero

under_rate over_rate g_weeks exp_worktime

cur_allocate cur_board cur_depart cur_ag_rboard cur_ag_rdepart
cur_undertime cur_overtime
end-initializations


emp_count := 0
forall(e in REG_EMP) do
emp_count := emp_count + 1
emp_array(emp_count) := e
end-do
role_count := 0
forall(r in ALL_ROLES) do
role_count := role_count + 1
role_array(role_count) := r
end-do


! Calculate the parameters for the Long Work variables, and declare these
overall_regular_max_work := max(e in REG_EMP) max_work(e)
overall_agency_max_work := max(r in ALL_ROLES) ag_max_work(r)
if(overall_regular_max_work > overall_agency_max_work) then
overall_max_work := overall_regular_max_work
else
overall_max_work := overall_agency_max_work
end-if

declarations
lambda = 1..overall_max_work
                          ! Index used for number of consecutive weeks
```

```
extension_chng_cost: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real ! Cost of
    CHANGES TO an employee working on board a vessel for longer than usual
cur_long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of integer       !Added
    for recovery problem - detail of current roster, and change variable
end-declarations

initializations from DATAFILE
extension_chng_cost cur_long_work
end-initializations


! The rest of these declarations are required for the Heuristics:
declarations
! The initial solution:
taskbased_sol: array(ALL_EMP, ALL_ROLES, TIME) of integer ! =1 if emp i carries
    out role r at time t in the task-based soltion, =0 otherwise


! Relating to main programme:
iteration: integer

update_done: boolean
modified: boolean
candidate_exist: boolean

! Relating to the various solutions which must be recorded:
transfer_sol_to, transfer_sol_from: string

emp_cost: array(SOL_TYPE, REG_EMP) of real
ag_cost: array(SOL_TYPE, ALL_ROLES) of real
total_cost: array(SOL_TYPE) of real

emp_changes: array(SOL_TYPE, REG_EMP) of real
ag_changes: array(SOL_TYPE, ALL_ROLES) of real
total_changes: array(SOL_TYPE) of real

ag_crewchange: array(SOL_TYPE, ALL_ROLES) of set of integer

allocate_sol: array(SOL_TYPE, ALL_EMP, ALL_ROLES, TIME) of integer
board_sol, depart_sol: array(SOL_TYPE, REG_EMP, VESSELS, TIME) of integer
ag_rboard_sol, ag_rdepart_sol: array(SOL_TYPE, ALL_ROLES, TIME) of integer
undertime_sol, overtime_sol: array(SOL_TYPE, GUARANTEED) of integer
long_work_sol: array(SOL_TYPE, lambda, ALL_EMP, ALL_ROLES, TIME) of integer
```

```
vacant_sol: array(ALL_ROLES, TIME) of integer


! Relating to calculating the costs:
emps_changed: set of string
roles_changed: set of string
ag_roles_changed: set of string
to_calculate: string
to_check: string
working_at_t: boolean

agency_used: boolean

work_total, rest_total: array(REG_EMP, TIME) of real      ! Used to track the
    consecutive working time / rest period requirements of each employee
ag_work_total: array(ALL_ROLES, TIME) of real                    ! Used to track
    the consecutive working time of the agency employees

chng_allocate: array(SOL_TYPE, ALL_EMP, ALL_ROLES, TIME) of integer
chng_board, chng_depart: array(SOL_TYPE, REG_EMP, VESSELS, TIME) of integer
chng_ag_rboard, chng_ag_rdepart: array(SOL_TYPE, ALL_ROLES, TIME) of integer
chng_undertime, chng_overtime: array(SOL_TYPE, GUARANTEED) of integer
chng_long_work: array(SOL_TYPE, lambda, ALL_EMP, ALL_ROLES, TIME) of integer


! Required for calculating the least-cost Agency crew movements:
divide_number, tracking_number: real
definite_crewchange, possible_crewchange, evaluate_crewchange: set of integer
crewchange_cost, min_crewchange_cost: real
poss_ag_rboard, poss_ag_rdepart: array(TIME) of integer
poss_chng_ag_rboard, poss_chng_ag_rdepart: array(TIME) of integer
poss_ag_long_work, poss_chng_ag_long_work: array(lambda, TIME) of integer


! Relating to the identification and manipulation of the working blocks:
emp_before, emp_after: string
assign_emp_before, assign_emp_after: boolean
count_before, count_after: integer
task_vacant: string
vessel_vacant: string

block_found: boolean
```

```
new_block: boolean

block_start, earliest: integer
block_end, latest: integer
block_len: integer

feasible: boolean
candidate_cost: real

candidate_emp: string


! Relating to determining the order in which employees are examined:
emp_order_number: array(REG_EMP) of real
role_order_number: array(ALL_ROLES) of real
emp_ordered_list: list of string
role_ordered_list: list of string
emp_added_set: set of string
role_added_set: set of string
min_no: real
min_emp: string
min_role: string

final_cost: real
final_changes: real
final_vacant: real
final_iters: real
final_time: real
final_feasible: string

final_solution: array(ALL_EMP, ALL_ROLES, TIME) of integer

end-declarations


!-------------------------------------------------------------------------
!-------------------------------------------------------------------------

procedure transfer_solution
! transfer all solution details from one solution type to another (eg when the '
    candidate' becomes new 'current' solution

if(transfer_sol_from in SOL_TYPE and transfer_sol_to in SOL_TYPE) then
```

```
total_cost(transfer_sol_to) := total_cost(transfer_sol_from)
total_changes(transfer_sol_to) := total_changes(transfer_sol_from)

forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
allocate_sol(transfer_sol_to,e,r,t) := allocate_sol(transfer_sol_from,e,r,t)
                  !; if(r = "Orio-01" and e = "C-45") then writeln("Transfering
    allocate solution (",e,",",r,",",t,") = ",allocate_sol(transfer_sol_from,e,r,
    t)," from ",transfer_sol_from," to ",transfer_sol_to); end-if
chng_allocate(transfer_sol_to,e,r,t) := chng_allocate(transfer_sol_from,e,r,t)
                 !; if(r = "Orio-01" and e = "C-45") then writeln("Transfering
    change allocate (",e,",",r,",",t,") = ",chng_allocate(transfer_sol_from,e,r,t
    )," from ",transfer_sol_from," to ",transfer_sol_to); end-if
forall(l in lambda) do
long_work_sol(transfer_sol_to,l,e,r,t) := long_work_sol(transfer_sol_from,l,e,r,t
    )     !; if(l = 1 and e = "C-23" and t > 10) then writeln("Set long work (",
    transfer_sol_to,",",l,",",e,",",r,",",t,") = long work (",transfer_sol_from
    ,",",l,",",e,",",r,",",t,") = ",long_work_sol(transfer_sol_to,l,e,r,t)); end-
    if
chng_long_work(transfer_sol_to,l,e,r,t) := chng_long_work(transfer_sol_from,l,e,r
    ,t)
end-do
end-do
forall(e in REG_EMP) do
emp_cost(transfer_sol_to,e) := emp_cost(transfer_sol_from,e)
emp_changes(transfer_sol_to,e) := emp_changes(transfer_sol_from,e)

forall(t in TIME, v in VESSELS) do
board_sol(transfer_sol_to,e,v,t) := board_sol(transfer_sol_from,e,v,t)
chng_board(transfer_sol_to,e,v,t) := chng_board(transfer_sol_from,e,v,t)
depart_sol(transfer_sol_to,e,v,t) := depart_sol(transfer_sol_from,e,v,t)
chng_depart(transfer_sol_to,e,v,t) := chng_depart(transfer_sol_from,e,v,t)
end-do
if(e in GUARANTEED) then
undertime_sol(transfer_sol_to,e) := undertime_sol(transfer_sol_from,e)
chng_undertime(transfer_sol_to,e) := chng_undertime(transfer_sol_from,e)
overtime_sol(transfer_sol_to,e) := overtime_sol(transfer_sol_from,e)
chng_overtime(transfer_sol_to,e) := chng_overtime(transfer_sol_from,e)
end-if
end-do
forall(r in ALL_ROLES) do
ag_cost(transfer_sol_to,r) := ag_cost(transfer_sol_from,r)
ag_changes(transfer_sol_to,r) := ag_changes(transfer_sol_from,r)
```

```
ag_crewchange(transfer_sol_to,r) := ag_crewchange(transfer_sol_from,r)

forall(t in TIME) do
ag_rboard_sol(transfer_sol_to,r,t) := ag_rboard_sol(transfer_sol_from,r,t)
chng_ag_rboard(transfer_sol_to,r,t) := chng_ag_rboard(transfer_sol_from,r,t)
ag_rdepart_sol(transfer_sol_to,r,t) := ag_rdepart_sol(transfer_sol_from,r,t)
chng_ag_rdepart(transfer_sol_to,r,t) := chng_ag_rdepart(transfer_sol_from,r,t)
end-do
end-do

else
writeln("ERROR - incorrect option selected for transfer")
end-if

transfer_sol_to := ""
transfer_sol_from := ""

end-procedure


!----------------------------------------------------------------------
!----------------------------------------------------------------------

procedure calculate_cost

writeln("Costs may have changed for ",getsize(emps_changed)," regular employees:
    ",emps_changed)
writeln("Costs may have changed for agency in ",getsize(ag_roles_changed)," roles
    : ",ag_roles_changed)
writeln

if(to_calculate in SOL_TYPE) then
! REQUIRED?                     forall(e in REG_EMP) list_sol("eval",e) :=
   list_sol(to_calculate,e)
! REQUIRED?                     forall(r in ALL_ROLES) ag_list_sol("eval",r)
    := ag_list_sol(to_calculate,r)

total_cost("eval") := 0
total_changes("eval") := 0

forall(e in REG_EMP) do
if(e not in emps_changed) then
```

```
emp_cost("eval",e) := emp_cost("current",e)                                 !;
     writeln("Emp ",e," cost is same as current = ",emp_cost("eval",e))
emp_changes("eval",e) := emp_changes("current",e)

forall(t in TIME) do
forall(r in ALL_ROLES) do
allocate_sol("eval",e,r,t) := allocate_sol("current",e,r,t)
                              !; if(r = "Orio-01" and e = "C-45") then writeln
    ("Transfering allocate solution (",e,",",r,",",t,") = ",allocate_sol("current
    ",e,r,t)," from 'current' to 'eval'"); end-if
chng_allocate("eval",e,r,t) := chng_allocate("current",e,r,t)
forall(l in lambda) do
long_work_sol("eval",l,e,r,t) := long_work_sol("current",l,e,r,t)
                 !; if(l = 1 and e = "C-23" and t > 10) then writeln("Set long
    work ('eval',",l,",",e,",",r,",",t,") = long work ('current',",l,",",e,",",r
    ,",",t,") = ",long_work_sol("eval",l,e,r,t)); end-if
chng_long_work("eval",l,e,r,t) := chng_long_work("current",l,e,r,t)
end-do
end-do
forall(v in VESSELS) do
board_sol("eval",e,v,t) := board_sol("current",e,v,t)
chng_board("eval",e,v,t) := chng_board("current",e,v,t)
depart_sol("eval",e,v,t) := depart_sol("current",e,v,t)
chng_depart("eval",e,v,t) := chng_depart("current",e,v,t)
end-do
end-do
if(e in GUARANTEED) then
undertime_sol("eval",e) := undertime_sol("current",e)
chng_undertime("eval",e) := chng_undertime("current",e)
overtime_sol("eval",e) := overtime_sol("current",e)
chng_overtime("eval",e) := chng_overtime("current",e)
end-if

else

emp_cost("eval",e) := 0
                                                  !; writeln("Emp ",e," cost
    must be recalculated, now = ",emp_cost("eval",e))
emp_changes("eval",e) := 0

consec_work := work_zero(e)
                                                  !; if(e = "C-15") then
    writeln("(Emp ",e," starts with work zero = ",work_zero(e),")"); end-if
```

