Solving Crew Scheduling Problem in Offshore Supply Vessels: Heuristics and Decomposition Methods

Thesis for the degree of Doctor of Philosophy

Seda Sucu
Department of Management Science, University of Strathclyde

January 4, 2018

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

This dissertation is submitted to The University of Strathclyde, in accordance with the requirements for the degree of Doctor of Philosophy in the Strathclyde Business School. Parts of this dissertation are the results of collaboration with Dr Alexander Leggate, Dr Kerem Akartunali and Dr Robert Van Der Meer. I declare that I have made a substantial contribution to this work and this dissertation is composed by myself. This dissertation has not been submitted to any other university or higher education institution, or for any other academic award in this university. Where use has been made of other people's work, it has been fully acknowledged and referenced. The papers contained in this dissertation, or upon which this dissertation is based, have either been published in, or been submitted for publication to, as indicated, at reviewed journals or conferences. The papers have not been edited except to fix typographical or spelling errors and to update references. They have, however, been reformatted for this dissertation and thus floating objects, such as figures and tables, may have moved about with respect to their surrounding text.

- Paper I A. Leggate, Sucu, S., Akartunalı, K., Van Der Meer, R., Modelling Crew Scheduling in Off-Shore Supply Vessels, *Journal of the Operational Research Society, doi:10.1080/01605682.2017.1390531.*

- Paper II S. Sucu, Leggate, A., Akartunalı, K., Van Der Meer, R., Modeling Uncertainty in Vessel Crew Scheduling, *Proceedings of the 7th Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2015, pages 805-808, Prague, Czech Republic, 2015.*

I will provide more details concerning joint work. In Section 4.1 (Paper I), Dr Leggate proposed the original idea of modelling, I extended the computational study and analysis. In Section 4.2.2, the motivation comes from Dr Leggate, who first suggested to consider the Time Windows model. We revised together the mathematical model and heuristics in Section 4.3. I implemented the related software and computational study for the heuristic on my own. I have written the extended abstract (Paper II). The Chapter 5 and 6 is based on solely my effort. In particular modification of the problem structure was suggested by me. I also identified the motivation problems and I implemented the software for running the experiments.

Signed: Seda Sucu

Date: October, 2017

Abstract

For the efficient utilisation of resources in various transportation settings, scheduling is a significant area of research. Having crew as the main resource for operation maintenance, scheduling crew have been a powerful decision making tool for optimisation studies. This research provides a detailed real case study analysis regarding the difficulties in planning crew in maritime industry. As a special case study, this thesis researches crew scheduling in offshore supply vessels which are used for specific operations of a global scaled company in oil and gas industry deeply with modified formulations, heuristics and decomposition methods.

An extended version of computational study for a simple formulation approach (Task Based Model) is applied as deeper analysis to Leggate (2016). Afterwards, more realistic approach to the same problem is revised. Following the revision, a customized and thorough computational study on the heuristic method with various settings is designed and implemented in C++.

After elaborated analysis completed on the suggested models firstly, a modification on Time Windows model is presented to increase the efficacy. This modification provides a sharp decrease in upper bounds within a short time compared to the previously suggested models. Through this suggestion, more economic schedules within a short period of time are generated.

Achieving high performances from the modified model, an application of a decomposition algorithm is provided. We implemented a hybrid solution of Benders Decomposition with a customized heuristic for the modified model. Although this hybrid solution does not provide high quality solutions, it evaluates the performance of possible decomposed models with potential improvements for future research.

An introduction to robust crew scheduling in maritime context is also given with a description of resources of uncertainty in this concept and initial robust formulations are suggested.

To my family...

Acknowledgements

I would first of all like to acknowledge and thank my supervisors Kerem Akartunali and Robert Van Der Meer whose comments helped to improve the content and presentation of my thesis. I am grateful to Kerem for his support and motivation throughout my Ph.D. study and for his guidance in this study and academic life.

I would like to express my gratitude to my reviewers Roberto Rossi and Ashwin Arulselwan for kindly accepting to be examiners of the examining committee, and for providing valuable feedback.

I would like to thank Ioannis Fragkos for fruitful discussion. I would also like to acknowledge my friends for making my experience in PhD joyful. I would like to thank to my family, who were there throughout it all.

This PhD was possible only through a PhD scholarship offered by the University of Strathclyde, which is also generously part-funded by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA9550-14-1-0203. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purpose notwithstanding any copyright notation thereon.

Contents

Chapter 1

Introduction

Scheduling is a field that concerns the careful assignment of limited resources to the most deserving jobs and activities in order to reach the highest efficiency in terms of these limited resources. These resources can be time, money, labours etc.The basic areas that use scheduling methods to increase the efficiency and assign limited resources to get optimal productivity are manufacturing systems, services, information technologies, hospitals, and transportation systems. The application of scheduling problems show differences in terms of our objectives and resource limitations. Apart from the different sources of limitations, the main target of scheduling may show variations. The objective functions for scheduling problems change according to what we schedule and for which system we need schedules. They can vary in terms of the resources, constraints and the field of scheduling. It can be maximising the throughput in a manufacturing system, minimising the completion times of production, minimising the delay times of the vehicle in a transportation system or minimising production and staff costs. Even in some studies, the problem does not hold any objective function at all, the objective becomes finding a feasible schedule that meets the constraints of the system.

To increase productivity, efficient utilisation of manpower is one of the main concerns of organizations. Solution methods for these problems often need large solution times to reach optimality, especially in large sized real life problems. Due to the importance of staff (crew) scheduling and complexity of these problems, staff scheduling has a remarkable place in the literature of scheduling problems (Cai and Li, 2000).

This thesis aims to find solution methods for cost minimisation problem based on

the changes in scheduling of vessel crews and quantify the effect of various problem parameters with a computational study. To improve the existing solution methods which are addressed in Leggate (2016), following the thorough literature review in the area, various optimisation tools are employed. The centre of this research is grounded in staff scheduling which is an important field of optimisation.

In this study, the main concern is minimising cost in a special type of vessels based on the changes and drawbacks of schedules depending on the crew in these vessels. This objective directs this research to the crew scheduling problems. In crew scheduling, it is aimed to obtain work timetables (schedules) for an organization to meet the demands of the required number of staff with eligibility the same time considering the regulations and rules in this organization, (Ernst et al., 2004b). 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 numerous exact solution methods to approximation algorithms and heuristics, (Ernst et al., 2004a,b). However, all these cases display a number of similarities common to crew scheduling problems in any given setting.

The basic input of the most crew scheduling problems is set of crew and the set of tasks that should be carried out according to the definition of tasks and skill level of employees. The common features of these problems can be stated like that the tasks should be completed in a defined time window, in an actual task station by taking into consideration the legal and contractual requirements, as well as physical constraints imposed by the geographical and temporal aspects of the tasks through the actual objective function. According to these characteristics, the solution methods search for the best allocation of staffs in general.

The discussion why the crew scheduling problems are important for this study begins with the huge effect of crew costs on the transportation industry. As the nature of enhancing the capability of current systems in a company, lowering the costs as much as possible is the proper action to take. In this sense, even small savings in the crew schedules can have an important effect on the actual cost.

Apart from the important effect of the crew costs, crew scheduling problems are inherently intractable problems and they have gained importance because of this feature. To obtain an optimal solution with a solver in a reasonable time is not generally possible for the actual size of these problems. This characteristic increases

the complexity level of this kind of problems. The studies on employee assignment with deciding the work and rest period with various settings are known as NP-Hard and NP-complete problem (Di Gaspero et al., 2007; Kyngäs et al., 2012). Reaching the optimal solution takes a long time since most of the decision variables are binary in these problems. This situation can be challenging for the companies, especially which need to obtain schedules in a definite time interval. If the company can not reach the optimal solution in a required time, then the quality of the solution may decrease. It means there is a trade off between time and the quality of solutions for the NP-Hard problems which can not be solved optimally in polynomial time (Garey and Johnson, 2002). Due to this trade off, these problems are extensively studied in the optimisation community. There are so many studies for developing new solution methods apart from classical mixed integer programming which can find better solutions in shorter time.

Even though the crew scheduling problems in the transportation industry have a significant place in literature, the most popular type among all other transportation settings is the airline crew scheduling. However, there have also been spectacular publications in the context of train and bus crew scheduling (reviews by (Ernst et al., 2004a), (Ernst et al., 2004b) and more recently (Van den Bergh et al., 2013) discuss a variety of application areas). This case is not valid for vessel crew scheduling problems. There are several reasons to explain the lack of studies for crew scheduling problems in vessels. These reasons can be listed as long planning time horizons in the maritime context, lower visibility of shipping industry compared to the rail, road and air, and the higher level of uncertainty and the requirements of more recovery schedules (Christiansen et al., 2007).

Another key point about this research is to consider the vessel crew scheduling problem with robustness. Robust optimisation seeks a solution under uncertainty. Data in our problem is also subject to uncertainty in a certain extent such as crew availability, demand changes and cost fluctuations.

In this study, since the crew is the main subject, the uncertainty is inevitable. Intuitively, people (crew) are the most important factor that affects the cost. This fact prevents having deterministic parameters. Due to the fact that human behaviour cannot be known exactly all the time, some non-deterministic perceptions are needed for drawing results as near to the reality as possible.Moreover, there may be environmental factors, primarily the weather conditions, that can lead to distortion. In addition to such factors on the supply side, uncertainties in demand may also cause

disruption in crew planning. For example, financial environment of vessel companies and related industries such as oil and gas or offshore energy might lead to variation in the cost of the crew.

Accordingly, rescheduling the crew is mostly necessary to decrease the harm of disruptions. Known as the crew recovery problem, 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).

In Section 1.1, the introduction of Offshore Supply Vessel concept will be discussed to provide a deeper understanding of the possible sources regarding the issues related to the current concept. After explaining the reasons related to the crew scheduling in this context, the outline of thesis follows.

1.1 Offshore Supply Vessels

There are several modes of operations in sea transportation, which can be grouped into three categories of Liner, Tramp and Industrial shipping (Christiansen et al., 2013). We note another category not suitable for this classification is the Offshore Supply (or Service) Vessel (OSV). The operations carried on offshore supply vessels are mostly in the interest of oil companies (Barret, 2005). In other words, while providing equipment to the construction teams located in the oceans, offshore vessels also help companies in oil exploration and drilling. In addition, offshore vessels can provide the transportation and relocation of crewing personnel to and from the operational locations in the high seas when it is necessary.

In addition to the oil and gas industry, an important area which requires the OSVs for operation maintenance is Offshore wind farms that recently broaden their sites offshore in deeper water especially in Europe, (Barlow et al., 2017).

Offshore vessels can be mainly classified into four main categories: Offshore support vessels, offshore production vessels, construction vessels and oil exploration, drilling vessels (Chopra, 2017). Among these four, offshore support vessels are the ones we focus on in our study. Offshore support vessels are the ones which supply the necessary manpower and technical support in order to let companies have unin-

terrupted operational processes in the high seas. Offshore supply vessels' main duty is to transport the necessary structural elements to the specific sectors in high seas as well as giving assistance to supply freight. These vessels can be built to supply the operational demands of the companies. In terms of their capabilities, these ships generally carry different types of common and speciality tools on their decks and most of them have a mixture of bulk and deck cargo under their deck.

As a result of the significant growth of the offshore oil and gas industries, a huge demand for offshore support vessels (OSV) exists to carry out the various operations. Oil companies usually charter offshore supply vessels rather than buying them. Hence, there is a need for logistics planning for the use of these vessels (Aas et al., 2009). In Section 1.2, we elaborate the importance of crew scheduling in the OSVs.

1.2 Impact of Crew Scheduling on Offshore Supply Vessels

Logistic planning of Offshore Supply Vessels has a highly complex structure due to the complexity of offshore operations, long planning horizons and high uncertainty in related sectors which require offshore maintenance operations. The problems related to this kind of logistic planning are known as vehicle routing problem in the literature. Despite the necessity and importance of planning OSVs, there are not enough number of studies dealing with this kind of problems in maritime settings by using optimisation and simulation methods as it is used in general.

Most of the studies in OSV planning are related to the optimisation of fleet assignment, fleet design and ship routing. To the best of our knowledge, there is no study on crew scheduling as a part of logistic planning in OSVs.

The crew cost for this transportation cost problem depends on different variables. When we investigate the crew cost in the maritime industry particularly, it can be seen that there are so many variables that affect crew costs such as salary of the crew, house and feed expenses as well as transportation cost of them. Transportation cost includes the travel expenses from a gateway city near crew members' home to the departure port of the ship, airfare, visa expenses, hotel, meals, and crew return expenses after their duty. Accordingly, the crew cost has a comparatively significant effect on the shipping companies and leading even $2-3\%$ decrease with better crew

planning saves hundreds of thousands of dollars for global companies (Giachetti et al., 2013).

Apart from the importance of the crew costs, completing the required operation on time by the workers with the sufficient skill level has high importance for the maintenance as well. Hence, the companies carry on offshore operations, have large scale business and require more complex planning, to implement feasible schedules is not a simple job.

Crew scheduling problems are inherently intractable and this feature makes them attractive as optimisation problems. To obtain an optimal solution with a solver in a reasonable time is not generally possible for problems of a realistic size. Most of the crew scheduling problems are known to be NP-Hard problems; as the size of the problem becomes larger, the complexity level of this kind of problems also increases. Even though crew scheduling problems in the transportation industry have a significant place in the scheduling literature, maritime crew scheduling problems are not as popular as airline settings. There are several reasons to explain the lack of studies for crew scheduling problems in vessels. These reasons can be explained by the long planning time horizons in the maritime context, low visibility of the shipping industry compared to rail, road and airline setting.

1.3 Research Motivation and Main Contributions

As the significant impact of crew scheduling within the transformation and logistics context is aforementioned, there is a considerable amount of studies published in transportation settings such as airline (Kasirzadeh et al., 2017), railway (Chen and Niu, 2012) and urban buses (Öztop et al., 2017). However, there is a limited number of studies on maritime context. The main motivation behind our research can be explained with the importance of crew planning in the maintenance of the offshore operations, the huge effect of crew costs in operation costs for the means of transportation and the lack of operational research application for crew scheduling in maritime settings.

In this thesis, our primary objective is to provide an elaborate crew scheduling which is cost and time efficient by satisfying the rules and regulations for a large maritime company conducting an Offshore Service Vessel type operation on a global scale. To carry out our main objective, we underlined smaller research aims that

are grounded in the analysis of proposed methods, and construction of practically efficient solution approaches for the crew scheduling problem described by industry partners. In accordance with our goals, we aimed to contribute a more realistic mathematical model, diagnose the reasons for low efficiency regarding the solution time of proposed methods and formulations by Leggate (2016). Leggate (2016) modelled the VCS problem in OSVs with two formulations named as Task Based (TB) and Time Windows (TW). This dissertation aims to improve solution methods in the light of the information obtained through computational analysis of previous methods for recovery schedules and finally suggest a robust counterpart for schedules from scratch.

In other words, in this research it is expected to find an answer to this question:

- How can the proposed models for VCS in OSVs be improved to generate feasible schedules against unexpected situations in a more time efficient way by using optimisation tools?

Additionally, these are the sub questions which support main research question:

- What are the weaknesses of proposed TB and TW models?
- How can one deal with the drawbacks of TW model?
- Are heuristics and exact methods that are suggested in this thesis effective ways to solve TW model?
- What are the strength and weaknesses of the suggested methods?
- How can one obtain schedules immune to uncertainty?

Based on these research questions and our objectives, the contributions of this thesis can be stated as follows. First of all, an extensive computational study is performed and analysed to have a deeper understanding of TB model which is presented in Section 4.1. This contribution consolidates the our joint paper (Leggate et al., 2017) which has been accepted in JORS. Second, revision of realistic model is also applied. The design and development of a customized heuristic for the case study which was started by Leggate (2016) are provided as a joint contribution to this research. The first attempt of the heuristic by Leggate (2016) was not elaborate enough, had limited options for enlarging the local search in case of getting

stuck at local optima, and abrupt termination settings. Therefore, a more thorough, exhaustive and correct customized heuristic algorithm is designed and developed in this thesis. Moreover, the implementation of this customized heuristic in C++ with Microsoft Visual Studio 2010 is solely accomplished in this dissertation (see Section 4.3).

In Chapter 5, as a more distinctive contribution, the existing model is analysed thoroughly and amended accordingly to increase the solution quality and state the problem faced by industrial partner more accurately. After reviewing the literature on optimisation techniques to solve VCS in OSV, we determined to use Benders Decomposition which solves multiple smaller partitioned problems rather than solving the whole problem with all decision variables and constraints simultaneously, (Taşkın et al., 2012).This technique is compatible with our problem and the problem can be decomposed into smaller problems. Accordingly, Benders Decomposition is applied not only with naive implementation but also with some improvement techniques suggested in the literature. Besides, a hybrid method Benders Decomposition Algorithm and the customized heuristic is proposed as an integrated solution method. Although we could not manage to obtain optimality in reasonable time frame with this method, the application of this decomposition technique provided profound analysis to understand crew recovery scheduling problem in OSVs from a wider horizon. In addition to this, for some problem sets the hybrid method gives higher quality results than the direct implementation of the modified model.

Lastly, an initial study is given for seeking the robustness in the phase of creating crew schedules for OSVs, in Chapter 6. This study initiates the robust planning for the crew in offshore settings. After the potential uncertainties are defined, a model that provides robustness against crew availability and demand uncertainty is suggested.

1.4 Thesis Plan

This thesis consists of 7 main chapters. In Chapter 1, offshore supply vessels, crew scheduling in this context, and research motivation are briefly explained. The following chapter deeply investigates, in an in-depth manner the areas such as crew scheduling, recovery scheduling problem, crew scheduling in the maritime industry, and robust optimisation. Additionally, Chapter 2 highlights the research methods

used and the research philosophy adopted in this study.

The problem that was brought forward by our industry partner and investigated through the case study method is presented in Chapter 3 with a business model of the managerial process.

The proposed formulations, solution methods and collaborative work with Leggate et al. (2017) in order to solve this problem is deeply analysed for further development of new solution methods in Chapter 4.

In Chapter 5, advantages of benders decomposition and suitability of this technique to our problem are explained. Following this, an analysis concerning this decomposition method is also suggested. Depending on the results of this analysis, a better functioning model that clearly reflected the intricacies of the problem that were pertinent to our case was provided by modifying the model that was investigated in Chapter 4.

Instead of responding to the uncertainties through recovery schedules, the investigation of robust optimisation techniques to generate more robust plans against to uncertainty is conducted in Chapter 6. The sources of uncertainties are further defined, and a robust counterpart is suggested to overcome these changes.

In Chapter 7 a detailed discussion regarding this study is provided. In addition to the thorough conclusion, research limitations, and scope for future research are also elaborated upon.

Chapter 2

Literature Review

Scheduling is used mostly in manufacturing systems, services sector, information technology, hospitals, educational institutions and transportation systems to increase the efficiency in each system with regards to the optimum utilization of limited resources. Scheduling problems have received significant research attention and these problems are wide-ranging, just like the ones we mentioned previously. The main interest of this study is crew scheduling, particularly in the maritime context as well as under other uncertainties. In this section, relevant literature to our problem will be provided in course of two main sections, *Crew Scheduling*, and *Robust Optimisation in Scheduling*, respectively.

2.1 Crew Scheduling

Crew scheduling has been studied for more than 50 years. The complexity of these problems and increasing demands in today's global world have kept this research area alive. Some survey papers that highlight comprehensive understanding for crew scheduling and rostering problems are presented in Ernst et al. (2004b), Brucker et al. (2011), Van den Bergh et al. (2013).

Personnel scheduling problems were first classified into three main groups such as shift scheduling, days off scheduling and tour scheduling by Baker (1976). In shift scheduling, the staff requirements on each shift can be treated independently in order to decide the feasible allocation of employees.This problem is generally encountered by industrial companies. In the days off scheduling, the length of the operating week in the facility and the length of an employee's working week are not equal to

each other. 5-day work weeks for employees and a 7-day operating week can be an example for this case. The tour scheduling is defined as a combination of the shift scheduling and the days off scheduling problem. Tour scheduling creates rosters for each staff member over the planning horizon. In this kind of personnel scheduling problems, organizations have the 7-day operating week, with more than one shift a day (e.g., airlines, hotels, hospitals, etc.). As there are certain regulations for giving breaks and the maximum amount of work, the particular tour (i.e., hours of the day and days of the week) in which the employee is expected to be allocated has to be specified.

For crew rostering problems Ernst et al. (2004b) distinguished six modules which are demand modelling, days off scheduling, shift scheduling, line-of-work construction (tour scheduling), task assignment and staff assignment. Demand modelling is divided into subgroups such as task-based demand, flexible demand, and shift-based demand. These groups show variety with respect to the level of uncertainty in demand. Mostly task and shift based demands are known in advance. In flexible demand, the required staff is predicted by using forecasting techniques.

In Brucker et al. (2011), they classify the studies according to four main categories, elaborately: i)characteristics of personnel, decision delineation and shifts definition; ii)constraints, performance measures and flexibility; iii)solution method and uncertainty incorporation; iv)application area and applicability of research. This study shows the substantial variation of studies and different problem structure in this area with the help of this four main categories. Transportation is stated as one of the main application areas for personnel scheduling and crew scheduling is represented as a special case for personnel scheduling problems. The significance of temporal and spatial characteristics of tasks is mentioned in Ernst et al. (2004b) and the limited studies in the maritime transportation is observed through the survey paper in Brucker et al. (2011). The complexity of our problem distinguishes from the literature regarding temporal and spatial characteristic such as carrying operations offshore and duties with a more extended time length.

Crew scheduling problems can also be categorized as assignment problems. A detailed survey of the assignment problems was presented by Pentico (2007).

Multiperiod staff assignment problems deal with assigning staff members to change tasks during a planning horizon with considering the operational requirements and eligibility of staff members (Franz and Miller, 1993). Franz and Miller (1993) aimed to maximise resident's schedule preferences while holding the goals of

hospital and staffs' contractual necessities for the 12-months period. They proposed a heuristic solution and compare their results with maximum possible and continuous objective function values for testing the performance of heuristic solutions. In our problem, we are also interested in assigning our crew to operations by thinking the required number of crew members to each vessel, each week and eligibility of them for certain tasks. By this characteristic, we can also call our problem as 0-1 assignment problem with side constraints.Mazzola and Neebe (1986) have drawn conclusions about the NP-completeness about 0-1 assignment problem with side constraints, known as (APSC) in literature. They emphasized that 0-1 assignment problems even they only have no more than one side constraint is NP-complete.

As the applications of personnel scheduling are substantial and have a large variety, relevant literature to our problem can be narrowed down to crew scheduling in transportation settings. Under Section 2.1, Section 2.1.1 particularly provides comprehensive literature about the airline crew scheduling which is the most popular area amongst all transportation settings for crew scheduling problem. Following the airline crew scheduling, Section 2.1.2 provides relevant literature on crew scheduling in other transportation settings. In Section 2.1.3, we presented the studies on recovery crew schedules in various transportation settings and investigated the crew scheduling problems in maritime settings by pointing out the characteristics that distinguish from our study.

2.1.1 Airline Crew Scheduling

The closest approach to crew scheduling in vessels can be stated as the airline context in terms of the complexity, size and uncertainty of the inputs. Airline crew scheduling has extensive literature over the decades.

Airline companies generally combine flights as a bundle including specific patterns such as pairings or crew rotations. These crew pairings start and end at a crew base Andersson et al. (1998). The pre-defined pairings are integrated into the crew rosters. Appropriately, the crew scheduling problem is decomposed into two problems as crew pairing and crew assignment. Then, these problems are solved respectively. Crew pairing problem is generally formulated as a set partitioning problem, (Zeghal and Minoux, 2006). This formulation tries to find the optimal pairing to minimise cost while covering all the flight segments. It brings explicit enumeration for all pairings and it is not possible to enumerate them especially for the big problems. After solving

the crew pairing problem, the next step is crew assignment problem. In this study, crew assignment problems are solved separately for each crew type. Since we try to solve recovery problem separately for captains, riggers etc., solution methods that we presented in (Leggate et al., 2017) show the similarity in terms of the rostering method in this study. However, the constraints used for crew assignment differs. The reason of that situation is based on the variation between vessels and airline settings regarding the rules and regulations.

Anbil (1993) conducted a joint study with IBM and American Airlines Decision Technologies to minimise cost by enumerating feasible pairings with an LP solver. To solve very large problems, Bixby et al. (1992) suggested a solution as a combination of interior point and simplex methods. On the other hand, Hoffman and Padberg (1993) are more interested in having solutions in reasonable time frame rather than having optimal one. They proposed a set partition formulation and solved it by a branch-and-cut approach. Klabjan et al. (2001) improve this solution method on by enumerating millions of random pairings. LP relaxation is followed by a heuristic method which is based on the information gained by the duality of LP relaxation formulation to obtain integer solutions.

Gamache et al. (1999) used set partitioning and column generation method to decide the pairing assignment and schedule the other activities of the crew. To obtain the best rosters between the set of possibilities, an IP formulation is applied to the master-problem. Sub-problem generates schedules for each employee by using the network formulation. In this formulation, nodes represent time points and arcs represent the tasks or rest periods. The definition arc costs are based on the reduced costs taken from the master-problem. Accordingly the shortest path gives the schedules for staff.

Barnhart et al. (2003) give a thorough analysis of airline crew scheduling by expressing the common elements and importance of crew scheduling in transportation context with respect to maintenance and cost. The constraints about the pairings and duty period time length, minimum and maximum amount of resting periods between duties, maximum elapsed time of a pairing show similarity with our problem as well. The rostering problem constraints, are to ensure the crew pairings have the appropriate number of selected schedules and all of the crew members are assigned one work. Due to the fact that both crew pairing and assignment problems have an enormous number of decision variables and these decision variables are all integer, it is really hard to solve these problems. Barnhart et al. (2003) provide information about

the solution approaches to these problems from the existing literature. They gave some basic ideas about network structures, partial generation of pairings, solving LP relaxations and branching methods for pairing problems.

Benders Decomposition method is another way to solve large scaled problems that are applied to crew scheduling in airline settings (Cordeau et al., 2001; Mercier et al., 2005). Cordeau et al. (2001) have proposed a solution method for the aircraft routing and crew scheduling, simultaneously. This study shows the importance of integrated decision making on aircraft routing and crew scheduling decisions.Benders decomposition is suggested to decompose the problem master and slave problems and column generation method is applied to be able to solve decomposed problems.

Cappanera and Gallo (2004) define a multicommodity network flow problem for crew scheduling problem. In this representation, a network shows the tasks and crew are symbolized as commodities. They defined some valid inequalities and used CPLEX to analyse their LP formulation. With the help of this LP model, they figure out finding the crew pairings, rest and training periods with respect to the rules and regulations.

There are many solution methods suggested for the assignment of airline crew to minimise crew pairing cost in terms of daily, weekly and monthly time horizon. Since monthly time horizons have the more practically applicable solution approach comparing daily and weekly time periods, the direction of airline crew studies is mostly based on the monthly planning. These problems are generally formalized as the set covering and set partitioning problem. Gopalakrishnan and Johnson (2005) and Klabjan (2005) provided comprehensive survey paper for staff scheduling in airline concept.

Nissen and Haase (2006) have an interesting study which analyses the variation between the US and European airlines. Their formulation is the duty-period-based and basic reason of the difference between these two regions is payment method of the crew. European airlines have crew fixed salaries and North American crew's salary depends on the time that they have worked. The predominance of fixed price gives advantage in terms of having shorter rescheduling horizons. Accordingly having smaller problems and, thus, faster solution times are possible.

In another study, Gamache et al. (2007) proposed a graph coloring model and a tabu search algorithm for solving a feasibility problem in monthly airline crew scheduling by giving more importance for the crew satisfaction more than minimising the crew cost. The proposed solution method gives feasible monthly airline

schedule for each employee. Since their method offers the possibility to handle each requirement either as a hard or a soft constraint, it can be said that it is a flexible approach. To check the feasibility of the schedule, graph colouring is used. In this method, nodes show the assigned tasks, and two nodes are connected if their associated tasks could not be performed by the same worker. If the number of staff that has to be allocated is less than minimum number of colours, then the schedule does not give feasible solution for the rest of crew.

One of the recent and efficient methods used in crew scheduling is constraint programming. Suraweera et al. (2013) improved an algorithm to produce constraint tree. They particularly focused on airline crew scheduling problem. The aim of the airline crew scheduling is assigning crew to operate flight legs on the airline. Their goal is exploring the flight requirements, government regulations, assignment of the crew, and contractual obligations. Getting information about these obligations, regulations take long process. Changes in airline rules, and the difficulties to understand incomprehensible rules are the main motivation of this study. They used an algorithm called as ComCon to infer constraints from the schedules that are based on constraint templates.

Tam et al. (2014) introduced an operational multi-crew scheduling problem in airline settings. Two models with objectives, which are minimising the total number of un-covered tasks or the total number of uncovered rights, are solved. Integer non-linear multi-commodity network flow formulation is suggested to solve the problem. Column generation method in a branch-and-bound scheme is proposed as an optimal solution method for this study.

Section 2.1.2 focuses the crew scheduling concept in other transportation settings rather than the airline industry.

2.1.2 Crew Scheduling In Other Transportation Settings

Obviously, the airline is not the only transportation setting that required crew schedules, even it is a limited amount, there have been some studies about bus and railway rostering problems as well.

In railway settings, one of the studies that focused on cost minimisation is conducted by Vaidyanathan et al. (2007). They used network flow model to assign the crew to the trains. In this network, crew districts are defined by railroad, and each crew district has crew pools. The rules for this problem are designated according

to these crew pools. To solve this network based approach, mathematical modelling is used but increase the efficiency they also used some relaxation techniques. They showed their results by analysing them with real data obtained from North American rail road. These studies (Vaidyanathan et al., 2007; De Leone et al., 2011) have similarities with our work in terms of assigning crew to the particular tasks, however, the planning horizon and regulation show significant differences.

De Leone et al. (2011) studied on one of these problems about bus crew scheduling. In their study, they formulated a mathematical model for this NP-Hard problem considering labour rules and safety regulations in Italian transportation. Mathematical model is not practical enough for the large problem instances. Due to this reason, De Leone et al. (2011) improved a Greedy Randomized Adaptive Search Procedure and analysed the performance of this heuristic by using real-world instances.

A railway crew scheduling problem is studied by Hanafi and Kozan (2014). This problem concerned the allocation of train services to the crew according to the specific train timetable. There are some operational and contractual requirements that need to be considered for having the feasible schedules. Based on the complexity of constraints and a large number of decision variables, the problem is defined as mathematically intractable. In order to deal with this difficulty, a hybrid constructive heuristic with the simulated annealing search algorithm is suggested as a solution method. According to the computational study, they concluded that the hybridization of a simulated annealing-based algorithm for solving a highly constrained combinatorial optimisation problem is an effective method.

Ma et al. (2016) focused on bus crew scheduling problem in Beijing. In this case study, a meta-heuristics approach is employed for solving real-world bus-driver scheduling problems. Time windows approach is used to define the duties of bus drivers. The problem is solved by a variable neighbourhood search algorithm and a case study of two depots of the Beijing Public Transport Group is used for evaluating the heuristics' performance. The results underlined that up to 18.1 % cost minimisation is obtained.

2.1.3 Crew Scheduling Problem with Recovery

In our study, we have also interest in the recovery of the current schedule. Since crew schedules in vessels tend to be affected by environmental factors, the schedules are required to be adjusted or updated. This feature changes the origin of the study

to the recovery scheduling problems.

However there are not too many studies for the recovery problem, some heuristic search algorithms, dynamic programming algorithm, and column generation methods exist in the literature. Wei et al. (1997) developed a multi commodity network flow for the crew management problem during airline irregular operations. Their objective is to minimise cost for returning to the original schedule. They gave a depth-first branch-and-bound search algorithm to solve their set covering formulation for this problem. Their algorithm provides flexibility in terms of the constraints defined by the business.

Yu et al. (2003) improved a decision-support system which is called as CrewSolver for Continental Airlines to generate crew-recovery solutions. Crew recovery model has two constraints which are covering the flight and enforcing the crew assignment to be assigned. Related penalty cost is applied to the uncovered flights, deadheading crew, and in the case of no assignment of a crew. They proved the complexity of the problem as NP-hard. As a result of the implementation of this CrewSolver, the savings from the major disruptions are declared as approximately US $ 40 million.

Lettovskỳ et al. (2000) presented a crew recovery model in airline context. In this model, the cost of adjusted pairings, reserve crew, dead-headed crews and cancellation are aimed to minimise. They propose a recovery plan for reassigning crews to deal with the disrupted crew schedule. The built a fast crew-pairing generator which enumerates feasible continuations of partially flown crew trips.

Guo (2005) has a different approach the recovery problem. This study is aimed to minimise the changes in the current schedule. The problem is formulated as set partitioning and he had both column generation approach and hybrid of a genetic algorithm with a local search. And he tested the performances both of the solution methods with a case study including 188 crew members and 85 daily flights for 5 days recovery period. After this test, it is noted that heuristic approach has a reasonable solution obtained within 3 minutes.

A thorough survey paper is provided by Clausen et al. (2010) including recent disruption management (recovery) methods in airline industry. They give comprehensive information about the recovery problem in the airline setting with respect to different objective functions for the different resources of the disruptions. In recovery problems, they conclude that finding solutions quickly is an important element. They aim to reduce the problem size by using the only time windows which are required re-planning, and only the affected crew with a certain number of candidate

crew.

Zhang et al. (2015) suggested a two stage heuristic algorithm in order to solve the integrated aircraft and crew recovery scheduling problem.The first stage, the integrated aircraft schedule and the flight-rescheduling problem with partial crew consideration are solved. A disruption cost of the original crew connection is added to the objective function, as well. They proposed solving the integrated crew recovery and flight re-scheduling with partial aircraft consideration, in the second step of solution method. Modelling the problem as multi commodity network flow rather than set covering method provided higher efficiency in solving MIP with CPLEX Solver for the 2nd stage of the problem. Additionally, an iterative algorithm approach is employed to solve the integrated problem until there is no improvement found.

Rezanova and Ryan (2010) focused on a crew recovery problem in railway context. They formulated the train driver recover problem as a set partitioning problem and they dealt with the infeasibility by adding further drivers or increasing the recovery time period. The solution method is based on solving the LP relaxation of the set partitioning problem with a dynamic column generation approach. They prefer using depth-first search as constraint branching strategy. Depending on the real-life data from the Danish passenger railway operator, their computational study underlines that applying optimisation techniques to crew recovery problems provide efficiency for decision-making process against disruptions.

2.1.4 Crew Scheduling in Maritime Industry

Among these crew scheduling problems, maritime crew scheduling problems are the main focus of our study. In Section 2.1.4, we give the motivation and novelty of our study while providing information about the up-to-date literature on this subject.

When the literature is compared in terms of crew scheduling problems in maritime industry and airline setting, it is easily observed that the quantity of paper for vessel crew is significantly less than airline settings. Wermus and Pope (1994) worked on a special type of navy crew scheduling. They tried to obtain a schedule that concerns about the equal workload for employees rather than minimising cost for a small-sized crew with 8 employees. Another study by Horn et al. (2007) considers an integrated vehicle and crew scheduling problem. The problem complexity shows similarity in terms of the crew scheduling. They tried to assign crew members as a team instead of individual assignment. In the formulation of this problem, integer

linear programming is used but since it is not sufficient, simulated annealing is also suggested.

Legato and Monaco (2004) addressed the cost minimisation problem, in the port of Gioia Tauro, raised from the change of crew. Two phased planing is suggested to deal with this problem. Similar to our case study, depending on the crew availability and sudden changes in weather conditions, long term planning for crew may not be easily maintained. Their solution method includes simple rotation for 5-days working and a 1-day resting period for each crew member for 30-day ahead planning in the first phase. The second phase is about ensuring the long term planning with linear programming methods for a day ahead preparation. Our study varies from Legato and Monaco (2004) with carrying more complicated health and safety requirements depending on the crew types.

Another study underlines the staff scheduling in maritime is conducted by Ammar et al. (2013). The objective of this study is finding a feasible solution by taking regulations, coverage rate and covering all journeys and increasing the worker satisfaction level rather than minimising the cost. To generate feasible solutions, the assignment of tasks to the staff are organized by teams. A mathematical formulation and two heuristics are suggested as solution methods and tested in a real case study.Koubaa et al. (2014) applied Artificial Bee Colony solution method on the Same seafaring staff scheduling problem which is studied by Ammar et al. (2013). Greedy Randomized Adaptive Search Procedure heuristic results of Ammar et al. (2013) are used as benchmarks to evaluate the performance of Artificial Bee Colony solution approach. Although the constraints and industrial settings of this staff scheduling problem in maritime context show similarity to our research, our study distinguishes with the objective function, planning time horizon and assignment of crew members within teams from the problem presented in Ammar et al. (2013).

Most recent study about vessel crew scheduling is conducted by Giachetti et al. (2013). In this study, they mainly focused on cruise ships and improved a solution for minimising the adjustments of crew. Minimising these adjustments helps to minimise cost due to crew schedule. In our study we set our objective as minimising cost depends on these adjustments. They used decision support system to decide the number of required crew. They tried to determine a feasible assignment to cover all tasks like in our problem. After deciding this number by stochastic overbooking model, integer programming is used for finding the cost of schedule. In our problem, there are two different kinds of employees called as contracted and agency. To prevent

infeasibility due to the lack of crew, it is assumed that agency can support employees to the required position. This study can be stated as the closest study to our study. The company is a global one like in our study. Also when the characteristic of crew is considered, transportation costs and international workers have common features with our study. On the other hand, we pay attention to the different skill levels of the employee. This reflects some consequences to the employee cost. Another point is, they worked with a cruise line and sudden changes are not valid for this type of vessels; although, the sudden changes are one of the important problems in our study.

Another aspect of our research is having robust schedules against sudden changes rather than searching for recovery planning. In order to increase the strength of our research regarding robust schedule aspect, we looked for robust optimisation techniques and their application, particularly in crew scheduling.

2.2 Robust Optimisation

Optimisation is about obtaining the best possible results by considering given conditions. It helps to take a decision in many areas like manufacturing, design, construction, planning, scheduling etc. to increase efficiency of aforementioned system. Given that optimisation techniques help to formulate the objectives as function and formulate the conditions and reach the best choices for efficient results, it is hard to have a completely deterministic data and construct a model without making strong assumptions . Since it is hard to have an information about the probabilistic distribution of unexpected events, researchers work on improving different techniques to treat the uncertainty in optimisation.

A post-optimisation tool *sensitivity analysis* suggests solving problems by fixing the values of parameters and trying to find out the effects of adjustments of these values on the optimal solution and feasibility. Clarifying that sensitivity analysis has an advantage regarding its ease to apply, it is limited for carrying out the robustness of mathematical models by allowing small perturbations during the analysis.

The related literature about robust optimisation is presented under two sub sections which are *Robust Optimisation in General Frame* and *Robust Optimisation in Crew Scheduling*, respectively.

2.2.1 Robust Optimisation in General Frame

A notable method coping with uncertainty is stochastic programming which works under a probability distribution assumption for uncertain parameter values, Birge and Louveaux (2011). In this method, the objective function is determined according to the random variables and the appropriate suitable function of it. Stochastic programming is a pro-active method that assumes uncertainties have probability distributions Eggenberg et al. (2010). The benefit of this tool is easy to model recourse; however, it is important to note that this solution method encounters with some optimisation issues for large-scale problems together with the heavy data requirements to apply this method Pinar (2012).

Robust optimisation can be stated as a complementary method to sensitivity analysis and stochastic programming. It is a modelling methodology dealing with optimisation problems in which the data are not certain and generally it is hard to make assumptions about the uncertain parameters Pinar (2012).

The very first study to improve immune models to uncertainty was conducted by Soyster (1973). This paper proposed a linear optimisation model which gives a feasible solution for every point of convex set under the assumption of parameters coming from a bounded, convex uncertainty set. While this method guarantees the feasibility for the entire set of uncertainty, this method is too conservative due to the fact that it highly relies on uncertainty set. This method considers the solutions of extreme cases without giving attention to the low probability of realization these extreme scenarios.

Ben-Tal and Nemirovski (1998), suggested ellipsoid uncertainty sets to deal with over conservatism issue and they ended up having a model with more tractable robust counterparts. Following these major studies, Ben-Tal and Nemirovski (2000) provided a comprehensive information on the methodology and application of robust optimisation, as well. They mentioned the characterization of a real-world optimisation problem such as inexact data, difficulties to implement accurate solutions into real-life problems, feasibility changes depending on the meaningful realizations, large-scale problems and inefficient optimal solutions that become infeasible with small adjustments. To cope with these real-life facts, they gave solid theoretical background on robust counterparts with different mathematical programming methods. Apart from the theoretical part, they also conducted a case study with 90 LP problems to show the big effects of small perturbations in the data and the achieve-

ments of robust optimisation methods on the inexact data.

Bertsimas and Sim (2004) extended the model of Soyster (1973) by controlling the degree of conservatism on every constraint. Their model maintains the feasibility when data changes and it is applied to discrete optimisation problems. Their core assumption of uncertain data is based on a symmetrical deviation between the bounds. They controlled the level of conservativeness with a parameter (Γ) and this parameter removes the necessity for non-linear programming applications. While ($\Gamma = 0$) implies no uncertainty, it would become Soyster's method under the case that (Γ) is equal to the set of the parameter subject to uncertainty.

Three seminal papers provided the fundamentals of the area of robust optimisation Soyster (1973), Ben-Tal and Nemirovski (1998), Bertsimas and Sim (2004). After robust optimisation gained popularity, different models have been developed on specific subjects with various methodologies and some survey papers organized such as Herroelen and Leus (2004) and Gabrel et al. (2014).

2.2.2 Robust Optimisation in Offshore Context

Halvorsen-Weare and Fagerholt (2011) study a supply vessel planning problem. This problem is a real life problem that Statoil energy company tries to deal with it. The objective is to minimise installation costs. There are some constraints are based on the maximum minimum duration of the voyage, the number of installations and the time between consecutive voyages. They gave the mathematical model for this problem. After explaining the deterministic model, they mentioned the weather impact on the installations. Due to the weather impact, deterministic schedules are not always so profitable. Accordingly, in this study, Halvorsen-Weare and Fagerholt (2011) suggested some robust approaches by adding slack to the voyages and schedules. These approaches can be listed as;

- for each day a supply vessel idle
- for each supply vessel at least one idle day in a week
- for each supply vessel no more than 2 voyages in a week

Considering these robustness approaches, they suggested an algorithm which is a combination of optimisation and simulation methods and they gave detailed computational study to present the test results. According to these results, predicted

cost decreases after adding the criteria of robustness. For the actual sized problems at least %3 savings are possible with different robust approaches based on the information of their simulation results. Their problem have some similarities with our problem in terms of the final schedule. They obtain schedules for vessels while we construct schedules for crew. Also their method of adding robustness can be expanded for our problem. Due to these reasons, this paper can be a good guide for our study.

In a liner shipping network, Brouer et al. (2013) is the first literature on suggesting a mathematical model for vessel disruption management. They proved the NP-completeness of recovery scheduling problems in vessels and they provided the computational study of the suggested model on a case study. They concluded that as the number of vessels are increasing, solution time increases with an exponential pattern. Qi (2015) emphasized the importance of real time recovery scheduling in the maritime industry. This paper focused on liner shipping with disruption both on a single vessel and multiple vessels on the same system and they analysed the benefits of port skipping and port swapping options under the case of longer delay from managerial aspect by developing a dynamic programming. Both studies provide reactive solutions rather than proactive robust planning from the beginning of the scheduling.

Since our problem is in the class of crew scheduling problems, in this literature review we preferred studying papers mostly on robust optimisation in crew scheduling. Since crew scheduling in airline transportation setting has significant economic effect, most of the robust crew scheduling studies focused on airline crew pairing problems. After giving a brief introduction to the description and methodology of robust optimisation in this section, more detailed summaries of robust optimisation in crew scheduling papers take place in the following Section 2.2.3.

2.2.3 Robust Optimisation in Crew Scheduling

Schaefer et al. (2005) consider airline crew scheduling under uncertainty. They emphasized that the difficulty raises in crew scheduling in airline industry due to the rules and regulations. This situation is valid for our study too. They grouped disruptions in severe and fractional disruptions. Airline crew scheduling problems are modelled like set partitioning problem with deterministic data, but in this kind of modelling assumes that every leg is operated perfectly as planned. It is ignored that

it is not possible in real life. In this study, they tried to compute the probabilities of disruptions. They improved an algorithm to calculate the expected total cost based on crew by taking into considerations of disruptions. They concluded that significant reduction in operational crew cost is obtained by using their algorithm rather than using deterministic method. In addition to this, even there is no computational proof, they think that considering each pairing in isolation instead of using expected cost as objective function may result a decrease in cost as well.

Yen and Birge (2006) proposed a model for airline crew scheduling with disruptions as an extension of Schaefer et al. (2005) study. They aimed to describe disruption costs raised from the delays of each crew and interactions among the pairings. They improved 2 stage stochastic integer programme. They describe disruption in terms of flight operation times. These times were defined as random vector and the elements of these vector represent disruption scenarios and these disruption scenarios occur with a probability. In the objective function, they try to minimise the summation of total cost and expected delay cost. There is no change in terms of constraints from the deterministic model. To solve minimising crew changes they used the delay of crews that are assigned to switch planes and improved a flight-pair branching algorithm. By this algorithm, they obtained pairings relatively easy when compared with crew-pairing branching. Furthermore, they analysed the performances of their algorithm and the advantages of the stochastic formulation over deterministic one. They showed that significant amount of savings is possible if delays are considered at the planning phase.

Shebalov and Klabjan (2006) discuss the minimisation of the crew cost. However, they have a different approach to model crew pairing called the crew pairing model with move-up crew count. Move-up crew means having the opportunity of swapping crews for a delayed flight by fulfilling the requirements of rules and regulations. Since maximising move-up crews provides to have more swappable crews, the crew pairing model with move-up crew counts, it is more likely to have more robust schedules. Their method has 2 stages. The 1^{st} stage minimises the crew pairing cost with traditional methods and in the 2^{nd} stage, they try to maximise the number of move-up crew. In our problem we are interested in assigning crews instead of crew pairing. It is hard to apply this method to our problem. However, it still can be a good approach to maximise the number of swappable crews to minimise the cost of crew changes. Shebalov and Klabjan (2006) tested the performance of their methods with an extensive analysis. They concluded that few move-up crews for many legs

are better than having too many move-up crews for few legs.

Lan et al. (2006) study on a robust aircraft routing model to minimise the expected propagated delay along aircraft routes. Firstly they improve an algorithm to generate delays and then they determine the distribution of the delay propagation. The objective of their formulation is minimising the expected total propagated delay of selected strings. Related constraints for this problem are covering constraints, flow balance constraints and counting constraint for the total number of aircraft in use at the count time and the number of aircraft in the fleet. They use a branch and bound technique to solve their mixed integer program and calculate propagated delay along individual strings when determining costs for the restricted master problem, but ignore delay when solving the sub-problem. They also consider minimising the expected total number of disrupted passengers.

Eggenberg et al. (2010) consider recovery airline scheduling with uncertain events. They gave detailed information about algorithms used under the uncertainty conditions. The first step for dealing with uncertainty is based on understanding the nature of uncertain set. To deal with uncertainties in scheduling, there are different approaches like proactive, reactive and predictive-reactive scheduling, (Lütjen et al., 2012). In proactive scheduling robust schedules are obtained to handle disruption. For this approach, uncertainties should be required to have distributions. On the other hand, reactive scheduling does not need initial information. Predictive-reactive is combination of both these approaches and after initial schedule is created and revised when necessary. Between the uncertain sets which carry probabilistic information are solved with stochastic optimisation algorithms.

Eggenberg et al. (2010) improved a worst case pro-active method based on reactive algorithm. The reason for this preference is based on the difficulties of obtaining a wide set of observations and determining the nature of this observations. They defined different robust formulation based on objective functions to deal with robustness, minimise cost of the worst possible case, minimise arithmetical mean worst best case over the whole uncertainty set including same reaction costs, minimise maximum regret (the goal of robustness is more important than the cost minimisation). They gave a comparison between existing methods and their shortest path method application on interval data. They also aimed to extend the worst case pro-active method based on reactive algorithm for airline scheduling problem; but, they faced three important difficulties regards to deciding infeasible solutions in a given scenario and defining partial cost depends on infeasibility; solving the underlying recovery;

identifying the worst scenario for a given schedule. Since computing the recovery cost for every scenario is in the classification of NP-hard problem, they constructed uncertainty set rather than using a given one, having more schedule-based computations to estimate the performance of recovery algorithm and working more on multi-objective approaches.

Dunbar et al. (2012) consider the effect of aircraft routing and crew pairing schedules together on the propagation of the delays through the flight network. They suggest that determining aircraft routing and crew pairing together is really important due to the common transfer of delays between these two elements. They improve a robust solution for both aircraft and crew to deal with this problem. They give the mathematical formulation of these two problems (aircraft routing and crew pairing). They firstly consider how crew delays affect the aircraft connection. Their assumption is having a feasible set of crew strings and propagated delays because of the crews. Their approach aims to minimise cost based on unplanned delays and they showed that this method gives cost effective results due to their computational study.

Muter et al. (2013) discuss on airline crew pairing problem. They suggested a column generation method to obtain robust crew pairings. They give the robustness concept in terms of the extra flights requirements during operations. They noticed that in the smaller airlines in Turkey, there is a need of adding extra flights with short notice very often. Accordingly, they explain the robustness of their solution being able to allocate extra flights at the time of operation by disrupting the original plans as minimum as possible. The application of this approach is possible in two ways:

- Inserting extra flight into an existing pair
- Partially swapping the flights in two existing pairings to cover an extra flight

Crew pairing problem is solved in two phase. In the first phase, the total cost is minimised with respect to general constraints that are assigning each crew member to a sequence of flight legs and being sure about covering all flights with the solution. After getting the minimum cost, crew assignments should be done. In the robust formulation, the objective function is to minimise the summation of the the total cost of crew pairing and cost of deadhead flight. Constraints of this model are classic set covering rules and covering flights according to the robust method. Also, there

are some constraints providing connections between different decision variables and preventing assigning crew pairings without assigning them to the flight legs. Since there are lots of binary decision variables and constraints, there are lots of possibilities to reach feasible solutions. This fact makes these problems more challenging. Muter et al. (2013) solved the problem with column generation. It is harder to solve the robust version than classic crew pairing problem based on the column generation procedure. They concluded that their method works well for the small sized problem with minor changes but it is better to improve this method to have good results for the real-sized problems depending on the computational study.

However, in most of the studies discussed above suggest that robust schedules provide benefit in terms of cost, one paper Atkinson et al. (2016) concludes that it is not valid every time. This study examines different robust schedule practices. These practices are based on the flexibility to swap aircraft, flexibility to swap gates and scheduled down time. They construct a model to estimate the cost of robust scheduling inputs and the value of their operational outcomes. They suggest that this model can be used by managers, industry regulators and policy makers to decide the efficiency of robust schedules. Although their suggestions, it is not practical in our settings, it is important to conduct studies regarding the efficiency of robustness beforehand.

2.3 Existing Solution Methods

Optimisation techniques help to formulate the objectives as function and formulate the conditions and reach the best choices for efficient results. Mathematical programming methods are the way of optimising systems under given set of constraints and predefined parameter sets. Linear programming is one of the well-known areas of optimisation methods. To be able to use linear programming methods, all of the functions lying down the formulation should be linear. The simplex method which solves linear programming (LP) problems optimally is developed by Dantzig Wolf. This method uses the edges of the polytope constructed by constraints. Using edges gives the opportunity to try the consecutive extreme points for each iteration. In simplex method, it is assumed that decision variables are continuous; however, in real life problems, there might be a need for integer decision variables. Since the values of decision variable in optimal solution do not have to carry integer solution

by simplex method, this method may not work necessarily for every integer programming problems. Due to the requirement of integer solutions for a significant range of problems, several integer programming methods are developed, and these methods take considerable place among optimisation techniques.

Depending on the comprehensive literature review on crew scheduling problems in various transportation settings including the robust applications in planning and scheduling, VCS problem in OSVs is perceived as a combinatorial optimisation problem. Combinatorial optimisation searches for maximum (or minimum) of an objective function in the domain is a discrete but large configuration space. Combinatorial optimisation problems can be solved by mathematical programming which includes Linear Programming, Integer Programming, Mixed Integer Programming, Column Generation, Benders Decomposition, Branch-and-price, Dynamic Programming, Lagrange Relaxation, Goal programming methods. On the other hand, meta-heuristics for the application to combinatorial optimisation problems is another significant field of research to obtain feasible solutions with high efficacy. Besides the extensive usage of exact and heuristics solution methods, the hybridisation of techniques can be applied to combinatorial optimisation problems.

2.3.1 Mixed Integer Linear Programming

Mixed Integer Linear Programming (MILP) is an essential tool for solving combinatorial optimisation problems especially with the advanced technologies in optimisation solvers and computing. MILP is often applied to crew scheduling problems, as well. In this section, there will be a brief description together with the summary of essential definitions and theorems of MILP recalled throughout the thesis. A general MILP problem P_{milp} can be described as in equation 2.1. In this equation, x represents the continous variables with n dimension while y shows the integer variables with p dimension. $c \in \mathbb{R}^n$ and $h \in \mathbb{Z}^p$ are the vectors of the objective function coefficients. Parameters A and B are the matrices of constraints, while $b \in \mathbb{R}^n$ is the vector of right hand side coefficients.

$$P_{milp} = min_{x,y}\{c^T x + h^T y | Ax + Dy \leq b, x \in \mathbb{R}^n, y \in \mathbb{Z}^p\} \tag{2.1}$$

Definition 2.3.1 *Let $x_1, .., x_k \in \mathbb{R}^n$ be any set of points. Convex combination of points $x_1, .., x_k$ is linear combination of $x = \sum_{i=1}^{k} \lambda_i * x_i$ under the condition of*

$\sum_{i=1} \lambda_i = 1$ *and* $\lambda_i \geq 0$.

Definition 2.3.2 *Let the feasible set of* P_{milp} *is set* $S \subseteq \mathbb{R}^n$. *S is convex set if convex combination of any two points* $x_1, x_2 \in S$ *in the set* S.

Definition 2.3.3 *The set of all convex combinations of points in S gives the convex hull of the set S which is denoted as* $conv(S)$.

Definition 2.3.4 *The convex hull of finitely many vectors is a bounded polyhedron which is called a polytope.*

As above definitions provide general characteristics of the feasible region of MILP problem, it can be concluded that solving the problem P_{milp} can be satisfied through finding the solution over the convex hull S. One of the ways of describing convex hull is using extreme points and rays. A formal definition is provided below.

Definition 2.3.5 *Let* $x \in S$ *be an extreme point of polyhedron S then it can not be represented as a convex combination of any two distinct points in S.*

Definition 2.3.6 *A ray r of S is called as an extreme ray if there do not exist rays* $r^1, r^2 \in S^0$ *and a scalar* μ *such that* $r = \mu r^1 = (1 - \mu) r^2$ *for* $r^1 \neq \mu r^2$ *for any* $\lambda \geq 0$ *and* $0 < \mu < 1$.

The representation of a polyhedron through extreme points and rays is provided by Minkowski Resolution theorem:

Theorem 2.3.1 *Let* $S \neq \emptyset$ *be a polyhedron,* $\{x^k\}_{k \in K}$ *be the set of extreme points of S and* $\{r^j\}_{j \in J}$ *be the set of extreme rays of S.*

This theorem can be used as basis for decomposition techniques.

In this section, we provide some general definitions regarding MIP context. As the optimisation solvers employ Branch-and-Bound and Branch-and-Cut techniques, we provide brief information about these methods.

2.3.1.1 Branch and Bound Algorithm

Branch and Bound (B&B) is an exact optimisation algorithm which has been used in order to solve discrete optimisation problem for more than 50 years.

The procedure of B&B algorithm starts with an incumbent solution and depending on the minimisation or maximisation of the problem this incumbent solution represents the lower or upper bound for the optimal solution. If there is no incumbent solution, then infinity becomes the upper bound value for minimisation problem. This algorithm branches the whole problem into subproblems, and LP relaxation of each tree node is evaluated. If infeasibility is detected or the solution is worse than the best-known solution then, this node is pruned. Another possibility is to reach an integer solution that the incumbent can be updated. In the case of LP has a fractional solution with a better incumbent, the subproblem is divided, and new nodes are investigated. This procedure is repeated until all subproblems are examined, and the incumbent is determined to be the optimal solution.

2.3.1.2 Branch and Cut Algorithm

This algorithm combines branch and bound technique with cutting planes. It aims to improve the B&B by tightening the search space with the help of cuts. Depending on the solution of LP relaxation, the valid inequalities are generated. These valid inequalities are implemented to the original problem in order to cut off the infeasibility. In every step, LP is solved with the newly added constraints. The procedure continues till no new valid inequalities are explored.

In this dissertation, B&B and B&C methods are used by solvers as black box although any manipulation of the tree search concerning these techniques has not been applied.

2.3.2 Bender's Decomposition

Benders' Decomposition (BD) is another well-known optimisation technique to solve mixed integer programming, nonlinear programming and stochastic programming problems (Rahmaniani et al., 2017). It is an efficient method to deal with large-scale optimisation problems. It was first proposed by Benders (1962). In order to solve mixed integer programming problems, BD suggests decomposing the problem into two problems and solving them iteratively until the optimal solution found. The

decomposed problems are called as the restricted master problem (RMP) and sub-problem (SP). The connection between each decomposed problem is held by using the value of decision variables that are obtained by solving the dual of sub-problem (DSP). Assuming that we are interested in minimising the system, the DSP provides a valid upper bound for the original problem while RMP generates a lower bound for the system. The fundamental idea behind BD is generating some valid cuts for RMP through DSP.

Original BD can be explained for MILP as stated in Costa (2005). Consider the problem P :

$$\min c^T x + f^T y \tag{2.2}$$
$$Ax + By \geq b \tag{2.3}$$
$$Dy \geq e \tag{2.4}$$
$$y \in \mathbb{Z}^n \tag{2.5}$$
$$x \geq 0 \tag{2.6}$$

where x and y are vectors of the continous and integer variables, vectors c and f are the cost function coefficients, matrices A, B and D are the constraint coefficients, vector b and e represents the right hand side value. For a fixed value $\hat{y}$ of the variable y, problem P reduces to the subproblem SP:

$$\min c^T x + f\hat{y} \tag{2.7}$$
$$Ax + B\hat{y} \geq b \tag{2.8}$$
$$x \geq 0 \tag{2.9}$$

where $\hat{y} \in Y$ and $Y = \{y | Dy \geq e, y \in \mathbb{Z}^n\}$ Then we can state the dual of SP (DSP) by defining a dual variable u as given below:

$$\max(b - B\hat{y})^T u \tag{2.10}$$
$$A^T u \leq c \tag{2.11}$$
$$u \geq 0 \tag{2.12}$$

The variable y is fixed to value $\hat{y}$ without consideration of the feasibility of SP and DSP. As the maximisation problem DSP is a LP, the solution space of DSP is a non empty space that can be bounded, unbounded or infeasible. If DSP is infeasible then it corresponds to infeasibility or unboundedness in SP. In the case of unboundedness in DSP, SP is infeasible and it can be said that the $\hat{y}$ does not provide a feasible solution for the original problem P. As the feasible region of DSP is assumed to be not an empty space, this region can be represented with extreme points u^p and extreme rays u^f while $p : \{1, .., P\}$ are the extreme points and $f : \{1, .., F\}$ are the extreme rays. For bounded DSP, the solution is one of the extreme points and for unbounded DSP, there is a direction u^f such that $(b - By)\hat{u}^f \geq 0$. Under the case of unbounded DSP, the infeasibility of primal SP is investigated and prevented by eliminating the fixed $\hat{y}$ values by the cut $(0 \geq (b - By)\hat{u}^f)$ in restricted master problem (RMP). DSP provides an upper bound and it can be formulated by using extreme points and extreme rays in restricted master problem (RMP) through the constraints (2.14) and (2.15). These are called as optimality and feasibility cuts, respectively.

The rest of original P gives the RMP with the help of auxiliary variable z:

$$\min z \tag{2.13}$$

$$z \geq f^T y + (b - By)\hat{u}^o \quad \forall o \in \{1, .., O\} \tag{2.14}$$

$$0 \geq (b - By)\hat{u}^f \quad \forall f \in \{1, .., F\} \tag{2.15}$$

$$y \in Y \tag{2.16}$$

Based on the equations (2.7) to (2.16) for P, The Benders Decomposition (BD) algorithm is given in Algorithm 2.1:

Algorithm 2.1 Benders Decomposition Algorithm

Require: y := initial feasible integer solution, UB:= $+\infty$, LB:= $-\infty$
While UB-LB$\leq \epsilon$ **Do**
Solve sub-problem: $\max_u \left\{ f^T\hat{y} + (b - B\hat{y})u | A^T u \leq c, u \geq 0 \right\}$
If Unbounded then: get extreme ray f ; add cut $(b - By)^T \hat{u^f} \leq 0$ to RMP
Else Get extreme point o ; add cut $z \geq f^T y + (b - By)^T \hat{u^o}$ to RMP
UB:=$\min \left\{ UB, f^T y + (b - By)^T \right\}$
End If
Solve RMP: $\min_y \{z | cuts, y \in Y\}$, LB:= z
End while

Benders' Decomposition method has wide variety of application areas and there are various techniques that might improve the efficiency of this algorithm. Applications of some of these methods for VCS in OSV's will be discussed in details in Section 5. In the following section, detailed knowledge on heuristic methods is provided.

2.3.3 Heuristics

The Combinatorial Optimisation problems, which are NP-Hard, might require exponential computation time to reach the optimality in the worst-case. As the requirement of long computational time might not fit for practical purposes, approximation algorithms that provide time-efficient solutions have been investigated for more than 40 years.

The approximation methods can be categorised as constructive methods and local search methods. Constructive methods build a solution from scratch. They are mostly based on the best choice in each iteration and typically the fastest approximate methods. On the other hand, local search methods start with a feasible solution and try to improve it progressively. Depending on the defined solution criteria, in every iteration, current solution has been replaced with better or at least the same quality results that are obtained from the neighbourhood of the current solution.

The local optimality can be demonstrated as following. Let's assume P_{CO} is a combinatorial optimisation problem with objective function f to be minimised and variables $X : \{x_1, ..., x_n\}$. The domain of variables can be shown as $D : \{D_1, ..., D_n\}$. Then the feasible region of P_{CO}, S expresses the search space which includes all feasible assignment that satisfies all the constraints in the given domain D. The optimal solution $s^* \in S$ has the minimum objective function value and s^* is a global optimal solution of (S, f) under the condition of satisfying $f(s^*) \leq f(s) \forall s \in S$. Accordingly, $\forall s \in S$ there is a set of neighbours $N(s) \subset S$ while $N(s)$ is neighbour of s. Based on the neighbourhood structure N, $\hat{s}$ is a solution that $\forall s \in N(\hat{s}) : f(\hat{s}) \leq f(s)$. If $f(\hat{s}) < f(s) \forall s \in N(s)$.

In addition to the local search algorithms, meta-heuristics are another type of approximation method in order to solve Combinatorial Optimisation problems. Metaheuristics simply combine the basic heuristic methods in higher level frameworks. Metaheuristic methods do not require specialized knowledge of the optimisation problem. Meta-heuristic algorithms have two main components that are the lo-

cal intensification and global diversification.The concept of diversification means the exploration of the search space, while intensification is the exploitation of the combined search experience. Tabu Search, Iterated Local Search, Variable Neighborhood Search and Simulated Annealing are the meta-heuristics that work on one or several neighbourhood structures. We will briefly explain these trajectory methods in this section.

Algorithm 2.2 Iterated Local Search Algorithm

Require: Initial feasible solution s, neighbourhood structure N

1: Until stopping criteria met do
2: Find the best solution in $N(s)$: $\hat{s}$
3: Apply perturbation on $\hat{s}$;
4: Complete local search in the new neighbourhood and obtain $s^{'}$.
5: Apply acceptance criteria
6: if $s^{'}$ satisfies the criteria $s^{'} := \hat{s}$
7: end-if
8: end-do

- **Tabu Search (TS)**:This search method is first introduced by Glover (1986). TS simply aims to create a search which prevents endless cycling by avoiding to return to the recently visited solutions. To be able to escape from the cycles, TS uses short-term memory which keeps the information on the most recently visited solutions and not allows the search in that direction. This search chooses the best solution from the non-tabu list and assigns this solution as the new current solution. After the elicitation of a new current solution, the previous solution is added to the tabu list while one of the solutions is removed from the list dynamically. This method provides efficient results especially when it is integrated with other heuristics.

- **Iterated Local Search (ILS)**: In this heuristic method the search starts with an initial solution and conducts a local search until it reaches a local optimum. As the aim is to avoid local optima, the solution is perturbed and the search on the new local optimum starts. The key point in this search is to determine the degree of the perturbation as it might cause either to start with a diverged solution or not diversify enough from the current solution. Apart from the perturbation, the performance of ILS depends on the quality of the initial solution, and the solution acceptance criteria during the search.

Algorithm 2.3 Tabu Search Algorithm

Require: Initial feasible solution s, neighbourhood structure N

1: Until stopping criteria met do
2: Tabu List:= $\emptyset$
3: Choose best solution from $N(s)$ which is not in Tabu List
4: Update the Tabu List
5: end-do

- **Variable Neighbourhood Search (VNS)**: This method is based on the idea of systematic neighbourhood change during the search. It was first suggested by Mladenović and Hansen (1997). The change in the neighbourhood occurs under the condition of local search trapped in a local optimum. This method employs three main phases to change the search neighbourhood which are shaking, local search and move. The shaking phase is about perturbing the solution with the aim of providing an efficient starting point for the local search.

Algorithm 2.4 Simulated Annealing Algorithm

Require: Generate Initial feasible solution:= s, neighbourhood structure:= N, probability parameter:= p

1: Until stopping criteria met do
2: Choose solution $s^{'}$ from a random $N(s)$
3: If $(f(s^{'}) < f(s))$ then $s^{'} := s$
4: Else $s^{'} := s$ with probability p
5: end-if
6: Decrease p
7: end-do

- **Simulated Annealing (SA)**: This is yet another technique, which is first suggested in Kirkpatrick et al. (1983), to prevent the search from getting stuck in the local minima. It allows the solutions which are worse than the current solution during the search and the probability of this move decreases throughout the search. This algorithm begins with generating an initial solution and define a parameter which determines the probability of choosing the lower quality solution. The decrease in the probability is based on the idea that there is a higher probability of having a direction for the uphill moves in the beginning of the search and this probability needs to decrease as the search is converging a basic iterative improvement algorithm.

Blum and Roli (2003) provides more comprehensive information about metaheuristics in Combinatorial Optimisation, in this section we want to explain the heuristic search methods employed in this dissertation.

Algorithm 2.5 Variable Neighbourhood Search

Require: Set of neighbourhood structures $N_k \; \forall \; k = 1, ..k_{max}$, find a solution s

1: Until stopping criteria met do
2: **for** $k = 1; \;\; k \in k_{max}; \;\; k++$ **do**
3: Choose solution $s^{'}$ from a random $N_k(s) \Rightarrow$ shaking phase
4: Find local optimum starting from solution $s^{'} := s^{"} \Rightarrow$ local search phase
5: If $f(s^{"}) < f(s^{'})$ then $s := s^{"}$, $k := 1$
6: else $k := k + 1$
7: end-if
8: **end for**
9: end-do

Section 2.3 provided general information about the optimisation techniques which are relevant to the solution methods in this dissertation. Section 2.4 underlines the preferred methods and philosophy for this research. In order to understand why the specific methods have been used and why specific models have been adopted to solve problems in this dissertation, Section 2.4 plays a significant role in this study.

2.4 Research Methods and Philosophy

Every research subject has different patterns and unique point of view to the research questions. Even though they differ from this side, the methods that are used to answer the questions and the philosophical background can show similarities. The philosophical basement of this research in terms of ontology, epistemology and methodology helped me to choose the appropriate research methods. As stating the aim of our research is to minimise the cost arising from the drawbacks of crew assignment, we worked on finding some solution methods for obtaining optimal schedule with minimum cost. Scheduling has a significant place in operations research as well as in management science and staff scheduling is one of the important sub areas of scheduling as mentioned in Sections 1 and 2. Optimizing the assignment of staff gives advantage not only for lowering the costs but also maintaining convenience of schedules. Based on the lack in literature in vessel crew scheduling, a significant gap towards uncertainty drew our attention. Intuitively, people (crew) are one of the

most important factors that affects the cost. This fact prevents having deterministic parameters and moves this research to the more stochastic side. Due to the fact that human behaviour cannot be known exactly all the time, some non-deterministic perceptions are needed for drawing near results to the reality. Apart from the human factor, there can be some environmental effects that lead distortion in crew. In this research we started with literature search investigating the state of the art methods to solve similar problems.

The literature related to crew scheduling covers different types of methodology. These methods are generally based on the optimisation methods. Inevitably, for this research area, mathematical modelling has big impact to deal with these type of problems. To understand and be able to apply these different methods to our problem, our main research methodology is modelling which means the interpretation of the objectives with the resource limitation into mathematical models.

From another point of view, obtaining data for this research has remarkable place to evaluate the performance of the study. Accordingly, case study can be the other helpful methods to understand the quality of the study and maintains realistic solutions to this problem. Since we have already had a contact with a company, we preferred using modelling and case study methods as a research methodology throughout our study.

When the literature is searched in details in a specific area of scheduling, it can be seen that most of the studies are inspired from the past works. There is such a development path that every new study tries to increase the efficiency. This situation is valid in the crew scheduling as well. For staff scheduling there are some artificial intelligence approaches, some constraint programming approaches, some heuristics and mathematical programming approaches available, (Morgado and Martins, 1993; Azaiez and Al Sharif, 2005; Burke et al., 2004). Giving attention to these kinds of connections is beneficial for observing similarities and differences between the past works and our problem. So in this area it is so important to review the past models. In our case, it is better to be inspired by the airline crew scheduling literature. Scanning literature with details gives more idea about the models that are applied to the crew scheduling.

Modelling gives some insights to understand the other perceptions, helps exploring new methods. Besides it is very helpful tool to improve organizations and enhance their capability. Accordingly, we organized system inputs and outputs, and then we improved our solution methods based on mathematical modelling or heuristic ap-

proaches. The other important challenge of this study was producing reasonable solutions to real life conditions and performance analysis of the suggested methods. To be able to obtain effective solutions and test the performance analysis of solution methods, the well-designed data was required. Therefore, the importance of reaching or obtaining data cannot be disregarded. We simulated our own data using modelling methods based on the information provided by companies.

As mathematical modelling was required for solving this problem, It was really important to explore the problem with all dimensions. Observing the effects of different parameters on the performance of modelling approaches enriches the research and makes it more complete. With the help of some statistical tools, better insight can be obtained. Accordingly, statistical methods took place in the research methodology, as well.

The other method that I preferred to use in my research is the case study. This method is used in so many different disciplines. Case study is explained as a detailed intensive study of a unit, such as a corporation or a corporate division that stresses factors contributing to its success or failure. Case study research can be used for providing a connection between the research and the real life example of this research. Generally, there are some basic steps that should be used when conducting a case study. The first step is defining the research subject and explaining the aim of subject. The second step is choosing the cases which you want to work on and deciding the methods of collecting data. The other steps are related to data preparation, collection, analysis and reporting. As data gathering is really important for finding reasonable solution methods and evaluating these methods properly. It is better to see the company at their own place to understand the reasons of scheduling drawbacks. If the company is visited on-site, it would be easier to observe the operations. Furthermore, case study in a shipping company would give advantage to understand the possible problems of the crew, and to observe the deterministic and non-deterministic parameters for our mathematical model. Another advantage of case study with a company is the possibility of a joint work with this company and reaching realistic results. This provides a plus for organizing a cost analysis. Accordingly, we can have a chance to evaluate how our suggestions work in the cost side; cost can be minimised with this study after operating a joint work with the company.

When talking about the philosophical paradigms of this research subject it is important to give information about the ontology of this study. In general speaking

it can be said that ontology is keen on the nature of the reality. This research subject is looking for a single reality. Accordingly we can state the main ontology of this research is based on objectivism.

Objectivist approach looks for reality through the scientific facts, laws and rules. In this research, we aimed to suggest different solution methods to optimize the schedule via robust optimisation methods which is subsection of mathematical programming. We will evaluate the results and performance of our study from this concept. In the light of these targets, this research is based on the scientifically proved facts. In addition to these, objectivism requires researchers to take external place. The researcher must take a positive stance and become an independent observer throughout the research process. Accordingly, we did not put our biased ideas into the study. Even, our research is based on the non-deterministic parameters, this does not affect the nature of reality. Nature of reality of this research was based on the single reality because we planned to approach this problem working with the simulated data which is based on the facts obtained from literature.

Another important philosophical foundation of the research is to understand the knowledge of this study and express the basic belief of the researcher. This knowledge defines the epistemological position of the research. The main epistemological positions in management science are positivism, critical realism and interpretivism. It can be roughly said that the ontological position of these paradigms change from objectivism to subjectivism and from positivism to interpretivism. Positivist philosophy accepts ideas that are capable of being proven scientifically. From this point of view, in this research the main paradigm is positivism. Within the light of this explanation, it can be stated that there is a consistent relation between the ontological and epistemological background of this study. At the same time, we defined crew related problems on costs in a realistic way. This gives us another research paradigm as realism. Actually, realism and positivism have similarity settings in terms of data gathering. Realism is not as deterministic as positivism but at the same time it does not have totally subjective view to the knowledge.

When looking for the truth has again external stance but researcher gives the idea of theoretical reasoning and experimentation. The reasoning of quantitative research can be stated as deductive. Deductive approach starts with a theory or hypothesis and then used observation for confirmation of the study. In this research we also start with literature review for existing theories rather than data. After that the model has to be computationally tested for confirmation. Last but not least it is believed

that the problem can be solved by using modelling, particularly to mathematical programming. Mathematical programming is a way to model the relation between different variables at the same time holding the importance of known parameters. In this research we attempt to develop a model using programming with the aim of minimising the total costs of vessel crew scheduling problem.

Additionally, we concentrated on some heuristic methods. By heuristics, optimality is not guaranteed for the nature of our problem but it is useful for preventing the lodging in a definite point. Accordingly, the methods and techniques of this research can be summarized as modelling with data generation methods based on the scientifically verified facts, simultaneously seeing the practicality of the research.

2.5 Summary for Literature

We have presented the variety of studies that are linked to vessel crew scheduling (VCS) by splitting them into four main groups. Based on the research in literature, various optimisation techniques have been employed to solve the related problems to VCS such as; branching techniques, column generation and decomposition methods, and heuristics. There are a significant amount of problems required for the combination of these methods.

However, there are some studies that have a similar concept to our problem. Vessel Crew Scheduling Problem with Offshore Supply Vessel setting has not been introduced in the literature. We believe that with the help of case study method, our problem can be distinguished from the studies in literature by having scheduling horizon spanning several months, the frequent need of rescheduling as conditions change over time and with the complexity of rules and regulations to be held by the company.

Chapter 3

Problem Description

Crew scheduling problems have variations in different contexts. In vessel crew scheduling, problems mostly arise from the planning time horizon of interest, the complexity of carrying out offshore operations, specific rules and regulations in the maritime context, and unexpected weather conditions. In addition to these, our problem gets explicitly harder in the sense of having a global scale of vessels, mixed nationality group of crew and having a different set of skills for offshore operations and more extended duty time periods. In Section 3.2, the current method using by the company for decision-making process to deal with the changes is provided and in Section 3.1, the problem which is obtained through the case study, is described with details.

3.1 Vessel Crew Scheduling in Offshore Supply Vessels

A case study is an essential element of our research. The problem description is based on the insights gained through interviews with a company, which is operating many OSVs to provide engineering, construction and services to the offshore energy industry worldwide, made by former PhD student Leggate (2016).

To state our problem clearly, we start by giving detailed information about crew set. There are three groups of the crew which are named as regular crew, contracted crew and agency crew. The set of all crew members are represented as (E).

The regular crew are shown as (R) in the mixed integer programming (MIP) model, and they have fixed contract with the company. They also have the limita-

tion concerning the maximum number of consecutive weeks they can carry without penalty cost and are also obliged to rest to fulfil the minimum amount of rest period after departing from any vessels. The second group of crew who are also the subset of the regular crew are contracted crew (G). Hiring these contracted crew costs less than hiring the other type of crew. Both regular crew and contracted crew are the subsets of crew set (E).

Apart from the regulations related to maximum consecutive weeks of working and minimum rest they need to take, they have guaranteed days to be assigned; they need to work at sea per year. Otherwise, it leads additional payment for the company with overtime cost or wastage of salary in the case that they work less than usual days.

We also have agency crew which is again the subset of (E) and has the index (m+1) who are found by external agencies. Although they are almost twice more expensive than regular crew, they become advantageous in some instances such as shortage of people and more demand for workers. Since they are available at short notice, always meet the demand of required skill level and there is no rest constraint for them, they are more flexible than the other categories. Due to this flexibility of the agency crews, infeasibility of the problem can be avoided with paying more considerable costs. Apart from the contract type of crew, there are some other points need to be stated regarding crew features. Crew availability has importance for being able to carry out the task at the given time for the necessary role. It is based on different conditions of the crew like physical availability (e.g. fatigue level, health condition, motion sickness), the nationality of the crew member (e.g. legal and contractual requirements), eligibility (e.g. experience level, training and skills). As we mentioned briefly maximum consecutive work and minimum rest period to explain legal and contractual requirements, there are also restrictions with regards to working permit for some nationals of the crew to work in ships registered under specific flags. This situation might lead to inefficient combinations concerning cost and mandatory paperwork. Each task is designed to be operated by a crew member at a time, and each crew member can be assigned only one job at a time.

The most common assignment pattern for the crew is to have 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. However, depending on a vessel's location, the main crew nationalities, and the crew role in question patterns can also include 2-on-4-off, 10-on-4-off, and others,

(European Union, 2003). In addition to the nationality mix constraints, the crew has different skill levels depends on their experience and training history. Accordingly, their eligibility for each task needs to be ensured to be able to carry out the roles that they are assigned.

Offshore supply vessels have a different kind of operations compared to other maritime settings. The operations mostly have been designated to occur in marine support operations; therefore it requires more complex organisations and longer planning horizon with extended duty period. Our industrial partner uses in practice a background of 13 weeks for their crew scheduling, and hence we apply the same length of the horizon in the remainder of the thesis. There is a need to arrange travel in advance for the crew, and also the logistics should be arranged for the crew up to four weeks before operations start offshore. Since the aimed planning horizon is long, to be able to ensure the robustness of schedules, planners encounter with problems related to practicality and maintenance of current plans which means 13-weeks in advance in our problem. Those characteristics of our problem lead to uncertainty in planning and make the conditions harder for crew availability. Accordingly, rolling horizon approach has been used, and changes have been applied when there is a need for it by using the current schedule as a starting point.

The current method applied by the company is back to back scheduling which can be stated as assigning a crew always on the same vessel and sharing their role with another crew in turn. In Section 3.2, the current method of the company is given with detail.

3.2 Decision Making Process in the Company

In the current system of our industrial partner, there is no decision support tool which can assist to find the schedules with the minimum cost for the crew or to recover the unexpected changes in crew planning with the minimum cost.

The usual business in our industrial partner relates to providing services to the clients for the maintenance of offshore operations. These clients are mostly offshore oil industry companies. The agreements between the industrial partner and their clients are based on the projects. The demand of the clients, the length of plans and operations are determined by the clients and the company together in advance.

Operations are controlled by both onshore and offshore staff to manage the plan-

ning process, more precisely. In the onshore group of staff Project Managers, Vessel Managers, Planners, and Crewing group take place. Offshore Managers work on the vessels similar to the crew members. Apart from our industrial partner, external travel agency and client companies are also involved in the decision-making process.

After the client and the company has agreed to the project, Project Manager contact with the clients and determine the requirements of the project to meet the clients' demand. Following this process, Project manager communicates with the Vessel Manager to arrange the assignment of the vessels to the related projects. Vessel Managers schedule the ships and provide the information on updated ship schedules.

The schedules are sent to the Planners, and they assign the crew members to the vessels by considering the rules and regulations explained in 3.1. It is essential to keep the crew schedule up-to-date for 13 weeks which is the general time horizon for projects in the company. Vessel managers are required to inform the Planners to deal with the change in the earliest convenience.

Crew assignments are always done on the same vessel, and the roles are operated by sharing with another crew. This method is not robust enough towards unexpected situations and requires some amendments to the default schedule. The Planners need to find a new feasible plan when the change is necessary and to be confirmed by the entitled crew, Crewing group and the Offshore Managers. As the Crewing team organise the required travel arrangements for crew members when they are transferred to the assigned vessel or port, they involve this decision-making process in this step.

Furthermore, to be able to provide the transfer of crew members, the bookings should be arranged with the Travel Agent. Four-week ahead planning is made regarding the crew change date. Although the reservations are completed with external Travel Agent, Crewing employees provide the specific changes regarding the time of crew change, vessel and location to the OffShore Managers.

Offshore Managers have administrative tasks. Together with the crew schedule proposal, the vessel information also needs to be checked. Therefore, there should be an information exchange between the Planners and the Vessel Managers. After the confirmation of the suggested schedules, Planners contact the Travel Agent to amend the bookings. There is a possibility of the refusal of the new plan; it means that the planners need to identify a new one again.

Following the whole communication process between the onshore and offshore

staff and confirmation of the changed schedules, the last step of planning is completed by the external travel agent. They obtained the information of required arrival times, routes for transferring the crew members, new bookings or cancellations. Depending on the collected data they organised the essential bookings and provided feedback to the Crewing group.

Crew members are an essential part of the team that operations cannot be completed without their workforces. In the planning process, they also have a substantial impact on the new schedules which needs to fit their needs and work standards.

The detailed business map regarding the current decision-making process of the company is provided in Leggate (2016) at pg 314.

The drawback of the current method is that it requires many manual operations. Accordingly, the process becomes more complicated, and planners' main objective becomes generating feasible schedules rather than minimising cost, increasing efficiency or measuring the performance of new plans. The manual processes cause inefficiency with regards to time, as well.

In this research we are looking for solution methods that can improve the current system and proposed solution methods while meeting the demand by operating the tasks for the desired time, taking rules and regulations into account, managing changes with low costs and generating recovery schedules in short time (i.e. 10 minutes). We first represent the model which organises schedules from scratch. In Chapter 4, two mixed integer linear programming (MILP) formulation are presented to be able to obtain feasible crew schedules with minimum cost. These MILP models are named as *Task Based* (TB) and *Time Windows* (TW) model, in order. TB and TW was originally proposed in Leggate (2016). The performance analysis of TB was extended in Section 4.1 while TW was substantially revised as collaborative work in Section 4.2.

Chapter 4

Formulations and Heuristics

This section includes two MILP models for the problem described in Section 3 with the exact and heuristic solution methods. In addition to the model and solution method representation, a thorough analysis of these solution methods are provided.

Vessel crew scheduling problem in OSV's is computationally intractable real-life problems with a significant amount of binary and integer decision variables. As the nature of integer programming problems, an increase in the problem size effects the complete enumeration in an exponential pattern. Therefore to deal with VSC in OSVs efficiently, we suggest both MILP and Heuristics solution methods.

In this section, MILP models of Task Based (TB) and Time Windows (TW) formulations mentioned above and the recovery form of these formulations which are introduced in Leggate (2016), are presented in Section 4.1 and 4.2.2, respectively.

Section 4.3 gives the detailed information about the customised Heuristic algorithm and an elaborate performance analysis of the Heuristic, and MILP solutions for TB and TW formulations will follow in 4.3.2.

4.1 A Task Based Approximation

This section includes part of joint paper Leggate et al. (2017) which is the first paper to introduce the VSC in OSV's and emphasize the importance of recovery crew scheduling in this concept. The first author have investigated the given formulation, whereas I carried out the extended computational study and analysis held in the paper.

We have a wide set of roles which must be covered by crew members, and which

can be broadly divided into two categories: 1) "Marine crew" include roles such as captain, bridge and engineering crew, and must be covered at all times, even when a vessel is unassigned or undergoing maintenance; 2) "Project crew" vary depending on a vessel's assignment, e.g. diving crew when a vessel is engaged in a diving project, and also include crew working on deck such as riggers and deck foremen, who will not be required when a vessel is unassigned or undergoing maintenance. Vessels require only one of certain role types, such as a captain, but may require multiple diving or engineering roles to be covered.

We consider two types of crew: regular crew, denoted by the set $\mathcal{F} = \{1, ..., m\}$, and agency crew, represented with the index {m+1}. We also define the set of all employees, denoted by $\mathcal{E} = \{1, ..., m + 1\}$. Regular crew are permanent employees of the company, and include a subset of fixed contract crew, denoted by $\mathcal{G}$. Fixed contract crew are paid a salary to work a certain number of days at sea per year, and using this contracted number of days (as well as the number of days worked so far in the year), the company estimates for each employee $i \in \mathcal{G}$ the number of days this employee is expected to work in a given planning period, which is denoted by G_i. This quantity is used in order to estimate the costs associated with overtime and undertime, as follows: if employee $i \in \mathcal{G}$ works more than G_i days in the planning period, each additional day will be charged by the overtime rate of Φ_i; on the other hand, if this employee works less than G_i days in the planning period, then this wastage of salary will be charged by the effective rate per day denoted by Υ_i. The remaining regular crew are the so-called day rate crew, who are paid per day at sea with a rate up to 50% higher than the rate of the fixed contract crew. Agency crew {m+1} are outsourced by external agencies and are available at short notice, albeit more expensive as up to twice of day rate crew. In general, the company relies on regular employees, and uses agency crew only when absolutely necessary.

Crew availability for a specific role at a given time depends on a number of factors, including other commitments (e.g., training), unexpected unavailability (e.g., illness), their training and experience level, their nationality, and legal and contractual requirements. For example, ships registered under certain flags can employ only crew from certain nationalities, costly visas may be required in certain regions, and there might be undesirable combinations of certain nationalities. On the other hand, each employee $i \in \mathcal{E}$ has a maximum duration they can be assigned to work, denoted by Ω_i, and a minimum duration of rest, denoted by P_i. Notwithstanding individual differences stemming from nationalities or contractual specifics, regular working pat-

terns for the vast majority of crew is 4-on-4-off (4 weeks at sea followed by a 4 week rest), and the rest 5-on-5-off (often in remote regions) (European Union, 2003). We assume that the regular assignment pattern on each vessel is pre-determined, thus dividing each role into four- or five-week duty periods, or tasks. We assume there are n tasks in total, which is composed of the set of tasks which are to be carried out, denoted by $\mathcal{J}$, and the set of rest tasks, denoted by $\mathcal{N}$. It is possible that a task $j \in \mathcal{J} \bigcup \mathcal{N}$ can cover a number of consecutive roles, and each task is designed for a single employee, with a starting time S_j and duration D_j. Without loss of generality, we assume that the indices of all tasks in the set $\mathcal{J} \bigcup \mathcal{N}$ are ordered in the nondecreasing order of starting times, i.e., $S_{j-1} \leq S_j$. Crew changes take place each week on each vessel, affecting only a subset of the crew, and the cost of assigning employee $i \in \mathcal{E}$ to task $j \in \mathcal{J}$, denoted by C_{ij}, considers all penalties and financial costs (including day rates of day rate and agency crew).