```
forall(t in TIME) do
working_at_t := false
forall(r in ALL_ROLES) do
forall(l in lambda) long_work_sol("eval",l,e,r,t) := 0
           !; if(e = "C-23" and t > 10) then writeln("Set long work ('eval',1,",e
    ,",",r,",",t,") = 0"); end-if

if(allocate_sol(to_calculate,e,r,t) = 1) then
allocate_sol("eval",e,r,t) := 1
                          !; if(r = "Orio-01" and e = "C-45") then writeln("
    Allocate solution (",e,",",r,",",t,") = ",allocate_sol(to_calculate,e,r,t),",
     so setting = 1 for 'eval'"); end-if
working_at_t := true
consec_work := consec_work +1
                          !; if(e = "C-15") then writeln("(Emp ",e," is working at
     time ",t,", so consec work = ",consec_work,")"); end-if
forall(l in 1..consec_work) long_work_sol("eval",l,e,r,t) := 1          !; if(e =
     "C-23" and t > 10) then writeln("Set long work ('eval',1,",e,",",r,",",t,")
    = 1"); end-if
else
allocate_sol("eval",e,r,t) := 0
                          !; if(r = "Orio-01" and e = "C-45") then writeln("
    Allocate solution (",e,",",r,",",t,") = ",allocate_sol(to_calculate,e,r,t),",
     so setting = 0 for 'eval'"); end-if
end-if
end-do
if(working_at_t = false) then
consec_work := 0
                                              !; if(e = "C-15") then writeln("(
    Emp ",e," NOT working at time ",t,", so consec work = ",consec_work,")"); end
    -if
end-if
end-do

forall(v in VESSELS) do
board_sol("eval",e,v,1) := 0
depart_sol("eval",e,v,1) := 0

if((sum(r in ROLES(v)) allocate_sol("eval",e,r,1)) < starting(e,v)) then
depart_sol("eval",e,v,1) := 1
elif((sum(r in ROLES(v)) allocate_sol("eval",e,r,1)) > starting(e,v)) then
board_sol("eval",e,v,1) := 1
```

```
end-if

forall(t in TIME | t > 1) do
board_sol("eval",e,v,t) := 0
depart_sol("eval",e,v,t) := 0

if((sum(r in ROLES(v)) allocate_sol("eval",e,r,t)) < (sum(r in ROLES(v))
    allocate_sol("eval",e,r,t-1))) then
depart_sol("eval",e,v,t) := 1
elif((sum(r in ROLES(v)) allocate_sol("eval",e,r,t)) > (sum(r in ROLES(v))
    allocate_sol("eval",e,r,t-1))) then
board_sol("eval",e,v,t) := 1
end-if
end-do
end-do

! Add costs:
if(e in GUARANTEED) then
if(g_weeks(e) > (exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(allocate_sol("
    eval",e,r,t)))) then
undertime_sol("eval",e) := integer(g_weeks(e) - (exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate_sol("eval",e,r,t))))
overtime_sol("eval",e) := 0
else
overtime_sol("eval",e) := integer((exp_worktime(e) + sum(r in ALL_ROLES, t in
    TIME)(allocate_sol("eval",e,r,t)))- g_weeks(e))
undertime_sol("eval",e) := 0
end-if
chng_undertime("eval",e) := integer(undertime_sol("eval",e) - cur_undertime(e))
chng_overtime("eval",e) := integer(overtime_sol("eval",e) - cur_overtime(e))
emp_cost("eval",e) := emp_cost("eval",e) + (under_rate(e)*chng_undertime("eval",e
    )) + (over_rate(e)*chng_overtime("eval",e))  !; if((under_rate(e)*
    chng_undertime(e)) + (over_rate(e)*chng_overtime(e)) <> 0) then writeln("Emp
    ",e,", UT & OT costs: ",(under_rate(e)*chng_undertime(e)) + (over_rate(e)*
    chng_overtime(e))); end-if
end-if

forall(r in ALL_ROLES, t in TIME) do
if(cur_allocate(e,r,t) = 0) then chng_allocate("eval",e,r,t) := allocate_sol("
    eval",e,r,t)                                                    !; if(r = "
    Orio-01" and e = "C-45") then writeln("Current allocate (",e,",",r,",",t,") =
     ",cur_allocate(e,r,t),", so for 'eval' set change allocate = allocate
    solution = ",chng_allocate("eval",e,r,t)); end-if
```

```
else chng_allocate("eval",e,r,t) := cur_allocate(e,r,t) - allocate_sol("eval",e,r
    ,t)                                                                  !; if(r = "
    Orio-01" and e = "C-45") then writeln("Current allocate (",e,",",r,",",t,") =
     ",cur_allocate(e,r,t),", so for 'eval' set change allocate = current
    allocate - allocate solution = ",chng_allocate("eval",e,r,t)); end-if
end-if
emp_cost("eval",e) := emp_cost("eval",e) + (work_chng_cost(e,r,t)*chng_allocate("
    eval",e,r,t))                                            !; if((work_chng_cost(
    e,r,t)*chng_allocate(e,r,t)) <> 0) then writeln("Emp ",e,", work change costs
    : ",(work_chng_cost(e,r,t)*chng_allocate(e,r,t))); end-if
emp_changes("eval",e) := emp_changes("eval",e) + chng_allocate("eval",e,r,t)
end-do

forall(v in VESSELS, t in TIME) do
if(cur_board(e,v,t) = 0) then chng_board("eval",e,v,t) := board_sol("eval",e,v,t)
else chng_board("eval",e,v,t) := cur_board(e,v,t) - board_sol("eval",e,v,t)
end-if
emp_cost("eval",e) := emp_cost("eval",e) + (board_chng_cost(e,v,t)*chng_board("
    eval",e,v,t))                                              !; if((
    board_chng_cost(e,v,t)*chng_board(e,v,t)) <> 0) then writeln("Emp ",e,",
    board change costs: ",(board_chng_cost(e,v,t)*chng_board(e,v,t))); end-if
end-do

forall(v in VESSELS, t in TIME) do
if(cur_depart(e,v,t) = 0) then chng_depart("eval",e,v,t) := depart_sol("eval",e,v
    ,t)
else chng_depart("eval",e,v,t) := cur_depart(e,v,t) - depart_sol("eval",e,v,t)
end-if
emp_cost("eval",e) := emp_cost("eval",e) + (depart_chng_cost(e,v,t)*chng_depart("
    eval",e,v,t))                                            !; if((
    depart_chng_cost(e,v,t)*chng_depart(e,v,t)) <> 0) then writeln("Emp ",e,",
    depart change costs: ",(depart_chng_cost(e,v,t)*chng_depart(e,v,t))); end-if
end-do

forall(l in lambda, r in ALL_ROLES, t in TIME) do
if(cur_long_work(l,e,r,t) = 0) then chng_long_work("eval",l,e,r,t) :=
    long_work_sol("eval",l,e,r,t)
else chng_long_work("eval",l,e,r,t) := cur_long_work(l,e,r,t) - long_work_sol("
    eval",l,e,r,t)
end-if
emp_cost("eval",e) := emp_cost("eval",e) + (extension_chng_cost(l,e,r,t)*
    chng_long_work("eval",l,e,r,t))                          !; if((
    extension_chng_cost(l,e,r,t)*chng_long_work(l,e,r,t)) <> 0) then writeln("Emp
```

```
     ",e,", longwork change costs: ",(extension_chng_cost(l,e,r,t)*chng_long_work
    (l,e,r,t))); end-if
end-do

end-if
end-do



forall(r in ALL_ROLES) do
if(r not in ag_roles_changed) then
ag_cost("eval",r) := ag_cost("current",r)

                                 !; writeln("Ag Role ",r," cost is same as current
     = ",ag_cost("eval",r))
ag_changes("eval",r) := ag_changes("current",r)
ag_crewchange("eval",r) := ag_crewchange("current",r)
forall(t in TIME) do
allocate_sol("eval","AGENCY",r,t) := allocate_sol("current","AGENCY",r,t)
chng_allocate("eval","AGENCY",r,t) := chng_allocate("current","AGENCY",r,t)
forall(l in lambda) do
long_work_sol("eval",l,"AGENCY",r,t) := long_work_sol("current",l,"AGENCY",r,t)
chng_long_work("eval",l,"AGENCY",r,t) := chng_long_work("current",l,"AGENCY",r,t)
end-do
ag_rboard_sol("eval",r,t) := ag_rboard_sol("current",r,t)
chng_ag_rboard("eval",r,t) := chng_ag_rboard("current",r,t)
ag_rdepart_sol("eval",r,t) := ag_rdepart_sol("current",r,t)
chng_ag_rdepart("eval",r,t) := chng_ag_rdepart("current",r,t)
end-do

else

ag_cost("eval",r) := 0

                                                !; writeln("Ag Role ",r," cost
    must be recalculated, now = ",ag_cost("eval",r))
ag_changes("eval",r) := 0

if(ag_starting(r) = 1) then
onboard := true
else
onboard := false
end-if
```

```
possible_crewchange := {}
definite_crewchange := {}
forall(t in TIME) do
if(allocate_sol(to_calculate,"AGENCY",r,t) = 1) then
allocate_sol("eval","AGENCY",r,t) := 1
if(onboard) = FALSE then
definite_crewchange += {t}
else
possible_crewchange += {t}
end-if
onboard := TRUE
else
allocate_sol("eval","AGENCY",r,t) := 0
if(onboard) = TRUE then
definite_crewchange += {t}
end-if
onboard := FALSE
end-if
end-do

ag_crewchange("eval",r) := {}
                                        !; writeln("Evaluating solution <> best
    or current solution for agency for role ",r)
if(possible_crewchange = {}) then
forall(p in definite_crewchange) ag_crewchange("eval",r) += {p}                !;
     writeln("\tPossible crewchange set is empty, so crewchange set = ",
    ag_evaluating_crewchange(r))
else
number_to_run := 2^(getsize(possible_crewchange))
                  !; writeln("\tPossible crewchange set = ",possible_crewchange
    ,"\t=> must examine ",number_to_run," combinations...")