Some of the projects run by our industrial partner require specialist knowledge, training or experience. We denote by $\mathcal{K}$ the set of these projects, and for any such project $k \in \mathcal{K}$, the company determines the minimum total experience required across the tasks in the project, denoted by H_k. In order to be able to calculate how much actual total experience is available in a project $k \in \mathcal{K}$ for a given allocation of crew, the company identifies the so-called experience score E_{ij} of employee $i \in \mathcal{E}$ for each task $j \in P_k$.

Before formally stating the mathematical formulation of the problem, we define next our decision variables. For each employee $i \in \mathcal{E}$ and task $j \in \mathcal{J} \bigcup \mathcal{N}$, the binary variable x_{ij} takes a value of 1 if employee i is allocated to task j, and 0 otherwise. For each fixed contract employee $i \in \mathcal{G}$, we define the continuous variables u_i and o_i to indicate the number of days under and over the guaranteed days expected in the planning period, respectively. Finally, for each employee $i \in \mathcal{E}$ and task $j \in \mathcal{J} \bigcup \mathcal{N}$, we define the variables w_{ij} and r_{ij} to represent the accumulated work (and rest, respectively) resource value for employee i once all tasks up to and including the task j have been considered, where w_{ij} is a continuous variable counting the number of consecutive working days whereas r_{ij} is a binary variable indicating whether the employee needs a rest at this point or not. In line with these variables and the ordered set of $\mathcal{J} \bigcup \mathcal{N}$, we also define the parameters w_{i0} and r_{i0}, where the former indicates the number of consecutive working days employee i has accumulated immediately prior to the start of the planning period, and the latter indicates whether employee i requires rest at the start of the planning period or not. Finally, in order to facilitate

the calculations of the accumulated work and rest resource value variables, we define the parameters W_j and R_j for each task $j \in \mathcal{J} \bigcup \mathcal{N}$, where the work resource value W_j is either set to the duration of the task j, i.e., $W_j = D_j$, if $j \in \mathcal{J}$, or to a sufficiently large negative number if j is a rest task, while the rest resource value R_j is either set to 1 if $j \in \mathcal{J}$, or to -1 otherwise.

We can now state the Task Based formulation (denoted as TB) as follows:

$$\min \sum_{i \in \mathcal{E}} \sum_{j \in \mathcal{J} \cup \mathcal{N}} C_{ij} x_{ij} + \sum_{i \in \mathcal{G}} (\Upsilon_i u_i + \Phi_i o_i) \tag{4.1}$$

$$\text{subject to:} \quad \sum_{i \in \mathcal{E}} x_{ij} = 1 \qquad \forall j \in \mathcal{J} \tag{4.2}$$

$$\sum_{j \in \mathcal{J} \cup \mathcal{N}} x_{ij} \leq 1 \qquad \forall i \in \mathcal{F} \tag{4.3}$$

$$\sum_{i \in \mathcal{E}} \sum_{j \in \mathcal{P}_k} E_{ij} x_{ij} \geq H_k \qquad \forall k \in \mathcal{K} \tag{4.4}$$

$$w_{i,j-1} + W_j x_{ij} \leq w_{ij} \qquad \forall j \in \mathcal{J} \cup \mathcal{N}, i \in \mathcal{F} \tag{4.5}$$

$$w_{ij} \leq \Omega_i \qquad \forall j \in \mathcal{J} \cup \mathcal{N}, i \in \mathcal{F} \tag{4.6}$$

$$r_{i,j-1} + R_j x_{ij} \leq r_{ij} \qquad \forall j \in \mathcal{J} \cup \mathcal{N}, i \in \mathcal{F} \tag{4.7}$$

$$u_i \geq G_i - \sum_{j \in \mathcal{J}} D_j x_{ij} \qquad \forall i \in \mathcal{G} \tag{4.8}$$

$$o_i \geq \sum_{j \in \mathcal{J}} D_j x_{ij} - G_i \qquad \forall i \in \mathcal{G} \tag{4.9}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \text{ s.t. } x_{ij} \text{ is defined} \tag{4.10}$$

$$r_{ij} \in \{0, 1\}, \; w_{ij} \geq 0 \qquad \forall j \in \mathcal{J} \cup \mathcal{N}, i \in \mathcal{F} \tag{4.11}$$

$$u_i, o_i \geq 0 \qquad \forall i \in \mathcal{G} \tag{4.12}$$

The objective function (4.1) minimises the sum of the direct costs of assigning employees to tasks and the costs incurred due to guaranteed days of fixed contract crew. Constraints (4.2) ensure each task is covered, and constraints (4.3) prevent an employee to be assigned two overlapping tasks, where the agency crew (indexed m+1) is not included since as many agency crew as needed are assumed to be available. The minimum experience required for certain projects is ensured by constraints (4.4), and employee's consecutive working period lengths are calculated using constraints (4.5), with constraints (4.6) enabling maximum permitted working durations. Similarly,

constraints (4.7) cover the minimum between-task rest period duration for employees, and finally, constraints (4.8) and (4.9) enable the calculation of under- and over-time for each fixed contract employee. We also note that in particular the constraints (4.4), (4.8) and (4.9) are specific to this maritime setting, whereas the remaining constraints can be found in various other transportation crew scheduling problems, expressed either implicitly or explicitly.

Although this model provides feasible schedules by meeting the regulations requirement fully, a penetrating cost function as explained in Section 4.2 with equation (4.15) is needed for a better planning. Apart from the cost defined in objective function, there is a fundamental difference in the selection of decision variables (x_{ij}) which shows allocation of employee i depending only on the role whereas in Time Windows approach, same decision variable is represented as (x_{ijt}) which is determined based on both week and role for each employee. This approach brings more possible combinations for alternative schedules.

4.1.1 Extending Task Based Model to a Recovery-type Formulation

The nature of our particular maritime problem requires frequent changes on the existing schedule in light of new information about crew availabilities or vessel requirements. To accommodate this, we first define a binary $(m+1) \times |\mathcal{J}|$ matrix $\mathbf{X}^*$ indicating assignments of the current schedule. Then, we define new binary decision variables y_{ij} to keep track of changes from the current schedule, which is 1 if there is a change to employee i's schedule with respect to task j, and 0 otherwise. For notational simplicity and preserving our previously defined formulation, we note that x_{ij} is now a dependent variable and can be written in terms y_{ij} and x^*_{ij}, where $x_{ij} = x^*_{ij} - y_{ij}$ if $x^*_{ij} = 1$ or $x_{ij} = x^*_{ij} + y_{ij}$ if $x^*_{ij} = 0$. This is equivalent to:

$$x_{ij} = x^*_{ij} + (1 - 2x^*_{ij})y_{ij}; \forall i \in \mathcal{E}, \forall j \in \mathcal{J} \cup \mathcal{N} \tag{4.13}$$

As we focus now on changes made to an existing schedule, the cost of changing an employee's assignment to a task (including e.g. costs/savings with respect to wage and transportation) becomes relevant. Hence, we define two additional parameters, C'_{ij}, the cost (or saving if $C'_{ij} < 0$) of changing the assignment of employee i with respect to task j, and Ξ_i, the additional overtime/undertime costs for employee $i \in \mathcal{G}$

with respect to $\mathbf{X}^*$. Then, the objective function is as follows:

$$\min \sum_{i \in \mathcal{E}} \sum_{j \in \mathcal{J} \cup \mathcal{N}} C'_{ij} y_{ij} + \sum_{i \in \mathcal{G}} (\Upsilon_i u_i + \Phi_i o_i - \Xi_i) \tag{4.14}$$

Finally, we formally state our formulation for recovery form of TB model denoted as (RF-TB), as follows: $\min_{y,r,w,u,o}$ $((4.14)|(y,r,w,u,o) \in X^{RF})$ and $X^{RF-TB} = \{(y,r,w,u,o)|$ $(4.2)-(4.9)$, $(4.11)-(4.13), y \in \{0,1\}^{|\mathcal{F}|\times|\mathcal{J}|}\}$.

In the following section, the data is described in detail and extended computational study takes place for RF-TB following the section articulating the data generation process.

4.1.2 Design of Computational Study of Task Based Method

Although access to real problem instances was not possible due to data confidentiality, our industrial partner was able to provide us several specially identified key parameters and possible values or value ranges for these parameters. The company was also interested to investigate the effect of such parameters on solving these problems, and this process enabled us to generate randomized but realistic data sets covering all possible combinations. We used a full-factorial design with different number of levels involving the following four experimental factors identified by the company, which were varied across all instances: i) The probability that an employee is available on a given day, which is dependent on their availability the previous day (for which we define the parameters p (and q), i.e., the probability of an employee being unavailable on a certain day given they were unavailable (available) on the previous day); ii) Use of a probability reduction factor, $r\,(d)$, that increases uncertainty about the availability of an employee; iii) A disruption penalty K (which is split into K_N for **N**ear-term (i.e. 4 weeks), and K_L for **L**ong-term); iv) An agency penalty factor K_{AG}. The levels for each of these factors is listed in Table 4.1, and the interested reader can refer to VCS-data (2015) for the data of the resulting 240 instances. We note that these instances concern problems with ship captains only in consistency with the approach used by the company solving separate problems for each employee category, where the number of employees was set to 48 in line with the biggest problems of the company with 40 vessels and 13 weeks planning horizon. Moreover, in order to provide extensive computational results in Section 4.1.2.1, we generated 240 very large scale instances with the number of captains doubled to 96

by replicating the data of the original 240 instances.

Table 4.1: Levels used for the full factorial design

Factor	Set of levels
p	{0.2, 0.5, 0.8}
$r\,(d)$	{Use, Not Use}
(K_N, K_L)	{(1,1), (2,1) ,(2,2), (5,1), (5,2), (5,5), (10,1), (10,2), (10,5), (10,10)}
K_{AG}	{1,2,5,10}

As discussed in Section 3 , the primary concern of the company is to find feasible solutions quickly. Hence, rather than investigating how long the optimal solution would take to find, the aim of the computational tests is to discover the number and quality of solutions found within a practically acceptable time limit. Since the planners may run a model multiple times altering settings, it was agreed with the company to test two time limit settings, namely 2 and 10 minutes. In order to provide comprehensive computational results, we also solve the test instances that could not be solved in 10 minutes with a time limit of 1 hour. All test runs were carried out on a Dell Optiplex 790 PC (Windows 7 32-bit, Intel Core i5 3.10 Ghz, 4GB RAM), and the formulations were implemented and tested using FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7).

For the cost-minimisation approach, our preliminary tests suggested changing the default settings to a maximum of 30 rounds of cover and 10 rounds of Gomory cuts, with an acceptable gap of 5% set as a cutoff.

4.1.2.1 Cost-minimisation results

In order to test the cost minimisation model, we solved 240 problem instances of the data set VCS-data (2015) with 48 captains as well as the extended 240 instances with 96 captains using 2-min, 10-min and 1 hour time limits. We note that a typical instance contains around 23 thousand rows and 17 thousand columns for problems with 48 captains, and 89 thousand rows and 67 thousand columns for problems with 96 captains. We present overall results in Table 4.2, where the number of test instances that were completed in 2-min/10-min/1-hour runs are noted in columns "Opt." (for optimal solutions) and columns "$< 5\%$" (for instances that were stopped due to gap in a run being less than the acceptable gap of 5%). In addition, the columns "Ave." indicate the average gaps and the columns "Med." indicate the

median gaps for all 240 instances in each set, where we calculated the gaps using the overall best bound for each instance, i.e., the bound obtained from the 1-hour run, rather than the bound obtained in each run with different time limits, in order to provide a clear comparison of solution qualities between different lengths of runs.

As Table 4.2 clearly indicates, the biggest instances the company deals with in practice, i.e., those with 48 captains, can be in general solved very effectively. Even using the 2-min time limit, only 13 out of 240 instances cannot be solved within 5% of optimality, and 2 of these instances can be brought within 5% of optimality using the 10-min time limit. This is certainly encouraging for the practical use of the approach, as only 11 out of 240 instances ($\approx 4.6\%$ of the total) do not achieve the desired optimality gap of 5%. On the other hand, extending the time limit to 1-hour does not provide any benefit for these instances, only with marginal improvement of gaps for the remaining 11 instances (and no further instance brought within 5% of optimality). Next, we discuss the instances with 96 captains that provide further insights with regards to the computational capability of the models proposed for large-scale problems. Although these significantly bigger problems are naturally more challenging, resulting in average percentage gaps of 33.04%, 9.81% and 9.70% for 2-min, 10-min and 1-hour runs, respectively, the proposed models still remain practically effective, indicating further possibility for use even in case of significant growth of the company, such as in case of a merger. As results indicate, 149 out of 240 instances ($\approx 60.1\%$ of the total) can achieve the desired optimality gap of 5% in a 10-min run. Although extending the time limit to 1-hour helps to achieve the desired optimality gap for 5 more instances, the improvement of gaps remain marginal, similar to our previous experience with the 48 captain instances. In addition, the histograms in Figure 4.1 provide a breakdown of solutions with respect to gaps for all these tests. These figures present a better understanding of the improvement of gaps with longer computational times, though the improvements remain marginal, in particular for the case of 48 captains.

The results obtained so far also motivated us to further investigate challenging test instances with extensive computational runs. Therefore, we have applied a 6 hour time limit and executed the approach on the 5 instances with 48 captains that could not be solved to optimality in 1 hour and had a gap of 6% or higher, in order to ensure not to be too close to the 5% gap cutoff. As the results in Table 4.3 indicate, these most challenging 5 instances improved only marginally between 1-hour to 6-hour runs, the best performing instance (R205) having still a gap of 6.1%. Our

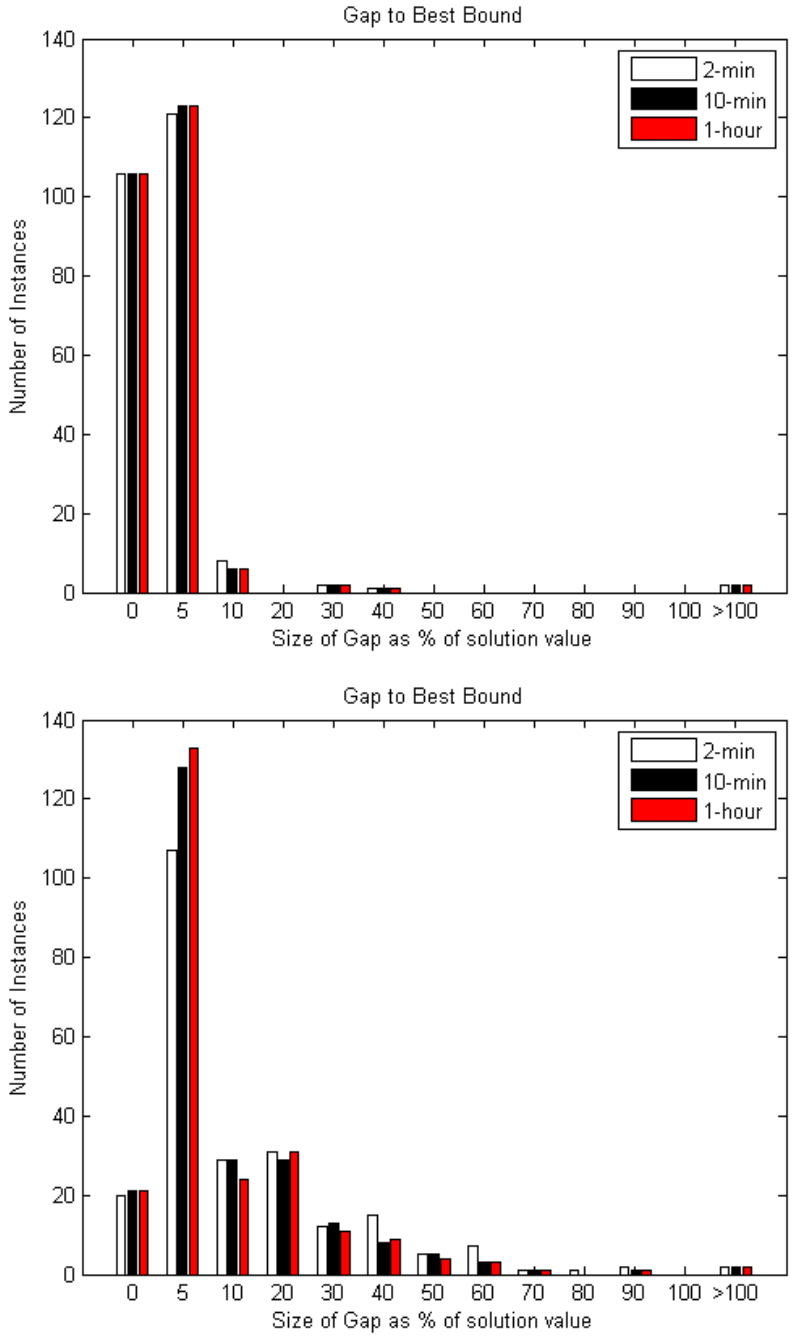

Figure 4.1: Gaps found in 2-min, 10-min and 1-hour cost-min runs for 48 captains (top) and 96 captains (bottom), based on best known bounds. x-axis indicates the integrality gaps, and y-axis indicates number of instances in each bracket.

Table 4.2: Cost minimisation results for instances with 48 and 96 captains.

	48 captains				96 captains			
Time	Opt.	$< 5\%$	Ave.	Med.	Opt.	$< 5\%$	Ave.	Med.
2-min	106	121	2.47%	0.27%	20	107	33.04%	4.76%
10-min	106	123	2.42%	0.27%	21	128	9.81%	4.22%
1-hr	106	123	2.41%	0.27%	21	133	9.70%	4.22%

preliminary tests with some of the unsolved 96 captain instances indicated a similar pattern of very limited improvement over extended runs, though it is also worth to note that some of these large scale instances experienced memory issues.

Table 4.3: 6-hour extensive runs for challenging instances with 48 captains.

Instance	Opt.(1-hour)	Opt.(6-hour)	Best Bound (6-hour)
R001	1827	1801	1029.25
R005	20.5	20.5	-50.515
R056	22620.8	22618.8	18822.11767
R081	-409	-409	-507.333
R205	-394	-396	-424.008

Finally, we note that we will discuss in Section 4.1.2.2 the effect of parameter values, using statistical analysis tools such as Kruskal-Wallis test.

4.1.2.2 Effect of parameter values on results

As discussed before, several parameters were varied for the generation of data instances. In this part, we investigate the effect of changing these parameter values using "Analysis of Variance". We used the ANOVA and F-test where possible; however, we note that for some of the parameters investigated, the required assumption of normal distribution did not necessarily hold and hence, the non-parametric equivalent "Kruskal-Wallis test" was carried out instead. Table 4.4 shows the p-values for these tests, with respect to the varied parameters and the key output measures. We grouped the outputs in 3 main groups depending on the main objective functions that we run for experimental results. In Table 4.4 these 3- main groups and the depending performance measurements are given on the y-axis and the parameters are given on the x- axis. The type of applied tests is also stated in the x-axis of Table 4.4.

The null hypothesis that we use for testing the parameter values effect is that there is no difference between different levels of a given parameter. Accordingly an asterisk next to a p-value indicates that, at the 5% significance level, there is a significant difference between the values of the given key output measure for instances generated using different values of the given parameter. For example, where the '# changes - 2mins (only)' row intersects with the 'p and q' column the value of $0.004*$ indicates that there is a significant difference between the number of changes in the min-cost solution between instances which were generated using different values of availability probabilities p and q. Those parameters which returned a significant result as shown in Table 4.4 required further investigation to determine in what way the change in parameter values was significant. We note that the quantity $\bar{K}$ is an average of the Near and Long disruption factors K_N and K_L, weighted according to the number of weeks to which each apply (i.e. 4 and 9 weeks in a 13-week planning period).

Further investigation allows us to make some observations about how the values of these parameters may influence the running of the models in practice. For example, it can be seen from the table that the factors relating to crew availability have a significant influence on specific outputs pertaining to change. As might be expected, we can conclude that if in reality, crew absences tend to be longer but less frequent, then we would wait to see fewer changes being found in the 2-min cost-minimisation.

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 have a considerably broader influence. If the company wishes to apply disruption factors (i.e. penalties for making changes to the schedule), then, in general, we expect the effects to be: i) shorter run times in both approaches, ii) fewer iterations for change-minimisation, iii) fewer changes in 2-min cost-minimisation, iv) smaller cost gaps both for the cost-minimisation approach.

The additional penalty for using agency crew has a similar influence concerning running time, iterations and gaps in cost, but has no significant effects on the number of changes. It should be noted that when the agency penalty is not applied, especially extended run times and large percentage gaps in cost are observed.

Table 4.4: Results of ANOVA and Kruskal-Wallis tests for influence of parameters used in instance generation

	Outputs of interest		K-W or ANOVA	Parameter					
				p and q	use $r(d)$?	K_N	K_L	$\bar{K}$	K_{AG}
Cost-min settings	Run time	10mins	K-W	0.283	0.137	0.006*	0.000*	0.000*	0.201
		2mins	K-W	0.217	0.351	0.003*	0.000*	0.000*	0.008*
	# changes	2mins (only)	ANOVA	0.004*	0.000*	0.000*	0.000*	0.000*	0.960
	Gap found in each run	Actual; 10mins	K-W	0.115	0.383	0.168	0.000*	0.010*	0.560
		Actual; 2mins	K-W	0.701	0.430	0.024*	0.000*	0.000*	0.015*
		Adjusted; 10mins	K-W	0.458	0.576	0.063	0.074	0.213	0.208
		Adjusted; 2mins	K-W	0.538	0.825	0.432	0.011*	0.104	0.056
	Gap to best known bound	Actual; 10mins	K-W	0.158	0.152	0.227	0.001*	0.033*	0.767
		Actual; 2mins	K-W	0.341	0.231	0.169	0.000*	0.001*	0.106
		Adjusted; 10mins	K-W	0.434	0.636	0.049*	0.054	0.151	0.032*
		Adjusted; 2mins	K-W	0.592	0.975	0.158	0.027*	0.277	0.101
Change-min settings	Run details	Running time	K-W	0.101	0.528	0.045*	0.001*	0.023*	0.000*
		Iterations	ANOVA	0.001*	0.000*	0.003*	0.125	0.037*	0.000*
	# of changes	First sol found	ANOVA	0.000*	0.002*	0.515	0.808	0.941	0.780
		Last sol found	ANOVA	0.002*	0.002*	0.003*	0.007*	0.010*	0.561
	Cost of soln.	First min-chng sol	ANOVA	0.280	0.393	0.001*	0.012*	0.005*	0.000*
		Chpst min-chng sol.	ANOVA	0.212	0.198	0.002*	0.029*	0.011*	0.000*
	% gap to best known bound	Chpst ovrll	K-W	0.187	0.481	0.010*	0.000*	0.000*	0.000*
		Adjusted chpst ovrll	K-W	0.446	0.846	0.308	0.001*	0.018*	0.000*
Comparison with 2-min min-cost	% diff in # of changes	First sol found	ANOVA	0.000*	0.002*	0.008*	0.000*	0.000*	0.868
		Last sol found	K-W	0.576	0.830	0.000*	0.000*	0.000*	0.413
	% diff in cost	First min-chng sol	K-W	0.439	0.074	0.002*	0.093	0.024*	0.000*
		Chpst min-chng sol	K-W	0.013*	0.122	0.008*	0.064	0.032*	0.000*
		Chpst ovrll	K-W	0.413	0.676	0.181	0.014*	0.168	0.000*

4.1.3 Summary of Findings from Task Based Model

By presenting the TB model as a solution method, a vessel crew scheduling problem about a large global company operating Offshore Supply Vessels (OSVs) providing services particularly to the oil and gas industry has been introduced in the literature with providing the first computational results in this area, (Leggate et al., 2017). Discussions with our industrial partner have also allowed us to present formulations that could serve as part of a decision support tool, allowing the planners to find better quality solutions in real time. The current practice for crew scheduling at our industrial partner involves manual processes carried out on various spreadsheets. Because of the difficulties involved in the process, and the lack of an automated tool to aid decision making, the primary concern is to obtain feasible solutions. Even this can be difficult, in particular when there is time pressure. Hence, TB approach effectively provides feasible, low-cost crew schedules when necessary.

A critical aspect of the TB model is the implicit assumption that the time over which a role requires to be covered can be divided into four- or five-week blocks. Although in the practical setting, employees could potentially be asked to work more irregular patterns even if not desirable. Accordingly, the TW model which is shown in Section 4.2 and 4.2.1 meet the need of a more sophisticated model for this problem. Even the TB model can be solved quickly, TW model, which is explained in Section 4.2 and 4.2.1 for generating schedules from scratch and finding recovery schedules respectively, increases the flexibility for the planners as well as the potential quality of schedules created, albeit with a need for more customized solution methodologies in order to obtain solutions in real time.

After this analysis, we explain the other solution methods and analyse their performances in the following sections. Section 4.2 shows another MILP model, which is called Time Windows (TW), to solve VCS in OSV's and provides further insights regarding the performance of TW model and compares the efficiency of these two models.

4.2 A Basic Scheduling Formulation: Time Windows Model

Since the problem faced by the company has been described, we now present the mixed integer linear programming formulation of the problem. This formulation was revised collaboratively and presented in Leggate (2016), initially. The models presented Leggate (2016) in is the first study for the vessel crew scheduling in OSV. We also put down this problem description and mathematical model as a basis for this thesis. Throughout the study, we develop new solution methods, revisit the formulation in Section 5.2 and compare our findings by considering Leggate's study as a reference point.

This model holds the assumption that scheduling is designed for the entire planning period from scratch, and there is no partial schedule in existence prior to the start of the scheduling process.

Sets and indices, parameters and decision variables used primarily during the thesis are stated below.

Sets and indices

t - time index in weeks; $t \in \{1, \ldots, T\}$.

$m+1$ - index of the agency crew(s).

m - number of regular crews.

E - set of crew combination of regular and agency crew; $E = \{1, \ldots, m+1\}$.

E_R - set of regular crew; $E_R = \{1, \ldots, m\}$.

G - set of fixed contract crew; $G \subseteq E_R$.

K - set of vessels which are being planned for during the planning horizon.

J - set of all roles being planned for over the planning period; $J = \{1, \ldots, n\}$.

V_k - set of roles to be planned for on board vessel $k \in K$; $V_k \subseteq J$; $\bigcup_{\forall k} V_k = J$; $V_k \cap V_{k'} = \emptyset \ \forall k, k' \neq k$.

λ - index to indicate an employee's number of consecutive working weeks.

Parameters

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

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

c_i^O - *over-time* paid per week that employee $i \in G$ works in excess of g_i, $\forall i \in G$.

c_i^U - effective *under-time* rate for employee $i \in G$, i.e. the weekly amount paid to employee i which is wasted if the employee works less than g_i in the year, $\forall i \in G$.

c_{ikt}^B - cost of employee i boarding vessel k prior to performing a task on board in period t, $\forall i \in E, k \in V_k, t \in \{1, \ldots, T\}$.

c_{ikt}^D - cost of employee i departing vessel k prior to period t (i.e. having completed a task on board in period $t-1$), $\forall i \in E, k \in V_k, t \in \{1, \ldots, T\}$.

c_{ijt}^W - cost *directly* associated with employee i carrying out role j in period t , $\forall i \in E, j \in J, t \in \{1, \ldots, T\}$.

$c_{\lambda ijt}^L$ - additional cost of employee i carrying out same role j for a λ^{th} consecutive week in period t, $\forall i \in E, j \in J, t \in \{1, \ldots, T\}$.

a_{jt} - indicates if role j requires an employee assigned to it in period t, $\forall i \in E, t \in \{1, \ldots, T\}$;

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

e_{ijt} - indicates whether an employee is eligible and available to carry out role j in period t,$\forall i \in E, j \in J, t \in \{1, \ldots, T\}$;

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

w_i^{max} - 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} - legal / contractual upper limit on the number of consecutive weeks at sea that an individual agency employee can be assigned to role j, $\forall j \in J$.

ρ_i - minimum length (in weeks) of the rest period to which crew $i \in E_R$ is entitled following a spell offshore.

s_{ik} - indicates assignment of employee i at the start of the planning period, $\forall i \in E_R, k \in V_k$;

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

$s_{m+1,k}$ - number of agency employees who are on board vessel k immediately prior to period 1, $\forall i \in E_R$.

σ_j - indicates assignment of an agency employee at the start of the planning period;

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

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

α_{j0} - number of consecutive weeks an agency employee has been assigned to role j prior to period t, $\forall j \in J$.

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

Decision Variables

x_{ijt} - the main decision variable, indicating each employee's assignment,$\forall i \in E, j \in J, t \in \{1, \ldots, T\}$;

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

o_i - estimated amount of *over-time* (in weeks) that crew $i \in G$ is expected to work in the financial year.

u_i - estimated amount of *under-time* (in weeks) that crew $i \in G$ is expected to work in the financial year.

b_{ikt} - indicates crew i boarding vessel k at week t, $\forall i \in E_R, k \in V_k, t \in \{1, \ldots, T\}$;

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

$b_{m+1,kt}$ - *number* of agency employees which must board vessel k prior to performing an assignment in period t, $\forall k \in V_k, t \in \{1, \ldots, T\}$. .

β_{jt} - indicates agency crew boarding to carry out a specific role j, $\forall j \in J, t \in \{1, \ldots, T\}$;

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

d_{ikt} - indicates crew i departing from vessel k at week t, $\forall i \in E_R, k \in V_k, t \in \{1, \ldots, T\}$;

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

$d_{m+1,kt}$ - *number* of agency employees which should 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;

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

w_{it} - number of consecutive weeks work offshore that employee $i \in E_R$ has been assigned up to and including week t, $\forall i \in E_R, t \in \{1, \ldots, T\}$.

α_{jt} - number of consecutive weeks that an agency employee has been assigned to role j up to an including week t, $\forall j \in J, t \in \{1, \ldots, T\}$.

r_{it} - 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 same role j, $\forall i \in E, j \in J, t \in \{1, \ldots, T\}$;

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

Using this notation, it is now possible to express the full formulation for VSC for OSV's using time windows approach:

min:

$$\sum_{i=1}^{m+1} \sum_{k \in V_k} \sum_{t=1}^{T} \left(c^B_{ikt} b_{ikt} + c^D_{ikt} d_{ikt}\right) + \sum_{i \in G} \left(c^U_i u_i + c^O_i o_i\right) + \sum_{i=1}^{m+1} \sum_{j \in J} \sum_{t=1}^{T} \left(c^W_{ijt} x_{ijt} + \sum_{\forall \lambda} c^L_{\lambda ijt} l_{\lambda ijt}\right) \tag{4.15}$$

subject to:

$$\sum_{i=1}^{m+1} e_{ijt} x_{ijt} = a_{jt} \qquad \forall j, t \tag{4.16}$$

$$\sum_{j \in J} x_{ijt} \leq 1 \qquad \forall t, i \in E_R \tag{4.17}$$

$$b_{ik1} \geq \sum_{j \in V_k} x_{ij1} - s_{ik} \qquad \forall k, i \in E_R \tag{4.18}$$

$$b_{ikt} \geq \sum_{j \in V_k} x_{ijt} - \sum_{j \in V_k} x_{ij,t-1} \qquad \forall k, i \in E_R, t \in \{2, .., T\} \tag{4.19}$$

$$d_{ik1} \geq s_{ik} - \sum_{j \in V_k} x_{ij1} \qquad \forall k, i \in E_R \tag{4.20}$$

$$d_{ikt} \geq \sum_{j \in V_k} x_{ij,t-1} - \sum_{j \in V_k} x_{ijt} \qquad \forall k, i \in E_R, t \in \{2, .., T\} \tag{4.21}$$

$$\beta_{j1} - \delta_{j1} = x_{m+1,j1} - \sigma_j \qquad \forall j \tag{4.22}$$

$$\beta_{jt} - \delta_{jt} = x_{m+1,jt} - x_{m+1,j,t-1} \qquad \forall j, t \tag{4.23}$$

$$b_{m+1,kt} \geq \sum_{j \in V_k} \beta_{jt} \qquad \forall k, t \tag{4.24}$$

$$d_{m+1,kt} \geq \sum_{j \in V_k} \delta_{jt} \qquad \forall k, t \qquad (4.25)$$

$$u_i \geq g_i - \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} x_{ijt} \right) \qquad \forall i \in G \qquad (4.26)$$

$$o_i \geq \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} x_{ijt} \right) - g_i \qquad \forall i \in G \qquad (4.27)$$

$$w_{it} \geq w_{i,t-1} + \sum_{j \in J} x_{ijt} - w_i^{max} \left(1 - \sum_{j \in J} x_{ijt} \right) \qquad \forall t, i \in E_R \qquad (4.28)$$

$$w_i^{max} l_{\lambda ij1} \geq w_{i,0} - w_i^{max}(1 - x_{ij1}) + x_{ij1} - (\lambda - 1) \qquad \forall j, i \in E_R, \lambda \qquad (4.29)$$

$$w_i^{max} l_{\lambda ijt} \geq w_{i,t-1} - w_i^{max}(1 - x_{ijt}) + x_{ijt} - (\lambda - 1) \qquad \forall j, t \in \{2, .., T\}, i \in E_R, \lambda \qquad (4.30)$$

$$\alpha_{jt} \geq \alpha_{j,t-1} + x_{m+1,jt} - \alpha_j^{max} \delta_{jt} \qquad \forall j, t \qquad (4.31)$$

$$\alpha_{jt} \geq x_{m+1,jt} \qquad \forall j, t \qquad (4.32)$$

$$\alpha_j^{max} l_{\lambda,m+1,jt} \geq \alpha_{jt} - (\lambda - 1) \qquad \forall j, t, \lambda \qquad (4.33)$$

$$r_{it} \geq r_{i,t-1} - \left(1 - \sum_{j \in J} x_{ijt} \right) \qquad \forall t, i \in E_R \qquad (4.34)$$

$$r_{it} \geq (\rho_i - 1) \sum_{k \in K} d_{ikt} \qquad \forall t, i \in E_R \qquad (4.35)$$

$$\rho_i \left(1 - \sum_{j \in J} x_{ijt} \right) \geq r_{i,t-1} \qquad \forall t, i \in E_R \qquad (4.36)$$

$$x_{ijt} \in \{0, 1\} \qquad \forall i, j, t \qquad (4.37)$$

$$l_{\lambda ijt} \in \{0, 1\} \qquad \forall i, j, t, \lambda \qquad (4.38)$$

$$b_{ikt}, d_{ikt} \in \{0, 1\} \qquad \forall k, t, i \in E_R \qquad (4.39)$$

$$\beta_{jt}, \delta_{jt} \in \{0, 1\} \qquad \forall j, t \qquad (4.40)$$

$$b_{m+1,kt}, d_{m+1,kt} \geq 0 \text{ and integer} \qquad \forall k, t \qquad (4.41)$$

$$u_i, o_i \geq 0 \qquad \forall i \in G \qquad (4.42)$$

$$w_{it}, r_{it} \geq 0 \qquad \forall i, t \qquad (4.43)$$

$$\alpha_{jt} \geq 0 \qquad \forall j \in J, t \qquad (4.44)$$

The objective function aims to minimise the cost of the crew. The sources of cost are boarding and departing of crews on vessels, working cost when they are assigned to the tasks and penalty costs raised from consecutive work on the same vessel without resting and also working more or less than the guaranteed time. Constraint

(4.16) ensures that the required number of crews are assigned to the vessels for the specified time windows while checking the eligibility of employees. Accordingly, constraint (4.17) prevents the overlapping assignments for crew for regular crew. Constraint (4.17) is not valid for agency crew since they can be assigned as needed. Boarding and departing of regular crews are determined by their allocations for consecutive weeks. While constraints (4.18) - (4.19) decide whether employees board or not, constraints (4.20) - (4.21) carry the same idea for departing. For agency crew boarding and departing need to be defined differently than the regular employees since agency crew appears at the time of need and leave after carrying out the role. Constraints (4.22) and (4.23), which allow 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. Constraints (4.24) and (4.25) calculate the number of agency crew boarded and departed for each vessel and each week, in order to evaluate the boarding and departing cost of agency crew, respectively. The legal and contractual obligations are stated starting with inequalities (4.26) and (4.27) that calculates the respective estimated number of weeks under- or over-utilised each fixed contract employee will be in the year. By defining $\lambda \in \{1, .., w_i^{max}\}$ through the model, the amount of consecutive work done by each employee from one period to the next needs to be calculated by constraint (4.28) and the time length spent at sea by an employee is determined by constraint (4.30) in order to figure out potential penalty cost from long work. 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. Constraint set (4.31) carries the similar calculations with constraint (4.30) for the agency employees from one period to the next but on a task-by-task basis, rather than employee-by-employee. To link the penalty, cost constraints (4.33) are employed. As keeping track of the continuous work has importance with regards to cost and regulations, the rest periods needs to be observed, too. The constraint (4.34) gives the decrease in weeks for necessary rest time when they are not assigned to any roles in that period. Constraint set (4.35) resets the required number of rest weeks and calculates the required rest period between departing from offshore work and before boarding for any other one. With the help of these constraints, the constraint set (4.36) prevents the assignment of employees to any roles if they are due to rest. Finally, constraints (4.37)-(4.44) states the characteristic of each decision variable (i.e. non-negative,

integer, binary).

4.2.1 Recovery Formulation of Time Windows Model

The formulation which is given in Section 4.2 is designed for generating schedules from scratch with minimum cost for the desired planning period. Although it is essential to have a plan in advance, in OSV's, there is a high demand for modifications in the schedules. According to the industrial partners, it is expressed as the most prominent issue to have a new schedule which covers the updated demand and/or crew availability in a short time with minimum cost.

In order to meet this demand, a model is created on a rolling basis, making sure the requirements for the coming 13 weeks as it is expressed by the company are kept up to date at all times. The fundamental steps of the process can be considered as follows:

1. At the end of the week $t-1$, the planners will have created a feasible schedule covering the next thirteen weeks, from week t to week $t+12$. Meanwhile, the crewing team will have confirmed the logistics for the crew changes in the coming four weeks, i.e. from week t to week $t+3$.

2. At the start of the new week (note that a *week* in planning terms need not necessarily start on Monday), the planners will receive updated information about vessel requirements, and may also have new details on crew availability. The schedules at this point cover the next 12 weeks (from $t+1$ to $t+12$), with logistics having been arranged for the next three weeks (from $t+1$ to $t+3$); however, some of these assignments may have been rendered redundant or infeasible by the new information received.

3. The planners work for the coming week can be broken down into three parts:

   (a) To make any essential changes to the schedule for weeks $t+1$ to $t+4$ (note that during this week, the crewing team will be making travel arrangements for the week $t+4$). As travel plans have already been made, any changes to the schedule for this period must involve consultation with the crewing team. It follows that it is usually more expensive, and hence less desirable, to make changes to the schedule during this period - it should therefore only be done if essential.

(b) To make any changes which may be necessary to the existing schedule for weeks $t+5$ to $t+12$ due to modifications to vessel requirements or crew availabilities or to account for knock-on effects of other changes.

(c) To schedule the crew for the currently unscheduled period $t+13$.

In order to model this problem, we firstly consider the partial schedule which is known at the start of the planning step, having been decided in the previous week. This can be described by the final values decision variables discussed in Section 4.2: x^*_{ijt}, o^*_i, u^*_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 week, and will be used to help evaluate the changes which are required to be made to the schedule. We will then define a corresponding new set of decision variables which will represent the new schedule which is to be created in the current week:

$$\hat{x}_{ijt}, \hat{o}_i, \hat{u}_i, \hat{b}_{ikt}, \hat{\beta}_{jt}, \hat{d}_{ikt}, \hat{\delta}_{jt}, \hat{w}_{it}, \hat{\alpha}_{jt}, \hat{r}_{it}, \hat{l}_{\lambda ijt}$$

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 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

$$\hat{x}_{ijt} = x^*_{ijt} + (1 - 2x^*_{ijt})x^{\pm}_{ijt} \quad \forall i \in E, j \in J, t \in t \in \{1, \ldots, T\} \tag{4.45}$$

Similarly, we can also define the rest of the decision variables have related cost coefficients in objective function by using the same formulation in (4.45) apart from

$\hat{o}_i$, $\hat{u}_i$. Since under-time and over-time are not binary variables, the changes to these can be calculated differently, and in particularly it should be noted that the change variables $u_i^{\pm}$ and $o_i^{\pm}$ could take negative values. They are defined as follows:

$$u_i^{\pm} = \hat{u}_i - u_i^* \quad i \in G \tag{4.46}$$

$$o_i^{\pm} = \hat{o}_i - o_i^* \quad i \in G \tag{4.47}$$

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 $w_{it}^{\pm}$, $\alpha_{jt}^{\pm}$, $r_{it}^{\pm}$.

As the focus of our problem is now on making changes to an existing schedule, the cost of making an assignment is now less important; instead, we must be concerned with the effect of *changing* an assignment. The objective may be defined as to minimise the number of changes (and therefore minimise disruption) required to achieve the new schedule; alternatively, the company may prefer to minimise the cost of the changes, where cost can incorporate a number of factors. These would include financial costs such as changes in wage payments and transportation costs, and may also include penalty costs in order to reflect the undesirable nature of some options (e.g. changing an assignment at very short notice). Note that the nature of the transportation arrangements means that some costs may not be recoverable if, for example, an employee has been booked on a flight but it then transpires that they are not required for that assignment. It is also important to highlight that cost of change can be negative for some case in this context.

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_{ijt}^W - cost of changing whether or not employee $i \in E$ works in role j in period t.

ϕ_{ikt}^B - cost of changing whether or not employee $i \in E_R$ boards vessel k in advance of period t.

ϕ_{ikt}^D - cost of changing whether or not employee $i \in E_R$ leaves vessel k in advance of period t.

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

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

$\phi^{L}_{\lambda ijt}$ - 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.

Using the quantities defined above, and with other quantities remaining as defined in Section 4.2 earlier, we can next formulate the recovery-type problem. The new cost contains change cost for related cost source and the appointed decision variables depending on existence of change as different than the formulation in 4.2.
min:

$$\sum_{i\in E_R}\sum_{\forall k,t}\left(\phi^{B}_{ikt}b^{\pm}_{ikt}+\phi^{D}_{ikt}d^{\pm}_{ikt}\right)+\sum_{\forall j,t}\left(\phi^{B_A}_{jt}\beta^{\pm}_{jt}+\phi^{D_A}_{jt}\delta^{\pm}_{jt}\right)+$$
$$\sum_{\forall i,j,t}\left(\phi^{W}_{ijt}x^{\pm}_{ijt}+\sum_{\forall\lambda}\phi^{L}_{\lambda ijt}l^{\pm}_{\lambda ijt}\right)+\sum_{i\in G}\left(c^{U}_{i}u^{\pm}_{i}+c^{O}_{i}o^{\pm}_{i}\right) \tag{4.48}$$

The objective (equation 4.48) states that the total cost of all changes should be minimised. Notice that the agency boarding and departing variables for individual tasks ($\beta^{\pm}_{jt}$ and $\delta^{\pm}_{jt}$) are used rather than those counting the total number of agency crew boarding and departing as in (4.15) 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.

The same constraints used in 4.2 has been applied for the recovery form with the renewed version of decision variables such as $\hat{x}_{ijt}$ instead x_{ijt}. Since x_{ijt} became a parameter as current solution for recovery schedule, this rule is valid for $\hat{x}_{ijt}$, $\hat{o}_i$, $\hat{u}_i$, $\hat{b}_{ikt}$, $\hat{\beta}_{jt}$, $\hat{d}_{ikt}$, $\hat{\delta}_{jt}$, and $\hat{l}_{\lambda ijt}$. The linking equation are used for defining these decision variables for recovery schedules similar to 4.45. Finally, there are binary variable and sign constraints as given below.

$$\hat{x}_{ijt}, x^{\pm}_{ijt} \in \{0,1\} \qquad \forall i \in E, j \in J, t \in \{1,\ldots,T\} \tag{4.49}$$
$$\hat{l}_{\lambda ijt}, l^{\pm}_{\lambda ijt} \in \{0,1\} \quad \forall i \in E, j \in J, t \in \{1,\ldots,T\}, \lambda \in \{1,\ldots,w^{max}_{i}\} \tag{4.50}$$

$$\hat{b}_{ikt}, b^{\pm}_{ikt}, \hat{d}_{ikt}, d^{\pm}_{ikt} \in \{0,1\} \qquad \forall kinK, t \in \{1,\ldots,T\}, i \in E_R \tag{4.51}$$

$$\hat{\beta}_{jt}, \beta^{\pm}_{jt}, \hat{\delta}_{jt}, \delta^{\pm}_{jt} \in \{0,1\} \qquad \forall j \in J, t \in \{1,\ldots,T\} \tag{4.52}$$

$$\hat{u}_i, \hat{o}_i \geq 0 \qquad \forall i \in G \tag{4.53}$$

$$\hat{w}_{it}, \hat{r}_{it} \geq 0 \qquad \forall i \in E_R, t \in \{1,\ldots,T\} \tag{4.54}$$

$$\hat{\alpha}_{jt} \geq 0 \qquad \forall j \in J, t \in \{1,\ldots,T\} \tag{4.55}$$

$$u^{\pm}_i, o^{\pm}_i \text{ are unrestricted} \qquad \forall i \in G \tag{4.56}$$

4.2.2 Computational Analysis of Time Windows Model

The recovery form of TW model given in Section 4.2.1 fundamentally differs from the TB approach with the complexity of decision variables used for determining the assignment of employees. In TB model, one tries to figure out whether an employee should be assigned to a pre-determined task block or not. On the other hand in TW, the primary decision variable becomes whether assigning an employee to a specific role at a particular week or not. The constraints described in TB and TW model differ from each other accordingly. In addition to the decision variables, some of the cost components distinguish from the TB model as well. TW model is more elaborate than the TB model, and have more integer decision variables. The complexity of constraints of TW is higher than TB model, similarly. It is important to note that reaching the optimal solution by TB formulation does not necessarily mean that this solution would be an optimal for the TW model.

The recovery version of TW model has been implemented in Dell Optiplex 790 PC (Windows 7 32-bit, Intel Core i5 3.10 Ghz, 4GB RAM), and tested using FICO ® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) for these 240 instances which are first designed by Leggate (2016), similar to the TB method. The study of Leggate (2016) initiates the VSC in OSV's. We note that Leggate (2016) implemented the first model as well as generated the first set of instances with 48 captains only. In this thesis, we improve this implementation further, not only by revising the implementation for correctness but also for optimal computational performance. Moreover, the initial set of data is extended in this thesis with newly generated and larger instances.Detailed material about the data sets are given in Section 4.1.2.

Since the recovery schedules are designed to respond to unexpected situations during operations, they are supposed to generate feasible and low-cost schedules in short time. Accordingly, we worked with various time limit settings as 2, 5,

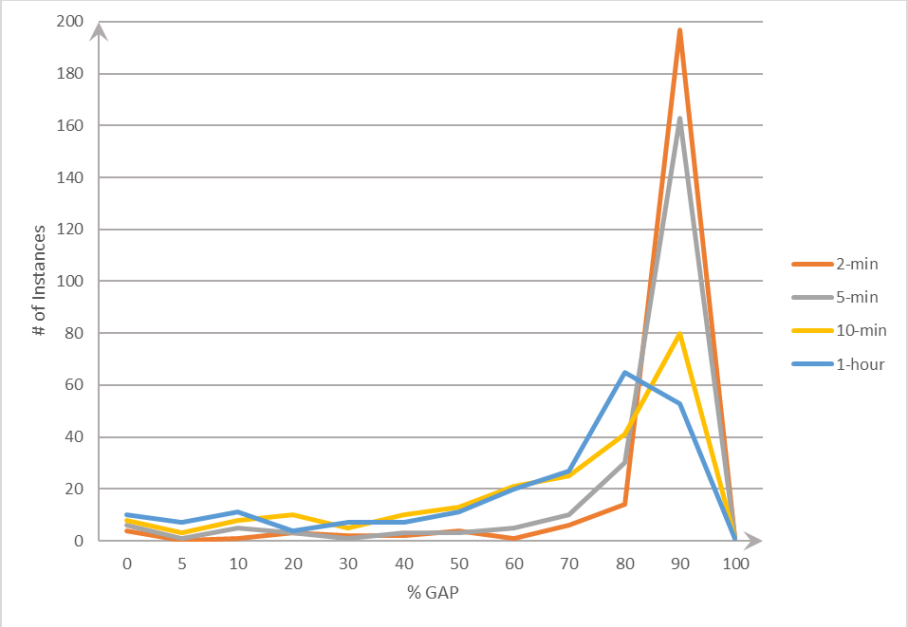

Figure 4.2: Gaps found in 2-min, 5-min, 10- min and 1-hour TW runs for 48 captains, to the best bound. x-axis indicates the integrality gaps, and y-axis indicates number of instances in each bracket.

10 minutes and 1 hour, to observe the direct approach performance. Apart from changing the time settings, the default settings were applied to run the computational study for 240 generated instances. We obtained the percentage gaps by evaluating the distance to the best lower bound amongst the 2-min, 5-min, 10-min and 1-hour run for each instance. According to the results we have obtained from runs with different time limits, we have observed that the number of optimal solutions is 5, 9, 15 and 17 at the end of 2, 5, 10 minutes and 1 hour runs, respectively for 240 problem sets. Considering the low number of optimality achieved by MILP, we used percentage gap to measure the performance of this solution method. The formula defined gap can be stated as:

*Percentage Gap = (ABS(Solution found- Best Bound)/ABS(Solution found)) * 100*

The Figure 4.2 shows the distribution of percentage gaps found with recovery TW model with time limits of 2-min, 5-min, 10-min and 1-hour by FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7).

This graph shows that most of the instances have 90-95 percentage gap from the best lower bound with 2-min solution time. As the time limit increases, we can

	GAP TW 2-min	GAP TW 5-min	GAP TW 10-min	GAP TW 1- hour
Mean	88.57%	82.81%	67.87%	64.33%
Standard Error	1.43%	1.78%	2.10%	2.11%
Median	95.64%	93.61%	80.53%	79.46%
Standard Deviation	0.22	0.28	0.33	0.33
Sample Variance	0.05	0.08	0.11	0.11
Kurtosis	8.58	3.47	-0.26	-0.56
Skewness	-3.08	-2.19	-0.92	-0.97

Figure 4.3: Summary of descriptive statistics on gaps found in 2-min, 5-min, 10- min and 1-hour TW runs for 48 captains, to the best bound.

see the change in trend between the lines. The lines for 5-min, 10-min and 1-hour have more dispersed gaps comparing to the 2-min results. Table 4.3 provides the basic statistics related to the gaps in the run within each time settings. It can be seen that from 2-min to 10-min the average gap decrease from 88.57 % to 67.87 %. The mean difference between 2-min and 10-min run are substantial. Although the time increases 10-min to 1-hour, this difference becomes just 3 %. Supporting the statement related to the distribution of gap, variance increases as the time limit increases. Skewness and kurtosis represent the symmetry and heavy-tailed or light-tailed relative to a standard distribution, respectively. Kurtosis changes from 8.58 to -0.56; while, skewness approaches to 0 from 2-min to 1-hour. In other words, instead of having heavy-tailed and asymmetric distribution, the gaps for 1-hour run disperse more equally comparing to 2-min, and 5-min run.

The number of rows, columns and non-zero elements of the recovery model for real sized problems is shown in Table 4.5. We can state that when we have the more realistic approach for modelling, we have the more complex problem for the same size problem by looking at the Table 4.5 for the large-sized problem in Section 4.1.

Problem Size	Task Based Model	Time Windows Model
Number of Constraints	22722	402075
Number of Columns	16804	415678
Number of Non-zero Elements	61092	1095295

Table 4.5: Number of constraints and decision variables for Task Based and Time Windows Models

Similarly to the idea of comparing the number of constraints and decision variables for both models (TB and TW), we can compare these two models concerning the distribution of gaps. Unless TW model, TB model has much better performance

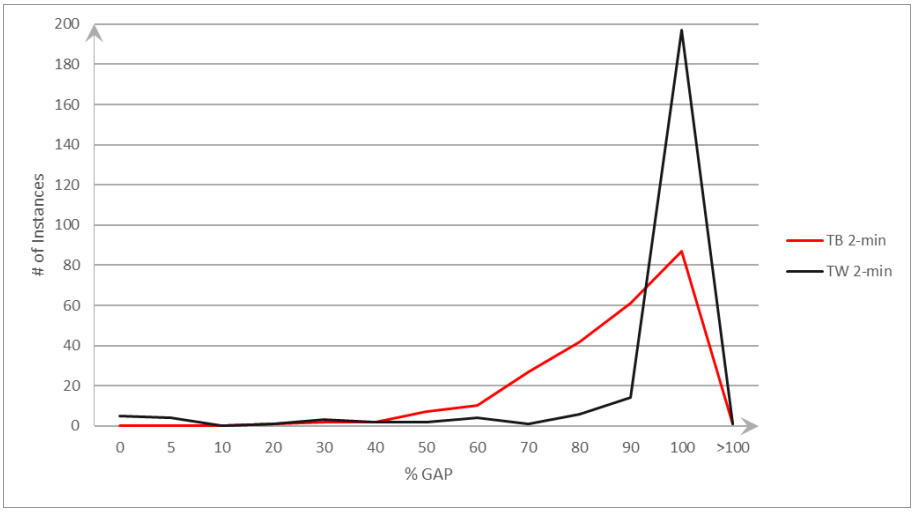

Figure 4.4: Gaps found in 2-min by Time Windows and Task Based model.

reaching optimality and having less gap. However, when we reflected the results of the cost comparison between these two models, we pointed the importance of realistic approach for cost minimisation.

After 2 minutes run of both models for same problem sets, we analysed that cost realised from TW model have more significant gaps. On the other hand, TB model is advantageous on TW model for the 2-min run; although, TB formulation is not able to find the optimum value for TW model. This difference between the performance of TB and TW is the outcome of TB model does not allow the same flexibility for the assignment as much as TW model does. As a result of the massive differences between the lower bound and best solution found through the direct application of these runs, we concluded that direct optimisation of TW and TB are not the economical methods in practice. Especially with the case of the Planners are often operating under time pressure and consequently waiting even ten minutes let alone an hour is not feasible.

The need for generation low-cost feasible schedules with reasonable time frame by our industrial partner, we decided to develop a heuristic algorithm that is capable of reaching better results within the short period. In Section 4.3, we propose a comprehensively customised heuristic algorithm for our problem as an approximation to TW recovery model which was explained in Section 4.2.1. In addition to the heuristics, we searched for decomposition methods, and the hybrid of decomposition and heuristic methods to be able to obtain optimality in a reasonable time frame

under Chapter 5.

4.2.3 Summary of Findings from Time Windows Model

The problems concerned by our industrial partner are formulated in two steps. While the TW model is presented in Section 4.2 describing the problem in detail and defining notation, the first model constructing all schedules from scratch is built considering the cost of the crew as objective, allowing us to set out the basics of the model. The second model explained in Section 4.2.1 is the extended version of the basic model with the consideration of the recovery-type nature of the real problem to minimise the cost of possible crew changes.

As there is a high demand for recovery schedules and a need for alternative schedules for the crew in a reasonable computational time, in Section 4.3, a heuristic method is proposed and analysed with details.

4.3 Heuristic Solution Method

Heuristic solution methods are preferred over direct solutions in staff scheduling literature widely as discussed in detail in Chapter 2. It is mostly the outcome of aiming feasibility for this kind of problems where especially it is hard to get optimality in short period of planning time, and unexpected situations occur, and planned schedules become no longer available. Based on the low efficiency and large gaps for most of the problem sets tested with direct solution approach on time windows model, we decided to solve the problem with heuristics. We have designed and implemented a heuristic approach which provides local neighbourhood search for decreasing the cost of change from an initial feasible solution for recovery scheduling problem. This part of the thesis is a collaborative work with Leggate (2016). The first attempt at the need of heuristic for this problem was discussed by Leggate (2016). After this research has started, the more customised and complicated procedures of the heuristic are designed, collaboratively. Furthermore, in addition to the design and development process, the implementation of the developed algorithm in C++ by using Microsoft Visual Studio 2010 is solely accomplished by my effort. The code which is used for computational study is also provided in Leggate (2016) under Appendix C between pages 750 and 1004.

The algorithm that is given in Algorithm 4.1 is the pseudo code of our case

specific algorithm to solve recovery problem and find alternative schedules on the spot. The essential part of the algorithm can be stated as the neighbourhood search. The primary functions that support the neighbourhood search are the extensions and swaps of the work blocks of employees. In order to start searching other feasible solutions, an initial feasible solution is required. To be able to explain the heuristic method comprehensively, the terminology regarding this method is provided below.

Initial Feasible Solution: The proposed heuristic search for a schedule which has a lower cost of change starts with an initial feasible schedule. This feasible schedule provides information on assignment for all regular employees (R) and agency crew member ($E\ R$) for each week by stating the vessel information if the crew member assigned for the week "t" or just stating that the employee is resting on time "t". Accordingly, we have the assignment information of all employees for whole planning horizon. This information assists us to be able to find new candidate solutions and compare them with the initial one by using the functions which are described next.

Work Block: If employee "e" is assigned to a job during the planning horizon under certain conditions, this assignment can be defined as a work block. The employees who already have been appointed to a role at the start of the planning period (i.e $s_{ik} = 1$) carry assignment that can be defined as a work block. Apart from the beginning of planning period, when an employee is assigned to the same role for at least one week, this assignment points out to a work block, similarly. The starting and finishing time, the length and the task which is subject to a possible extension and swap are explored by a function. The evaluation of the work blocks are done based on these features of the work block.

Block Extension: One of the methods to conduct the neighbourhood search is extending the work blocks in the initial feasible solution. After the usable work blocks are identified, the search can be done by extending it to either forward or backward. Forward extension implies earlier start of the block while backward extension means finishing this work block in a later week than the current week. In block extension, we first check the possible length of extension depending on the starting and finishing time (week) of the usable work block.

We cannot apply forward extension for a work block which has a work zero value bigger than 0 (i.e. $s_{ik} = 1$ or $w_{i0} > 0$). Also, the maximum extension length cannot be larger than the defined planning horizon. As an example, if work block starts in week 3 and ends in week 6, the possible forward extension length cannot be more than two weeks. In other words, this task block cannot start earlier than week 1.

Similarly, the length of backward extension cannot be more than seven weeks.

If an extension realises on a work block, then the rest of the schedule is affected by this change, and required amendments should be applied. The reason for this change is based on the constraint which prevents assigning more than one employee to the same role. Extension is not applied on the agency employees as they are generally paid higher than the regular crew. Additionally, the extensions do not guarantee the feasibility, during the search after every possible extension is applied, the viability of working conditions for each employee including agency employees are controlled with a feasibility check function.

Swap: Similar to the extensions, swap is another way of finding alternative schedules through the starting schedule. A swap is about exchanging a work block of an employee with another one. The work blocks of both employees should be almost in the same length and in the similar period. In this case, we describe one swapping block and one selected block. The swapping block cannot start more than one week earlier and end more than one week later the selected one. The blocks which start earlier than the first week of planning horizon are not preferred for swapping action. Swapping can be applied to the agency crew as well under the same conditions with the regular employees. Even the timescale of swapping and selected blocks suits technically, feasibility check is required to accept the swap.

The main idea behind the heuristic can be explained as follows. We start algorithm with an initial feasible solution and define the cost of initial solution as best cost. These actions are displayed as *Require* and *Ensure* in the pseudo code. Then, in Step 1 a randomised employee list to go through the iteration is made. We look for a usable block for the first employee e of the randomised list and looking for possible extensions in Step 2. If it is feasible to extend backwards the current task block of employee e, then we determine the new schedules and calculate the new cost. In the condition of cost improvement with the suggested schedule, the initial solution becomes tabu as the new one becomes the new current solution. We apply the same procedure for forward extension in the absence of the possible backward extension.If we cannot find any available block to extend forward or backward for employee e, we try to find a feasible employee to swap their current schedules with each other and check whether the new solution brings improvement on cost, it is feasible, or it is a tabu solution as it is stated in Step 3 to Step 9. For all employees in the randomised list, we go through the same procedures as the first employee in the list until the given time limit reached. We record all the tabu and infeasible schedules during the

search, accordingly do not allow them as new candidates.

Algorithm 4.1 Heuristic for Recovery Scheduling for Vessel Crew

Require: Initial feasible solution; randomize employees
Ensure: best cost:= initial cost
1: **for** $employee = e$; $employee \in Randomlist$; $employee++$ **do**
2: Find an usable block to search for extension or swap from current schedule of employee e
3: Extend the usable block with backward extension if possible ;
4: If not forward extension if possible;
5: If not swap the block with another employee
6: Calculate cost and check feasibility after every extension or swap
7: Accept if new cost $<$ best cost then best cost:= new cost
8: Assign as tabu solution if infeasible or no improvement on cost
9: Check there is a new candidate or not
10: If no candidate solution exists go to next employee in the randomized list
11: **end for**{employee}
12: **End** Repeat until the time limit is reached

The essential part of the algorithm can be stated as the local neighbourhood search. We organise the search with the idea of extensions as long as possible (maximum for both forward and backward extension) by checking the feasibility in every attempt to extend the time block for the concerned employee. If any later or earlier blocks (respectively for backward or forward extensions) had to be disrupted to maintain feasibility, then try to allocate the task to a more suitable employee, i.e. one who was assigned to an adjacent block in the current solution working periods where possible (with any tasks being removed as a knock-on effect being allocated to agency crew). As we mentioned above, another way of sustaining the search is *swapping*. The blocks can be swapped with another block which includes same time period with the one subject to change or can be one week longer at most. We proposed to make the swaps possible for unoccupied employees.

In addition to defining the search space, another fundamental information to be constructed is how to accept the suggested solutions after every new solution is obtained. We put feasibility check function at the end of every iteration, and we tagged them either as a *tabu* solution under the case of infeasibility detected or as *candidate* solutions in the case of feasibility. We preferred using the tabu solution concept especially to prevent having infeasible results repetitively as this would potentially save the significant amount of computational time.

Besides the core parts of the algorithm, we enrich the computational study and expand the selection of feasible schedules; we suggested extra settings that are described with details in Section 4.3.1 for the formal testing of customised heuristic.

The heuristic is coded in C++ and tested on 240 problem sets with $2 - minute$ time limit. Although the main idea behind the heuristic is explained in Algorithm 4.1, we used 48 different settings, which are discussed in detail under Section 4.3.1, to test the effect of decision points on cost improvement. Since our problem is too case-specific, it is important to see the results of different settings. Depending on the construction of this heuristic, it can stated that in addition to the local iterative search, the search is improved by applying some meta-heuristic methods. Keeping track of the list of infeasible schedules shows similarity to the tabu search. Defining different acceptance criteria is a different way of simulated annealing method and random kicks aim to change the neighbourhood like variable neighbourhood search. The extended computational study of the given solution method is presented in Section 4.3.2.

4.3.1 Different Settings of Heuristic Algorithm Test

The implementation process of the heuristics is made exhaustively with the consideration of the increment in the potential search area. The first application of heuristics heavily depends on the greedy search which has tendency of encountering with the problems such as getting stuck at local minimum points and plateaus, maintaining search without having the chance of revoking in the case of need for a change (Crama et al., 1995). In the light of this information, with the goal of having a more elaborate and flexible heuristic search we extended our algorithm by adding acceptance criteria, list ordering rules and some kick points for changing the area of search.

Although the algorithm 4.1 gives main idea behind heuristic simply, we used different settings for some decision points. We tested the heuristic and did computational analysis with the extended versions by using 5 main different parameter, listed below:

- Kick : No kick, 4+ iterations, 8+ iterations
- Initial Solution: Task Based Solution, Heuristic Initial Solution
- Employee Order List: Smart, Random

- Acceptance Rule: Best, Current
- Number of Employees Examined: 1/3rd, All

Based on these parameters and the settings applied for each parameter $3 * 2 * 2 * 2 * 2 = 48$ different versions of heuristic is obtained. For a clear understanding of the customized heuristic, it is essential to explain the reasoning behind each setting and their functions.

Kick: The idea of kick is based on the need of working on new search space to avoid as much as possible from the local optima and plateaus during the heuristic search. A kick is called during the search when we cannot observe any improvement in the results after certain number of repetitive iterations. When a kick is called, it provides a new space for search and amend the neighbourhood. The various settings has been suggested for starting the kick that are listed below during the computational study;

- Kick is not applied.
- If there is no change for best solution during at least four iterations, and a) kick has not been applied; or b) kick has not been activated for the last twenty iterations; or c) the current cost and the best cost achieved has the same value; or d) without taking into account the values of best solution, if there is no decrease on the current solution during at least four iterations
- If there is no change for best solution during at least eight iterations, and a) kick has not been applied; or b) kick has not been activated for the last forty iterations; or c) the current cost and the best cost achieved has the same value; or d) without taking into account the values of best solution, if there is no decrease on the current solution during at least eight iterations

.

Initial Solution: To be able to start the heuristic search, the initial feasible solution has quite a lot importance for the rest of the search. In order to initiate our customised heuristic, we used the TB method which was explained in details in Section 4.1 and a randomised heuristic algorithm which is proposed by Leggate (2016).

Employee Order List: While examining the employees for possible extensions (forward/backward) or swaps used for finding the alternative solutions, a list is

provided either random or organized with a rule. We called the ordering based on a rule as *smart* ordering. The logic behind the smart ordering can be stated as; when a kick has been activated or it is the first iteration, then the employees are sorted randomly; otherwise, if decrease in the cost is observed at the current iteration then employees are sorted by putting the employee with a recent change as first in the list. If any of the cases above has not been realized then the sort is organized accordingly putting the least changed one the top of the list.

Acceptance Rule: This rule is about deciding the solution to be continued with during the search. We have two options here to determine our current solution either assigning the best solution as current or without looking at the cost when we have a change in the cost by neglecting the increase or decrease, we continue with the most recent feasible solution. We applied both of these rules to test their impact on the performance of heuristic.

Number of Employees Examined: The reason of to be thought the number of employees as a heuristic criterion is about the low performance of the initial heuristic in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) and requiring long computational time on examining all of the employees. Although the speed of C++ is indisputably higher than the speed of FICO® Xpress-MP and there is no time concern in C++, we would like to see whether this setting makes a significant difference or not. Therefore, the number of examined employees are tested as *All employees* and *One-third of all employee*.

Based on these settings, 48 different version of heuristic is implementation of heuristics solution method. We applied all the methods to 240 problem sets that has been generated in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7), accordingly we obtained $48x240 = 11520$ data points to discuss the performance of heuristic method. To identify the effects of these 5 parameters on heuristic performance, full factorial design analysis is applied. The discussion of this analysis is presented at Section 4.3.2.

4.3.2 Computational Results

The rough numbers, which indicate the performance of direct solution method, were discussed in Section 4.2.2. In Section 4.3.2 we analysed those numbers with more detail.

Figure 4.5 underlines the performance of three different solution approaches. The

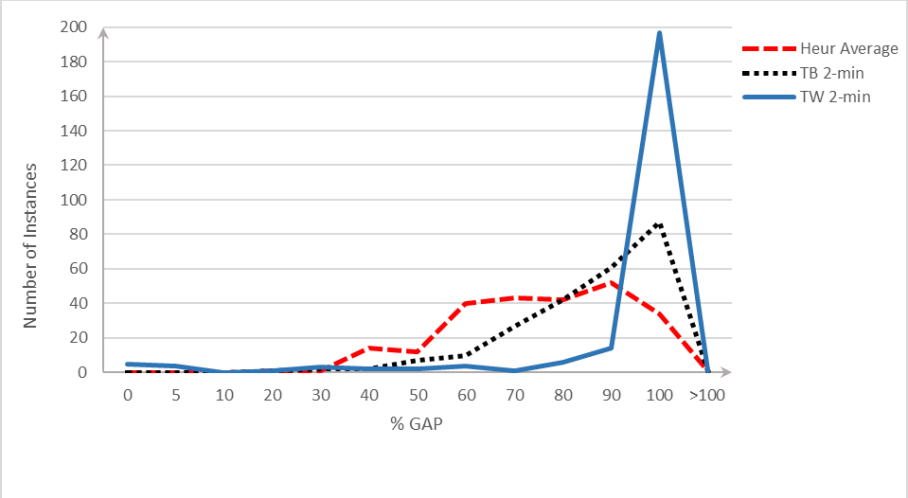

Figure 4.5: Percentage gap dispersion cross solution methods

methods which are compared are TB approximation, TW model and the customised heuristic. The results are obtained with the 2-minute time limit. As 48 different settings are applied for testing the heuristic, the average gap collected from these 48 configurations is presented in the figure 4.5. This figure shows that heuristic solution has higher efficiency than the other direct methods regarding less number of instances which have percentage gap higher than 60%.The average percentage gap of heuristics is 18 % lower than the direct solution method, and we observe more dispersed distribution for gap values of the heuristic rather than cumulated around 90% as in the direct approach. The number of sets, which have gaps up to 50% is much higher for the heuristics approach compared to the TB and TW model. On the other hand, it is observed that direct solution approach reaches optimality for 5 sets in 2-minutes while TB approximation or any heuristic methods cannot achieve the optimality in the same solution time.

Although heuristics do not perform well for reaching the optimality, they have higher performance regarding having the lower cost of a shorter solution time amongst the overall results. The figure 4.6 shows the difference between percentage gap of 1-hour TW model and 2-min heuristic method. To measure the performance of heuristics, we first highlight the minimum objective function values among 48 versions of heuristics for each problem set. The mean and median percentage gap of these minimum costs is shown in the second line in Table 4.6. In addition to the minimum costs, the costs found by the average of 48 version for each instance are calculated.

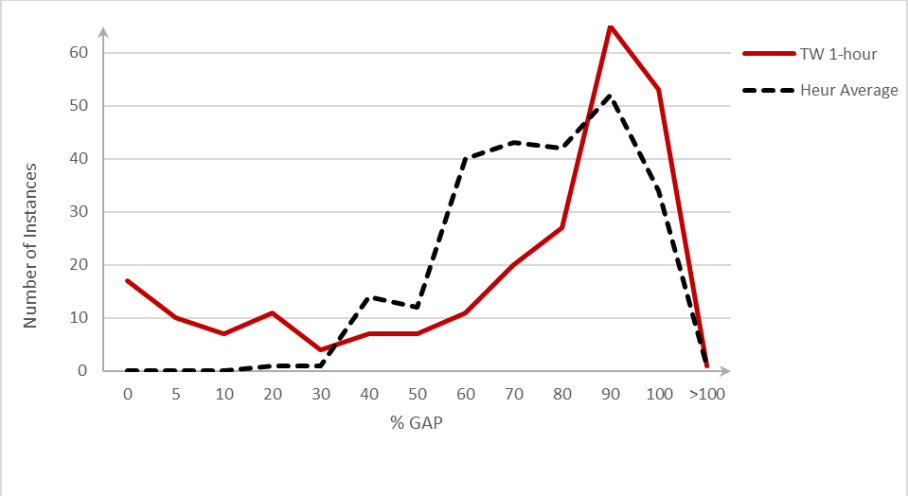

Figure 4.6: Percentage gap dispersion for TW and the heuristic method

The mean and median values of percentage gap depending on these average costs are highlighted in the third line of the Table 4.6. The last two lines of this table, mean and median percentage gaps obtained by direct solution approach (MIP of recovery formulation) are presented.

Afterwards, we compare these highlighted values with the objective function values reached after solving the direct formulation with 1-hour time limit. In the sequel of this comparison, we conclude that performance of heuristics with 2-minutes time limit overcomes even the 1-hour direct solution's performance concerning the median and mean values with almost 8% and 6%, respectively. The mean difference increases 30% when we check the results under the same time limits for optimisation solver and heuristics (see Table 4.6).

Solution Method	Mean	Median
Heuristic Solution (Min)- 2 min	58.83%	59.72%
Heuristic Solution (Average)- 2 min	70.89%	72.27%
Direct Solution - 2 min	88.57%	95.64%
Direct Solution - 1 hour	64.33%	79.46%

Table 4.6: Summary of Percentage Gaps

Besides achieving the lower gaps in, the shorter period through suggested method, it is essential to explore the cost improvement through the neighbourhood search. According to the computational test, there is a 34% decrease on average from the

initial solution to the best solution obtained after 2-minutes of search. As the initial feasible solution is an important parameter that can affect the heuristic search, the improvement of the different initial solution methods is also analysed.

Initial Solution	Average Improvement
Task Based Approximation	30.50%
Heuristic Initial Solution	38.86%

Table 4.7: Cost improvement through different initial solutions

When the percentage improvement is calculated based on the variation between the initial solution and the result after search completed for 11520 data points, the observed maximum is found as 88%. Additionally, the test results indicated that a decrease in the cost for all instances is detected. However, depending on the settings of heuristic that is applied, there are some data points no change has been examined from the initial cost.

Since we do not want to misread the results and figure out the best combination for heuristic amongst 48 versions, we conduct a statistical analysis considering the potential impacts caused by the characteristics of problem sets and variety of heuristic methods. Therefore, we investigate the effects of 4 parameters listed in Section 4.1.2 and 5 different settings for heuristics mentioned above through a full factorial design analysis. Depending on the fact that all independent variables (effects) are categorical while the explanatory variable (GAP) is continuous, we apply linear regression for categorical data analysis by using SPSS 24.0 with emphasizing the characteristics of all variables. We define the parameters related to problem set attributes as ordinal and the parameters associated to heuristic variations as nominal.

As a result of this statistical analysis, we conclude that apart from order and accept criterion; all parameters have a statistically significant effect on percentage gap with 99% confidence interval with a 56% adjusted R-square value. Based on the SPSS output of this analysis, we can state that rather than the heuristic settings, problem characteristic is more effective on the mean values of GAP. It is also a result of the elaboration of instances explained by these given parameters. Agency penalty factor and the probability of crew availability have the highest impact with the most elevated coefficient values, and the initial solution seems more important than the other solution criterion relying on the same indicator. Detailed output of SPSS results can be found in Figure A.1.

Acceptance	Best	70.82%
	Current	70.96%
Ordering	Random	71.00%
	Smarter	70.79%
Initial Solution	Heuristic	68.14%
	Task Based	73.65%
List Size	All Employees	71.28%
	One Third	70.51%
Kick Settings	4+ iters	71.57%
	8+ iters	71.02%
	no kick	70.09%

Figure 4.7: Average Percentage Gaps for Each Option

Since our primary aim is to show the existence of the relationship with the gap and these effects instead of the accuracy of the regression model, we annotate the results with the help of the pivot tables referencing the average gap values. The referenced gap values are shown in Figure 4.7. As a result of this simple analysis, we conclude that choosing one option over another affects the gap approximately with 1% almost for all settings. However, the selection of initial feasible solution can be considered as a stronger element for the structure of heuristic.

With the conclusion of there is no direct effect of settings apart from the initial solution decision, we looked for interactive effects of parameters. Figure 4.8 represent the mean gaps for 48 versions by grouping them depending on their initial feasible solution options. The numbers show that for both graphs, version 1, 7, and 8 have the minimum values for the mean gap, while version 4 has the highest gap. Although the difference between the highest and lowest mean gap scores is not more than 5%, it can be stated that some versions have better quality results based on the parameter settings.

The preferred versions, which have the lowest and highest mean gaps, are shown explicitly in Table 4.9 and the display for table of all 48 versions is provided in Appendix A.2.

Unfortunately, this information cannot give obvious conclusions, but we can see the common feature that kick is not activated for the three versions which provide the smaller gaps. The elements other than the initial solution and kick activation, do not show significant difference depending on one option to another.

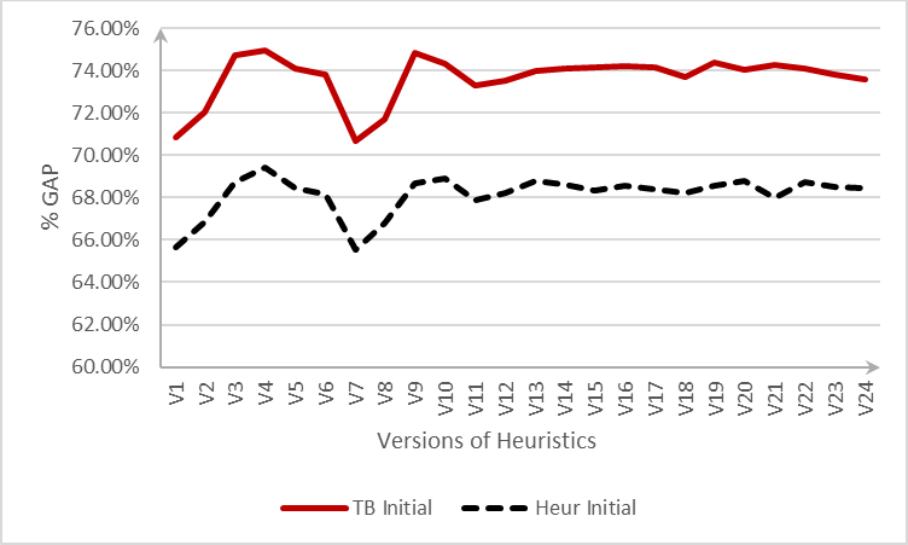

Figure 4.8: Average percentage gaps for heuristic combinations based on initial solution criterion

	Acceptance Rule	Kick Setting	Ordering Rule	Number of Employee
Version 1 (V1)	current	no kick	smarter	one third
Version 7 (V7)	best	no kick	smarter	one third
Version 8 (V8)	best	no kick	random	one third
Version 4 (V4)	current	4+ iters	random	one third

Figure 4.9: Explanation of Heuristic Combinations for Version 1, 7, 8 and 4

4.3.3 Summary of Findings from Heuristic Method

Heuristic method, which is fundamentally based on neighbourhood search and improved with tabu solutions, is implemented in C++ and tested on 240 problems with five different settings which provided 48 various selection of heuristic applications. As a result of exhaustive computational tests, high-quality improvement and the higher decrease in percentage gap have been succeeded with 2 minutes run of each heuristic.

Furthermore, it is concluded that the other dominant aspect of heuristic is the ability to generate the significant number of alternative recovery schedules in 2-minutes. Based on the 11520 data points, the number of candidate solutions found during the heuristic search is 273 on average. This aspect of the heuristic solution method provides the wide selection of recovery schedules to the planners.

In addition to the performance analysis of overall results, a statistical analysis is applied to understand the effect of heuristic settings and data characteristics. As a result of this analysis, we concluded that:

- Not Using Kicks
- Starting the search with Heuristic Initial Solution
- Using the Randomized Order List
- Accepting the first non-tabu solution which improves on the Current Solution
- Examining the one-third of all employees at each iteration

gives the less gap comparing the other settings. Furthermore,

- Low probability of crew availability
- Highest agency penalty factor
- Lack disruption factor

increase the GAP due to the hardness of the problem. Using heuristics to generate feasible schedules within the short time is an important decision tool for minimising the cost of crew changes and having a better planning for the efficiency.

To add more value to the research besides the practical impact and with the aim of gaining more insights for the problem structure, we continue our investigation with exact methods that can be applied to our problem and provide the potential improvement to the existing solution methods.

Chapter 5

Benders Decomposition for VCS in OSVs

In the VCS problem in OSVs, there are too many integer and binary variables due to the characteristics of the problem. By using the BD algorithm, our goal is to reduce the number of integer variables with the help of fixing specific binary variables. As a result of this, the rest of the problem (sub-problem) can be solved as an LP. Primarily, we aim to find the solution to our problem by decomposing it into two problems which are expected to be less complicated and have more efficient solution method. The principles of BD and the algorithm for MILP problem are provided in Section 2.3.2. Through this chapter, the BD algorithm is implemented on our problem with the acceleration techniques such as Combinatorial cuts, Pareto optimality cuts and with the heuristic.A thorough computational analysis of these applications is also designed and a discussion around this computational study is provided.

In Section 5.1, the preliminary study for the application of BD in our problem is represented.

5.1 Application of BD on Recovery Formulation

In order to show the compatibility of the BD for our problem structure, Theorem 5.1.1 is proposed. We use vector notation $\mathbf{b}$ whose component corresponds to $b_{ikt} \in \mathbb{R}, \forall i, k, t$ and likewise we use notations $\{\mathbf{x}, \mathbf{l}, \mathbf{o}, \mathbf{u}, \mathbf{b}, \mathbf{d}, \alpha, \beta, \delta, \mathbf{w}, \mathbf{r}\}$.

Theorem 5.1.1 *Define* $Q := \{(x,l) : (4.16),(4.17),(4.29),(4.37),(4.38)\}$. *Let* A *be the solution space for the linear programming relaxation of problem* $\{(4.15-4.44)\}$. *Now denote the vector* $y = \{\mathbf{o},\mathbf{u},\mathbf{b},\mathbf{d},\alpha,\beta,\delta,\mathbf{w},\mathbf{r}\}$. *For any* $\hat{\mathbf{x}},\hat{\mathbf{l}} \in Q$, $Proj_y A$ *is integral where* $Proj_y A = \{y : \exists x,l,(x,l,y) \in A\}$ *is the projection of* A *in the space of* y.

Proof: For any fixed values $\hat{x},\hat{l} \in Q$, the problem becomes separable and matrices that correspond to each sub-problem is either network or identity matrices and totally uni-modular.$\diamond$

By Theorem 5.1.1 we can ensure the suitability of Benders Decomposition Method for our problem. Additionally, the SP of our model can be stated as below (equations (5.1)-(5.23)) while x_{ijt} and $l_{\lambda ijt}$ are fixed.

min:

$$\sum_{i=1}^{m+1}\sum_{k\in V_k}\sum_{t=1}^{T}\left(c^B_{ikt}b_{ikt} + c^D_{ikt}d_{ikt}\right) + \sum_{i\in G}\left(c^U_i u_i + c^O_i o_i\right) \tag{5.1}$$

subject to:

$$b_{ik1} \geq \sum_{j\in V_k}\hat{x}_{ij1} - s_{ik} \qquad \forall k,i \in E_R \tag{5.2}$$

$$b_{ikt} \geq \sum_{j\in V_k}\hat{x}_{ijt} - \sum_{j\in V_k}\hat{x}_{ij,t-1} \qquad \forall k,i \in E_R, t \in \{2,..,T\} \tag{5.3}$$

$$\beta_{j1} - \delta_{j1} = \hat{x}_{m+1,j1} - \sigma_j \qquad \forall j \tag{5.4}$$

$$\beta_{jt} - \delta_{jt} = \hat{x}_{m+1,jt} - \hat{x}_{m+1,j,t-1} \qquad \forall j,t \tag{5.5}$$

$$b_{m+1,kt} \geq \sum_{j\in V_k}\beta_{jt} \qquad \forall k,t \tag{5.6}$$

$$d_{m+1,kt} \geq \sum_{j\in V_k}\delta_{jt} \qquad \forall k,t \tag{5.7}$$

$$u_i \geq g_i - \left(\Omega_i + \sum_{j\in J}\sum_{t=1}^{T}\hat{x}_{ijt}\right) \qquad \forall i \in G \tag{5.8}$$

$$o_i \geq \left(\Omega_i + \sum_{j\in J}\sum_{t=1}^{T}\hat{x}_{ijt}\right) - g_i \qquad \forall i \in G \tag{5.9}$$

$$w_{it} \geq w_{i,t-1} + \sum_{j\in J}\hat{x}_{ijt} - w_i^{max}\left(1 - \sum_{j\in J}\hat{x}_{ijt}\right) \qquad \forall t,i \in E_R \tag{5.10}$$

$$w_i^{max}\hat{l}_{\lambda ijt} \geq w_{i,t-1} - w_i^{max}(1-\hat{x}_{ijt}) + \hat{x}_{ijt} - (\lambda - 1) \quad \forall j,\lambda,t \in \{2,..,T\}, i \in E_R \tag{5.11}$$

$$\alpha_{jt} \geq \alpha_{j,t-1} + \hat{x}_{m+1,jt} - \alpha_j^{max}\delta_{jt} \qquad \forall j,t \in \{2,..,T\} \tag{5.12}$$

$$\alpha_{jt} \geq \hat{x}_{m+1,jt} \qquad \forall j,t \in \{2,..,T\} \tag{5.13}$$

$$\alpha_j^{max} \hat{l}_{\lambda,m+1,jt} \geq \alpha_{jt} - (\lambda - 1) \qquad \forall j,t,\lambda \tag{5.14}$$

$$r_{it} \geq r_{i,t-1} - \left(1 - \sum_{j \in J} \hat{x}_{ijt}\right) \qquad \forall t, i \in E_R \tag{5.15}$$

$$r_{it} \geq (\rho_i - 1) \sum_{k \in K} d_{ikt} \qquad \forall t, i \in E_R \tag{5.16}$$

$$\rho_i \left(1 - \sum_{j \in J} \hat{x}_{ijt}\right) \geq r_{i,t-1} \qquad \forall t \in \{2,..,T\}, i \in E_R \tag{5.17}$$

$$b_{ikt}, d_{ikt} \in \{0,1\} \qquad \forall k,t, i \in E_R \tag{5.18}$$

$$\beta_{jt}, \delta_{jt} \in \{0,1\} \qquad \forall j,t \tag{5.19}$$

$$b_{m+1,kt}, d_{m+1,kt} \geq 0 \text{ and integer} \qquad \forall k,t \tag{5.20}$$

$$u_i, o_i \geq 0 \qquad \forall i \in G \tag{5.21}$$

$$w_{it}, r_{it} \geq 0 \qquad \forall i,t \tag{5.22}$$

$$\alpha_{jt} \geq 0 \qquad \forall j,t \tag{5.23}$$

and RMP of this problem is expressed as:

$$\min \sum_{i=1}^{m+1} \sum_{j \in J} \sum_{t=1}^{T} \left(c_{ijt}^W x_{ijt} + \sum_{\forall \lambda} c_{\lambda ijt}^L l_{\lambda ijt}\right) + S \tag{5.24}$$

$$\sum_{i=1}^{m+1} e_{ijt} x_{ijt} = a_{jt} \qquad \forall j,t \tag{5.25}$$

$$\sum_{j \in J} x_{ijt} \leq 1 \qquad \forall t, i \in E_R \tag{5.26}$$

$$w_i^{max} l_{\lambda ij1} \geq w_{i,0} - w_i^{max}(1 - x_{ij1}) + x_{ij1} - (\lambda - 1) \qquad \forall j, i \in E_R, \lambda \tag{5.27}$$

$$\rho_i \left(1 - \sum_{j \in J} x_{ij1}\right) \geq r_{i,0} \qquad \forall i \in E_R \tag{5.28}$$

$$S \geq (b - Bx)^T \pi_p \quad \forall p \in \{extrem - points\} \tag{5.29}$$

$$0 \geq (b - Bx)^T \pi_p \quad \forall p \in \{extreme - rays\} \tag{5.30}$$

$$x_{ijt} \in \{0,1\} \qquad \forall i,j,t \tag{5.31}$$

$$l_{\lambda ijt} \in \{0,1\} \qquad \forall i,j,t,\lambda \tag{5.32}$$

$$S \in \{-\infty, +\infty\} \tag{5.33}$$

The auxiliary variable S is introduced for RMP to represent the objective function value obtained from DSP. With the help of constraint 5.29, S provides the connection

between SP and RMP. Meanwhile, π is obtained from the cost coefficients of DSP which calculates the boarding, departing, under and over time costs of employees. Sub-problem carries the constraints (4.18) − (4.36) excluding the ones in RMP, i.e. (5.25− 5.28).