forall(x in 1..integer(number_to_run)) do
divide_number := number_to_run
tracking_number := x-1
feas_crewchange := true
crewchange_cost := 0
evaluate_crewchange := {}

consec_work := ag_work_zero(r)

forall(t in TIME) do
if(t in possible_crewchange) then
```

```
divide_number := divide_number/2
if(tracking_number/divide_number < 1) then
poss_ag_rboard(t) := 0
poss_ag_rdepart(t) := 0
else
evaluate_crewchange += {t}
poss_ag_rboard(t) := 1
poss_ag_rdepart(t) := 1
tracking_number := tracking_number - divide_number
consec_work := 0
end-if

if(cur_ag_rboard(r,t) = 0) then
poss_chng_ag_rboard(t) := poss_ag_rboard(t)
else
poss_chng_ag_rboard(t) := cur_ag_rboard(r,t) - poss_ag_rboard(t)
end-if

if(cur_ag_rdepart(r,t) = 0) then
poss_chng_ag_rdepart(t) := poss_ag_rdepart(t)
else
poss_chng_ag_rdepart(t) := cur_ag_rdepart(r,t) - poss_ag_rdepart(t)
end-if

crewchange_cost := crewchange_cost + (ag_board_chng_cost(r,t)*poss_chng_ag_rboard
    (t)) + (ag_depart_chng_cost(r,t)*poss_chng_ag_rdepart(t))

end-if

forall(l in lambda) poss_ag_long_work(l,t) := 0

if(allocate_sol("eval","AGENCY",r,t) = 1) then
consec_work := consec_work +1
if(consec_work > ag_max_work(r)) then
feas_crewchange := false
else
forall(l in 1..consec_work) poss_ag_long_work(l,t) := 1
end-if
else
consec_work := 0
end-if

forall(l in lambda) do
```

```
if(cur_long_work(l,"AGENCY",r,t) = 0) then
poss_ag_chng_long_work(l,t) := poss_ag_long_work(l,t)
else
poss_ag_chng_long_work(l,t) := cur_long_work(l,"AGENCY",r,t) - poss_ag_long_work(
    l,t)
end-if
crewchange_cost := crewchange_cost + (extension_chng_cost(l,"AGENCY",r,t)*
    poss_ag_chng_long_work(l,t))
end-do

end-do

if(feas_crewchange = true) then
if(ag_crewchange("eval",r) = {}) then
forall(p in definite_crewchange) ag_crewchange("eval",r) += {p}
forall(p in evaluate_crewchange) ag_crewchange("eval",r) += {p}
min_crewchange_cost := crewchange_cost
!; writeln("\t\tCombination ",evaluate_crewchange," is first feasible one\tCost =
     ",crewchange_cost)
else
if(crewchange_cost < min_crewchange_cost) then
ag_crewchange("eval",r) := {}
forall(p in definite_crewchange) ag_crewchange("eval",r) += {p}
forall(p in evaluate_crewchange) ag_crewchange("eval",r) += {p}
min_crewchange_cost := crewchange_cost
!; writeln("\t\tCombination ",evaluate_crewchange," gives an improvement \tCost =
     ",crewchange_cost)
!                                                   else
!                                                          writeln("\t\tCombination
     ",evaluate_crewchange," is not an improvement\tCost = ",crewchange_cost)
end-if
end-if
!                                   else
!                                          writeln("\t\tCombination ",
    evaluate_crewchange," is infeasible")
end-if

end-do

!                            writeln("\tBest solution is to have crewchange set =
     ",ag_evaluating_crewchange(r))
end-if
```

```
consec_work := ag_work_zero(r)
forall(t in TIME) do
ag_rboard_sol("eval",r,t) := 0
ag_rdepart_sol("eval",r,t) := 0

if(t in ag_crewchange("eval",r)) then
consec_work := 0

if(t = 1) then
if(ag_starting(r) = 1) then
ag_rdepart_sol("eval",r,t) := 1
end-if
else
if(allocate_sol("eval","AGENCY",r,t-1) = 1) then
ag_rdepart_sol("eval",r,t) := 1
end-if
end-if

if(allocate_sol("eval","AGENCY",r,t) = 1) then
ag_rboard_sol("eval",r,t) := 1
end-if
end-if

forall(l in lambda) long_work_sol("eval",l,"AGENCY",r,t) := 0
if(allocate_sol("eval","AGENCY",r,t) = 1) then
consec_work := consec_work +1
forall(l in 1..consec_work) long_work_sol("eval",l,"AGENCY",r,t) := 1
else
consec_work := 0
end-if


if(cur_allocate("AGENCY",r,t) = 0) then chng_allocate("eval","AGENCY",r,t) :=
    allocate_sol("eval","AGENCY",r,t)
else chng_allocate("eval","AGENCY",r,t) := cur_allocate("AGENCY",r,t) -
    allocate_sol("eval","AGENCY",r,t)
end-if
ag_cost("eval",r) := ag_cost("eval",r) + (work_chng_cost("AGENCY",r,t)*
    chng_allocate("eval","AGENCY",r,t))                              !; if((
    work_chng_cost("AGENCY",r,t)*chng_allocate("AGENCY",r,t)) <> 0) then writeln
    ("Ag Role ",r,", work change costs: ",(work_chng_cost("AGENCY",r,t)*
    chng_allocate("AGENCY",r,t))); end-if                                   !; if
    (r = "Ospr-01" and to_calculate = "current") then writeln("In time period ",t
    ,", allocate = ",allocate_sol("eval","AGENCY",r,t)," => change = ",
    chng_allocate("AGENCY",r,t)," => added cost = ",work_chng_cost("AGENCY",r,t)
    ,"*",chng_allocate("AGENCY",r,t)); end-if
ag_changes("eval",r) := ag_changes("eval",r) + chng_allocate("eval","AGENCY",r,t)

if(cur_ag_rboard(r,t) = 0) then chng_ag_rboard("eval",r,t) := ag_rboard_sol("eval
    ",r,t)
else chng_ag_rboard("eval",r,t) := cur_ag_rboard(r,t) - ag_rboard_sol("eval",r,t)
end-if
ag_cost("eval",r) := ag_cost("eval",r) + (ag_board_chng_cost(r,t)*chng_ag_rboard
    ("eval",r,t))                                                  !; if((
    ag_board_chng_cost(r,t)*chng_ag_rboard(r,t)) <> 0) then writeln("Ag Role ",r
    ,", board change costs: ",(ag_board_chng_cost(r,t)*chng_ag_rboard(r,t))); end
    -if
           !; if(r = "Ospr-01" and to_calculate = "current") then writeln("In
    time period ",t,", board = ",ag_rboard_sol("eval",r,t)," => change = ",
    chng_ag_rboard(r,t)," => added cost = ",ag_board_chng_cost(r,t),"*",
    chng_ag_rboard(r,t)); end-if

if(cur_ag_rdepart(r,t) = 0) then chng_ag_rdepart("eval",r,t) := ag_rdepart_sol("
    eval",r,t)
else chng_ag_rdepart("eval",r,t) := cur_ag_rdepart(r,t) - ag_rdepart_sol("eval",r
    ,t)
end-if
ag_cost("eval",r) := ag_cost("eval",r) + (ag_depart_chng_cost(r,t)*
    chng_ag_rdepart("eval",r,t))                                         !; if((
    ag_depart_chng_cost(r,t)*chng_ag_rdepart(r,t)) <> 0) then writeln("Ag Role ",
    r,", depart change costs: ",(ag_depart_chng_cost(r,t)*chng_ag_rdepart(r,t)));
     end-if
           !; if(r = "Ospr-01" and to_calculate = "current") then writeln("In
    time period ",t,", depart = ",ag_rdepart_sol("eval",r,t)," => change = ",
    chng_ag_rdepart(r,t)," => added cost = ",ag_depart_chng_cost(r,t),"*",
    chng_ag_rdepart(r,t)); end-if

forall(l in lambda) do
if(cur_long_work(l,"AGENCY",r,t) = 0) then chng_long_work("eval",l,"AGENCY",r,t)
    := long_work_sol("eval",l,"AGENCY",r,t)
else chng_long_work("eval",l,"AGENCY",r,t) := cur_long_work(l,"AGENCY",r,t) -
    long_work_sol("eval",l,"AGENCY",r,t)
end-if
ag_cost("eval",r) := ag_cost("eval",r) + (extension_chng_cost(l,"AGENCY",r,t)*
    chng_long_work("eval",l,"AGENCY",r,t))  !; if((extension_chng_cost(l,"AGENCY
    ",r,t)*chng_long_work(l,"AGENCY",r,t)) <> 0) then writeln("Ag Role ",r,",
    longwork change costs: ",(extension_chng_cost(l,"AGENCY",r,t)*chng_long_work(
    l,"AGENCY",r,t))); end-if  !; if(r = "Ospr-01" and to_calculate = "current")
    then writeln("In time period ",t," for l = ",l,", long work = ",long_work_sol
    ("eval",l,"AGENCY",r,t)," => change = ",chng_long_work(l,"AGENCY",r,t)," =>
    added cost = ",extension_chng_cost(l,"AGENCY",r,t),"*",chng_long_work(l,"
    AGENCY",r,t)); end-if
end-do

end-do

end-if
end-do


! CALCULATE TOTALS

total_cost("eval") := (sum(e in REG_EMP) emp_cost("eval",e)) + (sum(r in
    ALL_ROLES) ag_cost("eval",r))
total_changes("eval") := (sum(e in REG_EMP) emp_changes("eval",e)) + (sum(r in
    ALL_ROLES) ag_changes("eval",r))

writeln("Cost for ",to_calculate," solution is:\t\t",total_cost("eval"))
writeln("Number of changes in this solution is:\t",total_changes("eval"))
if(to_calculate <> "trial") then writeln("\t(Number of vacant roles:\t",
    number_vacant,")"); end-if

transfer_sol_from := "eval"
transfer_sol_to := to_calculate
transfer_solution

else
writeln("ERROR - incorrect option selected for evaluation")
end-if

writeln
writeln("------------------------------------------------------")
writeln

end-procedure


!-------------------------------------------------------------------------
!-------------------------------------------------------------------------
```

```
! Define initial solution array values

procedure initialise

number_vacant := 0
forall(r in ALL_ROLES, t in TIME) do
vacant_sol(r,t) := 1
forall(e in ALL_EMP) do
allocate_sol("initial", e, r, t) := 0      !; if(r = "Orio-01" and e = "C-45")
    then writeln("Initialising - set allocate sol ('initial',",e,",",r,",",t,") =
     0"); end-if
allocate_sol("current", e, r, t) := 0      !; if(r = "Orio-01" and e = "C-45")
    then writeln("Initialising - set allocate sol ('current',",e,",",r,",",t,") =
     0"); end-if
allocate_sol("fixed",e,r,t) := 0                  !; if(r = "Orio-01" and e = "C
    -45") then writeln("Initialising - set allocate sol ('fixed',",e,",",r,",",t
    ,") = 0"); end-if

if(eligable(e,r,t) = 1) then
if(cur_allocate(e,r,t) = 1) then
allocate_sol("initial", e, r, t) := 1      !; if(r = "Orio-01" and e = "C-45")
    then writeln("eligable and current allocate (",e,",",r,",",t,") = 1, so set
    allocate sol ('initial',",e,",",r,",",t,") = 1"); end-if
allocate_sol("current", e, r, t) := 1      !; if(r = "Orio-01" and e = "C-45")
    then writeln("eligable and current allocate (",e,",",r,",",t,") = 1, so set
    allocate sol ('current',",e,",",r,",",t,") = 1"); end-if
vacant_sol(r,t) := 0
end-if
else
allocate_sol("fixed",e,r,t) := 1                        !; if(r = "Orio-01" and
    e = "C-45") then writeln("eligable (",e,",",r,",",t,") = 0, so set allocate
    sol ('fixed',",e,",",r,",",t,") = 1"); end-if
end-if
end-do

if(vacant_sol(r,t) = 1) then number_vacant := number_vacant + 1
end-if
end-do

emps_changed := {}
roles_changed := {}
ag_roles_changed := {}
```

```
forall(e in REG_EMP) emps_changed += {e}
forall(r in ALL_ROLES) ag_roles_changed += {r}


writeln
writeln("Original schedule has roles carried out as follows:")
forall(t in TIME) write("\tWeek ",t)
write("\n")
forall(r in ALL_ROLES) do
write(r)
forall(t in TIME) do
if(required(r,t) = 0) then
write("\t (n/a)")
else
covered := FALSE
forall(e in ALL_EMP) do
if(cur_allocate(e,r,t) = 1) then
write("\t",e)
covered := TRUE
end-if
end-do
if(covered = FALSE) then write("\t*TBC*"); end-if
end-if
end-do
write("\n")
end-do
writeln
writeln

writeln("... and crew assigned to roles as follows:")
forall(t in TIME) write("\tWeek ",t)
write("\n")
forall(e in REG_EMP) do
write(e)
forall(t in TIME) do
working := FALSE
forall(r in ALL_ROLES) do
if(cur_allocate(e,r,t) = 1) then
write("\t",r)
working := TRUE
end-if
end-do
if(working = FALSE) then
```

```
avail := FALSE
forall(r in ALL_ROLES) do
if(eligable(e,r,t) = 1) then avail := TRUE
end-if
end-do
if(avail = FALSE) then write("\t*unav*")
else write("\t")
end-if
end-if
end-do
write("\n")
end-do
writeln
writeln
writeln("And the following assigned to agency:")
agency_used := false
forall(r in ALL_ROLES) do
if(sum(t in TIME) (cur_allocate("AGENCY",r,t)) > 0) then
agency_used := true
write("Role ",r," in week(s): ")
forall(t in TIME) do
if(cur_allocate("AGENCY",r,t) = 1) then write(t," "); end-if
end-do
write("\n")
end-if
end-do
if(agency_used = false) then writeln("\t(none)"); end-if
writeln
writeln("------------------------------------------------------")
writeln


writeln("Number vacant = ",number_vacant,", consisting of:")
forall(r in ALL_ROLES, t in TIME) do
if(vacant_sol(r,t) = 1) then writeln("\tRole ",r," in Week ",t); end-if
end-do
writeln
writeln("------------------------------------------------------")
writeln

end-procedure
```

```
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------

procedure show_updates

writeln
writeln("Solution for the following roles has been updated as shown:")
forall(t in TIME) write("\tWeek ",t)
write("\n")
forall(r in roles_changed) do
write(r)
forall(t in TIME) do
if(required(r,t) = 0) then
write("\t (n/a)")
else
covered := FALSE
forall(e in ALL_EMP) do
if(allocate_sol("current",e,r,t) = 1) then
write("\t",e)
covered := TRUE
end-if
end-do
if(covered = FALSE) then write("\t*TBC*"); end-if
end-if
end-do
write("\n")
end-do
writeln
writeln

if(emps_changed = {}) then
writeln("(No regular crew have had their assignments updated)")
else
writeln("... and the following crew have had assignments updated as shown:")
forall(t in TIME) write("\tWeek ",t)
write("\n")
forall(e in emps_changed) do
write(e)
forall(t in TIME) do
working := FALSE
forall(r in ALL_ROLES) do
if(allocate_sol("current",e,r,t) = 1) then
```

```
write("\t",r)
working := TRUE
end-if
end-do
if(working = FALSE) then
avail := FALSE
forall(r in ALL_ROLES) do
if(eligable(e,r,t) = 1) then avail := TRUE
end-if
end-do
if(avail = FALSE) then write("\t*unav*")
else write("\t")
end-if
end-if
end-do
write("\n")
end-do
writeln
end-if
writeln

if(ag_roles_changed = {}) then
writeln("(Assignments have not changed for agency crew)")
else
writeln("And agency assignments to the following roles have changed as shown:")
agency_used := false
forall(r in ag_roles_changed) do
if(sum(t in TIME) (allocate_sol("current","AGENCY",r,t)) > 0) then
agency_used := true
write("Role ",r," in week(s): ")
forall(t in TIME) do
if(allocate_sol("current","AGENCY",r,t) = 1) then write(t," "); end-if
end-do
write("\n")
end-if
end-do
if(agency_used = false) then writeln("\t(none)"); end-if
end-if
writeln
writeln("------------------------------------------------------")
writeln

end-procedure
```

```
!----------------------------------------------------------------------
!----------------------------------------------------------------------

procedure initial_feas_check

writeln("Carry out preliminary checks on feasibility:")

emps_changed := {}
roles_changed := {}
ag_roles_changed := {}

update_done := false

forall(e in REG_EMP, v in VESSELS) do
if(starting(e,v) > 0) then
forall(r in ALL_ROLES) do
if(r not in ROLES(v) or (r in ROLES(v) and eligable(e,r,1) < 1)) then
forall(t in 1..min_rest(e)) do
if(allocate_sol("current",e,r,t) = 1) then
update_done := true

writeln(" -> remove employee ",e," from role ",r," at time ",t)
allocate_sol("current",e,r,t) := 0
vacant_sol(r,t) := 1
number_vacant := number_vacant + 1

emps_changed += {e}
roles_changed += {r}
end-if

allocate_sol("fixed",e,r,t) := 1
end-do
end-if
end-do
end-if
end-do

writeln
if(update_done = false) then
writeln("No changes were necessary at this point")
writeln
```

```
writeln("------------------------------------------------------")
writeln
else
writeln("Recalculate costs:")
writeln
to_calculate := "current"
calculate_cost

show_updates
end-if

end-procedure


!------------------------------------------------------------------------
!------------------------------------------------------------------------

procedure repair_vacancies

writeln("Block detail:\tstart: ",block_start,"\t end: ",block_end,"\t length: ",
    block_len)
writeln(" --> repair this block")
writeln

! First we find the employees who work before and after the block:
emp_before := "none"
emp_after := "none"
update_done := FALSE


forall(e in ALL_EMP) do
if(block_start > 1) then
if(allocate_sol("current",e,task_vacant,block_start-1) = 1) then emp_before := e;
     end-if
end-if
if(block_end < WEEKS_TO_PLAN) then
if(allocate_sol("current",e,task_vacant,block_end+1) = 1) then emp_after := e;
    end-if
end-if
end-do

writeln("Employee in role in previous period: ",emp_before)
writeln("Employee in role in subsequent period: ",emp_after)
```

```
writeln


! Now we look to repair this infeasibility
emps_changed := {}
roles_changed := {}
ag_roles_changed := {}
assign_emp_before := false
assign_emp_after := false


! Can we assign the employee before or after?
if(block_len = WEEKS_TO_PLAN) then
writeln("Role is vacant across entire planning horizon - fill with agency crew")

forall(t in TIME) do
allocate_sol("current","AGENCY",task_vacant,t) := 1
allocate_sol("fixed","AGENCY",task_vacant,t) := 1
vacant_sol(task_vacant,t) := 0
number_vacant := number_vacant -1
end-do
roles_changed += {task_vacant}
ag_roles_changed += {task_vacant}

update_done := true

elif(block_end = WEEKS_TO_PLAN and emp_before <> "none") then
writeln("Role is vacant at end of planning horizon")

if(emp_before = "AGENCY") then assign_emp_before := true
elif(block_len < max_work(emp_before)) then
if(long_work_sol("current",max_work(emp_before)-block_len+1,emp_before,
    task_vacant,block_start-1) = 0) then
if(sum(t in block_start..block_end, r in ALL_ROLES)(allocate_sol("current",
    emp_before,r,t)) < 1) then
if(sum(t in block_start..block_end)(eligable(emp_before,task_vacant,t)*(1-
    allocate_sol("fixed",emp_before,task_vacant,t))) = block_len) then
assign_emp_before := true
end-if
end-if
end-if
end-if
```

```
elif(block_start = 1 and emp_after <> "none") then
writeln("Role is vacant at start of planning horizon")

if(emp_after = "AGENCY") then assign_emp_after := true
elif(rest_zero(emp_after) = 0) then
if(sum(t in block_start..block_end, r in ALL_ROLES)(allocate_sol("current",
    emp_after,r,t)) < 1) then
if(sum(t in block_start..block_end)(eligable(emp_after,task_vacant,t)*(1-
    allocate_sol("fixed",emp_after,task_vacant,t))) = block_len) then
assign_emp_after := true
end-if
end-if
end-if

else
writeln("Role is occupied at both start and end of planning horizon")

if(emp_before <> "AGENCY" and emp_before <> "none") then
if(block_len <= max_work(emp_before)) then
if(long_work_sol("current",max_work(emp_before)-block_len+1,emp_before,
    task_vacant,block_start-1) = 0) then
if(sum(t in block_start..block_end, r in ALL_ROLES)(allocate_sol("current",
    emp_before,r,t)) < 1) then
if(sum(t in block_start..block_end)(eligable(emp_before,task_vacant,t)*(1-
    allocate_sol("fixed",emp_before,task_vacant,t))) = block_len) then
assign_emp_before := true
end-if
end-if
end-if
end-if
end-if

if(assign_emp_before = false and emp_after <> "none") then
if(emp_after = "AGENCY") then assign_emp_after := true
else
if(block_start <= min_rest(emp_after)) then
if(rest_zero(emp_after) < block_start and sum(t in 1..block_end, r in ALL_ROLES)(
    allocate_sol("current",emp_after,r,t)) < 1) then
if(sum(t in block_start..block_end)(eligable(emp_after,task_vacant,t)*(1-
    allocate_sol("fixed",emp_after,task_vacant,t))) = block_len) then
assign_emp_after := true
end-if
end-if
```