The described sub-problem and restricted master problems are the decomposed form of TW formulation for basic scheduling problem given in Section 4.2. Since, we are more interested in the recovery process for our case, we implemented BD algorithm mainly for recovery version of the problem which is provided in Section 4.2.1. We employed equations (4.45-4.46) and redefined them into (5.34) and converted the cost coefficient to utilize recovery feature.

$$x^{\pm}_{ijt} = (\hat{x}_{ijt} - x^{*}_{ijt})/(1 - 2x^{*}_{ijt}) \quad \forall i \in E, j \in J, t \in t \in \{1, \ldots, T\} \tag{5.34}$$

Note that the change in the decision variables for recovery schedules can be rephrased as in equation 5.34 and put it in the formulation by replacing the x_{ijt} with $x^{\pm}_{ijt}$. We applied the same rule on the decision variable for long work, board and depart. As we rephrase the decision variables, we can utilize the cost coefficient with cost of changes (ϕ^{W}_{ijt}, ϕ^{B}_{ikt},ϕ^{D}_{ikt}, $\phi^{B_A}_{jt}$, $\phi^{D_A}_{jt}$, $\phi^{L}_{\lambda ijt}$).

We conducted a preliminary study with a small sized problem (5 weeks, 13 employees, 3 vessels) that can be solved easily with direct solution approach in very short time (2 seconds). After we implemented the BD algorithm in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) (see appendix section B.1.1), we tested the performance with this small sized problem. The initial observations based on this experiment are:

- Can not generate any optimality cuts until the 23rd iteration
- No feasible solution obtained until the 140 seconds
- After 1 hour, the result obtained was not feasible

This preliminary study shows that the implementation of BD as we initially proposed is not an efficient way to solve our problem. Even though we do not aim to reach the optimality, it is not easy to obtain feasible solutions by applying this method. Accordingly, we sought after logical explanations for the weak cuts

and questioned the quality of our decomposition method with regards to the fixed decision variables and the size of master and slave problems.

We focused on searching the possible reasons of unboundedness in dual sub-problem (DSP). When we fixed decision variables l and x, the value of $w_{t-1,i}$ became a parameter under the condition of $t = 1$, and constraint (4.30) $w_i^{max} l_{\lambda ij1} \geq w_{i,0} - w_i^{max}(1 - x_{ij1}) + x_{ij1} - (\lambda - 1)$ become redundant constraint. This situation led us to divide the constraint set (4.30) between sub-problem and restricted master problem for $t = 1$ and $t \in 2..T$. We wanted to prevent possible infeasible subsystems due to this division and generated results through RMP that are feasible regarding the maximum consecutive work constraints. Accordingly, we decided to include w as fixed decision variables. Hence, the constraints contained only l, x and w were included in the RMP. In relation to this, the SP of second version of BD application had to reformulated by the following:

min:

$$\sum_{i=1}^{m+1} \sum_{k \in V_k} \sum_{t=1}^{T} \left(c_{ikt}^{B} b_{ikt} + c_{ikt}^{D} d_{ikt} \right) + \sum_{i \in G} \left(c_i^{U} u_i + c_i^{O} o_i \right) \tag{5.35}$$

subject to:

$$b_{ik1} \geq \sum_{j \in V_k} \hat{x}_{ij1} - s_{ik} \qquad \forall k, i \in E_R \tag{5.36}$$

$$b_{ikt} \geq \sum_{j \in V_k} \hat{x}_{ijt} - \sum_{j \in V_k} \hat{x}_{ij,t-1} \quad \forall k, i \in E_R, t \in \{2, .., T\} \tag{5.37}$$

$$\beta_{j1} - \delta_{j1} = \hat{x}_{m+1,j1} - \sigma_j \qquad \forall j \tag{5.38}$$

$$\beta_{jt} - \delta_{jt} = \hat{x}_{m+1,jt} - \hat{x}_{m+1,j,t-1} \qquad \forall j, t \tag{5.39}$$

$$b_{m+1,kt} \geq \sum_{j \in V_k} \beta_{jt} \qquad \forall k, t \tag{5.40}$$

$$d_{m+1,kt} \geq \sum_{j \in V_k} \delta_{jt} \qquad \forall k, t \tag{5.41}$$

$$u_i \geq g_i - \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt} \right) \qquad \forall i \in G \tag{5.42}$$

$$o_i \geq \left(\Omega_i + \sum_{j \in J} \sum_{t=1}^{T} \hat{x}_{ijt} \right) - g_i \qquad \forall i \in G \tag{5.43}$$

$$\alpha_{jt} \geq \alpha_{j,t-1} + \hat{x}_{m+1,jt} - \alpha_j^{max} \delta_{jt} \qquad \forall j, t \in \{2, .., T\} \tag{5.44}$$

$$\alpha_{jt} \geq \hat{x}_{m+1,jt} \qquad \forall j, t \in \{2, .., T\} \tag{5.45}$$

$$\alpha_j^{max}\hat{l}_{\lambda,m+1,jt} \geq \alpha_{jt} - (\lambda - 1) \qquad \forall j,t,\lambda \tag{5.46}$$

$$r_{it} \geq r_{i,t-1} - \left(1 - \sum_{j\in J} \hat{x}_{ijt}\right) \qquad \forall t, i \in E_R \tag{5.47}$$

$$r_{it} \geq (\rho_i - 1)\sum_{k\in K} d_{ikt} \qquad \forall t, i \in E_R \tag{5.48}$$

$$\rho_i\left(1 - \sum_{j\in J} \hat{x}_{ijt}\right) \geq r_{i,t-1} \qquad \forall t \in \{2,..,T\}, i \in E_R \tag{5.49}$$

$$b_{ikt}, d_{ikt} \in \{0,1\} \qquad \forall k,t,i \in E_R \tag{5.50}$$

$$\beta_{jt}, \delta_{jt} \in \{0,1\} \qquad \forall j,t \tag{5.51}$$

$$b_{m+1,kt}, d_{m+1,kt} \geq 0 \text{ and integer} \qquad \forall k,t \tag{5.52}$$

$$u_i, o_i \geq 0 \qquad \forall i \in G \tag{5.53}$$

$$r_{it} \geq 0 \qquad \forall i,t \tag{5.54}$$

$$\alpha_{jt} \geq 0 \qquad \forall j,t \tag{5.55}$$

and RMP of this problem:

$$\min \sum_{i=1}^{m+1}\sum_{j\in J}\sum_{t=1}^{T}\left(c_{ijt}^W x_{ijt} + \sum_{\forall\lambda} c_{\lambda ijt}^L l_{\lambda ijt}\right) + S \tag{5.56}$$

$$\sum_{i=1}^{m+1} e_{ijt}x_{ijt} = a_{jt} \qquad \forall j,t \tag{5.57}$$

$$\sum_{j\in J} x_{ijt} \leq 1 \qquad \forall t, i \in E_R \tag{5.58}$$

$$\rho_i\left(1 - \sum_{j\in J} x_{ij1}\right) \geq r_{i,0} \qquad \forall i \in E_R \tag{5.59}$$

$$w_{it} \geq w_{i,t-1} + \sum_{j\in J}\hat{x}_{ijt} - w_i^{max}\left(1 - \sum_{j\in J}\hat{x}_{ijt}\right) \qquad \forall t, i \in E_R \tag{5.60}$$

$$w_i^{max}\hat{l}_{\lambda ijt} \geq w_{i,t-1} - w_i^{max}(1 - \hat{x}_{ijt}) + \hat{x}_{ijt} - (\lambda - 1) \qquad \forall j,t,i \in E_R,\lambda \tag{5.61}$$

$$S \geq (b - Bx)^T\pi_p \quad \forall p \in \{extreme - points\} \tag{5.62}$$

$$0 \geq (b - Bx)^T\pi_p \quad \forall p \in \{extreme - rays\} \tag{5.63}$$

$$x_{ijt} \in \{0,1\} \qquad \forall i,j,t \tag{5.64}$$

$$l_{\lambda ijt} \in \{0,1\} \qquad \forall i,j,t,\lambda \tag{5.65}$$

$$w_{it} \geq 0 \qquad \forall i,t \tag{5.66}$$

$$S \in \{-\infty, +\infty\} \tag{5.67}$$

The summary of the test run on the same small instance can be listed as follow:

- The total solution time to find optimal solution is 3922 second while it can reach optimality in 14 seconds with Direct solution approach of recovery scheduling given in Section 4.2.1.
- The total number of iterations is 151 and first 27 of them are feasibility cuts.
- It has reached 10 % gap after the 53rd iteration and it struggled to reach optimality after that point,

Preliminary experiments indicated that model modification helped us to gain optimal results in a shorter time. Moreover, it provides smaller gaps between the iterations during the BD algorithm compared to the first attempt(i.e. 5.2- 5.32). We are now able to obtain optimality cuts and optimal solution after 1-hour run. On the other hand, it was still not comparable with direct solution approach, and for real sized problem it ended up giving memory errors. The insights obtained from the small instance is also not promising.

Although we managed to prevent the potential infeasibility originated from the maximum consecutive work constraint by fixing more decision variables and adding related constraints to the RMP, DSP mostly resulted in unboundedness.Additionally, the time spent on feasibility cuts was not efficient enough with respect to the quality of bounds. According to Saharidis and Ierapetritou (2010) generating optimality cut rather than feasibility cut can result in faster convergence and increase the quality of cuts during BD algorithm.

In the light of this information, we considered diagnosing the inefficiency. Codato and Fischetti (2006) suggested a method for fast minimum infeasible subsystem (MIS) search of an infeasible linear system. This approach advocates that given an infeasible system of inequalities, similar to the constraints (5.36)-(5.49), find an inclusion-minimal set of its rows yielding an infeasible system. It suggests that employing a dummy constraint which forces the objective function of dual sub-problem to 1 and converts sub-problem's right-hand side values to 0 by assuming the coefficient for the objective function of primal problem is to 1 to eliminate the dual unboundedness, (Gleeson and Ryan, 1990).

When we adopt this information to our problem, we can find the source of infeasibility by using the sub-problem P_{unb} that has resulted with unboundedness. Solving P_{unb} with modification for fast MIS search, the decision variables have non-zero solution values represents dual decision variables (primal constraints) as the source of infeasibility of primal sub.

After the application of fast MIS search, we have noticed that constraints (4.34-4.36) related to minimum rest regulations in Section 4.2 also cause dual unboundedness (primal infeasibility) in sub-problem. By the help of this change, the model became more suitable to work with combinatorial Benders' cuts (CB) of Codato and Fischetti (2006) in the case of the unboundedness in DSP. The detailed information for CB and how they designed in our problem take place in Section 5.3. As a result of this situation, we added the decision variable d, which states departure of regular employees, to the set of fixed decision variables aiming to satisfy the minimum rest constraints before sending the fixed decision variables' information to the sub-problem.

On the other hand, to apply BD algorithm on VSC in OSVs, we preferred to have a simplified and more flexible model to work with BD as a consequence of low performance in the preliminary result of the original model (4.2.1). We aimed to increase the efficiency of the algorithm, and generalise the solution method with high efficiency for original recovery model.

Section 5.2 explains under which assumptions the modified model is designed. Also the modified mathematical model is represented. Afterwards, in Section 5.3 we illustrate how we adopt the Combinatorial Benders' Cuts to our problem in detail and how we implement in the BD algorithm.

5.2 Simplification of the Recovery Form

The model given in Section 4.2 reflects the real case in the company. Although it is important to model reality, obtaining practical results out of it makes significant impact for the development of this study. To be able to take a further step and improve the current solutions, working with less realistic but more flexible models is required for our problem. Due to huge amount of binary variables and complicated constraints that the long work cost carries, it becomes harder to find the source of low performance of MIP and Benders Decomposition algorithm in our problem.

Therefore, we simplify the model by extracting the long work cost and keeping all other constraints the same. The original model which includes the long work variable is capable of finding lower costs than the modified model. The reason behind this situation is about realizing an optimisation through the long work decision variable, as well. As the modified model does not take long work variable into account, it does not optimise the cost depending on this variable. On the other hand, with the simplified characteristic of modified model, it is more likely to find better solutions with the modified model for hard instances in a limited computational time. Therefore, it can be stated that the cost of the modified model is realistic enough to compensate the extraction of the long work cost from our model.

The decision variables measuring the existence of long work is denoted by $l_{\lambda ijt}$ and becomes 1 if the carrying out of role j in period t is at least the λ^{th} consecutive working week for employee i.

At the same time, the associated constraints with $l_{\lambda ijt}$ are (4.30) in Section 4.2 to utilize this constraint we need decision variable which records the number of consecutive work up to the specified week, formulated as:

w_{it} - number of consecutive weeks work offshore that employee $i \in E_R$ has been assigned up to and including week t, $\forall i \in E_R, t \in \{1, \ldots, T\}$.

w_{it} aligns with constraint (4.28) in Section 4.2. When we excluded $l_{\lambda ijt}$ to simplify the model, we did not need to identify w_{it}, and linked constraints 4.30, 4.28. In correspondence with this, it is not necessary to keep track of consecutive working week information for agency type crews.

α_{jt} - number of consecutive weeks that an agency employee has been assigned to role j up to an including week t, $\forall j \in J, t \in \{1, \ldots, T\}$.

Similarly, we can express the related constraint sets (4.31 - 4.33) without the help of α_{jt}.

Thus, modified model excludes $\lambda mJT + mT + JT$ number of integer decision variables. Also, we replace the constraint sets (4.28 - 4.33) with the following constraints to control maximum consecutive working weeks.

$$w_i^{max} \geq w_{i,0} + \sum_{j \in J} \sum_{t=0}^{w_i^{max} - w_{i,0}} x_{ijt+1} \qquad \forall i \in E_R \tag{5.68}$$

$$w_i^{max} \geq \sum_{j \in J} \sum_{k=0}^{w_i^{max}} x_{ijt+k} \quad \forall i \in E_R, t \in 1..T - w_i^{max} \tag{5.69}$$

$$\alpha_j^{max} \geq \alpha_{r,0} + \sum_{t=0}^{\alpha_r^{max} - \alpha_{r,0}} x_{m+1,j,t+1} \quad \forall j \in J \tag{5.70}$$

$$\alpha_j^{max} \geq \sum_{k=0}^{\alpha_r^{max}} x_{m+1,j,t+k} \quad \forall j \in J, t \in 1..T - \alpha_j^{max} \tag{5.71}$$

Another point that varies from the original model is assuming that after departing from a vessel, it is not allowed to work without having(ρ_i) weeks rest for regular employees. The regulations suggest that after departure happens from a vessel, regular employees should not be assigned for another duty without completing the minimum rest constraint. In the original model, it is possible to assign an employee to a role in another vessel just after departing from a vessel. Constraint (5.72) is added to prevent assignment of a new job and departing at the same week. Constraints (5.73) and (5.74) are added with the expectation of obtaining tighter cuts and make it suitable for being able to use combinatorial Benders' cuts. The explained constraints take place in the modified model as below, while $c = \min\{\rho_i - 1, T - t\}$.

$$\sum_{k \in V} d_{ikt} + \sum_{j \in J} x_{ijt} \leq 1 \quad \forall t, i \in E_R \tag{5.72}$$

$$x_{i,j,t+y} = 0 \quad \forall i \in E_R, j \in J, t \in \{1, .., r_{i,0}\} \tag{5.73}$$

$$\sum_{j \in J} x_{i,j,t+y} + \sum_{k \in V} d_{ikt} \leq 1 \quad \forall i \in E_R, t \in \{1, .., T-1\}, y \in \{0, .., c\} \tag{5.74}$$

The objective function of the modified model becomes (5.75) after extracting the long work cost from the model.

$$\sum_{i=1}^{m+1} \sum_{k \in V_k} \sum_{t=1}^{T} \left(c_{ikt}^B b_{ikt} + c_{ikt}^D d_{ikt}\right) + \sum_{i \in G} \left(c_i^U u_i + c_i^O o_i\right) + \sum_{i=1}^{m+1} \sum_{j \in J} \sum_{t=1}^{T} \left(c_{ijt}^W x_{ijt}\right) \tag{5.75}$$

In accordance with the formulations and explanations above, we can state that modified model aims to minimise equation (5.75) subject to the equations given in Section 4.2 equation $(4.16 - 4.27)$, $(4.34 - 4.36)$ and modified constraints $(5.68 - 5.74)$.

Therefore the modified model becomes:

$\min \{(5.75) | (4.16 - 4.27); (4.34 - 4.36); (5.68 - 5.74)\}$

As a summary, the modified model does not carry the $l_{\lambda,i,j,t}$, w_{it}, $\alpha_{j,t}$ decision variables and the constraints $(4.28 - 4.33)$ which are related to these decision variables, any more. The number of integer/binary decision variables in the model are significantly less than the original model. Also, these constraints are expressed differently in modified version by just using the decision variable x_{ijt}, accordingly we can claim that modified model has a simpler design. It is easier to work with these constraints for advance programming and optimisation techniques; especially, to point out the reasons of longer solution times and diagnose errors while being suitable to possible adjustments.

Computational studies for the instances solved both by model in Section 4.2.1 and modified model are good indicators of the validity and practicality of this new model. Initially, we solved 240 instances, which we have tested before with direct solution and heuristics, by using LP relaxation of three models that were original formulation (Original) described in Section (4.2.1), modified model with (i.e. MOD-CB) and without (i.e. MOD) the constraints ((5.72)-(5.74)),respectively.

The full code implemented in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) for the computational study is presented in the appendix section B.1.2.

MIP differs from linear programming (LP) with having the feasible set of all integer-valued points within the polytope rather than the whole polytope. Accordingly, the optimal solution is searched in an extreme point of the convex hull of all feasible integral points. To relate LP solutions to the solution of MIP, we can express that LP has larger feasible region comparing the IP. This suggests that the optimal value to the LP gives a lower bound for the ILP. However the MIPs are computationally expensive, the relation between the optimal LP value and the optimal integral solution can be used as an indicator for the performance of different models on a specific problem.

Following this relation between ILP and LP, we applied the relaxed LP version of all these 3 models. We observed that Mod-CB had the highest objective function value for all 240 instances. We calculated the percentage gap between the objective function values obtained from the modified models and original model after lp relaxation. Depending on these results, the average gap between models MOD-CB and Original model is 29.58% while it is 0.22% for MOD and Original.

All instances are solved by all these three models which have the same settings with 10 minutes limit in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7). The

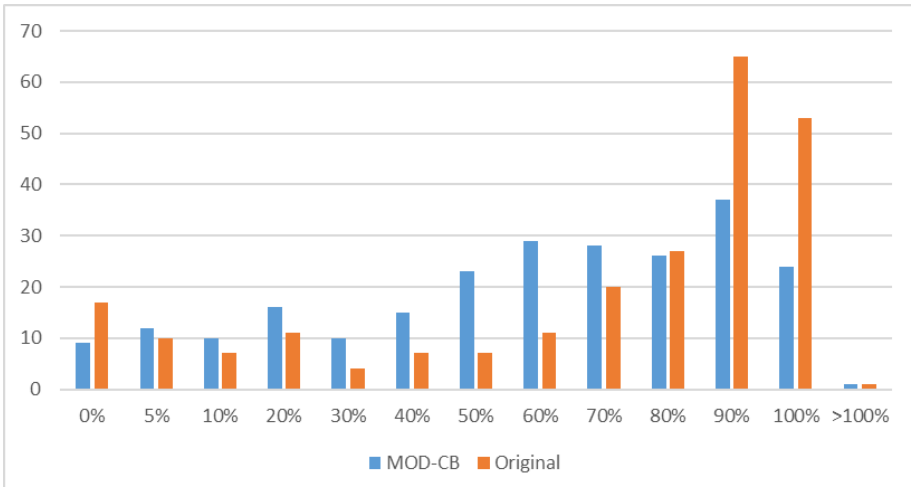

Figure 5.1: Comparison of GAPS to BB for MOD-CB 10-min run and Original 1-hr run

average gaps within best bound for MOD-CB, MOD and Original model are 51.75, 57.97 and 67.96 %, whereas the number of optimum solutions out of 240 sets are 18, 15 and 14, respectively. Additionally, MOD-CB has higher lower bound in comparison with MOD on the 169 instances out of 240.

Following the higher performance of MOD-CB, we carry the analysis based on the results of MOD-CB. In order to carry the computational study more concretely, the costs of recovery schedules obtained from MOD-CB and Original model have to be comparable under the same objective function. In other words, the results of x_{ijt} coming from the both models are applied to the original model which holds long work cost for each instance. Through this study, two models became suitable to be compared on the same plaque.Related results showed that the significant difference with respect to percentage gap between the MOD-CB and Original model is still being held with, 54.18 % and 64.40 % respectively. Since we would like to see the effectiveness of the modified model in a short time, we run the models with 10 minute time limit. At the same time, we paid attention to carry the comparison of 1 hour run with original model versus the 10 minute run with modified model. Even with this variation in time limit, the renewed model reached smaller costs compared to the original model on 178 instances out of 240. Additionally, on average, modified model has less gap to the best bound with 10% in average.

Besides, the constraints (5.72)-(5.74) are sufficient to express constraints related

to the rest. There is no need to keep track of weeks that shows the resting period of employees by using decision variables $r_{i,t}$. However, using them as combinatorial cuts (see Section 5.3) is preferred to strengthen dual sub-problem cuts and have less constraints in RMP. We added these combinatorial Benders cuts as it is needed in the sequel of DSP rather than adding all of them simultaneously.

Number of Constraints	84921
Number of Columns	95854
Number of Non-zero Elements	406060

Table 5.1: Summary of Simplified Recovery Model

In addition to the computational study for comparing the modified model and original model, we have tested this new economic model on both medium and large instances to observe the performance with and without constraints $(5.72 - 5.74)$. Our objective by this revision is to be able to obtain better bounds with reasonable solution times. Accordingly, the first assumption is that modified model finds more alternative schedules with lower cost than the original model in the shortage of time to solve the problem. The second assumption is about the constraints added for having stronger cuts related to minimum rest (e.g. 5.72- 5.74) improves the quality of bounds in comparison to the bounds of modified model without adding these cuts.

5.3 Combinatorial Cuts

Preliminary studies of BD algorithm application to our problem show that feasibility cuts are generally very weak to obtain good lower bounds. As suggested in Yang and Lee (2012), weak feasibility cuts lead to slow convergence of BD algorithm. We experienced that the problem had difficulty to find feasible results to get efficient cuts from sub-problem. To solve the problems raised from DSP unboundedness issues, Codato and Fischetti (2006) suggested Combinatorial Benders Cuts for minimal infeasible subsystems. We want to leave out the infeasible region with the help of combinatorial cuts. As a result of this situation, we add the constraint set (5.74) to the RMP in the case of obtaining extreme rays from sub-problem and to prevent finding the same infeasible solutions in the next iterations.

$$\min \sum_{i=1}^{m+1} \sum_{j \in J} \sum_{t=1}^{T} \left(c_{ijt}^{W} x_{ijt} + \sum_{\forall \lambda} c_{\lambda ijt}^{L} l_{\lambda ijt} \right) + S \tag{5.76}$$

$$\sum_{i=1}^{m+1} e_{ijt} x_{ijt} = a_{jt} \qquad \forall j, t \tag{5.77}$$

$$\sum_{j \in J} x_{ijt} \leq 1 \qquad \forall t, i \in E_R \tag{5.78}$$

$$w_i^{max} \geq w_{i,0} + \sum_{j \in J} \sum_{t=0}^{w_i^{max} - w_{i,0}} x_{ijt+1} \qquad \forall i \in E_R \tag{5.79}$$

$$w_i^{max} \geq \sum_{j \in J} \sum_{k=0}^{w_i^{max}} x_{ijt+k} \qquad \forall t, i \in E_R \tag{5.80}$$

$$\alpha_j^{max} \geq \alpha_{r,0} + \sum_{t=0}^{\alpha_r^{max} - \alpha_{r,0}} x_{m+1,j,t+1} \qquad \forall j \tag{5.81}$$

$$\alpha_j^{max} \geq \sum_{k=0}^{\alpha_r^{max}} x_{m+1,j,t+k} \qquad \forall j, t \in 1..T - \alpha_j^{max} \tag{5.82}$$

$$\sum_{k \in V} d_{ikt} + \sum_{j \in J} x_{ijt} \leq 2 \qquad \forall t, i \in E_R \tag{5.83}$$

$$x_{i,j,t+y} = 0 \quad \forall i \in E_R, j, t \in \{1, .., r_{i,0}\} \tag{5.84}$$

$$\sum_{j \in J} x_{i,j,t+y} + \sum_{k \in V} d_{ikt} \leq 1 \quad \forall i \in E_R, t, y \in \{0, .., c\} \tag{5.85}$$

$$\tag{5.86}$$

$$d_{ik1} \geq s_{ik} - \sum_{j \in V_k} x_{ij1} \qquad \forall k, i \in E_R \tag{5.87}$$

$$d_{ikt} \geq \sum_{j \in V_k} x_{ij,t-1} - \sum_{j \in V_k} x_{ijt} \quad \forall k, i \in E_R, t \in \{2, .., T\} \tag{5.88}$$

$$S \geq (b - Bx)^T \pi_p \quad \forall p \in \{extreme - points\} \tag{5.89}$$

$$0 \geq (b - Bx)^T \pi_p \quad \forall p \in \{extreme - rays\} \tag{5.90}$$

$$x_{ijt} \in \{0, 1\} \qquad \forall i, j, t \tag{5.91}$$

$$S \in \{-\infty, +\infty\} \tag{5.92}$$

$$d_{ikt} \in \{0, 1\} \qquad \forall k, t, i \in E_R \tag{5.93}$$

$$r_{it} \geq 0 \qquad \forall t, i \in E_R \tag{5.94}$$

Since there is a high possibility to encounter with infeasibility from sub-problem and solving RMP with weak cuts, we decided to apply these cuts to enhance the efficacy of Benders' method. From now on we are working with the model in Section (5.3) It is way more efficient than the initial attempts for the small problem but still not working well for real sized problems. This model starts solving the RMP without constraint set (5.74) which means $(i * t)$ number of constraints do not need to be activated from the beginning of algorithm. They are added to the problem when infeasibility detected in the SP. This method might be beneficial for excluding infeasibility and spend less time arbitrary / irrelevant solutions.

5.4 Modern Benders Application with Lazy Constraint Callbacks

More than 50 years past from the BD invention, the technology in solvers have been improved a lot. In classical BD model, we can add cuts only when we solve the RMP and determine which type of cut we are planning to add, accordingly. Although the modern solvers provide the opportunity of checking every incumbent while RMP are being solved, it is not possible to maintain the search to pause in every incumbent without callbacks. In the case of working with a solver supporting callback, it is possible to get the information of incumbent and test what type of cut would be added and what would be the UB can be obtained through the questioned incumbent solution. With the advanced programming knowledge, it is an option to carry the search through one tree in BD algorithm, rather than waiting for the optimal results of the RMP which is an IP for our problem.

Thorsteinsson (2001) defends the benefits of not constructing a new master search tree. Since there are limitations in terms of memory and time, validating the convergence of the BD technique may not be attainable. On the other hand, it is sufficient for a decision-maker to have a feasible solution in many practical applications. Therefore, there is no need for decision makers to have an optimal solution. Besides, good feasible solutions are usually obtained earlier during the solution process (Rahmaniani et al., 2017) .

By observing the performance of our problem during solution of master problem, we concluded that for large sized and medium sized problem it spends very long time to reach optimality starting from the first iteration. Therefore, working through

one tree (modern approach) can be beneficial to improve our solution quality. The primary aim of using integer callbacks is to prevent rework and not solving the problem from scratch. This new method can be advantageous since it avoids from revisiting a node, does not construct a new search three and spend time on an arbitrary solution. On the other hand, it can be disadvantageous to prefer modern approach over classical one due to the fact that solving sub-problem every incumbent can be costly. Especially, if the cuts that would be avoided by classical approach is added as a result of the preference of modern approach.

Pseudo code of one-tree search is shown in Algorithm 5.1.

Algorithm 5.1 Modern Benders Decomposition Algorithm

Require: y := initial feasible integer solution, UB:= $+\infty$, LB:= $-\infty$
While UB-LB$\leq \epsilon$ **Do**
Solve sub-problem: $\max_u \left\{ f^T\hat{y} + (b - B\hat{y})u | A^T u \leq c, u \geq 0 \right\}$
If Unbounded then: get ray u ; add cut $(b - By)^T\hat{u} \leq 0$ to RMP
Else Get extreme point u ; add cut $z \geq f^T y + (b - By)^T\hat{u}$ to RMP
UB:=$\min \left\{ UB, f^T y + (b - By)^T \right\}$
End If
Solve RMP: $\min_y \{z | cuts, y \in Y\}$, LB:= z
Pause when an incumbent found
End while

In our formulation to solve VCS in OSV's, we are facing the difficulty to find a new incumbent solution, and it is like a black box to foresee the reaction to the solver. To solve our problem we implemented classical approach in the beginning and as a result of low performance we have tried to apply callback concept in FICO® Xpress-MP by using Mosel language (Mosel v3.6.0, Xpress-MP v7.7). The procedure of how callbacks work in FICO® Xpress-MP solver is visualised in Figure 5.2.

Callbacks are to help users for defining their own routines and they can be called during the optimisation and interrupt the solver while maintaining the search with the new requests. It is possible with Mosel to obtain the values of decision variables by using one of the callback functions to obtain the values of current integer solution, and send these solutions to the sub-problem to generate new dual variables. Although the new cuts can not directly change the structure of the problem in Mosel, it is possible to collect data from every incumbent. Only way of prompting solver for new cuts is using cut manager callbacks through branch and bound callbacks (FICO® Xpress-MP Optimization (2007)).

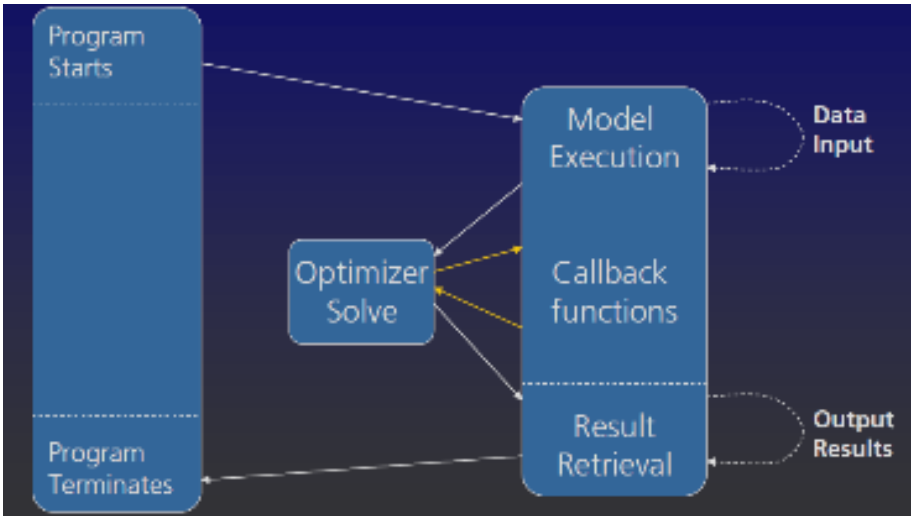

Figure 5.2: Callback Procedure (Optimization, 2007)

Similar to the innovation of callbacks, lazy constraints have been helpful in solving some problems that have been formulated with many constraints. For some of these problems with many constraints, it can be the case that some of these constraints might not be activated during the solution process. The main method to develop a problem using lazy constraints is to eliminate a complex and confusing part of the problem. If a solver can manage to find an incumbent, that solution is tested against the eliminated constraints. In situations when the solution is not feasible, a new constraints is included to the problem in order to interrupt any possible solutions as well as the specific one. Lazy constraints are employed as subtour elimination constraints in travelling salesman problems (Aguayo et al., 2017), and, Pearce and Forbes (2017) used them in a logic puzzle problem to show the application of lazy constraints and composite variables.

Regards to this information, another attempt in the implementation of BD related to callbacks is using lazy constraint logic for Recovery VCS problem. We have tested the efficacy of lazy constraints by eliminating the integer solutions that violate the rest constraints in RMP by reintroducing the constraints 5.86 as lazy cuts. We applied the lazy cuts with the help of *preintsol* callback in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7), which rejects integer solution if it violates the minimum rest constraints in our problem. Besides, in the case of rejection the solver does not allow to send the values of fixed variables to the sub-problem. Reminding the information

of not being able to add cuts apart from branch and bound callbacks, we had to call an extra callback function to add cuts without resolving the problem. Alternatively, we could exploit time limit to add all violated cuts and start solving the next iteration for BD. Although using preintsol callback and add lazy cuts when it is required was advantageous for small sized problems in medium and large instances it resulted with spending most of the time find a feasible integer satisfying all constraints and kept generating feasibility cuts for hours. As an example, for Set-1 it run purely with preintsol callback for 6 hours and not able to find one feasible integer solution. Based on the experience we encountered with different implementation for callbacks, we decided to carry the computational analysis and improve the algorithm by using all constraints as user cuts.

5.5 Pareto Optimality Cuts

Depending on the preliminary observations, we can state that classical BD algorithm shows very poor performance compared to the direct solution for our problem. Therefore, we conducted comprehensive research in BD literature for some enhancement in MIP problems. In the pursuit of finding better cuts to increase time efficiency for reaching optimal solutions, pareto-optimal cuts are suggested widely in the area of operations research. These type of cuts are important because pareto-optimal cuts cannot be dominated by any other cuts. Pareto-optimal cuts to reinforce the BD are first defined by Magnanti and Wong (1981). They called their technique as acceleration technique, which is the selection by alternate optima of the Benders sub-problem. In their study, Magnanti and Wong (1981) showed that accelerating cut generation technique provides efficient results for network location problems.

Following Magnanti and Wong's study, Papadakos (2008) suggested some enhancement for pareto optimal cuts for finding alternative core point with independent version of Magnanti and Wong (1981). Based on the paper of Papadakos (2008), domination between two points can be defined in BD algorithm with the comparison of two points in equation 2.14. If: $(b - B\hat{y})^T u_1 \geq (b - B\hat{y})^T u_2 \forall y \in Y$ where u_1 and $u_2 \in u$ then u_1 dominates u_2. Under the light of this information, Magnanti and Wong employed core points to define pareto optimality. Core point y^0 provides pareto-optimal cuts for optimal solution u for problem:

$$\max(b - By^0)^T u \tag{5.95}$$

subject to

$$(b - By^0)^T u = z(\hat{u}) \tag{5.96}$$

$$A^T u \leq c \tag{5.97}$$

$$u \geq 0 \tag{5.98}$$

Papadakos described independent version of Magnanti-Wong's (MW) problem under the condition of feasible sub-problem. In order not to deal with unboundedness, one can get pareto-optimality through the solution of the system $max\{(b - By^0)^T u | A^T u \leq c, u \geq 0\}$.

Apart from the difficulty based on the solution of sub-problem, discovery of core points is challenging for our problem. Papadakos suggested using the convex combination of current points obtained by restricted master problem as an alternative way of obtaining Magnanti-Wong points to generate pareto-optimal cuts. Based upon our research in literature (Magnanti and Wong, 1981) and (Papadakos, 2008), as an enhancement method for our BD implementation, we suggest to define dynamic core points that can be generated after every iteration following the master problem. Reminding that the fixed decision variables for our problem is x_{ijt} and d_{ikt} where i, j, t represents employee, task and week respectively:

$$x^0_{ijt} := (0.3) * x^{initial}_{ijt} + (0.7) * \hat{x}_{ijt} \tag{5.99}$$

$$d^0_{ikt} := (0.3) * d^{initial}_{ikt} + (0.7) * \hat{d}_{ikt} \tag{5.100}$$

With the help of these equations (5.99) - (5.100), we can find core points to use in MW through initial feasible solutions and the solution of the most recent iteration from RMP. Through this method, we give more weight on the result of current iteration and less weight on the result of first iteration. Accordingly, we implement pareto-optimal cuts within BD algorithm. The pseudo code of algorithm with MW method is given in Algorithm 5.2 and the implementation of this algorithm can be found in appendix section B.1.3. We start algorithm with an initial feasible solution for RMP. This initial solution is employed to solve DSP. Based upon the solution of DSP, we either solve MW problem or we pass the results of unboundedness with a feasibility cut to the RMP. In the case of optimality in DSP, before sending the

results with an optimality cut to the RMP, we solve MW with core points find from equations (5.99 - 5.100). If we have an optimal solution through MW problem, new dual variables are sent to the RMP with optimality cut instead of the ones found in DSP. We continue to apply the algorithm to the problem until we reached the optimality.

Algorithm 5.2 Benders Decomposition with MW Problem

Require: y := initial feasible integer solution, UB:= $+\infty$, LB:= $-\infty$
While UB-LB<= ϵ **Do**
Solve sub-problem: $\max_u \left\{ f^T\hat{y} + (b - B\hat{y})u | A^T u <= c, u >= 0 \right\}$
If Unbounded then: get ray u ; add cut $(b - By)^T u <= 0$ to RMP
Else Get extreme point u ;
Solve MW-problem: $\max_u \left\{ f^T\hat{y} + (b - By^0)u | A^T u <= c, (b - By^0)^T u = z(\hat{u}), u >= 0 \right\}$
If Unbounded then: use result of SP ; add the cut accordingly
Else Get extreme point u ; and add cut $z >= f^T y + (b - By)^T \hat{u}$ to RMP coming from MWP
UB:=$\min \left\{ UB, f^T y + (b - By)^T \right\}$
End If
Solve RMP: $\min_y \{ z | cuts, y \in Y \}$, LB:= z
End while

However, between the iterations for large sized problems it is not common to reach optimal result from RMP. Therefore, we stated the theoretical approach behind application of our problem to BD with MW Pareto optimal cuts. We gave the detailed description of our tune settings and results of various implementation of BD algorithm for recovery vessel crew scheduling in Section 5.7.

5.6 Using Heuristics in Benders Decomposition Algorithm

Looking from a heuristic perspective, the BD method can be considered as an appealing technique, since it can benefit from distinctive structures and provides a well-structured framework for the architecture of effective search processes (Raidl et al., 2015; Cote and Laughton, 1984).

The need for solving a sequence of problematic integer master problems is an essential challenge in large-scale problems. Some researchers have discovered the employment of meta-heuristics for the master problems. For example, in their paper,

Poojari and Beasley (2009) adopted a genetic algorithm together with a feasibility pump. This method helped Poojari and Beasley (2009) to include multiple cuts for each iteration, which generated a more significant increase in the lower bounds. In the end, good results were acquired even though the RMP was not solved to desired optimality.

Similarly, we combined heuristics within BD to improve the performance of the algorithm. By this application, we aimed to have stronger optimality cuts from the slave problem, help the reduction of upper bound and generate more alternative solutions. Besides, we would like to get some help of heuristics when IP is stuck in an integer solution and could not progress from the incumbent found by the optimizer and hence having memory error.

The heuristic has been implemented in the BD algorithm and arranged to be called every time when the solver reaches the specified time limit. The limit has been set to avoid spending long times to find a new integer solution, while solving the RMP. If an improvement on the upper or lower bounds can not be observed within the time limit, we stopped solving the RMP and extracting the most recent integer solution from solver. We used these integer solutions to initiate the heuristics with a feasible solution and conduct a neighbourhood search by the help of these values. As heuristic search completed, we solved dual problem regards to the heuristic results and added the new cuts to the integer problem.

The final implementation of BD algorithm with heuristic method is presented in Section 5.7.

5.7 Implementation of Benders Decomposition

The final algorithm that we preferred as the most efficient way of using BD algorithm and our customized heuristic is the hybrid of these two methods strengthened with pareto optimal cuts. The pseudo code of this hybrid method can be demonstrated as in Algorithm 5.3 and the actual codes for implementation of Algorithm 5.3 is provided in appendix section B.1.4. Additionally, during the computational study we refer the results that are obtained through Algorithm 5.3 by calling it as Hybrid Method.

In Section 5.8, computational efficiency of the hybrid method is also analysed with other BD algorithms (i.e. Algorithm 5.2, and 5.1).

Algorithm 5.3 Benders Decomposition and Heuristic for Recovery VSC

Require: y := initial feasible integer solution, UB:= $+\infty$, LB:= $-\infty$, iteration:=1
While UB-LB<= ϵ or Running Time $\leq$ TIME LIMIT **Do**
Solve sub-problem: $\max_u \left\{ f^T\hat{y} + (b - B\hat{y})u | A^T u <= c, u >= 0 \right\}$
If Unbounded then: get ray u ; add cut $(b - By)^T u <= 0$ to RMP
Else Get extreme point u ;
Solve MW-problem: $\max_u \left\{ f^T\hat{y} + (b - By^0)u | A^T u <= c, (b - By^0)^T u = z(\hat{u}), u >= 0 \right\}$
If Unbounded then: use result of SP ; add the cut accordingly
Else If Feasible then: Get extreme point u ; and add cut $z >= f^T y + (b - By)^T \hat{u}$ to RMP coming from MWP
UB:=$\min \left\{ UB, f^T y + (b - By)^T \right\}$
End If
Solve RMP : $\min_y \{ z | cuts, y \in Y \}$, LB:= z
If Number of incumbent found = iteration then:
iteration++
Else Continue till time limit is reached
End If
Go to Step 1 and solve sub-problem with the most recent $\hat{y}$, store new cuts
If Feasible then: Apply customized heuristic and obtain new $\hat{y}$
End If
End while

5.8 Computational Study for Benders Decomposition with Mixed Techniques

Development of BD algorithm and mixing it with accelerating techniques requires a detailed study to have accurate results. Especially with models including many different constraints and various (0-1) integer decision variables, having faulty results is highly probable. For BD it is really important to check accuracy based on dual problem and ensure that the decomposed models complete each other with linking constraints. Hence, we put significant amount of effort to debug our solution methods with various settings. We improved our updated solution method through the insights we gained from the preliminary results.

Here, we start to present computational study with the performance of selection of BD algorithm on MOD-CB by using one set as an example. We test this example instance with these three versions of the classical BD Algorithm:

- Classical BD,
- Classical BD with Magnanti and Wong Pareto Optimality Cuts,

Set	Classic BD		Classic BD with MW		Classic BD with Independert MW	
	CPU Time	UB	CPU Time	UB	CPU Time	UB
Set R001	99.96	5348.5	100.13	1855	100.13	1855
	249.96	1855	250.15	-203	250.15	-203
	449.88	-145.5	450.56	-203	450.25	-203
	699.88	-145.5	700.59	-203	700.54	-203
	1000.12	-207	1000.18	-224	1000.39	-224
	1349.85	-207	1350.21	-224	1350.28	-224
	1749.94	-207	1750.35	-224	1750.89	-224
	2200.14	-207	2201.61	-224	2201.26	-224
	2699.92	-212	2702.20	-224	2701.23	-224
	3250.54	-315.5	3252.82	-363	3251.17	-224
	3600.22	-315.5	3601.43	-363	3599.53	-224

Figure 5.3: Preliminary Result for Classical BD on Medium Sized Sample Set R001

- Classical BD with independent Pareto Optimality (Papadakos, 2008).

We applied 1-hour time limit and keep the time under control within each iteration while running the solver. The results are written after every iteration and they are shown in Figure 5.3.