```
else
if(sum(t in block_start-min_rest(emp_after)..block_end, r in ALL_ROLES)(
    allocate_sol("current",emp_after,r,t)) < 1) then
if(sum(t in block_start..block_end)(eligable(emp_after,task_vacant,t)*(1-
    allocate_sol("fixed",emp_after,task_vacant,t))) = block_len) then
assign_emp_after := true
end-if
end-if
end-if
end-if
end-if

if(assign_emp_after = false and emp_before = "AGENCY") then assign_emp_before :=
    true
end-if


if(assign_emp_after = false and assign_emp_before = false) then
writeln("--> Cannot cover task by either previous or subsequent employee")
writeln(" => Look at removing shortest period employee")

count_after := 0
working_at_t := true
forall(t in block_end+1..WEEKS_TO_PLAN) do
if(working_at_t = true) then
if(allocate_sol("current",emp_after,task_vacant,t) = 1 and allocate_sol("fixed",
    emp_after,task_vacant,t) = 0) then
count_after := count_after + 1
else
working_at_t := false
end-if
end-if
end-do

count_before := 0
working_at_t := true
forall(t in 1..block_start-1) do
if(working_at_t = true) then
if(allocate_sol("current",emp_before,task_vacant,block_start-t) = 1 and
    allocate_sol("fixed",emp_before,task_vacant,block_start-t) = 0) then
count_before := count_before + 1
else
working_at_t := false
```

```
end-if
end-if
end-do

!                    if(block_len <= count_before and block_len <= count_after
    and (block_len < count_before or block_len < count_after)) then
if(block_len <= 2 or count_before <= 2 or count_after <= 2 or (block_len +
    count_before) <= 4 or (block_len + count_after) <= 4) then

if(count_before < count_after and count_before > 0) then
writeln(" ---> Employee before has shorter working period (",count_before," to ",
    count_after,")")
forall(t in TIME) do
if(t >= block_start-count_before and t <= block_start-1) then
allocate_sol("current",emp_before,task_vacant,t) := 0                          !;
     if(task_vacant = "Orio-01" and emp_before = "C-45") then writeln("Set
    allocate sol ('current',",emp_before,",",task_vacant,",",t,") = 0"); end-if
allocate_sol("fixed",emp_before,task_vacant,t) := 1
           !; if(task_vacant = "Orio-01" and emp_before = "C-45") then writeln("
    Set allocate sol ('fixed',",emp_before,",",task_vacant,",",t,") = 1"); end-if
vacant_sol(task_vacant,t) := 1
number_vacant := number_vacant + 1
elif(t <= rest_zero(emp_before) or (block_start-count_before = 1 and t <=
    min_rest(emp_before))) then
forall(r in ALL_ROLES) do
if(allocate_sol("current",emp_before,r,t) = 1) then
allocate_sol("current",emp_before,r,t) := 0
allocate_sol("fixed",emp_before,r,t) := 1
vacant_sol(r,t) := 1
number_vacant := number_vacant + 1

roles_changed += {r}
end-if
end-do
end-if
end-do


emps_changed += {emp_before}
roles_changed += {task_vacant}
update_done := true ! consider update to be done - can be filled at next
    iteration
elif(count_after > 0) then
```

```
writeln(" ---> Employee before does not have shorter working period (",
    count_before," to ",count_after,")")
forall(t in block_end+1..block_end+count_after) do
allocate_sol("current",emp_after,task_vacant,t) := 0                           !;
     if(task_vacant = "Orio-01" and emp_after = "C-45") then writeln("Set
    allocate sol ('current',",emp_after,",",task_vacant,",",t,") = 0"); end-if
allocate_sol("fixed",emp_after,task_vacant,t) := 1
           !; if(task_vacant = "Orio-01" and emp_after = "C-45") then writeln("
    Set allocate sol ('fixed',",emp_after,",",task_vacant,",",t,") = 1"); end-if
vacant_sol(task_vacant,t) := 1
number_vacant := number_vacant + 1
end-do
emps_changed += {emp_after}
roles_changed += {task_vacant}
update_done := true ! consider update to be done - can be filled at next
    iteration
end-if

else
writeln(" ---> No removal will take place - block length not short enough, nor
    periods before and after (",count_before," and ",count_after,")")
end-if

end-if
end-if


! If we can assign the employee before or after:
if(assign_emp_before = true) then
writeln("-> Can assign employee from previous period, being ",emp_before)
forall(t in block_start..block_end) do
allocate_sol("current",emp_before,task_vacant,t) := 1                          !;
     if(task_vacant = "Orio-01" and emp_before = "C-45") then writeln("Set
    allocate sol ('current',",emp_before,",",task_vacant,",",t,") = 1"); end-if
allocate_sol("fixed",emp_before,task_vacant,t) := 1
           !; if(task_vacant = "Orio-01" and emp_before = "C-45") then writeln("
    Set allocate sol ('fixed',",emp_before,",",task_vacant,",",t,") = 1"); end-if
vacant_sol(task_vacant,t) := 0
number_vacant := number_vacant -1
end-do

roles_changed += {task_vacant}
```

```
if(emp_before = "AGENCY") then
ag_roles_changed += {task_vacant}
else
emps_changed += {emp_before}

forall(t in block_end+1..block_end+min_rest(emp_before) | t <= WEEKS_TO_PLAN) do
forall(r in ALL_ROLES) do
if(allocate_sol("current",emp_before,r,t) = 1) then
allocate_sol("current",emp_before,r,t) := 0
allocate_sol("fixed",emp_before,r,t) := 1
vacant_sol(r,t) := 1
number_vacant := number_vacant + 1

roles_changed += {r}
end-if
end-do
end-do
end-if



elif(assign_emp_after = true) then
writeln("-> Can assign employee from subsequent period, being ",emp_after)
forall(t in block_start..block_end) do
allocate_sol("current",emp_after,task_vacant,t) := 1                          !;
     if(task_vacant = "Orio-01" and emp_after = "C-45") then writeln("Set
    allocate sol ('current',",emp_after,",",task_vacant,",",t,") = 1"); end-if
allocate_sol("fixed",emp_after,task_vacant,t) := 1
           !; if(task_vacant = "Orio-01" and emp_after = "C-45") then writeln("
    Set allocate sol ('fixed',",emp_after,",",task_vacant,",",t,") = 1"); end-if
vacant_sol(task_vacant,t) := 0
number_vacant := number_vacant -1
end-do

roles_changed += {task_vacant}

if(emp_after = "AGENCY") then
ag_roles_changed += {task_vacant}
else
emps_changed += {emp_after}

consec_work := block_len
forall(t in block_end+1..WEEKS_TO_PLAN) do
if(allocate_sol("current",emp_after,task_vacant,t) = 1) then
if(consec_work = max_work(emp_after)) then
allocate_sol("current",emp_after,task_vacant,t) := 0                          !;
     if(task_vacant = "Orio-01" and emp_after = "C-45") then writeln("Set
    allocate sol ('current',",emp_after,",",task_vacant,",",t,") = 0"); end-if
allocate_sol("fixed",emp_after,task_vacant,t) := 1
           !; if(task_vacant = "Orio-01" and emp_after = "C-45") then writeln("
    Set allocate sol ('fixed',",emp_after,",",task_vacant,",",t,") = 1"); end-if
vacant_sol(task_vacant,t) := 1
number_vacant := number_vacant +1
else
consec_work := consec_work + 1
end-if
else
consec_work := 0
end-if
end-do

end-if

elif(update_done = false) then
writeln("-> cannot assign employee from previous or subsequent period...")

candidate_exist := false
candidate_cost := 0
candidate_emp := ""
forall(e in emp_ordered_list | e <> emp_before and e <> emp_after) do
emps_changed := {}
roles_changed := {}
ag_roles_changed := {}

earliest := block_start - min_rest(e)
if(earliest < 1) then earliest := 1; end-if
latest := block_end + min_rest(e)
if(latest > WEEKS_TO_PLAN) then latest := WEEKS_TO_PLAN; end-if

if(sum(t in earliest..block_end, r in ALL_ROLES)(allocate_sol("current",e,r,t)) <
     1 and block_len <= max_work(e)) then
if(sum(t in block_start..block_end)(eligable(e,task_vacant,t)*(1-allocate_sol("
    fixed",e,task_vacant,t))) = block_len) then
if(rest_zero(e) < block_start) then
if((block_start > 1 and (sum(v in VESSELS)(starting(e,v)) < 1 or block_start >
    min_rest(e))) or (work_zero(e) <= max_work(e) - block_len and sum(v in
    VESSELS | task_vacant not in ROLES(v))(starting(e,v)) < 1)) then
```

```
if(latest = WEEKS_TO_PLAN or sum(t in block_end+1..latest, r in ALL_ROLES)(
    allocate_sol("fixed",e,r,t)*allocate_sol("current",e,r,t)) < 1) then

writeln("--> could assign employee ",e," - evaluate cost of doing this:")

transfer_sol_from := "current"
transfer_sol_to := "trial"
transfer_solution

forall(t in TIME) do
if(t >= block_start and t <= block_end) then
allocate_sol("trial",e,task_vacant,t) := 1
!; if(task_vacant = "Orio-01" and e = "C-45") then writeln("Set allocate sol ('
    trial',",e,",",task_vacant,",",t,") = 1"); end-if
elif(t > block_end and t <= block_end + min_rest(e)) then
forall(r in ALL_ROLES) allocate_sol("trial",e,r,t) := 0
!; if(e = "C-45") then writeln("Set allocate sol ('trial',",e,",Orio-01,",t,") =
    0"); end-if
end-if
end-do
emps_changed += {e}

to_calculate := "trial"
calculate_cost

if(candidate_exist = false) then
candidate_exist := true
candidate_emp := e
candidate_cost := total_cost("trial")
else
if(total_cost("trial") < candidate_cost) then
candidate_emp := e
candidate_cost := total_cost("trial")
end-if
end-if
end-if
end-if
end-if
end-if
end-if
end-do

if(candidate_exist = false) then
```

```
candidate_exist := true
candidate_emp := "AGENCY"
else

writeln("--> could assign AGENCY employee - evaluate cost of doing this:")

transfer_sol_from := "current"
transfer_sol_to := "trial"
transfer_solution

emps_changed := {}
roles_changed := {}
ag_roles_changed := {}

forall(t in block_start..block_end) allocate_sol("trial","AGENCY",task_vacant,t)
    := 1
ag_roles_changed += {task_vacant}

to_calculate := "trial"
calculate_cost

if(total_cost("trial") < candidate_cost) then
candidate_emp := "AGENCY"
candidate_cost := total_cost("trial")
end-if
end-if

if(candidate_exist = FALSE or candidate_emp not in ALL_EMP) then
writeln("ERROR - NO CANDIDATE FOUND")
else

emps_changed := {}
roles_changed := {}
ag_roles_changed := {}

writeln("-> Select employee ",candidate_emp," to fill vacancy")
forall(t in block_start..block_end) do
allocate_sol("current",candidate_emp,task_vacant,t) := 1
           !; if(task_vacant = "Orio-01" and candidate_emp = "C-45") then writeln
    ("Set allocate sol ('current',",candidate_emp,",",task_vacant,",",t,") = 1");
     end-if
allocate_sol("fixed",candidate_emp,task_vacant,t) := 1
           !; if(task_vacant = "Orio-01" and candidate_emp = "C-45") then writeln
    ("Set allocate sol ('fixed',",candidate_emp,",",task_vacant,",",t,") = 1");
    end-if
vacant_sol(task_vacant,t) := 0
number_vacant := number_vacant -1
end-do

roles_changed += {task_vacant}

if(candidate_emp = "AGENCY") then
ag_roles_changed += {task_vacant}
else
emps_changed += {candidate_emp}

consec_work := block_len
forall(r in ALL_ROLES) do
forall(t in TIME) do
if(t > block_end and t <= block_end + min_rest(candidate_emp)) then
if(allocate_sol("current",candidate_emp,r,t) = 1) then
allocate_sol("current",candidate_emp,r,t) := 0                         !; if(r =
     "Orio-01" and candidate_emp = "C-45") then writeln("Set allocate sol ('
    current',",candidate_emp,",",r,",",t,") = 0"); end-if
allocate_sol("fixed",candidate_emp,r,t) := 1                           !; if(r =
     "Orio-01" and candidate_emp = "C-45") then writeln("Set allocate sol ('fixed
    ',",candidate_emp,",",r,",",t,") = 0"); end-if
vacant_sol(r,t) := 1
number_vacant := number_vacant +1

roles_changed += {r}
end-if
end-if
end-do
end-do
end-if

end-if

end-if


writeln("... and now calculate cost:")
writeln
to_calculate := "current"
calculate_cost
```

```
show_updates

end-procedure


!------------------------------------------------------------------------
!------------------------------------------------------------------------

procedure find_vacancies

forall(j in role_ordered_list) do

! Look for 'blocks'  which require repairing
task_vacant := ""
vessel_vacant := ""
block_start := 0
block_end := 0
block_len := 0
block_found := FALSE

modified := false

forall(t in TIME) do
new_block := FALSE
if(modified = false and block_found = FALSE and vacant_sol(j,t) = 1) then
new_block := TRUE
task_vacant := j
block_start := t
forall(v in VESSELS, k in ROLES(v)) do
if(j = k) then
vessel_vacant := v
end-if
end-do

writeln("Start of block - role ",task_vacant," (vessel: ",vessel_vacant,") is
    vacant from time ",block_start)
writeln

!      elif(block_found = TRUE and vacant_sol(r,t) = 1) then DO NOTHING

elif(modified = false and block_found = TRUE and vacant_sol(j,t) = 0) then
block_end := t-1
```

```
block_len := (block_end - block_start) +1
repair_vacancies
modified := true

! and reset...
block_found := FALSE
block_start := 0
block_end := 0
block_len := 0
task_vacant := ""
vessel_vacant := ""

end-if

if(new_block = TRUE) then
block_found := TRUE
new_block := FALSE
end-if

! if we are at the end of the planning period, conclude any remaining blocks
if(modified = false and t = WEEKS_TO_PLAN and block_found = TRUE) then
block_end := t
block_len := (block_end - block_start) +1
repair_vacancies
modified := true

end-if
end-do

end-do

end-procedure


!-------------------------------------------------------------------------
!-------------------------------------------------------------------------

procedure check_feasibility

feasible := TRUE
```

```
if(to_check in SOL_TYPE) then

! Job Cover constraints
JCfeas := TRUE
forall(r in ALL_ROLES, t in TIME | required(r,t) > 0) do
if(sum(e in ALL_EMP)(eligable(e,r,t)*allocate_sol("eval",e,r,t)) <> required(r,t)
    ) then
JCfeas := FALSE
writeln("\tINFEASIBILITY detected - Job Cover constraint (",r,",",t,")\t",sum(e
    in ALL_EMP)(eligable(e,r,t)*allocate_sol("eval",e,r,t))," <> ",required(r,t))
end-if
end-do
if(JCfeas = FALSE) then feasible := FALSE; end-if

! Overlap constraints
if(feasible = true) then
OLfeas := TRUE
forall(e in emps_changed, t in TIME) do
if(sum(r in ALL_ROLES) allocate_sol("eval",e,r,t) > 1) then
OLfeas := FALSE
writeln("\tINFEASIBILITY detected - Overlap constraint (",e,",",t,")\t",sum(r in
    ALL_ROLES) allocate_sol("eval",e,r,t)," > 1")
end-if
end-do
if(OLfeas = FALSE) then feasible := FALSE; end-if
end-if

! Boarding constraints
if(feasible = true) then
Brdfeas := TRUE
forall(e in emps_changed, v in VESSELS) do
if(board_sol("eval",e,v,1) < sum(r in ROLES(v))(allocate_sol("eval",e,r,1)) -
    starting(e,v)) then
Brdfeas := FALSE
writeln("\tINFEASIBILITY detected - Boarding constraint (",e,",",v,",1)\t",
    board_sol("eval",e,v,1)," < ",sum(r in ROLES(v))(allocate_sol("eval",e,r,1))
    - starting(e,v))
end-if
forall(t in 2..WEEKS_TO_PLAN) do
if(board_sol("eval",e,v,t) < sum(r in ROLES(v))(allocate_sol("eval",e,r,t)) - sum
    (r in ROLES(v))(allocate_sol("eval",e,r,(t-1)))) then
Brdfeas := FALSE
```

```
writeln("\tINFEASIBILITY detected - Boarding constraint (",e,",",v,",",t,")\t",
    board_sol("eval",e,v,t)," < ",sum(r in ROLES(v))(allocate_sol("eval",e,r,t))
    - sum(r in ROLES(v))(allocate_sol("eval",e,r,(t-1))))
end-if
end-do
end-do
if(Brdfeas = FALSE) then feasible := FALSE; end-if
end-if

! Departing constraints
if(feasible = true) then
Dprtfeas := TRUE
forall(e in emps_changed, v in VESSELS) do
if(depart_sol("eval",e,v,1) < starting(e,v) - sum(r in ROLES(v))(allocate_sol("
    eval",e,r,1))) then
Dprtfeas := FALSE
writeln("\tINFEASIBILITY detected - Departing constraint (",e,",",v,",1)\t",
    depart_sol("eval",e,v,1)," < ",starting(e,v) - sum(r in ROLES(v))(
    allocate_sol("eval",e,r,1)))
end-if
forall(t in 2..WEEKS_TO_PLAN) do
if(depart_sol("eval",e,v,t) < sum(r in ROLES(v))(allocate_sol("eval",e,r,(t-1)))
    - sum(r in ROLES(v))(allocate_sol("eval",e,r,t))) then
Dprtfeas := FALSE
writeln("\tINFEASIBILITY detected - Departing constraint (",e,",",v,",",t,")\t",
    depart_sol("eval",e,v,t)," < ",sum(r in ROLES(v))(allocate_sol("eval",e,r,(t
    -1))) - sum(r in ROLES(v))(allocate_sol("eval",e,r,t)))
end-if
end-do
end-do
if(Dprtfeas = FALSE) then feasible := FALSE; end-if
end-if

! Agency board / depart constraints
if(feasible = true) then
AGBDfeas := TRUE
forall(r in ag_roles_changed) do
if(ag_rboard_sol("eval",r,1) - ag_rdepart_sol("eval",r,1) <> allocate_sol("eval
    ","AGENCY",r,1) - ag_starting(r)) then
AGBDfeas := FALSE
writeln("\tINFEASIBILITY detected - AG board/depart constraint (",r,",1)\t",
    ag_rboard_sol("eval",r,1) - ag_rdepart_sol("eval",r,1)," <> ",allocate_sol("
    eval","AGENCY",r,1) - ag_starting(r))
```

```
end-if
forall(t in 2..WEEKS_TO_PLAN) do
if(ag_rboard_sol("eval",r,t) - ag_rdepart_sol("eval",r,t) <> allocate_sol("eval
    ","AGENCY",r,t) - allocate_sol("eval","AGENCY",r,(t-1))) then
AGBDfeas := FALSE
writeln("\tINFEASIBILITY detected - AG board/depart constraint (",r,",",t,")\t",
    ag_rboard_sol("eval",r,t) - ag_rdepart_sol("eval",r,t)," <> ",allocate_sol("
    eval","AGENCY",r,t) - allocate_sol("eval","AGENCY",r,(t-1)))
end-if
end-do
end-do
if(AGBDfeas = FALSE) then feasible := FALSE; end-if
end-if

! Undertime constraints
if(feasible = true) then
UTfeas := TRUE
forall(e in emps_changed | e in GUARANTEED) do
if(undertime_sol("eval",e) < g_weeks(e) - (exp_worktime(e) + sum(r in ALL_ROLES,
    t in TIME)(allocate_sol("eval",e,r,t)))) then
UTfeas := FALSE
writeln("\tINFEASIBILITY detected - Undertime constraint (",e,")\t",undertime_sol
    ("eval",e)," < ",g_weeks(e) - (exp_worktime(e) + sum(r in ALL_ROLES, t in
    TIME)(allocate_sol("eval",e,r,t))))
end-if
end-do
if(UTfeas = FALSE) then feasible := FALSE; end-if
end-if

! Overtime constraints
if(feasible = true) then
OTfeas := TRUE
forall(e in emps_changed | e in GUARANTEED) do
if(overtime_sol("eval",e) < (exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(
    allocate_sol("eval",e,r,t)))- g_weeks(e)) then
OTfeas := FALSE
writeln("\tINFEASIBILITY detected - Overtime constraint (",e,")\t",overtime_sol("
    eval",e)," < ",(exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(allocate_sol
    ("eval",e,r,t)))- g_weeks(e))
end-if
end-do
if(OTfeas = FALSE) then feasible := FALSE; end-if
end-if
```

```
! Long Work constraints
if(feasible = true) then
! First, calculate work resource values:
forall(e in emps_changed) do
work_total(e,1) := work_zero(e) + sum(r in ALL_ROLES)(allocate_sol("eval",e,r,1))
     - max_work(e)*(1-(sum(r in ALL_ROLES)(allocate_sol("eval",e,r,1))))
if(work_total(e,1) < 0) then
work_total(e,1) := 0
end-if
forall(t in 2..WEEKS_TO_PLAN) do
work_total(e,t) := work_total(e,(t-1)) + sum(r in ALL_ROLES)(allocate_sol("eval",
    e,r,t)) - max_work(e)*(1-(sum(r in ALL_ROLES)(allocate_sol("eval",e,r,t))))
if(work_total(e,t) < 0) then
work_total(e,t) := 0
end-if
end-do
end-do