This instance is generated with the intention of testing medium sized problem. Therefore it has 24 employees, 10 weeks and 14 vessels as main characteristics of data. Although it is medium sized, it was still hard to solve this instance in reasonable solution time. At the end of 1 hour run with MIP, the objective function reached is -430.5 and the lower bound is -1065.24. When we compare this result with BD application, we saw that the solution found by MOD-CB overcomes the classical BD performance for all three versions. In the end of 1-hour run with classical BD method, the best result obtained is -363. This result is obtained by the version supported by MW cuts.

Similar to the application of classical BD approach, we tested three versions of Modern BD performance on the same instance:

- Modern BD,
- Modern BD with MW Pareto Optimality Cuts creating the search tree with the same fixed binaries,
- Modern BD with MW Pareto Optimality Cuts creating the search tree with the updated fixed binaries.

For the modern approach, the independent MW is not applied on the sample set. Instead of testing the independent MW, the selection of values for fixed variables which feeds the sub-problem is tested.

Set	Modern BD (same intial)		Modern BD with MW (same intial)		Modern BD with MW	
	Time	OF	Time	OF	Time	OF
Set R001	1.54	6913.5	1.63	6913.5	39.83	2312
	3.05	6913.5	3.85	2258	99.77	-25.5
	15.35	2330.5	6.17	2258	179.86	-85.5
	22.93	1261	14.20	1698.5	279.75	-85.5
	31.99	1261	45.49	1698.5	399.95	-85.5
	574.82	-238.5	72.64	914	540.82	-260.5
	605.63	-238.5	103.38	542	701.89	-260.5
	636.66	-238.5	123.02	542	882.81	-260.5
	670.70	-238.5	139.46	200.5	1083.88	-260.5
	703.61	-238.5	156.82	-217.5	1305.01	-260.5
	1460.16	-238.5	1357.75	-257.5	1546.80	-260.5
	1492.35	-238.5	1544.28	-257.5	1808.03	-260.5
	1545.90	-238.5	2051.29	-257.5	2089.80	-260.5
	2449.32	-249	2087.95	-257.5	2390.77	-260.5
	3599.85	-358	2120.54	-257.5	2713.02	-260.5
	3599.85	-358	2548.52	-257.5	3054.85	-260.5
	3599.85	-358	3603.72	-333	3601.83	-276

Figure 5.4: Preliminary Results for Modern BD on Medium Sized Sample Set R001

Figure 5.4 underlines the performance of Modern BD for different settings and it can be seen that MW cuts do not perform well with this approach.

Even the performances between Modern and classical approaches are quite similar to the each other, MIP found the lowest cost compared to the various implementations of BD algorithm.

Further analysis with classical and modern approach on the generated medium and large scale instances are conducted and presented in Section 5.8.1, and 5.8.2.

5.8.1 Computational Results of Classical BD

To gain further insights, we carry out the computational study first with generated medium-sized instances. Thirty-five medium-sized instances are randomly generated, and these instances are solved by using MIP and BD algorithm. The solution approaches which are applied to these randomly generated instances are MOD-CB model, classical BD with original MW (BD-MW) and independent MW (BD-AC) cuts. As a result of this computation study, it is observed that BD algorithm performs less efficient than the MIP (see Figure 5.5) under the same computational time limit and using the default settings of optimisation solver. The results also indicate that BD-MW (i.e. classical MW cuts, Magnanti and Wong (1981)) provides better quality results than BD-AC (i.e.independent version of MW, Papadakos (2008)) on 24 instances over 35. In addition to this, the average percentage gap for BD-AC has the highest gap with 152.87% while this value for BD-MW and MOD-CB are 116.16%, and 113.87%, respectively.

Another observation has a minimal improvement for MOD-CB from 10-minute solution to the 1-hour. In other words, even MOD-CB finds fewer costs, to be able to

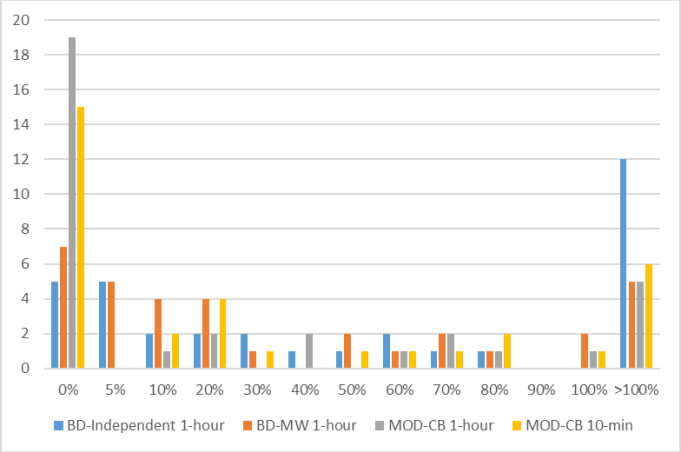

Figure 5.5: Dispersion of Percentage Gaps between the Lower Bound and Objective Function Value for MOD-CB and BD on 35 Instances

reach optimality; different exact methods are still needed. As a result of this result, some techniques to enhance the BD method are employed.

This initial study was helpful for determining the direction of further computational tests. The methods BD with MW and MIP are applied for additional 35 instances for medium sized. The percentage gaps are calculated for 70 medium instances with ((UB-LB)/LB). The mean values for this gap which is obtained through BD with MW and MIP within 1-hour time limit are 76.82% and 72.91%, respectively. When each data points are observed, some extraordinary gap values can be seen. Due to the existence of extremely high percentages such as 1493% and 693%, the mean may not be the best representative, and median values can be analysed. The median value of parentage gap for solution BD with MW is 7.15%, while it is 0 for MIP. Based on the mean and median values, MIP solution outperforms the classical BD with MW for the medium-sized problems. In 45 of the 70 instances, MIP method has better results, and the percentage difference between these two solution methods is measured by ((BD-MIP)/MIP). MIP achieved to reach smaller costs than BD with MW reached, and the difference on average is 10.2% based on these calculations. There is no distinct pattern that BD performs better or worse on the particular type of instances, but it can be stated for medium-sized problem BD-MW has quite a similar efficiency with MIP. In comparison to the first attempt of BD on our problem, model modification and additional techniques developed the quality of solutions significantly.

5.8.2 Computational Results of Modern BD

Through this dissertation, the instances which reflect the situation in our industrial partner are mainly tested to find the best practice. Based on this perspective, preliminary computational study is conducted for large sized instances, which have 48 employees, 25 vessels, 13 weeks as main parameters, to examine the BD performance. Since the classical BD has already been applied to 70 medium scale instances, the modern (one-tree) approach (i.e. Algorithm 5.1) is tested for large scale instances with a small adaptation.

Various versions of BD Algorithm are implemented on optimisation solver FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7). Since cut adding during the optimizing is not possible apart from the MIP branch-and-bound callbacks, we preferred using integer solution callback which is able to pause the MIP and solve DSP with storing the cuts. After certain time limit, all the cuts that are stored in previous iteration are added in the beginning of new iteration to solve RMP. In order to apply this method, we chose to keep the initial values for fixed variables to initiate the RMP for every new iteration.

Accordingly, 80-large scaled instances are solved by hybrid method that is explained in Section 5.7 with Algorithm 5.3 and classical approach. Both versions of BD are designed with MW cuts. The results, which are obtained through these two versions of BD, are compared with the MIP.

Sol-Method	Mod-CB	BD-MW	Hybrid Method
Mean	42.71 %	101.85 %	70.88 %
Median	9.24 %	56.98 %	38.26 %

Table 5.2: Results of Modern BD on Large Sized

The numbers in Table 5.2 indicate that for large sized instances the efficiency of BD method is quite low relatively to the MIP. Although, the hybrid method performed better than the classical BD, there is a significant difference between the MIP and BD performances regarding the mean and median values of percentage gaps. In addition to Table 5.2, Figure 5.6 shows the performances of each method visually based on the percentage gaps. The distances of the BD algorithms' results from the MIP solutions are 52.71% and 26.25%, respectively. The performance of BD decreased against MIP when the problem size increased. However, one tree approach is superior than the classical for 60 instances out of 80 with 7.25% on average.

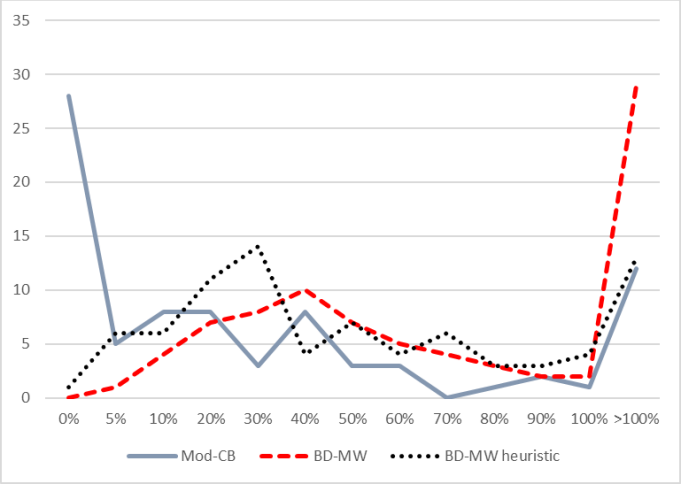

Figure 5.6: Dispersion of Percentage Gaps between the Lower Bound and Objective Function Value for MOD-CB and BD on 80 Instances

Similar to the results in Section 5.8.2, any special pattern cannot be found neither for low nor high efficiency for the hybrid method. However, while there are some instances that MIP has high percentage gap such as 137% from the best bound, hybrid method has low gap around 29.78%. The fundamental source of this situation cannot be identified since this case is not valid for the majority of the instances.

As the detailed analysis of computational study was presented in Section 5.8, the concluding remarks for the study about Benders Decomposition in Chapter 5.

5.9 Summary of Findings from Benders Decompositions

Here, in Chapter 5, we provided a comprehensive study on Benders Decomposition Method and its application to VCS in OSV's. In the beginning, the recovery model is used to apply BD, but it could not provide good quality results. Afterwards, a modified model is suggested, and BD is also tried on this modified model. It is explored that modified model outperformed the recovery model which is first proposed in Leggate (2016). Accordingly, the study on BD method is developed on the modified model (MOD-CB). The main advantage of MOD-CB can provide optimality cuts for RMP in BD applications and be more capable of potential developments.

Following this modification, a more reasonable solution is achieved for small sized

instance, and various improvement techniques for BD are tested on medium and large sized instances. The improvement techniques are mainly working with Pareto optimality cuts suggested by Magnanti and Wong (1981) and Papadakos (2008) and, employing the one-tree approach in an integrated method with the heuristic.

Based on mean and median values of percentage gaps, MOD-CB outperformed the performance of various BD methods on the sample instances. On the other hand, BD method seems promising and open to acceleration techniques to have better quality results. Another important point is for our problem; it is better to have an integrated heuristic method and work with a time limitation for every iteration instead of waiting to obtain optimal results from RMP. It is a critical remark since we encountered with either memory failure while searching optimal results for terminating just one iteration of RMP.

The potential development for future study about increasing the performance of BD for VCS in OSV are discussed in Section 7.3. The following chapter provides an initial study of the sources of uncertainties and a practical model suggestion to increase the robustness against uncertainty for this problem.

The potential development for future study about increasing the performance of BD for VCS in OSV are discussed in Section 7.3. The following chapter provides an initial study for the sources of uncertainties and a practical model suggestion to increase the robustness against uncertainty for this problem.

Chapter 6

Robust Counterpart for VCS in OSVs

Creating recovery schedules is practical regarding sudden changes, and decreases the cost due to unexpected situations. Another potential option can be to construct more robust schedules from scratch and minimise the cost for these unexpected situations. Accordingly, it is intended to suggest alternative solution methods to this problem by using robust optimisation methods.

Organizing schedules without the assumption of uncertainty are not the best practice. Despite the fact that we search the efficient methods for generating recovery schedules towards the unexpected changes during the planning horizon, it is also essential to measure the impact of having robust schedules from the start of the planning period.

The need for robust structures in maritime context was implied by a number of studies (e.g. Halvorsen-Weare and Fagerholt (2011); Scholz-Reiter et al. (2010)) and there are some applications in the literature to encourage the research in offshore. Scholz-Reiter et al. (2010) presented a study on the robust design of planning and control methods for offshore installations (i.e. an area of utilising offshore supply vessels). In their paper Scholz-Reiter et al. (2010) also focused on finding optimal schedules for installation under various weather conditions. Finally, they drew attention to the potential research area of vessel crew scheduling in the context of offshore industries.

As a result of volatile weather conditions at sea, crew availability and performance are subject to certain limitations (Stevens and Parsons, 2002). Based on the insights

we obtained through our case study, we focus on generating robust schedules against the uncertainty in the concept of crew availability. In order to reach this objective, the robust optimisation techniques are explored. According to the study of Gorissen et al. (2015), robust optimisation procedure of practical application can be further investigated under six main steps. These steps include the determination of the sources of uncertainty, measurement of the robustness level of nominal solution, application of required adjustment for variables, robust counterpart formulation and implementation, and finally the analysis of the robust solution based on conservatism level. While the first two steps of the procedure are already discussed in Chapter 3 by explaining the need of higher robustness, the remaining steps can be used as guidance.

Motions of vessel depend on the weather conditions a lot; it can cause an increase in energy expenditure and drowsiness level for crew members(Stevens and Parsons, 2002). In this dissertation, we also cover weather conditions and its adverse effect on crew availability by employing the eligibility parameter that belongs to an uncertainty set. In the model which is presented in Section 4.2, eligibility is represented with e_{ijt} and indicates whether the employee is available and skilled enough to be able to carry out the task. This parameter takes binary values depending on the availability of the crew which merely is shown as:

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

In the seminal papers of robust optimisation literature, such as (Soyster, 1973; Ben-Tal and Nemirovski, 2000; Bertsimas and Sim, 2004) the uncertainty is defined over a convex set of the parameters, which allows changes on the nominal values of parameters within certain deviation. As an example, in Bertsimas and Sim (2004) study, if it is assumed, coefficient of uncertain parameter is shown with a, then the uncertainty set can be stated as $[a - \hat{a}, a + \hat{a}]$ and this value can take any value from this set.

In our problem, it is not possible to define the eligibility with this kind of convex set as it is given above. Under the case of the values of the parameter e_{ijt} can only either be 0 or 1, it is not suitable to use set such as $[0, 1]$ since the non-integer values in this range do not reflect the problem compatible with real life. Accordingly, to the best of our knowledge, the proposed methods in the literature are not applicable

to our problem under this assumption. Therefore, it is suggested that having a base value for parameter e shown as $\bar{e} = 1$ can be adjusted by $\hat{e}$ value that can only be equal to 1. This proposition suggests that an eligible crew for a specific time window and role can be unavailable due to the volatile weather condition. In order to suggest a robust practice for our problem, as following another step suggested in Gorissen et al. (2015), in Section 6.1 the complete formulation of the robust counterpart is presented.

6.1 Formulation of Robustness

As the desire of robustness against crew availability is aforementioned, equation 4.16 in the MOD-CB (modified model) provided in Section 5.2 is subject to uncertainty.

This equation ensures that the required number of crews are assigned to the roles based on their eligibility (availability).

$$\sum_{i=1}^{m+1} e_{ijt} x_{ijt} = a_{jt} \quad \forall j, t \tag{6.1}$$

To express the uncertainty for this constraint, the equation 6.2 can be proposed as;

$$\sum_{i=1}^{m+1} \left(\bar{e}_{ijt} - \hat{e}_{ijt} z_{ijt}\right) x_{ijt} = a_{jt} \quad \forall j, t \tag{6.2}$$

where z_{ijt} is the decision variable that determines the existence of change in availability as given below:

$$z_{ijt} = \begin{cases} 1 & \text{if crew } i \text{ becomes unavailable to carry out role } j \text{ in period } t \\ 0 & \text{otherwise} \end{cases}$$

Under the assumption of uncertainty set is given in the range of [0,1] and to maintain the feasibility this parameter can only take values $\{0, 1\}$. This situation shows a similarity to the work of Soyster (1973) regarding having the feasibility for an entire convex set of uncertainty and relying highly on uncertainty set. Our suggestion is working with the extreme values of the set which can be perceived as high degree conservatism on robustness is covered by the method suggested in

Soyster (1973). Although when the practical application is observed, the chances of all crew become unavailable is quite low. Accordingly, a robust counterpart (RC) as proposed in Bertsimas and Sim (2004) to control the level of uncertainty study can be suggested.

In order to control the level of uncertainty, an additional parameter $\Gamma_t \in \mathbb{R}^+$ is employed. This parameter specifies the maximum number of crews who become unavailable for each week during the planning horizon however they are available at the beginning of planning (i.e. e_{ijt} changes from 1 to 0) on different vessels for each period. As the existence of the agency crew, the maximum value Γ_t can take for each period is the multiplication of the total number of crews with the roles defined on that period. In the case of $(\Gamma_t = |J| * |E| \ \forall t)$ shows the worst case scenario for our problem. At the same time, if $\sum_{t=1}^{T} \Gamma_t = 0$, in this case, the problem is solved with nominal values.

By holding the information about parameter Γ_t, which defines the level of conservatism, new decision variable z_{ijt} for robustness and constraint expressed with equation 6.2, further investigation can be done to construct the whole idea for providing more robust schedules to our industrial partner.

The equation 6.2 is non-linear, due to the multiplication of two binary decision variables. To linearise this equation an additional decision variable $K_{i,j,t} = z_{ijt} * x_{i,j,t}$ is defined and supported by constraints:

$$\sum_{i=1}^{m+1} \bar{e}_{ijt}.x_{ijt} - \sum_{i=1}^{m+1} K_{ijt} = a_{jt} \qquad \forall j,t \tag{6.3}$$

$$\sum_{j=1}^{J} \sum_{i=1}^{m} z_{ijt} \geq \Gamma_t \qquad \forall t \tag{6.4}$$

$$K_{ijt} \leq z_{ijt} \quad \forall j,t,i \in E_R \tag{6.5}$$

$$K_{ijt} \leq x_{ijt} \quad \forall j,t,i \in E_R \tag{6.6}$$

$$K_{ijt} \geq z_{ijt} + x_{ijt} - 1 \quad \forall j,t,i \in E_R \tag{6.7}$$

$$z_{ijt}, K_{ijt} \in \{0,1\} \quad \forall j,t,i \in E_R \tag{6.8}$$

Therefore the robust problem becomes:

$$\min \{(5.75) \,|(4.17-4.27); (4.34-4.36); (5.68-5.74), (6.3-6.8)\} \tag{6.9}$$

This problem aims to minimise the crew cost depending on work, boarding, departing and under-over time in the defined planning horizon by the objective function provided with equation (5.75). The constraints (4.17)-(4.27) have been explained in Section 3 with details and they ensure that the rules and regulations are followed. Additionally, constraints (4.34)-(4.36) and (5.68)-(5.74) are added to complete the whole problem with other requirements in their modified form as described in Section 5.2. Constraint (6.3) is the new form of covering the tasks to provide the robust counterpart. Constraint (6.4) ensures that the minimum number of changes in the availability of crew on different task are occurred for each week by the help of decision variable z_{ijt}. The constraints (6.5-6.7) are employed to linearise the non-linear equation (6.2). Finally, constraint (6.8) indicates the binary variables.

This model provides the control of uncertainty level by changing the values of Γ_t for each week. Based on the value of Γ_t , the cost of crew can change. From another point of view, one can see the flexibility of the current system under a defined cost. In this case, it is possible to generate the alternative schedules with maximum change under the pre-defined cost (Ξ) by using the robust model. To be able to find these schedules, a constraint (i.e. equation (6.10)) which guarantees an objective function value to be smaller or equal than the pre-defined cost is required. In addition to this constraint, Γ_t can be defined as integer decision variable instead of parameter.

$$\sum_{i=1}^{m+1}\sum_{k\in V_k}\sum_{t=1}^{T}\left(c^{B}_{ikt}b_{ikt}+c^{D}_{ikt}d_{ikt}\right)+\sum_{i\in G}\left(c^{U}_{i}u_{i}+c^{O}_{i}o_{i}\right)+\sum_{i=1}^{m+1}\sum_{j\in J}\sum_{t=1}^{T}\left(c^{W}_{ijt}x_{ijt}\right)\leq \Xi \quad (6.10)$$

Another parameter that can be potentially subjected to uncertainty is the number of required crews for each task and period which is represented by a_{jt}. The uncertainty based on this parameter can be rephrased as demand uncertainty in OSVs. If an unexpected change occurs in the policy that the company is no longer able to maintain offshore operations in a certain location, it can result in no need for the crew to operate in the related vessels in that location. On the other hand, based on the oil price fluctuations it is also possible to observe an uncertainty in demand for OSVs (Gaspar et al., 2015). In this case, there is a potential for both trends (i.e. increase and decrease).

Parameter a_{jt} is similar to the parameter e_{ijt}, in terms of convex uncertainty set. It cannot take the non-integer values, and it is limited to be either 0 or 1. In

the case of a decrease in demand, the current planning would be still feasible, but this feasible solution cannot guarantee the optimality for minimum cost. On the other hand, in case of an increase in the demand, the current schedule can guarantee neither feasibility nor optimality. Therefore, a more comprehensive RC can still be considered as a significant decision support tool for better planning in the context of VCS in OSVs.

As a suggestion to deal with the uncertainty in the required number of crew (a_{jt}), another binary decision variables (y_{jt}) can be defined to allow the change in this parameter with an additional parameter to decide the level of conservatism (Θ).

$$y_{jt} = \begin{cases} 1 & \text{if change in } a_{jt} \text{ occurs for role } j \text{ in period } t \\ 0 & \text{otherwise} \end{cases}$$

The more robust formulation for the equation which is affected by this uncertainty is the same one that is previously discussed for availability in equation 4.16. In order to formulate robustness for this case, two different groups of equation are proposed depending on the nominal value of a_{jt}.

$$\text{RC for equation 4.16} = \begin{cases} \sum_{i=1}^{m+1} e_{ijt} x_{ijt} = a_{jt}.y_{ijt} & \text{if } a_{ijt} = 1 \\ \sum_{i=1}^{m+1} e_{ijt} x_{ijt} = a_{jt} + y_{ijt} & \text{if } a_{ijt} = 0 \end{cases} \tag{6.11}$$

Furthermore, for the determination of uncertainty level equation 6.12 is proposed:

$$\text{RC for } a_{jt} = \begin{cases} \sum_{j=1}^{J} \sum_{t=1}^{T} y_{ijt} \leq \Theta^u & \text{if } a_{ijt} = 1 \\ \sum_{j=1}^{J} \sum_{t=1}^{T} y_{ijt} \geq \Theta^l & \text{if } a_{ijt} = 0 \end{cases} \tag{6.12}$$

By equation 6.12 the number cases result in demand increase are bounded below, and potential decrease in demand is bounded above with parameter Θ^l and Θ^u, respectively. The right hand side of the equation 6.11 can be combined with the equation 6.3 which is provided for robustness against uncertainty in crew availability. Finally, the constraint 6.11 can be added simultaneously to the problem stated in equation 6.9. The implementation of proposed methods for robust models on FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) are provided in Appendix B.2.1 and

B.2.2, respectively.

The combined formulation of uncertainty in crew availability and demand uncertainty is applied to a sample problem set. This problem includes 48 employee, 13 weeks long planning horizon, and 25 vessels as it is proposed by our industrial partner. The number of possible disrupted availability is determined as at least 32 in total for each time windows (week). This number represents at least 5% perturbation from the original schedule for each week. The schedules are generated by the original formula and also with the addition of RC. After 1-hour run of both models, the original formula found a schedule which has cost 15149 and 66% gap; while the formula with RC reaches 50.26% optimality gap with cost 10263. Although it requires much more elaborate computational study, depending on the example, the robust formulation seems promising.

In addition to the crew availability and demand uncertainty, transportation costs for crew can be hardly assumed to be entirely deterministic based on the timing of sudden changes. This situation leads to uncertainty in the cost coefficients for the objective function. Thus, defining the parameter of transportation cost with intervals instead of exact values and then to minimise the crew cost may provide more robust solutions. To model this uncertainty, defining a convex uncertainty set for additional transportation cost is sufficient. Although, in this dissertation, there is no study provided to decide the range of this cost element, a simple implementation of this suggestion in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) is presented in Appendix B.2.3.

Chapter 7

Conclusions and Future Research

In this chapter, we mainly conclude our findings and contributions which is discussed in this dissertation. The conclusions are also supported by the research limitations and direction of potential work for future study.

7.1 Research Findings

In this dissertation, we searched for efficient solution methods to provide feasible schedules to the problem faced by our industrial partner and draw conclusions about the characteristic of our problem by making a deeper analysis based on the results obtained through suggested methods.

After searching the existing literature for the related subjects, we figured out the lack of crew scheduling studies in maritime context with the similar complexity of our problem settings. By reviewing the crew scheduling studies in various transportation settings, and the study of Leggate (2016) which is the only study represents the same case as our problem, potential optimisation techniques and their compatibility to our structure have been investigated.

Further computational study for understanding the TB method for larger instances also with longer computational time has been conducted, and it is observed that the TB method is not useful for mega instances (96-captains) as it is for large instances (48-instances). It is also concluded that increase in the computational time from 2-min to 10-min results in substantial improvement for mega instances. However, longer computational time (1-hour) does not effect the quality of the solution for both large and mega instances. The improvement of the percentage gap was

negligible with 0.01% and 0.09%, respectively.

When the more realistic model (Time Windows) is applied, the number of constraints and decision variables increase substantially comparing the Task Based model. As a result of our findings and the necessity of alternative crew schedules against sudden changes, a customised heuristic algorithm is implemented in C++. The primary purpose of the heuristic method is to generate economic and feasible schedules. Due to the high performance of this implementation, a good number of recovery schedules can be obtained even in less than a minute. After 2-minutes run, the number of schedules obtained for each instance is found 273 on average. It is important to underline that in addition to the design of this algorithm, solving this method in C++ instead of an optimisation software also provides a significant difference in efficacy. This method provides 18% decrease in percentage gaps in average compared to the direct application of MIP formulation given in Section 4.2.1. Also, it shows 34% improvement from the provided initial solutions to the best solution found at the end of the algorithm.

In addition to the heuristic method, with the goal of reaching optimality, Benders Decomposition algorithm is applied to the proposed model.This study initiated further adjustments and practical developments in the model which is earlier suggested in Leggate (2016). A complete application for Benders Decomposition algorithm is suggested with acceleration techniques. Moreover, a preliminary computational study on eighty large sized and seventy medium-sized instances is conducted. The classical approach shows similar performance to the MIP application of MOD-CB for medium sized instances. However, this situation cannot be maintained for the real (large) sized problems. BD displays lower performance than the MIP for the large instances. On the other hand, the modern BD approach outperforms the classical one for the large-scale instances. The study on this technique shows that this algorithm is open to further improvement for the development of hybrid method we suggested in Section 5.7.

Another critical point we initiated in this research is the modelling uncertainty for our problem. The primary sources of uncertainty are discussed as the potential change in the availability of crew and uncertainty in demand. We concluded that these uncertainties could only deviate from 0 to 1 or the other way around in the existing models. Accordingly, the uncertainties can be modelled by using simple MIP techniques in order to start planning with more robust schedules compared to current planning methods that are applied by our industrial partners. There is

only one computational study provided for the robust formulation in which the crew availability and demand uncertainty combined. Through this sample instance, 10% improvement is observed with a more robust schedule which allows 5% uncertainty in crew availability. The implementation of this model on FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) is presented in Section B.2.

We have previously stated the research questions in Section 1.3. In order to provide precise information regarding our study, we bring these questions back and address our findings by replying these questions.

- How can the proposed models for VCS in OSVs be improved to generate feasible schedules against unexpected situations in a more time efficient way by using optimisation tools?

As this is our main research question, we first provided answers for the sub questions.

- What are the weaknesses of proposed TB and TW models?

  Although TB model is good at reaching optimality in short computational time, it was not effective enough providing the minimum cost schedules due to the not holding all the requirements and work with pre-defined task blocks. When the solutions obtained through TB model approximated to the TW model, it has been observed that there is 81.25% gap between the TB and lower bound found from 1-hour run of TW model on average for 240 instances. The TB model in itself has 2.47% gap for the same instances. Accordingly finding optimality through TB model does not explain reaching optimality on more realistic model. On the other hand, TW model for the same settings have 88.57% gap after 2-min run. The weakness of TW model is that having low efficiency in terms of time like having 64% gap even after 1-hour run. Another disadvantage of TB is that modelling some of the regulations in overly complicated way and not being flexible enough for applying different optimisation techniques.

- How can one deal with the drawbacks of TW model?

  As the drawbacks of TW model is that having high gaps, complicated constraints and not flexible model, we worked on simplifying the model while keeping the model realistic enough for the business application. The TW model

is modified and applied on 240 instances similarly. The generated recovery schedules from modified model and TW model are compared on the same cost function. The results showed that the modified model is 13% closer to the optimality bound comparing the original TW model after 10-min run.

- Are heuristics and exact methods that are suggested in this thesis effective ways to solve TW model?

  We concluded that heuristics provide the most efficient solution against time limitation. This method generates 273 different schedules and improve the initial solution with 34 % in average. Additionaly, it has the advantage of suitability for various implementation and it is open for adjustments and improvements. While the mean percentage gap which is found through the implementation of TW model in Fico-Xpress optimisation solver is 64.33% in 1-hour, this value is 58.53% from the heuristic in 2-min which are based on the best result amongst 48 different heuristic. The classical Bender's Decomposition method performed low efficiency against the direct solution methods. Depending on the high efficiency of customised heuristic method, it is suggested that using hybrid methods of heuristics with Benders' Decomposition Algorithm might also provide cost efficiency. The hybrid method was compared with the modified model and the computational experiment suggested that the hybrid method could not over perform the modified model. While hybrid method has 70% gap, the modified model has 42% gap after 1 hour run. Although Benders' Decomposition method does not provide high efficiency, it helps to find the research direction and identify the weakness of TW model.

- What are the strength and weaknesses of the suggested methods?

  The strength of suggested heuristic method is the time efficiency and the weakness of this method is not being able to reach optimality. On the other hand, Bender's Decomposition method is not timely efficient for VCS problem in OSV's, it gives space for improvement with some acceleration techniques. This characteristic of this method is promising in terms of providing a future research scope. The combination of acceleration techniques like Pareto optimality cuts, one-tree search, combinatorial cuts and getting help from heuristic provided substantial convergence to MIP, as the first trial of classical Benders' Decomposition method was not able to provide efficient optimality cuts even for small

sized problems.

- How can one obtain schedules immune to uncertainty?

  In order to deal with the uncertainty, the dissertation suggests solutions mostly regarding recovery solutions. Apart from the recovery schedules, a simple robust counterpart with the assumption of unexpected change in crew availability and demand is proposed. It is understood that with a preliminary study, creating schedules with the consideration of possible change from the beginning might help cost savings and measuring the costs with different scenarios.

Based on the summary of our findings, our study contributes to the existing literature with new solution methods and benchmarks for the crew scheduling problem in transportation settings, particularly in maritime context. An extended analysis with valuable insights that explain the strength and weaknesses of our formulations and applications of suggested techniques are provided for future implications of similar problems with complicated settings and long planning horizon.

Following these, we give further analysis for limitations through this research in Section 7.2 and the potential for future studies in Section 7.3.

7.2 Research Limitations

Like many other studies, it is inevitable not to have any limitation during the research period. Providing research limitations is as important as highlighting the research findings since they also include some critical concluding remarks.

The major limitation we face in this dissertation is working with bug-free implementation of different solution methods. Initially, during the progress of implementing heuristics in C++, it is difficult to ensure that our code works correctly. The reasons behind this situation can be summarised as the complexity of constraints and structure of the problem which is hardly based on binary variables. Additionally, while there are many functions which are integrated to each other, the heuristic is also highly customised, and using C++ for the implementation is too complicated for this structure as well. An elaborate design and careful coding with lengthy debugging processes are needed in the trade of fast calculations obtained by using C++.

Similarly, we encounter a similar problem with the application of Benders Decomposition method. Even BD algorithm is not complicated; it requires working with duality for the evaluation of sub-problem which can be tricky to model for the complexity of our problem. It is vital to managing duality successfully in the BD algorithm implementation. A comprehensive debugging is also required for this method. In addition to the results checking process, to figure out the low efficiency of BD algorithm for our problem took long process time. As a result of the unexpectedly low efficiency, a various number of applications are tried for both acceleration techniques and way to decompose the model into two models.

Another limitation is about the power of the processing system in our computer during the computational runs and facing with memory failures in the process of master problems. Although the memory failures were helpful for us to change our direction to some other applications for BD, they were obstacles to measure the efficiency, properly.

Last but not least, the fact that this topic has not been widely discussed in the literature can be underlined as another limitation which negatively influenced our research.

7.3 Future Research

Finally, here in this section, we are going to discuss the potential future work which may use this research as guidance. Most of the suggestions are related to the improvements of Benders Decomposition Method which is widely discussed in Section 5 for our problem. Besides, the robust formulations also lead other research opportunities in this area.

In order to increase the efficiency of BD algorithm, the implementation can be made on an optimisation solver by using a language which provides more flexibility such as concert technology supported by C++. This implementation may help researchers to apply more complicated techniques simpler than having implementations in Mosel language. Also, the customised heuristic, which is explained in Section 4.3 and written in C++, can be modified for model MOD-CB (see Section 5.2) and applied in C++ for the Hybrid method. In addition to the different implementation approach, a more comprehensive computational study can be organised.

In the majority of this dissertation, we approached the problem by defining it as a

recovery crew scheduling problem rather than scheduling from scratch. As in Chapter 6, robust scheduling techniques are discussed with some suggestions for robust modelling. This study can be extended with a more elaborate model by exploring all possible sources which create uncertainties and gaining some more information from the industrial partner about the coefficient range and frequency of the uncertain parameters.

Bibliography

Aas, B., Halskau Sr, Ø. and Wallace, S. W. (2009). The role of supply vessels in offshore logistics, *Maritime Economics & Logistics* **11**(3): 302–325.

Aguayo, M. M., Sarin, S. C. and Sherali, H. D. (2017). Solving the single and multiple asymmetric traveling salesmen problems by generating subtour elimination constraints from integer solutions.

Ammar, M. H., Benaissa, M. and Chabchoub, H. (2013). Grasp for seafaring staff scheduling: Real case, *Advanced Logistics and Transport (ICALT), 2013 International Conference on*, IEEE, pp. 427–433.

Anbil, R. (1993). Crew-pairing optimization at american airlines decision technologies, *Optimization in industry* pp. 32–36.

Andersson, E., Housos, E., Kohl, N. and Wedelin, D. (1998). Crew pairing optimization, *Operations Research in the Airline Industry* pp. 228–258.

Atkinson, S. E., Ramdas, K. and Williams, J. W. (2016). Robust scheduling practices in the us airline industry: Costs, returns, and inefficiencies, *Management Science* **62**(11): 3372–3391.

Azaiez, M. N. and Al Sharif, S. S. (2005). A 0-1 goal programming model for nurse scheduling, *Computers & Operations Research* **32**(3): 491–507.

Baker, K. R. (1976). Workforce allocation in cyclical scheduling problems: A survey, *Journal of the Operational Research Society* **27**(1): 155–167.

Barlow, E., Öztürk, D. T., Revie, M., Akartunalı, K., Day, A. H. and Boulougouris, E. (2017). A mixed-method optimisation and simulation framework for supporting logistical decisions during offshore wind farm installations, *European Journal of Operational Research* .

Barnhart, C., Cohn, A. M., Johnson, E. L., Klabjan, D., Nemhauser, G. L. and Vance, P. H. (2003). Airline crew scheduling, *Handbook of transportation science*, Springer, pp. 517–560.

Barret, D. (2005). The offshore supply boat sector, *Sector Note, New York, USA: Fortis Bank* .

Ben-Tal, A. and Nemirovski, A. (1998). Robust convex optimization, *Mathematics of operations research* **23**(4): 769–805.

Ben-Tal, A. and Nemirovski, A. (2000). Robust solutions of linear programming problems contaminated with uncertain data, *Mathematical programming* **88**(3): 411–424.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems, *Numerische mathematik* **4**(1): 238–252.

Bertsimas, D. and Sim, M. (2004). The price of robustness, *Operations research* **52**(1): 35–53.

Birge, J. R. and Louveaux, F. (2011). *Introduction to stochastic programming*, Springer Science & Business Media.

Bixby, R. E., Gregory, J. W., Lustig, I. J., Marsten, R. E. and Shanno, D. F. (1992). Very large-scale linear programming: A case study in combining interior point and simplex methods, *Operations Research* **40**(5): 885–897.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM Computing Surveys (CSUR)* **35**(3): 268–308.

Brouer, B. D., Dirksen, J., Pisinger, D., Plum, C. E. and Vaaben, B. (2013). The vessel schedule recovery problem (vsrp)–a mip model for handling disruptions in liner shipping, *European Journal of Operational Research* **224**(2): 362–374.

Brucker, P., Qu, R. and Burke, E. (2011). Personnel scheduling: Models and complexity, *European Journal of Operational Research* **210**(3): 467–473.

Burke, E. K., De Causmaecker, P., Berghe, G. V. and Van Landeghem, H. (2004). The state of the art of nurse rostering, *Journal of scheduling* **7**(6): 441–499.

Cai, X. and Li, K. (2000). A genetic algorithm for scheduling staff of mixed skills under multi-criteria, *European Journal of Operational Research* **125**(2): 359–369.

Cappanera, P. and Gallo, G. (2004). A multicommodity flow approach to the crew rostering problem, *Operations Research* **52**(4): 583–596.

Chen, M. and Niu, H. (2012). A model for bus crew scheduling problem with multiple duty types, *Discrete Dynamics in Nature and Society* **2012**.

Chopra, K. (2017). What are offshore vessels. http://www.marineinsight.com/types-of-ships/what-are-offshore-vessels/.

Christiansen, M., Fagerholt, K., Nygreen, B. and Ronen, D. (2007). Maritime transportation, *Handbooks in operations research and management science* **14**: 189–284.

Christiansen, M., Fagerholt, K., Nygreen, B. and Ronen, D. (2013). Ship routing and scheduling in the new millennium, *European Journal of Operational Research* **228**(3): 467–483.

Clausen, J., Larsen, A., Larsen, J. and Rezanova, N. J. (2010). Disruption management in the airline industry concepts, models and methods, *Computers & Operations Research* **37**(5): 809–821.

Codato, G. and Fischetti, M. (2006). Combinatorial benders' cuts for mixed-integer linear programming, *Operations Research* **54**(4): 756–766.

Cordeau, J.-F., Stojković, G., Soumis, F. and Desrosiers, J. (2001). Benders decomposition for simultaneous aircraft routing and crew scheduling, *Transportation science* **35**(4): 375–388.

Costa, A. M. (2005). A survey on benders decomposition applied to fixed-charge network design problems, *Computers & operations research* **32**(6): 1429–1450.

Cote, G. and Laughton, M. A. (1984). Large-scale mixed integer programming: Benders-type heuristics, *European Journal of Operational Research* **16**(3): 327–333.

Crama, Y., Kolen, A. W. and Pesch, E. (1995). Local search in combinatorial optimization, *Artificial Neural Networks*, Springer, pp. 157–174.

De Leone, R., Festa, P. and Marchitto, E. (2011). A bus driver scheduling problem: a new mathematical model and a grasp approximate solution, *Journal of Heuristics* **17**(4): 441–466.

Di Gaspero, L., Gärtner, J., Kortsarz, G., Musliu, N., Schaerf, A. and Slany, W. (2007). The minimum shift design problem, *Annals of operations research* **155**(1): 79–105.

Dunbar, M., Froyland, G. and Wu, C.-L. (2012). Robust airline schedule planning: Minimizing propagated delay in an integrated routing and crewing framework, *Transportation Science* **46**(2): 204–216.

Eggenberg, N., Salani, M. and Bierlaire, M. (2010). Constraint-specific recovery network for solving airline recovery problems, *Computers & operations research* **37**(6): 1014–1026.

Ernst, A., Jiang, H., Krishnamoorthy, M. and Sier, D. (2004a). An annotated bibliography of personnel scheduling and rostering., *Annals of Operations Research* **127**(1-4): 21–144.

Ernst, A., Jiang, H., Krishnamoorthy, M. and Sier, D. (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.

Franz, L. S. and Miller, J. L. (1993). Scheduling medical residents to rotations: solving the large-scale multiperiod staff assignment problem, *Operations Research* **41**(2): 269–279.

Gabrel, V., Murat, C. and Thiele, A. (2014). Recent advances in robust optimization: An overview, *European journal of operational research* **235**(3): 471–483.

Gamache, M., Hertz, A. and Ouellet, J. O. (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., Soumis, F., Marquis, G. and Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems, *Operations research* **47**(2): 247–263.

Garey, M. R. and Johnson, D. S. (2002). *Computers and intractability*, Vol. 29, wh freeman New York.

Gaspar, H. M., Brett, P., Erikstad, S. O. and Ross, A. M. (2015). Quantifying value robustness of osv designs taking into consideration medium to long term stakeholdersâ expectations, *12th International Marine Design Conference (IMDC)*, Vol. 2, pp. 247–259.

Giachetti, R. E., Damodaran, P., Mestry, S. and Prada, C. (2013). Optimization-based decision support system for crew scheduling in the cruise industry, *Computers & Industrial Engineering* **64**(1): 500–510.

Gleeson, J. and Ryan, J. (1990). Identifying minimally infeasible subsystems of inequalities, *ORSA Journal on Computing* **2**(1): 61–63.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence, *Computers & operations research* **13**(5): 533–549.

Gopalakrishnan, B. and Johnson, E. L. (2005). Airline crew scheduling: state-of-the-art, *Annals of Operations Research* **140**(1): 305–337.

Gorissen, B. L., Yanıkoğlu, İ. and den Hertog, D. (2015). A practical guide to robust optimization, *Omega* **53**: 124–137.

Guo, Y. (2005). A decision support framework for the airline crew schedule disruption management with strategy mapping, *Operations Research Proceedings 2004*, Springer, pp. 158–165.

Halvorsen-Weare, E. and Fagerholt, K. (2011). Robust supply vessel planning, *Network optimization* pp. 559–573.

Hanafi, R. and Kozan, E. (2014). A hybrid constructive heuristic and simulated annealing for railway crew scheduling, *Computers & Industrial Engineering* **70**: 11–19.

Herroelen, W. and Leus, R. (2004). Robust and reactive project scheduling: a review and classification of procedures, *International Journal of Production Research* **42**(8): 1599–1620.

Hoffman, K. L. and Padberg, M. (1993). Solving airline crew scheduling problems by branch-and-cut, *Management Science* **39**(6): 657–682.

Horn, M. E., Jiang, H. and Kilby, P. (2007). Scheduling patrol boats and crews for the royal australian navy, *Journal of the Operational Research Society* **58**(10): 1284–1293.

Kasirzadeh, A., Saddoune, M. and Soumis, F. (2017). Airline crew scheduling: models, algorithms, and data sets, *EURO Journal on Transportation and Logistics* **6**(2): 111–137.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P. et al. (1983). Optimization by simulated annealing, *science* **220**(4598): 671–680.

Klabjan, D. (2005). Large-scale models in the airline industry, *Column generation* pp. 163–195.

Klabjan, D., Johnson, E. L., Nemhauser, G. L., Gelman, E. and Ramaswamy, S. (2001). Solving large airline crew scheduling problems: Random pairing generation and strong branching, *Computational Optimization and Applications* **20**(1): 73–91.

Koubaa, M., Elloumi, S. and Dhouib, S. (2014). Artificial bee colony to solve seafaring staff scheduling problem: A real case, *Advanced Logistics and Transport (ICALT), 2014 International Conference on*, IEEE, pp. 314–318.