LWfeas := TRUE
forall(l in lambda, e in emps_changed, r in ALL_ROLES | exists(long_work_sol("
    eval",l,e,r,1))) do
if(max_work(e)*long_work_sol("eval",l,e,r,1) < work_zero(e) - max_work(e)*(1-
    allocate_sol("eval",e,r,1)) + allocate_sol("eval",e,r,1) - (l-1)) then
LWfeas := FALSE
!              writeln("\tINFEASIBILITY detected - Long Work constraint (",l,",",e
    ,",",r,",1)\t",max_work(e)*long_work_sol("eval",l,e,r,1)," < ",work_zero(e) +
     allocate_sol("eval",e,r,1) - (l-1))
writeln("\tINFEASIBILITY detected - Long Work constraint (",l,",",e,",",r,",1)\t
    ",max_work(e),"*",long_work_sol("eval",l,e,r,1)," < ",work_zero(e)," - ",
    max_work(e),"*",(1-allocate_sol("eval",e,r,1))," + ",allocate_sol("eval",e,r
    ,1)," - (",l,"-1)")
end-if
forall(t in 2..WEEKS_TO_PLAN) do
if(max_work(e)*long_work_sol("eval",l,e,r,t) < work_total(e,(t-1)) - max_work(e)
    *(1-allocate_sol("eval",e,r,t)) + allocate_sol("eval",e,r,t) - (l-1)) then
LWfeas := FALSE
!                     writeln("\tINFEASIBILITY detected - Long Work constraint (",
    l,",",e,",",r,",",t,")\t",max_work(e)*long_work_sol("eval",l,e,r,t)," < ",
    work_total(e,(t-1)) + allocate_sol("eval",e,r,t) - (l-1))
writeln("\tINFEASIBILITY detected - Long Work constraint (",l,",",e,",",r,",",t
    ,")\t",max_work(e),"*",long_work_sol("eval",l,e,r,t)," < ",work_total(e,(t-1)
    )," - ",max_work(e),"*",(1-allocate_sol("eval",e,r,t))," + ",allocate_sol("
    eval",e,r,t)," - (",l,"-1)")
end-if
end-do
end-do
if(LWfeas = FALSE) then feasible := FALSE; end-if
end-if

! Agency Long Work constraints
if(feasible = true) then
! First, calculate Agency work resource values
forall(r in ag_roles_changed) do
ag_work_total(r,1) := ag_work_zero(r) + allocate_sol("eval","AGENCY",r,1) -
    ag_max_work(r)*ag_rdepart_sol("eval",r,1)
if(ag_work_total(r,1) < allocate_sol("eval","AGENCY",r,1)) then
ag_work_total(r,1) := allocate_sol("eval","AGENCY",r,1)
end-if
forall(t in 2..WEEKS_TO_PLAN) do
ag_work_total(r,t) := ag_work_total(r,(t-1)) + allocate_sol("eval","AGENCY",r,t)
    - ag_max_work(r)*ag_rdepart_sol("eval",r,t)
if(ag_work_total(r,t) < allocate_sol("eval","AGENCY",r,t)) then
ag_work_total(r,t) := allocate_sol("eval","AGENCY",r,t)
end-if
end-do
end-do

AGLWfeas := TRUE
forall(l in lambda, r in ag_roles_changed, t in TIME | exists(long_work_sol("eval
    ",l,"AGENCY",r,t))) do
if(ag_max_work(r)*long_work_sol("eval",l,"AGENCY",r,t) < ag_work_total(r,t) - (l
    -1)) then
AGLWfeas := FALSE
writeln("\tINFEASIBILITY detected - Agency Long Work constraint (",l,",",r,",",t
    ,")\t",ag_max_work(r)*long_work_sol("eval",l,"AGENCY",r,t)," < ",
    ag_work_total(r,t) - (l-1))
end-if
end-do
if(AGLWfeas = FALSE) then feasible := FALSE; end-if
end-if

! Rest vs Work constraints
if(feasible = true) then
! First, calculate rest resource values
forall(e in emps_changed) do
```

```
rest_total(e,1) := rest_zero(e) - (1-(sum(r in ALL_ROLES)(allocate_sol("eval",e,r
    ,1))))                      !;if(e = "C-37") then writeln("rest_total(",e,",1)
     := ",rest_zero(e)," - (1-",(sum(r in ALL_ROLES)(allocate(e,r,1))),")"); end-
    if
if(rest_total(e,1) < (min_rest(e)-1)*(sum(v in VESSELS)(depart_sol("eval",e,v,1))
    )) then
rest_total(e,1) := (min_rest(e)-1)*(sum(v in VESSELS)(depart_sol("eval",e,v,1)))
                                !;if(e = "C-37") then writeln("rest_total(",e,",1)
     < (",min_rest(e),"-1)*",(sum(v in VESSELS)(depart(e,v,1)))," => reset, so
    rest_total(",e,",1) := (",min_rest(e),"-1)*",(sum(v in VESSELS)(depart(e,v,1)
    ))); end-if
end-if
forall(t in 2..WEEKS_TO_PLAN) do
rest_total(e,t) := rest_total(e,(t-1)) - (1-(sum(r in ALL_ROLES)(allocate_sol("
    eval",e,r,t))))  !;if(e = "C-37") then writeln("rest_total(",e,",",t,") := ",
    rest_total(e,(t-1))," - (1-",(sum(r in ALL_ROLES)(allocate(e,r,t))),")"); end
    -if
if(rest_total(e,t) < (min_rest(e)-1)*(sum(v in VESSELS)(depart_sol("eval",e,v,t))
    )) then
rest_total(e,t) := (min_rest(e)-1)*(sum(v in VESSELS)(depart_sol("eval",e,v,t)))
                        !;if(e = "C-37") then writeln("rest_total(",e,",",t,") <
    (",min_rest(e),"-1)*",(sum(v in VESSELS)(depart(e,v,t)))," => reset, so
    rest_total(",e,",",t,") := (",min_rest(e),"-1)*",(sum(v in VESSELS)(depart(e,
    v,t)))); end-if
end-if
end-do
end-do

RvWfeas := TRUE
forall(e in emps_changed) do
if(min_rest(e)*(1-(sum(r in ALL_ROLES)(allocate_sol("eval",e,r,1)))) < rest_zero(
    e)) then
RvWfeas := FALSE
writeln("\tINFEASIBILITY detected - Rest vs Work constraint (",e,",1)\t",min_rest
    (e)*(1-(sum(r in ALL_ROLES)(allocate_sol("eval",e,r,1))))," < ",rest_zero(e))
end-if
forall(t in 2..WEEKS_TO_PLAN) do
if(min_rest(e)*(1-(sum(r in ALL_ROLES)(allocate_sol("eval",e,r,t)))) < rest_total
    (e,(t-1))) then
RvWfeas := FALSE
!                       writeln("\tINFEASIBILITY detected - Rest vs Work constraint
    (",e,",",t,")\t",min_rest(e)*(1-(sum(r in ALL_ROLES)(allocate(e,r,t))))," <
    ",rest_total(e,(t-1)))
```

```
writeln("\tINFEASIBILITY detected - Rest vs Work constraint (",e,",",t,")\t",
    min_rest(e),"*(1-",(sum(r in ALL_ROLES)(allocate_sol("eval",e,r,t))),") < ",
    rest_total(e,(t-1)))
end-if
end-do
end-do
if(RvWfeas = FALSE) then feasible := FALSE; end-if
end-if

! Variable linking constraints
if(feasible = true) then
Linkfeas := TRUE
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME | e in emps_changed or (e = "
    AGENCY" and r in ag_roles_changed)) do
if(cur_allocate(e,r,t) = 0) then
if(chng_allocate("eval",e,r,t) <> allocate_sol("eval",e,r,t)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of allocate variables (",e,",",r,",",t
    ,")")
end-if
else
if(chng_allocate("eval",e,r,t) <> cur_allocate(e,r,t) - allocate_sol("eval",e,r,t
    )) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of allocate variables (",e,",",r,",",t
    ,")")
end-if
end-if
end-do
forall(e in emps_changed, v in VESSELS, t in TIME) do
if(cur_board(e,v,t) = 0) then
if(chng_board("eval",e,v,t) <> board_sol("eval",e,v,t)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of board variables (",e,",",v,",",t,")")
end-if
else
if(chng_board("eval",e,v,t) <> cur_board(e,v,t) - board_sol("eval",e,v,t)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of board variables (",e,",",v,",",t,")")
end-if
end-if
end-do
forall(e in emps_changed, v in VESSELS, t in TIME) do
```

```
if(cur_depart(e,v,t) = 0) then
if(chng_depart("eval",e,v,t) <> depart_sol("eval",e,v,t)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of depart variables (",e,",",v,",",t,")
    ")
end-if
else
if(chng_depart("eval",e,v,t) <> cur_depart(e,v,t) - depart_sol("eval",e,v,t))
    then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of depart variables (",e,",",v,",",t,")
    ")
end-if
end-if
end-do
forall(r in ag_roles_changed, t in TIME) do
if(cur_ag_rboard(r,t) = 0) then
if(chng_ag_rboard("eval",r,t) <> ag_rboard_sol("eval",r,t)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of AG board variables (",r,",",t,")")
end-if
else
if(chng_ag_rboard("eval",r,t) <> cur_ag_rboard(r,t) - ag_rboard_sol("eval",r,t))
    then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of AG board variables (",r,",",t,")")
end-if
end-if
end-do
forall(r in ag_roles_changed, t in TIME) do
if(cur_ag_rdepart(r,t) = 0) then
if(chng_ag_rdepart("eval",r,t) <> ag_rdepart_sol("eval",r,t)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of AG depart variables (",r,",",t,")")
end-if
else
if(chng_ag_rdepart("eval",r,t) <> cur_ag_rdepart(r,t) - ag_rdepart_sol("eval",r,t
    )) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of AG depart variables (",r,",",t,")")
end-if
end-if
end-do
```

```
forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME | e in emps_changed
    or (e = "AGENCY" and r in ag_roles_changed)) do
if(cur_long_work(l,e,r,t) = 0) then
if(chng_long_work("eval",l,e,r,t) <> long_work_sol("eval",l,e,r,t)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of long work variables (",l,",",e,",",r
    ,",",t,")")
end-if
else
if(chng_long_work("eval",l,e,r,t) <> cur_long_work(l,e,r,t) - long_work_sol("eval
    ",l,e,r,t)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of long work variables (",l,",",e,",",r
    ,",",t,")")
end-if
end-if
end-do
forall(e in emps_changed | e in GUARANTEED) do
if(chng_undertime("eval",e) <> undertime_sol("eval",e) - cur_undertime(e)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of undertime variables (",e,")")
end-if
if(chng_overtime("eval",e) <> overtime_sol("eval",e) - cur_overtime(e)) then
Linkfeas := FALSE
writeln("\tINFEASIBILITY detected - Link of overtime variables (",e,")")
end-if
end-do
if(Linkfeas = FALSE) then feasible := FALSE; end-if
end-if