Kyngäs, N., Nurmi, K. and Kyngäs, J. (2012). Optimizing large-scale staff rostering instances, *Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists, Hong Kong*.

Lan, S., Clarke, J.-P. and Barnhart, C. (2006). Planning for robust airline operations: Optimizing aircraft routings and flight departure times to minimize passenger disruptions, *Transportation science* **40**(1): 15–28.

Legato, P. and Monaco, M. F. (2004). Human resources management at a marine container terminal, *European Journal of Operational Research* **156**(3): 769–781.

Leggate, A. (2016). *A vessel crew scheduling problem: formulations and solution methods*, PhD thesis, Dept. of Management Science, University of Strathclyde. Available online via http://ethos.bl.uk/.

Leggate, A., Sucu, S., Akartunalı, K. and Van Der Meer, R. (2017). Modelling crew scheduling in off-shore supply vessels. Journal of the Operational Research Society.

Lettovskỳ, L., Johnson, E. L. and Nemhauser, G. L. (2000). Airline crew recovery, *Transportation Science* **34**(4): 337–348.

Lütjen, M., Karimi, H. R. et al. (2012). Approach of a port inventory control system for the offshore installation of wind turbines, *The Twenty-second International Offshore and Polar Engineering Conference*, International Society of Offshore and Polar Engineers.

Ma, J., Ceder, A. A., Yang, Y., Liu, T. and Guan, W. (2016). A case study of beijing bus crew scheduling: a variable neighborhood-based approach, *Journal of Advanced Transportation* **50**(4): 434–445.

Magnanti, T. L. and Wong, R. T. (1981). Accelerating benders decomposition: Algorithmic enhancement and model selection criteria, *Operations research* **29**(3): 464–484.

Mazzola, J. B. and Neebe, A. W. (1986). Resource-constrained assignment scheduling, *Operations Research* **34**(4): 560–572.

Mercier, A., Cordeau, J.-F. and Soumis, F. (2005). A computational study of benders decomposition for the integrated aircraft routing and crew scheduling problem, *Computers & Operations Research* **32**(6): 1451–1476.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search, *Computers & operations research* **24**(11): 1097–1100.

Morgado, E. M. and Martins, J. P. (1993). An ai-based approach to crew scheduling, *Artificial Intelligence for Applications, 1993. Proceedings., Ninth Conference on*, IEEE, pp. 71–77.

Muter, İ., Birbil, Ş. İ., Bülbül, K., Şahin, G., Yenigün, H., Taş, D. and Tüzün, D. (2013). Solving a robust airline crew pairing problem with column generation, *Computers & Operations Research* **40**(3): 815–830.

Nissen, R. and Haase, K. (2006). Duty-period-based network model for crew rescheduling in european airlines, *Journal of Scheduling* **9**(3): 255–278.

Optimization, D. (2007). Xpress-optimizer reference manual, *Dash Optimization Ltd., Englewood Cliffs, NJ* .

Öztop, H., Eliiyi, U., Eliiyi, D. T. and Kandiller, L. (2017). A bus crew scheduling problem with eligibility constraints and time limitations, *Transportation Research Procedia* **22**: 222–231.

Papadakos, N. (2008). Practical enhancements to the magnanti–wong method, *Operations Research Letters* **36**(4): 444–449.

Pearce, R. H. and Forbes, M. A. (2017). Puzzleâthe fillomino puzzle, *INFORMS Transactions on Education* **17**(2): 85–89.

Pentico, D. W. (2007). Assignment problems: A golden anniversary survey, *European Journal of Operational Research* **176**(2): 774–793.

Pinar, C. M. (2012). Robust optimization. https://www.ie.bilkent.edu.tr/ mustafap/courses/rt4.pdf.

Poojari, C. A. and Beasley, J. E. (2009). Improving benders decomposition using a genetic algorithm, *European Journal of Operational Research* **199**(1): 89–97.

Qi, X. (2015). Disruption management for liner shipping, *Handbook of Ocean Container Transport Logistics*, Springer, pp. 231–249.

Rahmaniani, R., Crainic, T. G., Gendreau, M. and Rei, W. (2017). The benders decomposition algorithm: A literature review, *European Journal of Operational Research* **259**(3): 801–817.

Raidl, G. R., Baumhauer, T. and Hu, B. (2015). Boosting an exact logic-based benders decomposition approach by variable neighborhood search, *Electronic Notes in Discrete Mathematics* **47**: 149–156.

Rezanova, N. J. and Ryan, D. M. (2010). The train driver recovery problem a set partitioning based model and solution method, *Computers & Operations Research* **37**(5): 845–856.

Saharidis, G. K. and Ierapetritou, M. G. (2010). Improving benders decomposition using maximum feasible subsystem (mfs) cut generation strategy, *Computers & chemical engineering* **34**(8): 1237–1245.

Schaefer, A. J., Johnson, E. L., Kleywegt, A. J. and Nemhauser, G. L. (2005). Airline crew scheduling under uncertainty, *Transportation science* **39**(3): 340–348.

Scholz-Reiter, B., Heger, J., Lütjen, M. and Schweizer, A. (2010). A milp for installation scheduling of offshore wind farms, *International Journal Of Mathematical Models And Methods In Applied Sciences* **5**(2): 371–378.

Shebalov, S. and Klabjan, D. (2006). Robust airline crew pairing: Move-up crews, *Transportation science* **40**(3): 300–312.

Soyster, A. L. (1973). Convex programming with set-inclusive constraints and applications to inexact linear programming, *Operations research* **21**(5): 1154–1157.

Stevens, S. C. and Parsons, M. G. (2002). Effects of motion at sea on crew performance: A survey, *Marine Technology* **39**(1): 29–47.

Suraweera, P., Webb, G. I., Evans, I. and Wallace, M. (2013). Learning crew scheduling constraints from historical schedules, *Transportation research part C: emerging technologies* **26**: 214–232.

Tam, B., Ryan, D. and Ehrgott, M. (2014). Multi-objective approaches to the unit crewing problem in airline crew scheduling, *Journal of Multi-Criteria Decision Analysis* **21**(5-6): 257–277.

Taşkın, Z. C., Smith, J. C. and Romeijn, H. E. (2012). Mixed-integer programming techniques for decomposing imrt fluence maps using rectangular apertures, *Annals of Operations Research* **196**(1): 799–818.

Thorsteinsson, E. (2001). Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming, *Principles and Practice of Constraint ProgrammingâCP 2001*, Springer, pp. 16–30.

Vaidyanathan, B., Jha, K. C. and Ahuja, R. K. (2007). Multicommodity network flow approach to the railroad crew-scheduling problem, *IBM Journal of Research and Development* **51**(3.4): 325–344.

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E. and De Boeck, L. (2013). Personnel scheduling: A literature review, *European Journal of Operational Research* **226**(3): 367–385.

VCS-data (2015). Vessel crew scheduling for off-shore supply vessels: A library of test instances. http://dx.doi.org/10.15129/5dddf224-deaa-4b4b-aeb2-ef60c0d910c9.

Wei, G., Yu, G. and Song, M. (1997). Optimization model and algorithm for crew management during airline irregular operations, *Journal of Combinatorial Optimization* **1**(3): 305–321.

Wermus, M. and Pope, J. A. (1994). Scheduling harbor pilots, *Interfaces* **24**(2): 44–52.

Yang, Y. and Lee, J. M. (2012). A tighter cut generation strategy for acceleration of benders decomposition, *Computers & Chemical Engineering* **44**: 84–93.

Yen, J. W. and Birge, J. R. (2006). A stochastic programming approach to the airline crew scheduling problem, *Transportation Science* **40**(1): 3–14.

Yu, G., Argüello, M., Song, G., McCowan, S. M. and White, A. (2003). A new era for crew recovery at continental airlines, *Interfaces* **33**(1): 5–22.

Zeghal, F. and Minoux, M. (2006). Modeling and solving a crew assignment problem in air transportation, *European Journal of Operational Research* **175**(1): 187–209.

Zhang, D., Lau, H. H. and Yu, C. (2015). A two stage heuristic algorithm for the integrated aircraft and crew schedule recovery problems, *Computers & Industrial Engineering* **87**: 436–453.

Appendix A

Heuristic Analysis

Categorical regression is applied on *Percentage GAP* obtained after to see the effect of data characteristics and heuristic application settings.

Model Summary

	Multiple R	R Square	Adjusted R Square	Apparent Prediction Error
Standardized Data	.755	.570	.569	.430

Dependent Variable: Gap

Predictors: Kick Order Initial Employee Ag_Pen Use Near P Long Accept

ANOVA

	Sum of Squares	df	Mean Square	F	Sig.
Regression	6564.362	17	386.139	896.226	.000
Residual	4955.638	11502	.431		
Total	11520.000	11519			

Dependent Variable: Gap

Predictors: Kick Order Initial Employee Ag_Pen Use Near P Long Accept

Coefficients

	Standardized Coefficients		df	F	Sig.
	Beta	Bootstrap (1000) Estimate of Std. Error			
Kick	-.029	.006	2	21.443	.000
Order	.006	.005	1	1.608	.205
Initial	.130	.006	1	450.878	.000
Employee	-.020	.006	1	9.980	.002
Ag_Pen	.594	.007	3	7139.870	.000
Use	.092	.006	1	229.299	.000
Near	-.045	.006	2	57.330	.000
P[a]	.423	.	2	.	.
Long	-.082	.007	3	158.350	.000
Accept	.002	.004	1	.365	.545

Dependent Variable: Gap

a. The tolerance for this variable is lower than 0.0001.

Figure A.1: SPSS Output for Categorical Regression on GAP

Version	Number of Employee	Ordering Rule	Acceptance Rule	Kick Setting	Initial Solution
TBA Init-V1	one third	smarter	current	no kick	Task-Based
TBA Init-V2	one third	random	current	no kick	Task-Based
TBA Init-V3	one third	smarter	current	4+	Task-Based
TBA Init-V4	one third	random	current	4+	Task-Based
TBA Init-V5	one third	smarter	current	8+	Task-Based
TBA Init-V6	one third	random	current	8+	Task-Based
TBA Init-V7	one third	smarter	best	no kick	Task-Based
TBA Init-V8	one third	random	best	no kick	Task-Based
TBA Init-V9	one third	smarter	best	4+	Task-Based
TBA Init-V10	one third	random	best	4+	Task-Based
TBA Init-V11	one third	smarter	best	8+	Task-Based
TBA Init-V12	one third	random	best	8+	Task-Based
TBA Init-V13	all	smarter	current	no kick	Task-Based
TBA Init-V14	all	random	current	no kick	Task-Based
TBA Init-V15	all	smarter	current	4+	Task-Based
TBA Init-V16	all	random	current	4+	Task-Based
TBA Init-V17	all	smarter	current	8+	Task-Based
TBA Init-V18	all	random	current	8+	Task-Based
TBA Init-V19	all	smarter	best	no kick	Task-Based
TBA Init-V20	all	random	best	no kick	Task-Based
TBA Init-V21	all	smarter	best	4+	Task-Based
TBA Init-V22	all	random	best	4+	Task-Based
TBA Init-V23	all	smarter	best	8+	Task-Based
TBA Init-V24	all	random	best	8+	Task-Based
Heur Init- V1	one third	smarter	current	no kick	Heuristic
Heur Init- V2	one third	random	current	no kick	Heuristic
Heur Init- V3	one third	smarter	current	4+	Heuristic
Heur Init- V4	one third	random	current	4+	Heuristic
Heur Init- V5	one third	smarter	current	8+	Heuristic
Heur Init- V6	one third	random	current	8+	Heuristic
Heur Init- V7	one third	smarter	best	no kick	Heuristic
Heur Init- V8	one third	random	best	no kick	Heuristic
Heur Init- V9	one third	smarter	best	4+	Heuristic
Heur Init- V10	one third	random	best	4+	Heuristic
Heur Init- V11	one third	smarter	best	8+	Heuristic
Heur Init- V12	one third	random	best	8+	Heuristic
Heur Init- V13	all	smarter	current	no kick	Heuristic
Heur Init- V14	all	random	current	no kick	Heuristic
Heur Init- V15	all	smarter	current	4+	Heuristic
Heur Init- V16	all	random	current	4+	Heuristic
Heur Init- V17	all	smarter	current	8+	Heuristic
Heur Init- V18	all	random	current	8+	Heuristic
Heur Init- V19	all	smarter	best	no kick	Heuristic
Heur Init- V20	all	random	best	no kick	Heuristic
Heur Init- V21	all	smarter	best	4+	Heuristic
Heur Init- V22	all	random	best	4+	Heuristic
Heur Init- V23	all	smarter	best	8+	Heuristic
Heur Init- V24	all	random	best	8+	Heuristic

Figure A.2: Explanation of Heuristic Versions

Appendix B

Code

B.1 Benders Decomposition Algorithm

The Benders Decomposition Algorithms are coded in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) and presented under this section.

B.1.1 First Trial Benders Decomposition

The implementation of the first version of BD is provided below.

```
model "Benders (master model)"
uses "mmxprs", "mmjobs", "mmsystem"

parameters
ALG = 1
DATAFILE = "small_data.txt"
end-parameters

forward procedure start_solution
forward procedure solve_primal_int(ct: integer)
forward procedure solve_cont
forward procedure solve_mwp
forward function eval_solution: boolean
forward procedure print_solution

declarations
```

```
STEP_0=2                          ! Event codes sent to submodels
STEP_1=3
STEP_2=4
STEP_3=5
EVENT_SOLVED=6                    ! Event codes sent by submodels
EVENT_INFEAS=7
EVENT_READY=8

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

sol_obj: real                     ! Objective function value (primal)
MC: array(range) of linctr        ! Constraints generated by alg.
RM: range                         ! Model indices
cut_type: real
stepmod: array(RM) of Model       ! Submodels
end-declarations
!DECLARATION OF PARAMETERS AND DECISION VARIABLES
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
allocate: dynamic array(ALL_EMP, ALL_ROLES, TIME) of real ! Variable for
   allocating employee to role during given time period
work_total: array(REG_EMP, TIME) of real
allocate_dual: dynamic array(ALL_EMP, ALL_ROLES, TIME) of real ! Variable
   for allocating employee to role during given time period
work_total_dual: array(REG_EMP, TIME) of real


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

!Reading from txt file
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



!Continue to define parameters and decision variables
```

```
declarations
lambda = 1..overall_max_work
                                ! Index used for number of consecutive
   weeks
long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real ! Used to
   indicate if a special bonus / penalty payment relating to consecutive
   time at sea is required
long_work_dual: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real ! 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



!solution of dual sub problem
sol_dual_3, sol_dual_5: array(REG_EMP, VESSELS) of real
sol_dual_4, sol_dual_6: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of real

                                      ! or takes a non-negative integer
   value for agency crew
sol_dual_7: array(ALL_ROLES) of real
sol_dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_9: array(GUARANTEED) of real
sol_dual_10: array(GUARANTEED) of real
sol_dual_14: array(lambda, REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_15: array(ALL_ROLES) of real
sol_dual_16: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of real                 !
   Used to track the consecutive working time of the agency employees
sol_dual_17: array(ALL_ROLES, TIME) of real
sol_dual_18: array(lambda, ALL_ROLES, TIME) of real
sol_dual_19:array(REG_EMP) of real
```

```
sol_dual_21:array(REG_EMP, TIME) of real
sol_dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of real
sol_dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_24:array(REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_25: array(lambda,REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of real
! Objective Value of dual problem
Dual_cost:real
UB:real
end-declarations

! reading the parameters
initializations from DATAFILE
extension_chng_cost cur_long_work long_work allocate work_total
end-initializations
declarations
!constraints for dual problem
dual_cons_board: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart: array(REG_EMP, VESSELS, TIME) of linctr        ! =1 if
   employee boards / departs vessel in given time period, 0 otherwise
! or takes a non-negative integer value for agency crew
dual_cons_ag_rboard: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart: array(ALL_ROLES, TIME) of linctr    ! =1 if agency
   crew starts / ends working on a role in given time period, 0 otherwise
dual_cons_undertime: array(GUARANTEED) of linctr
dual_cons_overtime: array(GUARANTEED) of linctr                   !
   Variables to calculate the amount of under/overtime carried out by
   employee
dual_cons_rest_total: array(REG_EMP, TIME) of linctr ! Used to track the
   consecutive working time / rest period requirements of each employee
dual_cons_work_total: array(REG_EMP, TIME) of linctr ! Used to track the
   consecutive working time / rest period requirements of each employee
dual_cons_ag_work_total: array(ALL_ROLES, TIME) of linctr
          ! Used to track the consecutive working time of the agency
   employees

!constraints for integer problem
All_covered: dynamic array(ALL_ROLES, TIME) of linctr
```

```
No_overlap: array(REG_EMP, TIME) of linctr
Long_work_count: dynamic array(lambda, REG_EMP, ALL_ROLES, TIME) of linctr
Rest_vs_work: array(REG_EMP,1..1) of linctr
Work_count: array(REG_EMP, TIME) of linctr


end-declarations
!sharing data info with submodels
initializations to "bin:shmem:probdata" ! Save data for submodels
REG_EMP GUARANTEED VESSELS WEEKS_TO_PLAN ROLES
under_rate over_rate g_weeks exp_worktime
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
extension_chng_cost cur_long_work long_work allocate work_total
end-initializations

! **** Submodels ****


! Compile + load all submodels
if compile("benders_int_recovery3.mos")<>0 then exit(1); end-if
create(stepmod(1)); load(stepmod(1), "benders_int_recovery3.bim")
if compile("benders_mwp_recovery3.mos")<>0 then exit(3); end-if
create(stepmod(0)); load(stepmod(0), "benders_mwp_recovery3.bim")
if compile("benders_dual_recovery3.mos")<>0 then exit(2); end-if
if ALG=1 then
create(stepmod(2)); load(stepmod(2), "benders_dual_recovery3.bim")
else
create(stepmod(0)); load(stepmod(0), "benders_dual_recovery3.bim")
if compile("benders_cont_recovery3.mos")<>0 then exit(3); end-if
create(stepmod(2)); load(stepmod(2), "benders_cont_recovery3.bim")
run(stepmod(0))
```

```
end-if
! Start the execution of the submodels


run(stepmod(1))
run(stepmod(2))
run(stepmod(0))

forall(m in RM) do
wait                              ! Wait for "Ready" messages
ev:= getnextevent
writeln("m: ", m)
writeln("RM: ",RM)
writeln("ev: ", getclass(ev))
if getclass(ev) <> EVENT_READY then
writeln("Error occurred in a subproblem")
exit(4)
end-if
end-do

! **** Solution algorithm ****

(! start_solution                    ! 0. Initial solution for getting dual
    variables
ct:= 1
repeat
writeln("\n**** Iteration: ", ct)
solve_primal_int(ct)             ! 1. Solve master problem with dual
   variables values and.
solve_cont
if(cut_type=0)then
solve_mwp                        ! 2. Solve problem with fixed int.
end-if
ct+=1
until eval_solution               !   Test for optimality
print_solution                    ! 3. Retrieve and display the solution
!)
```

```
prog_starttime_big:= gettime
start_solution                    ! 0. Initial solution for getting dual
    variables
ct:= 1
repeat
writeln("\n**** Iteration: ", ct)
prog_starttime := gettime
solve_primal_int(ct)             ! 1. Solve master problem with dual
    variables values and.
prog_endtime := gettime
writeln("Iteration: ", ct)
writeln("Running time:\t",prog_endtime-prog_starttime)
solve_cont
!if(cut_type=0)then
!solve_mwp
! end-if                     ! 2. Solve problem with fixed int.
ct+=1
until eval_solution!  Test for optimality
prog_endtime_big:= gettime
print_solution                    ! 3. Retrieve and display the solution
writeln("Running time:\t",prog_endtime_big-prog_starttime_big)

! **** Cleaning up temporary files ****
fdelete("benders_int_recovery3.bim")
fdelete("benders_dual_recovery3.bim")
fdelete("benders_mwp_recovery3.bim")
if ALG<>1 then fdelete("benders_cont_recovery3.bim"); end-if
fdelete("shmem:probdata")
fdelete("shmem:sol")

!-------------------------------------------------------------
! Produce an initial solution for the dual problem
procedure start_solution
if ALG=1 then                     ! Start the problem solving
send(stepmod(2), STEP_0, 0)
else
```

```
send(stepmod(0), STEP_0, 0)
end-if
wait                            ! Wait for the solution
ev:=getnextevent
if getclass(ev)=EVENT_INFEAS then
writeln("Problem is infeasible")
exit(6)
end-if
end-procedure

!-------------------------------------------------------------
! Solve a modified version of the primal problem, replacing continuous
! variables by the solution of the dual
procedure solve_primal_int(ct: integer)
send(stepmod(1), STEP_1, ct)    ! Start the problem solving
wait                            ! Wait for the solution
ev:=getnextevent
sol_obj:= getvalue(ev)          ! Store objective function value

initializations from "bin:shmem:sol" ! Retrieve the solution
allocate_dual long_work_dual
end-initializations
end-procedure

!-------------------------------------------------------------
! Solve the Step 2 problem (dual)
! for given solution values of allocate, work_total, long_work
procedure solve_cont
send(stepmod(2), STEP_2, 0)     ! Start the problem solving
wait                            ! Wait for the solution
dropnextevent

initializations from "bin:shmem:sol" ! Retrieve the solution
sol_dual_3 sol_dual_5 sol_dual_4 sol_dual_6 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_15
sol_dual_16 sol_dual_17 sol_dual_18 sol_dual_19 sol_dual_21
```

```
sol_dual_20   sol_dual_23 sol_dual_24 sol_dual_25 Dual_cost cut_type
   allocate_dual long_work_dual UB
end-initializations
end-procedure

!-------------------------------------------------------------
! Solve the Step 3 problem (dual)
! for given solution values of allocate, work_total, long_work
procedure solve_mwp
send(stepmod(0), STEP_3, 0)     ! Start the problem solving
wait                            ! Wait for the solution
dropnextevent

initializations from "bin:shmem:sol" ! Retrieve the solution
sol_dual_3 sol_dual_5 sol_dual_4 sol_dual_6 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_15
sol_dual_16 sol_dual_17 sol_dual_18 sol_dual_19 sol_dual_21
sol_dual_20   sol_dual_23 sol_dual_24 sol_dual_25 Dual_cost cut_type
end-initializations
end-procedure

!-------------------------------------------------------------

!The obj function value from dual and master problem are compared here

function eval_solution: boolean

write("Test optimality: ",UB - sol_obj , " = ",0.5)
returned:= UB- sol_obj<=0.5
! write("Test optimality: ",sol_obj - sum(e in ALL_EMP, r in ALL_ROLES, t
   in TIME)((work_cost(e,r,t)*allocate_dual(e,r,t)) + sum(l in lambda)(
   extension_cost(l,e,r,t)*long_work_dual(l,e,r,t))), " = ", (sum(e in
   REG_EMP, v in VESSELS)(sol_dual_3(e,v)*(sum(r in ROLES(v))(
   allocate_dual(e,r,1)) - starting(e,v)))+
!sum(r in ALL_ROLES)(sol_dual_7(r)*( allocate_dual("AGENCY",r,1) -
   ag_starting(r)))+
```

```
!sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_8(r,t)*(allocate_dual
   ("AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
!sum(e in GUARANTEED)(sol_dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
!sum(e in GUARANTEED)(sol_dual_10(e)*((exp_worktime(e) + sum(r in
   ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
!sum(r in ALL_ROLES)(sol_dual_15(r)*( ag_work_zero(r) + allocate_dual("
   AGENCY",r,1)))+
!sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) (sol_dual_16(r,t)*
   allocate_dual("AGENCY",r,t))+
!sum(r in ALL_ROLES, t in TIME)(sol_dual_17(r,t)*allocate_dual("AGENCY",r,
   t))+
!sum(l in lambda, r in ALL_ROLES, t in TIME)(sol_dual_18(l,r,t)*(-
   ag_max_work(r)*long_work_dual(l,"AGENCY",r,t) - (l-1)))+
!sum(e in REG_EMP)(sol_dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)
   (allocate_dual(e,r,1))))))+
!sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
!sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_23(e,t)*(- min_rest(e)
   *(1-(sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
!sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_4(e,v,t)
   *(sum(r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(
   allocate_dual(e,r,(t-1)))))+
!sum(e in REG_EMP, v in VESSELS)(sol_dual_5(e,v)*(starting(e,v) - sum(r in
    ROLES(v))(allocate_dual(e,r,1))))+
!sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_6(e,v,t)
   *( sum(r in ROLES(v))(allocate_dual(e,r,(t-1))) - sum(r in ROLES(v))(
   allocate_dual(e,r,t))))))

! returned:= ((sol_obj - sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((
   work_cost(e,r,t)*allocate_dual(e,r,t)) + sum(l in lambda)((
   extension_cost(l,e,r,t)*long_work_dual(l,e,r,t)))))= (sum(e in REG_EMP,
    v in VESSELS)(sol_dual_3(e,v)*(sum(r in ROLES(v))(allocate_dual(e,r,1)
   ) - starting(e,v)))+
!sum(r in ALL_ROLES)(sol_dual_7(r)*( allocate_dual("AGENCY",r,1) -
   ag_starting(r)))+
```

```
!sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_8(r,t)*(allocate_dual
   ("AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
!sum(e in GUARANTEED)(sol_dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
!sum(e in GUARANTEED)(sol_dual_10(e)*((exp_worktime(e) + sum(r in
   ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
!sum(r in ALL_ROLES)(sol_dual_15(r)*( ag_work_zero(r) + allocate_dual("
   AGENCY",r,1)))+
!sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) (sol_dual_16(r,t)*
   allocate_dual("AGENCY",r,t))+
!sum(r in ALL_ROLES, t in TIME)(sol_dual_17(r,t)*allocate_dual("AGENCY",r,
   t))+
!sum(l in lambda, r in ALL_ROLES, t in TIME)(sol_dual_18(l,r,t)*(-
   ag_max_work(r)*long_work_dual(l,"AGENCY",r,t) - (l-1)))+
!sum(e in REG_EMP)(sol_dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)
   (allocate_dual(e,r,1))))))+
!sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
!sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_23(e,t)*(- min_rest(e)
   *(1-(sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
!sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_4(e,v,t)
   *(sum(r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(
   allocate_dual(e,r,(t-1)))))+
!sum(e in REG_EMP, v in VESSELS)(sol_dual_5(e,v)*(starting(e,v) - sum(r in
    ROLES(v))(allocate_dual(e,r,1))))+
!sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_6(e,v,t)
   *( sum(r in ROLES(v))(allocate_dual(e,r,(t-1))) - sum(r in ROLES(v))(
   allocate_dual(e,r,t))))))
writeln(if(returned, " : true", " : false"))
end-function

!-------------------------------------------------------------
procedure print_solution
! Retrieve results
initializations from "bin:shmem:sol"
allocate_dual work_total_dual long_work_dual
end-initializations
```

```
forall(m in RM) stop(stepmod(m)) ! Stop all submodels

write("\n**** Solution (Benders): ", sol_obj)
writeln("long_work: [")
forall(l in lambda) do
forall(e in ALL_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) write((long_work_dual(l,e,r,t)),"\t")

end-do
write("\n")
end-do
write("\n")
end-do
write("]\n")

writeln("allocate: [")

forall(e in ALL_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) write((allocate_dual(e,r,t)),"\t")
end-do
write("\n")
end-do
write("]\n")

end-procedure
end-model
!-----------------------------DSP---------------------------------
model "Benders (dual problem)"
uses "mmxprs", "mmjobs"

parameters
BIGM = 100000
end-parameters
```

```
forward procedure save_solution

declarations
STEP_0=2                          ! Event codes sent to submodels
STEP_2=4
STEP_3=5
EVENT_SOLVED=6                    ! Event codes sent by submodels
EVENT_INFEAS=7
EVENT_READY=8

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 "bin:shmem:probdata"
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
allocate_master: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
work_total_master: array(REG_EMP, TIME) of mpvar
```

```
allocate: dynamic array(ALL_EMP, ALL_ROLES, TIME) of real
work_total: array(REG_EMP, TIME) of real
allocate_dual: dynamic array(ALL_EMP, ALL_ROLES, TIME) of real
work_total_dual: array(REG_EMP, TIME) of real
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
cut_type: real
end-declarations

initializations from "bin:shmem:probdata" ! Save data for submodels
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



declarations
lambda = 1..overall_max_work
long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real
long_work_master: array(lambda, ALL_EMP, ALL_ROLES, TIME) of mpvar
long_work_dual: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real

!parameters
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


! Discrete variables

dual_3, dual_5: array(REG_EMP, VESSELS) of mpvar
dual_4, dual_6: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar

                                        ! or takes a non-negative integer
   value for agency crew
dual_7: array(ALL_ROLES) of mpvar
dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9: array(GUARANTEED) of mpvar
dual_10: array(GUARANTEED) of mpvar
dual_14: array(lambda, REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_15: array(ALL_ROLES) of mpvar
dual_16: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar                    !
   Used to track the consecutive working time of the agency employees
dual_17: array(ALL_ROLES, TIME) of mpvar
dual_18: array(lambda, ALL_ROLES, TIME) of mpvar
dual_19:array(REG_EMP) of mpvar
dual_21:array(REG_EMP, TIME) of mpvar
dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of mpvar
dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of mpvar
dual_24:array(REG_EMP,2..WEEKS_TO_PLAN) of mpvar
dual_25: array(lambda,REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar

sol_dual_3, sol_dual_5: array(REG_EMP, VESSELS) of real
sol_dual_4, sol_dual_6: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of real

                                        ! or takes a non-negative integer
   value for agency crew
sol_dual_7: array(ALL_ROLES) of real
sol_dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_9: array(GUARANTEED) of real
```

```
sol_dual_10: array(GUARANTEED) of real
sol_dual_14: array(lambda, REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_15: array(ALL_ROLES) of real
sol_dual_16: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of real                 !
   Used to track the consecutive working time of the agency employees
sol_dual_17: array(ALL_ROLES, TIME) of real
sol_dual_18: array(lambda, ALL_ROLES, TIME) of real
sol_dual_19:array(REG_EMP) of real
sol_dual_21:array(REG_EMP, TIME) of real
sol_dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of real
sol_dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_24:array(REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_25: array(lambda,REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of real

UB:real
Dual_cost:real

end-declarations

initializations from "bin:shmem:probdata"
extension_chng_cost cur_long_work long_work allocate work_total
end-initializations


declarations

dual_cons_board: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart: array(REG_EMP, VESSELS, TIME) of linctr        ! =1 if
   employee boards / departs vessel in given time period, 0 otherwise
! or takes a non-negative integer value for agency crew
dual_cons_ag_rboard: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart: array(ALL_ROLES, TIME) of linctr    ! =1 if agency
   crew starts / ends working on a role in given time period, 0 otherwise
dual_cons_undertime: array(GUARANTEED) of linctr
dual_cons_overtime: array(GUARANTEED) of linctr                  !
   Variables to calculate the amount of under/overtime carried out by
   employee
```

```
dual_cons_rest_total: array(REG_EMP, TIME) of linctr ! Used to track the
   consecutive working time / rest period requirements of each employee
dual_cons_work_total: array(REG_EMP, TIME) of linctr ! Used to track the
   consecutive working time / rest period requirements of each employee
dual_cons_ag_work_total: array(ALL_ROLES, TIME) of linctr
          ! Used to track the consecutive working time of the agency
   employees


Dual_first:linctr

end-declarations

Dual_first:=(sum(e in REG_EMP, v in VESSELS)(dual_3(e,v)*(sum(r in ROLES(v
   ))(allocate(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4(e,v,t)*(sum(
   r in ROLES(v))(allocate(e,r,t)) - sum(r in ROLES(v))(allocate(e,r,(t-1)
   ))))+
sum(r in ALL_ROLES)(dual_7(r)*( allocate("AGENCY",r,1) - ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8(r,t)*(allocate("AGENCY",
   r,t) - allocate("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r in
   ALL_ROLES, t in TIME)(allocate(e,r,t)))))+
sum(e in GUARANTEED)(dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES, t
   in TIME)(allocate(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES)(dual_15(r)*( ag_work_zero(r) + allocate("AGENCY",r,1))
   )+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) (dual_16(r,t)*allocate("AGENCY
   ",r,t))+
sum(r in ALL_ROLES, t in TIME)(dual_17(r,t)*allocate("AGENCY",r,t))+
sum(l in lambda, r in ALL_ROLES, t in TIME)(dual_18(l,r,t)*(- ag_max_work(
   r)*long_work(l,"AGENCY",r,t) - (l-1)))+
sum(e in REG_EMP)(dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
   allocate(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20(e,t)*(- (1-(sum(r in
   ALL_ROLES)(allocate(e,r,t))))))+
```

```
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23(e,t)*(- min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, v in VESSELS)(dual_5(e,v)*(starting(e,v) - sum(r in
    ROLES(v))(allocate(e,r,1))))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_6(e,v,t)*( sum
    (r in ROLES(v))(allocate(e,r,(t-1))) - sum(r in ROLES(v))(allocate(e,r,
    t))))+
sum(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(
    dual_25(l,e,r,t)* ( (max_work(e)*long_work(l,e,r,t)) + (max_work(e)*(1-
    allocate(e,r,t))) - allocate(e,r,t) + (l-1)))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_24(e,t)*(sum(r in ALL_ROLES)
    (allocate(e,r,t)) - max_work(e)*(1-(sum(r in ALL_ROLES)(allocate(e,r,t)
    ))))))

!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board(e,v,1):=dual_3(e,v)<=(
    board_chng_cost(e,v,1))/(1-(2*cur_board(e,v,1)))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_cons_board(
    e,v,t):=dual_4(e,v,t)<=(board_chng_cost(e,v,t))/(1-(2*cur_board(e,v,t))
    )

!depart
forall(e in REG_EMP, v in VESSELS) dual_cons_depart(e,v,1):=dual_5(e,v)-
    ((min_rest(e)-1)*dual_21(e,1))<=(depart_chng_cost(e,v,1))/(1-(2*
    cur_depart(e,v,1)))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_cons_depart
    (e,v,t):=dual_6(e,v,t)-(min_rest(e)-1)*dual_21(e,t)<=(depart_chng_cost(
    e,v,t))/(1-(2*cur_depart(e,v,t)))

!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard(r,1):= dual_7(r)<=(
    ag_board_chng_cost(r,1)/(1-(2*cur_ag_rboard(r,1))))
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard(r,t) :=
    dual_8(r,t)<=(ag_board_chng_cost(r,t)/(1-(2*cur_ag_rboard(r,t))))

!ag_rdepart
```

```
forall(r in ALL_ROLES) dual_cons_ag_rdepart(r,1):= -dual_7(r)+(ag_max_work
    (r)*dual_15(r))<=(ag_depart_chng_cost(r,1)/(1-(2*cur_ag_rdepart(r,1))))

forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart(r,t) :=
    -dual_8(r,t)+(ag_max_work(r)*dual_16(r,t))<=(ag_depart_chng_cost(r,t)
    /(1-(2*cur_ag_rdepart(r,t))))



!undertime
forall(e in GUARANTEED) dual_cons_undertime(e):= dual_9(e)<=under_rate(e)

!overtime
forall(e in GUARANTEED) dual_cons_overtime(e):= dual_10(e)<=over_rate(e)


!ag_work_total

forall(r in ALL_ROLES) dual_cons_ag_work_total(r,1):= dual_15(r)-dual_16(r
    ,2)+dual_17(r,1)-sum(l in lambda)dual_18(l,r,1)<=0
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN-1) dual_cons_ag_work_total(r,t
    ):= dual_16(r,t)-dual_16(r,t+1)+dual_17(r,t)-sum(l in lambda)dual_18(l,
    r,t)<=0
forall(r in ALL_ROLES,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_ag_work_total(r,t):= dual_16(r,t)+dual_17(r,t)- sum(l in
    lambda)dual_18(l,r,t)<=0


!rest_total

forall(e in REG_EMP) dual_cons_rest_total(e,1):= dual_19(e)-dual_20(e,2)+
    dual_21(e,1)-dual_23(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total(e,t):=
    dual_20(e,t)-dual_20(e,t+1)+dual_21(e,t)-dual_23(e,t+1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_rest_total(e,t):= dual_20(e,t)+dual_21(e,t)<=0
```

```
!work_total

forall(e in REG_EMP) dual_cons_work_total(e,1):= - dual_24(e,2)+ sum(l in
   lambda,r in ALL_ROLES)dual_25(l,e,r,2)<=0
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN-1) dual_cons_work_total(e,t):=
   dual_24(e,t) - dual_24(e,(t+1)) + sum(l in lambda,r in ALL_ROLES)
   dual_25(l,e,r,t+1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
   dual_cons_work_total(e,t):= dual_24(e,t)<=0




forall(e in REG_EMP, v in VESSELS) dual_3(e,v)>=0
forall(e in REG_EMP, v in VESSELS) dual_5(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4(e,v,t)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_6(e,v,t)>=0
forall(r in ALL_ROLES) dual_7(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8(r,t) is_free
forall(e in GUARANTEED) dual_9(e)>=0
forall(e in GUARANTEED) dual_10(e)>=0
forall(r in ALL_ROLES) dual_15(r)>=0
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_16(r,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_17(r,t)>=0
forall(l in lambda, r in ALL_ROLES,t in TIME) dual_18(l,r,t)>=0
forall(e in REG_EMP) dual_19(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23(e,t)>=0
forall(l in lambda,e in REG_EMP, r in ALL_ROLES,t in 2..WEEKS_TO_PLAN)
   dual_25(l,e,r,t)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_24(e,t)>=0

send(EVENT_READY,0)              ! Model is ready (= running)
```

```
! (Re)solve this model until it is stopped by event "STEP_3"
repeat
wait
ev:= getnextevent
Alg:= getclass(ev)

if Alg=STEP_0 then              ! Produce an initial solution for the
! dual problem using a dummy objective
maximize(XPRS_BAR,Dual_first)

if(getprobstat = XPRS_INF) then
writeln("Problem is infeasible")
send(EVENT_INFEAS,0)          ! Problem is infeasible
else
write("**** Start solution: ")
write("**** Problem is not infeasible: ")
if(getprobstat = XPRS_UNB) then
writeln("Problem is unbounded")
cut_type:=0
writeln("Cut_type",cut_type)
BigM:= (sum(e in REG_EMP, v in VESSELS)(dual_3(e,v)*(sum(r in ROLES(v))(
    allocate(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4(e,v,t)*(sum(
    r in ROLES(v))(allocate(e,r,t)) - sum(r in ROLES(v))(allocate(e,r,(t-1)
    ))))+
sum(r in ALL_ROLES)(dual_7(r)*( allocate("AGENCY",r,1) - ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8(r,t)*(allocate("AGENCY",
    r,t) - allocate("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate(e,r,t)))))+
sum(e in GUARANTEED)(dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES, t
    in TIME)(allocate(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES)(dual_15(r)*( ag_work_zero(r) + allocate("AGENCY",r,1))
    )+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) (dual_16(r,t)*allocate("AGENCY
    ",r,t))+
sum(r in ALL_ROLES, t in TIME)(dual_17(r,t)*allocate("AGENCY",r,t))+
```

```
sum(l in lambda, r in ALL_ROLES, t in TIME)(dual_18(l,r,t)*(- ag_max_work(
    r)*long_work(l,"AGENCY",r,t) - (l-1)))+
sum(e in REG_EMP)(dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23(e,t)*(- min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, v in VESSELS)(dual_5(e,v)*(starting(e,v) - sum(r in
    ROLES(v))(allocate(e,r,1))))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_6(e,v,t)*( sum
    (r in ROLES(v))(allocate(e,r,(t-1))) - sum(r in ROLES(v))(allocate(e,r,
    t))))+
sum(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(
    dual_25(l,e,r,t)* ( (max_work(e)*long_work(l,e,r,t)) + (max_work(e)*(1-
    allocate(e,r,t))) - allocate(e,r,t) + (l-1)))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_24(e,t)*(sum(r in ALL_ROLES)
    (allocate(e,r,t)) - max_work(e)*(1-(sum(r in ALL_ROLES)(allocate(e,r,t)
    ))))))<=BIGM

maximize(XPRS_BAR,Dual_first)

else
writeln("Problem is feasible")
cut_type:=1
writeln("Cut_type:",cut_type)
end-if
save_solution
BigM:= 0
end-if
else                              ! STEP 2: Solve the dual problem for
! given solution values of y

initializations from "bin:shmem:sol"
allocate_dual long_work_dual
end-initializations
```

```
Obj:= (sum(e in REG_EMP, v in VESSELS)(dual_3(e,v)*(sum(r in ROLES(v))(
    allocate_dual(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4(e,v,t)*(sum(
    r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(allocate_dual
    (e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7(r)*( allocate_dual("AGENCY",r,1) - ag_starting(
    r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8(r,t)*(allocate_dual("
    AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
sum(e in GUARANTEED)(dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES, t
    in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES)(dual_15(r)*( ag_work_zero(r) + allocate_dual("AGENCY",
    r,1)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) (dual_16(r,t)*allocate_dual("
    AGENCY",r,t))+
sum(r in ALL_ROLES, t in TIME)(dual_17(r,t)*allocate_dual("AGENCY",r,t))+
sum(l in lambda, r in ALL_ROLES, t in TIME)(dual_18(l,r,t)*(- ag_max_work(
    r)*long_work_dual(l,"AGENCY",r,t) - (l-1)))+
sum(e in REG_EMP)(dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23(e,t)*(- min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, v in VESSELS)(dual_5(e,v)*(starting(e,v) - sum(r in
    ROLES(v))(allocate_dual(e,r,1))))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_6(e,v,t)*( sum
    (r in ROLES(v))(allocate_dual(e,r,(t-1))) - sum(r in ROLES(v))(
    allocate_dual(e,r,t))))+
sum(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(
    dual_25(l,e,r,t)* ( (max_work(e)*long_work_dual(l,e,r,t)) + (max_work(e
    )*(1-allocate_dual(e,r,t))) - allocate_dual(e,r,t) + (l-1)))+
```

```
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_24(e,t)*(sum(r in ALL_ROLES)
    (allocate_dual(e,r,t)) - max_work(e)*(1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,t)))))))


maximize(XPRS_BAR, Obj)
cut_type:=1

if(getprobstat=XPRS_UNB) then
write("Dual Unbounded ")
cut_type:=0
writeln("Cut_type",cut_type)
BigM:= (sum(e in REG_EMP, v in VESSELS)(dual_3(e,v)*(sum(r in ROLES(v))(
    allocate_dual(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4(e,v,t)*(sum(
    r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(allocate_dual
    (e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7(r)*( allocate_dual("AGENCY",r,1) - ag_starting(
    r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8(r,t)*(allocate_dual("
    AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
sum(e in GUARANTEED)(dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES, t
    in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES)(dual_15(r)*( ag_work_zero(r) + allocate_dual("AGENCY",
    r,1)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) (dual_16(r,t)*allocate_dual("
    AGENCY",r,t))+
sum(r in ALL_ROLES, t in TIME)(dual_17(r,t)*allocate_dual("AGENCY",r,t))+
sum(l in lambda, r in ALL_ROLES, t in TIME)(dual_18(l,r,t)*(- ag_max_work(
    r)*long_work_dual(l,"AGENCY",r,t) - (l-1)))+
sum(e in REG_EMP)(dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
```

```
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23(e,t)*(- min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, v in VESSELS)(dual_5(e,v)*(starting(e,v) - sum(r in
    ROLES(v))(allocate_dual(e,r,1))))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_6(e,v,t)*( sum
    (r in ROLES(v))(allocate_dual(e,r,(t-1))) - sum(r in ROLES(v))(
    allocate_dual(e,r,t))))+
sum(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(
    dual_25(l,e,r,t)* (- max_work(e)*long_work_dual(l,e,r,t) - max_work(e)
    *(1-allocate_dual(e,r,t)) + allocate_dual(e,r,t) - (l-1)))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_24(e,t)*(sum(r in ALL_ROLES)
    (allocate_dual(e,r,t)) - max_work(e)*(1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,t))))))+
sum(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(
    dual_25(l,e,r,t)* ( (max_work(e)*long_work_dual(l,e,r,t)) + (max_work(e
    )*(1-allocate_dual(e,r,t))) - allocate_dual(e,r,t) + (l-1)))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_24(e,t)*(sum(r in ALL_ROLES)
    (allocate_dual(e,r,t)) - max_work(e)*(1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,t)))))))<=BIGM
maximize(XPRS_BAR, Obj)
end-if

write("Step 2: ")
writeln("Cut_type",cut_type)

save_solution                   ! Write solution to memory
BigM:= 0

! Reset the 'BigM' constraint
end-if
until false

!-------------------------------------------------------------
! Process solution data
procedure save_solution
! Store values of u and x
forall(e in REG_EMP, v in VESSELS)do
```

```
sol_dual_3(e,v):=getsol(dual_3(e,v))
end-do

forall(e in REG_EMP, v in VESSELS)do
sol_dual_5(e,v):=getsol(dual_5(e,v))
end-do

forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_4(e,v,t):=getsol(dual_4(e,v,t))
end-do

forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_6(e,v,t):=getsol(dual_6(e,v,t))
end-do

forall(r in ALL_ROLES)do
sol_dual_7(r):=getsol(dual_7(r))
end-do

forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_8(r,t):= getsol(dual_8(r,t))
end-do

forall(e in GUARANTEED)do
sol_dual_9(e):=getsol(dual_9(e))
end-do

forall(e in GUARANTEED) do
sol_dual_10(e):=getsol(dual_10(e))
end-do

forall(r in ALL_ROLES) do
sol_dual_15(r):=getsol(dual_15(r))
end-do

forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_16(r,t):=getsol( dual_16(r,t))
```

```
end-do

forall(r in ALL_ROLES, t in TIME)do
sol_dual_17(r,t):= getsol(dual_17(r,t))
end-do

forall(l in lambda, r in ALL_ROLES, t in TIME)do
sol_dual_18(l,r,t):= getsol(dual_18(l,r,t))
end-do

forall(e in REG_EMP)do
sol_dual_19(e):=getsol(dual_19(e))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_20(e,t):=getsol(dual_20(e,t))
end-do

forall(e in REG_EMP, t in TIME) do
sol_dual_21(e,t):=getsol(dual_21(e,t))
end-do
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_23(e,t):=getsol( dual_23(e,t))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_24(e,t):=getsol( dual_24(e,t))
end-do

forall(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_25(l,e,r,t):= getsol(dual_25(l,e,r,t))
end-do


if Alg=STEP_0 then
Dual_cost:=(getobjval+sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((
    work_chng_cost(e,r,t)*((allocate(e,r,t)-cur_allocate(e,r,t))/(1-2*(
```

```
    cur_allocate(e,r,t)))))+ sum(l in lambda)(extension_chng_cost(l,e,r,t)
    *((long_work(l,e,r,t) -cur_long_work(l,e,r,t))/(1-2*(cur_long_work(l,e,
    r,t)))))))
UB:=100000000000
else
Dual_cost:=(getobjval+sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((
    work_chng_cost(e,r,t)*((allocate_dual(e,r,t)-cur_allocate(e,r,t))
    /(1-2*(cur_allocate(e,r,t)))))+ sum(l in lambda)(extension_chng_cost(l,
    e,r,t)*((long_work_dual(l,e,r,t) -cur_long_work(l,e,r,t))/(1-2*(
    cur_long_work(l,e,r,t)))))))
end-if

if( cut_type=0)then
UB:=UB

else
if(UB>=(Dual_cost+
sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*(
    cur_board(e,v,t)/((2*cur_board(e,v,t))-1)))+ (depart_chng_cost(e,v,t)*(
    cur_depart(e,v,t)/((2*cur_depart(e,v,t)-1))))) +
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e)))))then

UB:=(Dual_cost+
sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*(
    cur_board(e,v,t)/((2*cur_board(e,v,t))-1)))+ (depart_chng_cost(e,v,t)*(
    cur_depart(e,v,t)/((2*cur_depart(e,v,t)-1))))) +
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e))))
else
UB:=UB
```

```
end-if
end-if
initializations to "bin:shmem:sol"
sol_dual_3 sol_dual_5 sol_dual_4 sol_dual_6 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_15
sol_dual_16 sol_dual_17 sol_dual_18 sol_dual_19 sol_dual_21
sol_dual_20   sol_dual_23 sol_dual_24 sol_dual_25 Dual_cost cut_type
   allocate_dual long_work_dual UB
end-initializations

send(EVENT_SOLVED, getobjval)

write("Dual_cost: ",Dual_cost)
writeln("UB: ", UB)

! forall(j in Ctrs) write(sol_u(j), " ")
!write("\n x: ")
!forall(i in CtVars) write(getdual(CtrD(i)), " ")
writeln
fflush
end-procedure

end-model
!-------------------------------model MWP-------------------------------
 "Benders (mwp problem)"
uses "mmxprs", "mmjobs"

parameters
BIGM = 10000
end-parameters

forward procedure save_solution

declarations
STEP_0=2                        ! Event codes sent to submodels
STEP_2=4
STEP_3=5
```

```
EVENT_SOLVED=6                  ! Event codes sent by submodels
EVENT_INFEAS=7
EVENT_READY=8

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 "bin:shmem:probdata"
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
allocate_master: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
!allocate_mwp: dynamic array(ALL_EMP, ALL_ROLES, TIME) of real
work_total_master: array(REG_EMP, TIME) of mpvar
allocate: dynamic array(ALL_EMP, ALL_ROLES, TIME) of real
work_total: array(REG_EMP, TIME) of real
allocate_dual: dynamic array(ALL_EMP, ALL_ROLES, TIME) of real
work_total_dual: array(REG_EMP, TIME) of real
```

```
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
cut_type: real
```

```
end-declarations

initializations from "bin:shmem:probdata" ! Save data for submodels
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



declarations
lambda = 1..overall_max_work
long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real
long_work_master: array(lambda, ALL_EMP, ALL_ROLES, TIME) of mpvar
long_work_mwp: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real
long_work_dual: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real

!parameters
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
```

```
! Discrete variables

dual_3, dual_5: array(REG_EMP, VESSELS) of mpvar
dual_4, dual_6: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar

                                        ! or takes a non-negative integer
   value for agency crew
dual_7: array(ALL_ROLES) of mpvar
dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9: array(GUARANTEED) of mpvar
dual_10: array(GUARANTEED) of mpvar
dual_14: array(lambda, REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_15: array(ALL_ROLES) of mpvar
dual_16: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar                    !
   Used to track the consecutive working time of the agency employees
dual_17: array(ALL_ROLES, TIME) of mpvar
dual_18: array(lambda, ALL_ROLES, TIME) of mpvar
dual_19:array(REG_EMP) of mpvar
dual_21:array(REG_EMP, TIME) of mpvar
dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of mpvar
dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of mpvar

sol_dual_3, sol_dual_5: array(REG_EMP, VESSELS) of real
sol_dual_4, sol_dual_6: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of real

                                        ! or takes a non-negative integer
   value for agency crew
sol_dual_7: array(ALL_ROLES) of real
sol_dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_9: array(GUARANTEED) of real
sol_dual_10: array(GUARANTEED) of real
sol_dual_14: array(lambda, REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_15: array(ALL_ROLES) of real
sol_dual_16: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of real                 !
   Used to track the consecutive working time of the agency employees
```

```
sol_dual_17: array(ALL_ROLES, TIME) of real
sol_dual_18: array(lambda, ALL_ROLES, TIME) of real
sol_dual_19:array(REG_EMP) of real
sol_dual_21:array(REG_EMP, TIME) of real
sol_dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of real
sol_dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of real
allocate_mwp:array(ALL_EMP,ALL_ROLES,TIME) of real
Dual_cost:real
end-declarations

initializations from "bin:shmem:probdata"
extension_chng_cost cur_long_work long_work allocate work_total
end-initializations




declarations

dual_cons_board: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart: array(REG_EMP, VESSELS, TIME) of linctr        ! =1 if
   employee boards / departs vessel in given time period, 0 otherwise
! or takes a non-negative integer value for agency crew
dual_cons_ag_rboard: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart: array(ALL_ROLES, TIME) of linctr    ! =1 if agency
   crew starts / ends working on a role in given time period, 0 otherwise
dual_cons_undertime: array(GUARANTEED) of linctr
dual_cons_overtime: array(GUARANTEED) of linctr                   !
   Variables to calculate the amount of under/overtime carried out by
   employee
dual_cons_rest_total: array(REG_EMP, TIME) of linctr ! Used to track the
   consecutive working time / rest period requirements of each employee
dual_cons_work_total: array(REG_EMP, TIME) of linctr ! Used to track the
   consecutive working time / rest period requirements of each employee
dual_cons_ag_work_total: array(ALL_ROLES, TIME) of linctr
          ! Used to track the consecutive working time of the agency
```

```
    employees


Dual_cost_mwp:linctr
Obj_mwp:linctr
Obj_mwp_value:mpvar
end-declarations




send(EVENT_READY,0)              ! Model is ready (= running)

! (Re)solve this model until it is stopped by event "STEP_3"
repeat
wait
ev:= getnextevent
! Alg:= getclass(ev)

initializations from "bin:shmem:sol"
allocate_dual long_work_dual work_total_dual Dual_cost cut_type
end-initializations

writeln("Dual_cost: ",Dual_cost)

Obj_mwp:=Obj_mwp_value=0

Dual_cost_mwp:=(sum(e in REG_EMP, v in VESSELS)(dual_3(e,v)*(sum(r in
    ROLES(v))(allocate_dual(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4(e,v,t)*(sum(
    r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(allocate_dual
    (e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7(r)*( allocate_dual("AGENCY",r,1) - ag_starting(
    r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8(r,t)*(allocate_dual("
    AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
```

```
sum(e in GUARANTEED)(dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
sum(e in GUARANTEED)(dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES, t
    in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES)(dual_15(r)*( ag_work_zero(r) + allocate_dual("AGENCY",
    r,1)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) (dual_16(r,t)*allocate_dual("
    AGENCY",r,t))+
sum(r in ALL_ROLES, t in TIME)(dual_17(r,t)*allocate_dual("AGENCY",r,t))+
sum(l in lambda, r in ALL_ROLES, t in TIME)(dual_18(l,r,t)*(- ag_max_work(
    r)*long_work_dual(l,"AGENCY",r,t) - (l-1)))+
sum(e in REG_EMP)(dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23(e,t)*(- min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, v in VESSELS)(dual_5(e,v)*(starting(e,v) - sum(r in
    ROLES(v))(allocate_dual(e,r,1))))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_6(e,v,t)*( sum
    (r in ROLES(v))(allocate_dual(e,r,(t-1))) - sum(r in ROLES(v))(
    allocate_dual(e,r,t)))))=1

!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board(e,v,1):=dual_3(e,v)<=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_cons_board(
    e,v,t):=dual_4(e,v,t)<=0
!depart
forall(e in REG_EMP, v in VESSELS) dual_cons_depart(e,v,1):=dual_5(e,v)-
    ((min_rest(e)-1)*dual_21(e,1))<=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_cons_depart
    (e,v,t):=dual_6(e,v,t)-(min_rest(e)-1)*dual_21(e,t)<=0

!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard(r,1):= dual_7(r)<=0
```

```
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard(r,t) :=
    dual_8(r,t)<=0
!ag_rdepart

forall(r in ALL_ROLES) dual_cons_ag_rdepart(r,1):= -dual_7(r)+(ag_max_work
    (r)*dual_15(r))<=0

forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart(r,t) :=
    -dual_8(r,t)+(ag_max_work(r)*dual_16(r,t))<=0


!undertime
forall(e in GUARANTEED) dual_cons_undertime(e):= dual_9(e)<=0

!overtime
forall(e in GUARANTEED) dual_cons_overtime(e):= dual_10(e)<=0


!ag_work_total

forall(r in ALL_ROLES) dual_cons_ag_work_total(r,1):= dual_15(r)-dual_16(r
    ,2)+dual_17(r,1)-sum(l in lambda)dual_18(l,r,1)<=0
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN-1) dual_cons_ag_work_total(r,t
    ):= dual_16(r,t)-dual_16(r,t+1)+dual_17(r,t)-sum(l in lambda)dual_18(l,
    r,t)<=0
forall(r in ALL_ROLES,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_ag_work_total(r,t):= dual_16(r,t)+dual_17(r,t)- sum(l in
    lambda)dual_18(l,r,t)<=0


!rest_total

forall(e in REG_EMP) dual_cons_rest_total(e,1):= dual_19(e)-dual_20(e,2)+
    dual_21(e,1)-dual_23(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total(e,t):=
    dual_20(e,t)-dual_20(e,t+1)+dual_21(e,t)-dual_23(e,t+1)<=0
```

```
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
   dual_cons_rest_total(e,t):= dual_20(e,t)+dual_21(e,t)<=0

forall(e in REG_EMP, v in VESSELS) dual_3(e,v)>=0
forall(e in REG_EMP, v in VESSELS) dual_5(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4(e,v,t)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_6(e,v,t)>=0
forall(r in ALL_ROLES) dual_7(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8(r,t) is_free
forall(e in GUARANTEED) dual_9(e)>=0
forall(e in GUARANTEED) dual_10(e)>=0
forall(r in ALL_ROLES) dual_15(r)>=0
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_16(r,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_17(r,t)>=0
forall(l in lambda, r in ALL_ROLES,t in TIME) dual_18(l,r,t)>=0
forall(e in REG_EMP) dual_19(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23(e,t)>=0


maximize(XPRS_BAR, Obj_mwp_value)


writeln("Step 3_mwp: ")
writeln("Cut_type",cut_type)
save_solution                   ! Write solution to memory
! BigM:= 0                        ! Reset the 'BigM' constraint

until false

!-------------------------------------------------------------
! Process solution data
procedure save_solution
! Store values of u and x
forall(e in REG_EMP, v in VESSELS)do
sol_dual_3(e,v):=getsol(dual_3(e,v))
```

```
end-do

forall(e in REG_EMP, v in VESSELS)do
sol_dual_5(e,v):=getsol(dual_5(e,v))
end-do

forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_4(e,v,t):=getsol(dual_4(e,v,t))
end-do

forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_6(e,v,t):=getsol(dual_6(e,v,t))
end-do

forall(r in ALL_ROLES)do
sol_dual_7(r):=getsol(dual_7(r))
end-do

forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_8(r,t):= getsol(dual_8(r,t))
end-do

forall(e in GUARANTEED)do
sol_dual_9(e):=getsol(dual_9(e))
end-do

forall(e in GUARANTEED) do
sol_dual_10(e):=getsol(dual_10(e))
end-do

forall(r in ALL_ROLES) do
sol_dual_15(r):=getsol(dual_15(r))
end-do

forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_16(r,t):=getsol( dual_16(r,t))
end-do
```

```
forall(r in ALL_ROLES, t in TIME)do
sol_dual_17(r,t):= getsol(dual_17(r,t))
end-do

forall(l in lambda, r in ALL_ROLES, t in TIME)do
sol_dual_18(l,r,t):= getsol(dual_18(l,r,t))
end-do

forall(e in REG_EMP)do
sol_dual_19(e):=getsol(dual_19(e))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_20(e,t):=getsol(dual_20(e,t))
end-do

forall(e in REG_EMP, t in TIME) do
sol_dual_21(e,t):=getsol(dual_21(e,t))
end-do
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_23(e,t):=getsol( dual_23(e,t))
end-do


!Obj_mwp_value:=getobjval

initializations to "bin:shmem:sol"
sol_dual_3 sol_dual_5 sol_dual_4 sol_dual_6 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_15
sol_dual_16 sol_dual_17 sol_dual_18 sol_dual_19 sol_dual_21
sol_dual_20   sol_dual_23 cut_type Dual_cost
end-initializations

send(EVENT_SOLVED, getobjval)

writeln("Dual_obj_func_mwp: ",getobjval)
```

```
writeln("Dual_cost: ",Dual_cost)
! forall(j in Ctrs) write(sol_u(j), " ")
!write("\n x: ")
!forall(i in CtVars) write(getdual(CtrD(i)), " ")
!writeln
fflush
end-procedure
end-model
!--------------------------------RMP-------------------------------
model "Benders (integer problem)"
uses "mmxprs", "mmjobs"

parameters
STEP_0=2                          ! Event codes sent to submodels
STEP_1=3
EVENT_SOLVED=6                    ! Event codes sent by submodels
EVENT_READY=8
BIGM = 100000
OUTPUTFILE = "Logfile -bigdata.txt"
end-parameters

!setparam("XPRS_miprelstop",0.2)
!setparam("XPRS_maxtime",-300)

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

sol_obj: real                    ! Objective function value (primal)
MC: array(range) of linctr       ! Constraints generated by alg.
RM: range                        ! Model indices
stepmod: array(RM) of Model      ! Submodels
cut_type: real
end-declarations

initializations from "bin:shmem:probdata"
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

!the values of dv from master problem transferred to dual
allocate_dual: dynamic array(ALL_EMP, ALL_ROLES, TIME) of real
work_total_dual: array(REG_EMP, TIME) of real

!decision variables for master
allocate_master: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
work_total_master: array(REG_EMP, TIME) of mpvar

!cost parameters
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

!reading data
initializations from "bin:shmem:probdata"
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_master(e,r,t))
end-if
end-do
declarations
!parameter
lambda = 1..overall_max_work
long_work: array(lambda, ALL_EMP, ALL_ROLES, TIME) of real
!decision variable
long_work_master: array(lambda, ALL_EMP, ALL_ROLES, TIME) of mpvar

!parameters
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
```

```
! results obtained from dual model
sol_dual_3, sol_dual_5: array(REG_EMP, VESSELS) of real
sol_dual_4, sol_dual_6: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of real

                                        ! or takes a non-negative integer
   value for agency crew
sol_dual_7: array(ALL_ROLES) of real
sol_dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_9: array(GUARANTEED) of real
sol_dual_10: array(GUARANTEED) of real
sol_dual_14: array(lambda, REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_15: array(ALL_ROLES) of real
sol_dual_16: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of real                  !
   Used to track the consecutive working time of the agency employees
sol_dual_17: array(ALL_ROLES, TIME) of real
sol_dual_18: array(lambda, ALL_ROLES, TIME) of real
sol_dual_19:array(REG_EMP) of real
sol_dual_21:array(REG_EMP, TIME) of real
sol_dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of real
sol_dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_24:array(REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_25: array(lambda,REG_EMP, ALL_ROLES, 2..WEEKS_TO_PLAN) of real
z_best: array(range) of real
end-declarations
initializations from "bin:shmem:probdata"
extension_chng_cost cur_long_work
end-initializations
declarations
!constraints for master
All_covered: dynamic array(ALL_ROLES, TIME) of linctr
No_overlap: array(REG_EMP, TIME) of linctr
Long_work_count: dynamic array(lambda, REG_EMP, ALL_ROLES, TIME) of linctr
Rest_vs_work: array(REG_EMP,1..1) of linctr
Work_count: array(REG_EMP, TIME) of linctr
AG_long_work: dynamic array(lambda, ALL_ROLES, TIME) of linctr
z:mpvar
Primal_cost:linctr
```

```
end-declarations

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_master(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_master(e,r,t) <= 1


forall(e in REG_EMP) Work_count(e,1) := work_total_master(e,1) >=
    work_zero(e) + sum(r in ALL_ROLES)(allocate_master(e,r,1)) - max_work(e
    )*(1-(sum(r in ALL_ROLES)(allocate_master(e,r,1))))

forall(l in lambda, e in REG_EMP, r in ALL_ROLES | exists(long_work_master
    (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_master(l,e,r,1) >=
    work_zero(e) - max_work(e)*(1-allocate_master(e,r,1)) + allocate_master
    (e,r,1) - (l-1)
end-do

forall(e in REG_EMP) Rest_vs_work(e,1) := min_rest(e)*(1-(sum(r in
    ALL_ROLES)(allocate_master(e,r,1)))) >= rest_zero(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_master(e,
    r,t))) allocate_master(e,r,t) is_binary

forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(
    long_work_master(l,e,r,t)) ) long_work_master(l,e,r,t) is_binary
```

```
z is_free
send(EVENT_READY,0)              ! Model is ready (= running)

repeat
wait
ev:= getnextevent
ct:= integer(getvalue(ev))

initializations from "bin:shmem:sol"
sol_dual_3 sol_dual_5 sol_dual_4 sol_dual_6 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_15
sol_dual_16 sol_dual_17 sol_dual_18 sol_dual_19 sol_dual_21
sol_dual_20   sol_dual_23 sol_dual_24 sol_dual_25 cut_type
end-initializations

z_best(1):=-1000000000000000

setparam("XPRS_miprelstop",(0.55-(0.05*ct)+0.01))
! Add a new constraint as optimality cut
if(cut_type=1)then
writeln("Cut_type",cut_type)
MC(ct):= z >= (sum(e in REG_EMP, v in VESSELS)(sol_dual_3(e,v)*(sum(r in
    ROLES(v))(allocate_master(e,r,1)) - starting(e,v)))+
sum(r in ALL_ROLES)(sol_dual_7(r)*( allocate_master("AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_8(r,t)*(
    allocate_master("AGENCY",r,t) - allocate_master("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(sol_dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_master(e,r,t)))))+
sum(e in GUARANTEED)(sol_dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_master(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES)(sol_dual_15(r)*( ag_work_zero(r) + allocate_master("
    AGENCY",r,1)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) (sol_dual_16(r,t)*
    allocate_master("AGENCY",r,t))+
sum(r in ALL_ROLES, t in TIME)(sol_dual_17(r,t)*allocate_master("AGENCY",r
    ,t))+
```

```
sum(l in lambda, r in ALL_ROLES, t in TIME)(sol_dual_18(l,r,t)*(-
    ag_max_work(r)*long_work_master(l,"AGENCY",r,t) - (l-1)))+
sum(e in REG_EMP)(sol_dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_master(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_master(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_23(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_master(e,r,t))))))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_4(e,v,t)*(
    sum(r in ROLES(v))(allocate_master(e,r,t)) - sum(r in ROLES(v))(
    allocate_master(e,r,(t-1)))))+
sum(e in REG_EMP, v in VESSELS)(sol_dual_5(e,v)*(starting(e,v) - sum(r in
    ROLES(v))(allocate_master(e,r,1))))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_6(e,v,t)*(
     sum(r in ROLES(v))(allocate_master(e,r,(t-1))) - sum(r in ROLES(v))(
    allocate_master(e,r,t))))+
sum(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(
    sol_dual_25(l,e,r,t)* ( (max_work(e)*long_work_master(l,e,r,t)) + (
    max_work(e)*(1-allocate_master(e,r,t))) - allocate_master(e,r,t) + (l
    -1)))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_24(e,t)*(sum(r in
    ALL_ROLES)(allocate_master(e,r,t)) - max_work(e)*(1-(sum(r in ALL_ROLES
    )(allocate_master(e,r,t)))))))
end-if

if(cut_type=0)then
writeln("Cut_type",cut_type)
MC(ct):= (sum(e in REG_EMP, v in VESSELS)(sol_dual_3(e,v)*(sum(r in ROLES(
    v))(allocate_master(e,r,1)) - starting(e,v)))+
sum(r in ALL_ROLES)(sol_dual_7(r)*( allocate_master("AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_8(r,t)*(
    allocate_master("AGENCY",r,t) - allocate_master("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(sol_dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_master(e,r,t)))))+
sum(e in GUARANTEED)(sol_dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_master(e,r,t)))- g_weeks(e)))+
```

```
sum(r in ALL_ROLES)(sol_dual_15(r)*( ag_work_zero(r) + allocate_master("
    AGENCY",r,1)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) (sol_dual_16(r,t)*
    allocate_master("AGENCY",r,t))+
sum(r in ALL_ROLES, t in TIME)(sol_dual_17(r,t)*allocate_master("AGENCY",r
    ,t))+
sum(l in lambda, r in ALL_ROLES, t in TIME)(sol_dual_18(l,r,t)*(-
    ag_max_work(r)*long_work_master(l,"AGENCY",r,t) - (l-1)))+
sum(e in REG_EMP)(sol_dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_master(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_master(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_23(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_master(e,r,t))))))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_4(e,v,t)*(
    sum(r in ROLES(v))(allocate_master(e,r,t)) - sum(r in ROLES(v))(
    allocate_master(e,r,(t-1)))))+
sum(e in REG_EMP, v in VESSELS)(sol_dual_5(e,v)*(starting(e,v) - sum(r in
    ROLES(v))(allocate_master(e,r,1))))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_6(e,v,t)*(
     sum(r in ROLES(v))(allocate_master(e,r,(t-1))) - sum(r in ROLES(v))(
    allocate_master(e,r,t))))+
sum(l in lambda, e in REG_EMP, r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(
    sol_dual_25(l,e,r,t)* ( (max_work(e)*long_work_master(l,e,r,t)) + (
    max_work(e)*(1-allocate_master(e,r,t))) - allocate_master(e,r,t) + (l
    -1)))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_24(e,t)*(sum(r in
    ALL_ROLES)(allocate_master(e,r,t)) - max_work(e)*(1-(sum(r in ALL_ROLES
    )(allocate_master(e,r,t)))))))<=0
end-if
Primal_cost:=(z+sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((
    work_chng_cost(e,r,t)*((allocate_master(e,r,t)-cur_allocate(e,r,t))
    /(1-2*(cur_allocate(e,r,t)))))+ sum(l in lambda)(extension_chng_cost(l,
    e,r,t)*((long_work_master(l,e,r,t) -cur_long_work(l,e,r,t))/(1-2*(
    cur_long_work(l,e,r,t))))))+
sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*(
    cur_board(e,v,t)/((2*cur_board(e,v,t))-1)))+ (depart_chng_cost(e,v,t)*(
```

```
    cur_depart(e,v,t)/((2*cur_depart(e,v,t)-1))))) +
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e))))
fopen(OUTPUTFILE, F_OUTPUT)
setparam("XPRS_verbose",true)
!minimize(Total_cost)
minimize(Primal_cost)
! Store solution values of y
forall(l in lambda, e in ALL_EMP, r in ALL_ROLES, t in TIME)do
long_work_dual(l,e,r,t):=getsol(long_work_master(l,e,r,t))
end-do
!allocate_master: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_dual(l,r,t):=getsol(allocate_master(l,r,t))
end-do
initializations to "bin:shmem:sol"
allocate_dual long_work_dual work_total_dual
end-initializations
send(EVENT_SOLVED, getobjval)
write("Master: ",(getobjval), " ")
if(ct=1)then
if(getparam("XPRS_BESTBOUND")>=z_best(1))then
z_best(2):=getparam("XPRS_BESTBOUND")
end-if
end-if
if(ct>=2)then
if(z_best(ct)<=getparam("XPRS_BESTBOUND"))then
z_best(ct+1):=getparam("XPRS_BESTBOUND")
else
z_best(ct+1):= z_best(ct)
end-if
end-if
writeln("z_best: ", z_best(ct+1))
write("\n Slack: ")
```

```
forall(j in 1..ct) write(getslack(MC(j)), " ")
writeln
writeln("long_work: [")
forall(l in lambda) do
forall(e in ALL_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) write(getsol(long_work_dual(l,e,r,t)),"\t")
end-do
write("\n")
end-do
write("\n")
end-do
write("]\n")
writeln("allocate: [")
forall(e in ALL_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) write(getsol((allocate_dual(e,r,t))),"\t")
end-do
write("\n")
end-do
write("]\n")
fflush
fclose(F_OUTPUT)
write("Master: ",(getobjval), " ")

writeln("z_best: ", z_best(ct+1))
until false
end-model
```