! Status of variables
if(feasible = true) then
Statfeas := TRUE
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME | e in emps_changed or (e = "
    AGENCY" and r in ag_roles_changed)) do
if(allocate_sol("eval",e,r,t) <> 0 and allocate_sol("eval",e,r,t) <> 1) then
    Statfeas := FALSE; end-if
if(chng_allocate("eval",e,r,t) <> 0 and chng_allocate("eval",e,r,t) <> 1) then
    Statfeas := FALSE; end-if
forall(l in lambda) do
if(long_work_sol("eval",l,e,r,t) <> 0 and long_work_sol("eval",l,e,r,t) <> 1)
    then Statfeas := FALSE; end-if
```

```
if(chng_long_work("eval",l,e,r,t) <> 0 and chng_long_work("eval",l,e,r,t) <> 1)
    then Statfeas := FALSE; end-if
end-do
end-do
forall(e in emps_changed, v in VESSELS, t in TIME) do
if(board_sol("eval",e,v,t) <> 0 and board_sol("eval",e,v,t) <> 1) then Statfeas
    := FALSE; end-if
if(chng_board("eval",e,v,t) <> 0 and chng_board("eval",e,v,t) <> 1) then Statfeas
     := FALSE; end-if
if(depart_sol("eval",e,v,t) <> 0 and depart_sol("eval",e,v,t) <> 1) then Statfeas
     := FALSE; end-if
if(chng_depart("eval",e,v,t) <> 0 and chng_depart("eval",e,v,t) <> 1) then
    Statfeas := FALSE; end-if
end-do
forall(r in ag_roles_changed, t in TIME) do
if(ag_rboard_sol("eval",r,t) <> 0 and ag_rboard_sol("eval",r,t) <> 1) then
    Statfeas := FALSE; end-if
if(chng_ag_rboard("eval",r,t) <> 0 and chng_ag_rboard("eval",r,t) <> 1) then
    Statfeas := FALSE; end-if
if(chng_ag_rdepart("eval",r,t) <> 0 and chng_ag_rdepart("eval",r,t) <> 1) then
    Statfeas := FALSE; end-if
if(ag_rdepart_sol("eval",r,t) <> 0 and ag_rdepart_sol("eval",r,t) <> 1) then
    Statfeas := FALSE; end-if
end-do
forall(e in emps_changed | e in GUARANTEED) do
if(undertime_sol("eval",e) < 0) then Statfeas := FALSE; end-if
if(overtime_sol("eval",e) < 0) then Statfeas := FALSE; end-if
end-do
forall(e in emps_changed, t in TIME) do
if(work_total(e,t) < 0) then Statfeas := FALSE; end-if
if(rest_total(e,t) < 0) then Statfeas := FALSE; end-if
end-do
if(Statfeas = FALSE) then
writeln("\tINFEASIBILITY detected - status of variables (unspecified)")
feasible := FALSE
end-if
end-if


if(feasible = FALSE) then
writeln("Error detected - the solution evaluated is infeasible")
!              no_infeas := no_infeas +1
else
```

```
writeln("(Suggested solution is feasible)")
end-if



else
writeln("ERROR - incorrect option selected for evaluation")
end-if

end-procedure



!-----------------------------------------------------------------------
!-----------------------------------------------------------------------

procedure sort_employee_list

forall(e in REG_EMP) emp_order_number(e) := random
emp_ordered_list := []
emp_added_set := {}

forall(x in REG_EMP) do
min_no := 1
min_emp := ""

forall(e in REG_EMP | e not in emp_added_set) do
if(emp_order_number(e) < min_no) then
min_no := emp_order_number(e)
min_emp := e
end-if
end-do

emp_ordered_list += [min_emp]
emp_added_set += {min_emp}
end-do

writeln("Employee list sorted as follows: ",emp_ordered_list)
writeln

end-procedure



!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
```

```
procedure sort_role_list

forall(r in ALL_ROLES) role_order_number(r) := random
role_ordered_list := []
role_added_set := {}

forall(x in ALL_ROLES) do
min_no := 1
min_role := ""

forall(r in ALL_ROLES | r not in role_added_set) do
if(role_order_number(r) < min_no) then
min_no := role_order_number(r)
min_role := r
end-if
end-do

role_ordered_list += [min_role]
role_added_set += {min_role}
end-do

writeln("Role list sorted as follows: ",role_ordered_list)
writeln

end-procedure


!-------------------------------------------------------------------------
!-------------------------------------------------------------------------
!
! MAIN PROGRAMME

fopen(OUTPUTFILE, F_OUTPUT)

run_starttime := gettime

writeln

initialise

writeln("Calculate cost of solution at start:")
writeln
```

```
to_calculate := "initial"
calculate_cost

transfer_sol_from := "initial"
transfer_sol_to := "current"
transfer_solution

initial_feas_check

sort_employee_list
sort_role_list

writeln
writeln("------------------------------------------------------")
writeln

iteration := 0
while(number_vacant > 0 and iteration <= 20) do
find_vacancies
writeln("--------------------------------------------------")
writeln("--------------------------------------------------")
writeln("             END OF ITERATION ",iteration)
writeln("--------------------------------------------------")
writeln("--------------------------------------------------")
writeln
iteration := iteration + 1
end-do

! FINAL SOLUTION
writeln
writeln("FINAL SOLUTION IS AS FOLLOWS:")
forall(t in TIME) write("\tWeek ",t)
write("\n")
forall(r in ALL_ROLES) do
write(r)
forall(t in TIME) do
if(required(r,t) = 0) then
write("\t (n/a)")
else
covered := FALSE
forall(e in ALL_EMP) do
if(allocate_sol("current",e,r,t) = 1) then
write("\t",e)
```

```
covered := TRUE
end-if
end-do
if(covered = FALSE) then write("\t*TBC*"); end-if
end-if
end-do
write("\n")
end-do
writeln
writeln

writeln("... with crew assigned to roles as follows:")
forall(t in TIME) write("\tWeek ",t)
write("\n")
forall(e in REG_EMP) do
write(e)
forall(t in TIME) do
working := FALSE
forall(r in ALL_ROLES) do
if(allocate_sol("current",e,r,t) = 1) then
write("\t",r)
working := TRUE
end-if
end-do
if(working = FALSE) then
avail := FALSE
forall(r in ALL_ROLES) do
if(eligable(e,r,t) = 1) then avail := TRUE
end-if
end-do
if(avail = FALSE) then write("\t*unav*")
else write("\t")
end-if
end-if
end-do
write("\n")
end-do
writeln
writeln
writeln("And the following assigned to agency:")
agency_used := false
forall(r in ALL_ROLES) do
if(sum(t in TIME) (allocate_sol("current","AGENCY",r,t)) > 0) then
```

```
agency_used := true
write("Role ",r," in week(s): ")
forall(t in TIME) do
if(allocate_sol("current","AGENCY",r,t) = 1) then write(t," "); end-if
end-do
write("\n")
end-if
end-do
if(agency_used = false) then writeln("\t(none)"); end-if
writeln
writeln


! ... and check feasibility:
forall(e in REG_EMP) emps_changed += {e}
forall(r in ALL_ROLES) ag_roles_changed += {r}
to_check := "current"
check_feasibility

writeln
writeln("------------------------------------------------------")
writeln

run_endtime := gettime

final_cost := total_cost("current")
final_changes := total_changes("current")
final_vacant := number_vacant
final_iters := iteration
final_time := run_endtime - run_starttime

if(feasible = true) then final_feasible := "YES"
else final_feasible := "NO"
end-if

forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) final_solution(e,r,t) :=
    allocate_sol("current",e,r,t)

writeln("Solution cost:  ",final_cost)
writeln("No of changes:  ",final_changes)
writeln("No of vacancies: ",final_vacant)
writeln
writeln("No of iterations: ",final_iters)
writeln("Running time:   ",final_time)

prog_endtime := gettime
prog_runtime := prog_endtime - prog_starttime
writeln
writeln("Total running time: ",prog_runtime)
fclose(F_OUTPUT)

fopen(SUMMARYFILE, F_APPEND)
write(InstanceName)
write("\t",prog_runtime)
write("\t",final_cost,"\t",final_changes)
write("\t",final_iters,"\t",final_time)
write("\t",final_vacant,"\t",final_feasible)
write("\n")
fclose(F_APPEND)



! Print solution to a solution file, so it can be used as an initial solution for
     the heuristic procudure
fopen(SOLUTIONFILE, F_OUTPUT)
writeln("!Solution for ",InstanceName)
writeln("!\tfound using a the Heuristic Initial solution algorithm and")
writeln("!\twritten in representation proposed for carrying out heuristics.")
writeln
writeln("!Feasible for TW? ",final_feasible)
writeln("!Total run time: ",final_time)
writeln("!Solution value: ",final_cost)
writeln
write("!Roles are given in the following order:")
forall(r in ALL_ROLES) write("\t",r)
write("\n")
writeln
write("! Time: ")
forall(r in ALL_ROLES) do
forall(t in TIME) write(t,"\t")
end-do
write("\n")
writeln("heur_init_sol: [")
forall(e in REG_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) do
if(allocate_sol("current",e,r,t) > 0.9) then
```

```
write("\t1")
else
write("\t0")
end-if
end-do
end-do
write("\n")
end-do
forall(r in ALL_ROLES) do
forall(t in TIME) do
if(allocate_sol("current","AGENCY",r,t) > 0.9) then
write("\t1")
else
write("\t0")
end-if
end-do
end-do
write(" ]\n")
fclose(F_OUTPUT)


fopen(COLLATEDSOLUTIONS, F_OUTPUT)
writeln("!Solution for ",InstanceName)
writeln("!\tfound using a the Heuristic Initial solution algorithm and")
writeln("!\twritten in representation proposed for carrying out heuristics.")
writeln
writeln("!Feasible for TW? ",final_feasible)
writeln("!Total run time: ",final_time)
writeln("!Solution value: ",final_cost)
writeln
write("!Roles are given in the following order:")
forall(r in ALL_ROLES) write("\t",r)
write("\n")
writeln
write("! Time: ")
forall(r in ALL_ROLES) do
forall(t in TIME) write(t,"\t")
end-do
write("\n")
writeln("heur_init_sol: [")
forall(e in REG_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) do
```

```
if(allocate_sol("current",e,r,t) > 0.9) then
write("\t1")
else
write("\t0")
end-if
end-do
end-do
write("\n")
end-do
forall(r in ALL_ROLES) do
forall(t in TIME) do
if(allocate_sol("current","AGENCY",r,t) > 0.9) then
write("\t1")
else
write("\t0")
end-if
end-do
end-do
write(" ]\n")
fclose(F_OUTPUT)


end-model
```