B.1.2 Modified Recovery Model

The simplified and modified version of recovery model on FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) is shown in this section.

```
model ModelName
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver
parameters
```

```
DATE = "6-09-17"
PARAMETERFILE = "batch-input-parameters.dat"
end-parameters

declarations
InstanceName: string
end-declarations

initializations from PARAMETERFILE
InstanceName
end-initializations


DATAFILE := "Time-Windows-medium - Captains - "+InstanceName+".txt"
LOGFILE := InstanceName+"\\Logfile - TW 10min- Modified Integer Medium- "+
    InstanceName+" - "+DATE+".txt"
SUMMARYFILE := "Results -TW 10 min- Modified Integer Medium- "+DATE+".txt"

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
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
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
Work_count_start: array(REG_EMP) of linctr
AG_work_count: array(ALL_ROLES, TIME) of linctr
AG_work_count_start: array(ALL_ROLES) 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

!for controlling rest with modified version
Rest_new: array(REG_EMP,ALL_ROLES,range) of linctr
Rest_new1: array(REG_EMP, 1..WEEKS_TO_PLAN-1,range) of linctr
Depart_linking:array(REG_EMP, TIME) of linctr

!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_undertime: array(GUARANTEED) of linctr
Update_overtime: array(GUARANTEED) of linctr

a:integer
b:integer
c:integer
end-declarations
prog_setup_time := gettime

!objective function- cost calculation
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(e in GUARANTEED)((under_rate(e)*chng_undertime(e))+ (over_rate(e)*
    chng_overtime(e))))

!covering tasks
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

!no overlap
forall(e in REG_EMP, t in TIME) No_overlap(e,t) := sum(r in ALL_ROLES)
    allocate(e,r,t) <= 1

!board
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)))

!depart
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))

!-------------depart linking-------------------------------------
forall(e in REG_EMP, t in TIME) Depart_linking(e,t):=sum(v in VESSELS)
    depart(e,v,t)+sum(r in ALL_ROLES) allocate(e,r,t)<=1
!-------------depart linking-------------------------------------
```

```
!agency board and depart
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))

!under & over time
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)

!max consecutive work
forall(e in REG_EMP|work_zero(e)>=1) Work_count_start(e) := max_work(e) >=
     work_zero(e) + sum(r in ALL_ROLES, t in 0..max_work(e)-work_zero(e))(
    allocate(e,r,t+1))
forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-max_work(e)) Work_count(e,t) :=
     max_work(e) >= sum(r in ALL_ROLES, k in 0..max_work(e))(allocate(e,r,t
    +k))

!agency max consecutive work
forall(r in ALL_ROLES|ag_work_zero(r)>=1) AG_work_count_start(r) :=
    ag_max_work(r) >= ag_work_zero(r) + sum( t in 0..ag_max_work(r)-
    ag_work_zero(r))(allocate("AGENCY",r,t+1))
forall(r in ALL_ROLES, t in 1..WEEKS_TO_PLAN-ag_max_work(r)) AG_work_count
    (r,t) := ag_max_work(r) >= sum( k in 0..ag_max_work(r))(allocate("
    AGENCY",r,t+k))

!min rest constraints
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))


!---------------------------comb cut
    ----------------------------------------------------------------
forall( e in REG_EMP ,r in ALL_ROLES,t in 1..rest_zero(e)) do
if(rest_zero(e)>=1)then
Rest_new(e,r,t):= allocate(e,r,t)=0
end-if
end-do


forall( e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)
if(b<=a) then
c:=b
else
c:=a
end-if
end-if
forall(y in 1..c)do
Rest_new1(e,t,y) := sum(r in ALL_ROLES)(allocate(e,r,t+y))+sum (v in
    VESSELS) depart(e,v,t)<=1
end-do
end-do
!---------------------------comb cut
    ----------------------------------------------------------------

! 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(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(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

!-------------------------------------------------------------------------------

setparam("XPRS_verbose",true)
minimize(Total_cost)
```

```
end-model
```

B.1.3 Classical Benders Decomposition Algorithm

The implementation of Classical Benders Decomposition method with MW pareto optimality cuts in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) is given in this section. There are two files one is about to maintain the main algorithm and the other one is dual sub problem.

```
(!*******************************************************
Mosel Example Problems
======================

file benders_main.mos
'''''''''''''''''''''
Benders decomposition for solving a simple MIP.
- Master model -

*** ATTENTION: This model will return an error if ***
*** no more than one Xpress licence is available. ***

(c) 2008 Fair Isaac Corporation
author: S. Heipcke, Jan. 2006, rev. Jan. 2013
*******************************************************!)

model "Benders (master model)"
uses "mmxprs", "mmjobs", "mmsystem"



parameters
ALG = 1
BIGM = 100000000

DATE = "25-09-17"
```

```
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"
LOGFILE := InstanceName+"\\Logfile-1hr -Benders-mw "+InstanceName+" - "+
    DATE+".txt"
SUMMARYFILE := "Results -Benders 1 hr-mw"+DATE+".txt"

forward procedure solve_cont
forward procedure save_solution_dual
forward procedure evaluate_solution
forward function subprob : real
!forward function paretoprob : real
forward function heuristic_solve : real
forward procedure solve_primal_int(ct: integer)

forward procedure define_dualprob(prob:mpproblem)
forward function solve_dualprob(prob:mpproblem): real
forward procedure define_intprob(prob:mpproblem)
forward function solve_intprob(prob:mpproblem, ct:integer): real


declarations

STEP_0=2                        ! Codes sent to subproblems
STEP_1=3
STEP_2=4
STAT_SOLVED=6                   ! Status codes returned by subproblems
STAT_INFEAS=7
```

```
STAT_READY=8

UB:real !upperbound
LB: real !lowerbound
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
mip_x:integer
sol_obj: real                    ! Objective function value (primal)
MC: array(range) of linctr       ! Constraints generated by alg.
cuts_added:array(range) of integer
RM: range                        ! Model indices
cut_type: real
accept: real
Dualmodel: Model
Heuristic: Model
!Pareto:Model
Primal_cost:linctr
mybasis:basis
stepprob: array(RM) of mpproblem
best_bound:real
z_best:array (range) of real
end-declarations
!DECLARATION OF PARAMETERS AND DECISION VARIABLES
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
allocate: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer ! Variable
   for allocating employee to role during given time period
allocate_dual: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer !
   Variable for allocating employee to role during given time period
depart_dual: array(REG_EMP, VESSELS, TIME) of integer
depart: array(REG_EMP, VESSELS, TIME) of integer
allocate_best: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer
allocate_iteration: dynamic array(range,ALL_EMP, ALL_ROLES, TIME) of
   integer ! Variable for allocating employee to role during given time
   period
depart_iteration: array(range,REG_EMP, VESSELS, TIME) of integer


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 integer



end-declarations

!Reading from txt file
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

!Continue to define parameters and decision variables
```

```
declarations

!Added for recovery problem - detail of current roster, and change
    variable

!solution of dual sub problem
dual_3: array(REG_EMP, VESSELS) of mpvar
dual_4: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar
dual_7: array(ALL_ROLES) of mpvar
dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9: array(GUARANTEED) of mpvar
dual_10: array(GUARANTEED) of mpvar
dual_30: array(ALL_ROLES,TIME) of mpvar
dual_31: array(ALL_ROLES,TIME) of mpvar
dual_40: array(REG_EMP, VESSELS, TIME) of mpvar
dual_19:array(REG_EMP) of mpvar
dual_21:array(REG_EMP, TIME) of mpvar
dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of mpvar
dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of mpvar



dual_3_new: array(REG_EMP, VESSELS) of mpvar
dual_4_new: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar
dual_7_new: array(ALL_ROLES) of mpvar
dual_8_new: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9_new: array(GUARANTEED) of mpvar
dual_10_new: array(GUARANTEED) of mpvar
dual_30_new: array(ALL_ROLES,TIME) of mpvar
dual_31_new: array(ALL_ROLES,TIME) of mpvar
dual_40_new: array(REG_EMP, VESSELS, TIME) of mpvar
dual_19_new:array(REG_EMP) of mpvar
dual_21_new:array(REG_EMP, TIME) of mpvar
dual_20_new:array(REG_EMP, 2..WEEKS_TO_PLAN) of mpvar
dual_23_new:array(REG_EMP,2..WEEKS_TO_PLAN) of mpvar

sol_dual_3: array(range,REG_EMP, VESSELS) of real
```

```
sol_dual_4: array(range,REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of real

                                        ! or takes a non-negative integer
    value for agency crew
sol_dual_7: array(range,ALL_ROLES) of real
sol_dual_8: array(range,ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_9: array(range,GUARANTEED) of real
sol_dual_10: array(range,GUARANTEED) of real
sol_dual_19:array(range,REG_EMP) of real
sol_dual_21:array(range,REG_EMP, TIME) of real
sol_dual_20:array(range,REG_EMP, 2..WEEKS_TO_PLAN) of real
sol_dual_23:array(range,REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_30: array(range,ALL_ROLES,TIME) of real
sol_dual_31: array(range,ALL_ROLES,TIME) of real
sol_dual_40: array(range,REG_EMP, VESSELS, TIME) of real
dummy:linctr
dummy2:array(range) of linctr
! Objective Value of dual problem
Dual_cost:real
end-declarations

! reading the parameters
initializations from DATAFILE
allocate depart
end-initializations
declarations
!constraints for dual problem
dual_cons_board: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart: array(REG_EMP, VESSELS, TIME) of linctr        ! =1 if
    employee boards / departs vessel in given time period, 0 otherwise

                                        ! or takes a non-negative integer
    value for agency crew
dual_cons_ag_rboard: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart: array(ALL_ROLES, TIME) of linctr    ! =1 if agency
    crew starts / ends working on a role in given time period, 0 otherwise
dual_cons_undertime: array(GUARANTEED) of linctr
```

```
dual_cons_overtime: array(GUARANTEED) of linctr                  !
   Variables to calculate the amount of under/overtime carried out by
   employee

dual_cons_board_new: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart_new: array(REG_EMP, VESSELS, TIME) of linctr    ! =1 if
   employee boards / departs vessel in given time period, 0 otherwise

                                          ! or takes a non-negative integer
   value for agency crew
dual_cons_ag_rboard_new: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart_new: array(ALL_ROLES, TIME) of linctr       ! =1 if
   agency crew starts / ends working on a role in given time period, 0
   otherwise
dual_cons_undertime_new: array(GUARANTEED) of linctr
dual_cons_overtime_new: array(GUARANTEED) of linctr                      !
   Variables to calculate the amount of under/overtime carried out by
   employee
dual_cons_rest_total:array(REG_EMP, TIME) of linctr

!constraints for integer problem
All_covered: dynamic array(ALL_ROLES, TIME) of linctr
No_overlap: array(REG_EMP, TIME) of linctr
Work_count: array(REG_EMP, TIME) of linctr
Work_count_start: array(REG_EMP) of linctr
Depart_constr: array(REG_EMP, VESSELS, TIME) of linctr
Rest_vs_work: array(REG_EMP, TIME) of linctr
!Rest_new: array(REG_EMP) of linctr
!Rest_new1: array(REG_EMP, TIME) of linctr
Rest_new: array(REG_EMP,ALL_ROLES,range) of linctr
Rest_new1: array(REG_EMP, 1..WEEKS_TO_PLAN-1,range) of linctr
Depart_linking: array(REG_EMP, TIME) of linctr
ct:integer
int_started:integer
allocate_master: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
depart_master: array(REG_EMP, VESSELS, TIME) of mpvar     ! =1 if employee
   boards / departs vessel in given time period, 0 otherwise
```

```
Obj: linctr
Dual_first:linctr
z:mpvar
status: array(mpproblem) of integer ! Subproblem status
AG_work_count_start:array(ALL_ROLES) of linctr
AG_work_count:array(ALL_ROLES, TIME) of linctr
extra:array(REG_EMP) of linctr
z1:array(REG_EMP, VESSELS) of mpvar
z2:array(ALL_ROLES)of mpvar
z4:array(REG_EMP)of mpvar
z3:array(GUARANTEED)of mpvar
MC10: array(range,REG_EMP, VESSELS) of linctr ! Constraints generated by
    alg.
MC20: array(range,ALL_ROLES) of linctr
MC30: array(range,GUARANTEED) of linctr
MC40: array(range,REG_EMP) of linctr
Obj_mwp: linctr
Obj_mwp_value:mpvar
UB1:real
Cut:dynamic array(range)of linctr
type:dynamic array(range)of integer
cutid: array(range) of integer
finish:real
begin:real

end-declarations

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_master(e,r,t))
end-if
end-do

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_dual(e,r,t))
end-if
```

```
end-do

forall(a in 1..ct,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_iteration(a,e,r
    ,t))
end-if
end-do

!sharing data info with submodels
ct:=0
UB_first:=10000000000
x:=1
best_bound:=-10000000000
z_best(0):= best_bound
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_iteration(0,l,r,t):=allocate(l,r,t)
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_iteration(0,r,v,t):=depart(r,v,t)
end-do

create(stepprob(1)); define_intprob(stepprob(1))

create(stepprob(2)); define_dualprob(stepprob(2))



procedure solve_primal_int(ct: integer)
sol_obj:= solve_intprob(stepprob(1), ct)
end-procedure

procedure solve_cont
! Start the problem solving
res:= solve_dualprob(stepprob(2))

end-procedure
```

```
fixed:=(sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)
    *(cur_board(e,v,t)/((2*cur_board(e,v,t))-1))))+
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e))))

!-------------------------------------------------------------
! Define the dual problem
procedure define_dualprob(prob:mpproblem)
!board
with prob do

!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board(e,v,1):=dual_3(e,v)+
    dual_40(e,v,1)<=(board_chng_cost(e,v,1))/(1-(2*cur_board(e,v,1)))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_cons_board(
    e,v,t):=dual_4(e,v,t)+dual_40(e,v,t)<=(board_chng_cost(e,v,t))/(1-(2*
    cur_board(e,v,t)))
forall(e in REG_EMP, v in VESSELS, t in TIME)sethidden(dual_cons_board(e,v
    ,t),false)

!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard(r,1):= dual_7(r)+dual_31(r,1)
    <=(ag_board_chng_cost(r,1)/(1-(2*cur_ag_rboard(r,1))))
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard(r,t) :=
    dual_8(r,t)+dual_31(r,t)<=(ag_board_chng_cost(r,t)/(1-(2*cur_ag_rboard(
    r,t))))
forall(r in ALL_ROLES,t in TIME) sethidden( dual_cons_ag_rboard(r,t),false
    )
!ag_rdepart

forall(r in ALL_ROLES) dual_cons_ag_rdepart(r,1):= -dual_7(r)+dual_30(r,1)
    <=(ag_depart_chng_cost(r,1)/(1-(2*cur_ag_rdepart(r,1))))
```

```
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart(r,t) :=
    -dual_8(r,t)+dual_30(r,t)<=(ag_depart_chng_cost(r,t)/(1-(2*
    cur_ag_rdepart(r,t))))
forall(r in ALL_ROLES,t in TIME) sethidden( dual_cons_ag_rdepart(r,t),
    false)


!undertime
forall(e in GUARANTEED) dual_cons_undertime(e):= dual_9(e)<=under_rate(e)
forall(e in GUARANTEED) sethidden( dual_cons_undertime(e),false)
!overtime
forall(e in GUARANTEED) dual_cons_overtime(e):= dual_10(e)<=over_rate(e)
forall(e in GUARANTEED) sethidden( dual_cons_overtime(e),false)

!rest_total

forall(e in REG_EMP) dual_cons_rest_total(e,1):= dual_19(e)-dual_20(e,2)+
    dual_21(e,1)-dual_23(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total(e,t):=
    dual_20(e,t)-dual_20(e,t+1)+dual_21(e,t)-dual_23(e,t+1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_rest_total(e,t):= dual_20(e,t)+dual_21(e,t)<=0
forall(e in REG_EMP,t in TIME) sethidden(dual_cons_rest_total(e,t),false)


forall(e in REG_EMP, v in VESSELS) dual_3(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4(e,v,t)>=0
forall(r in ALL_ROLES) dual_7(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8(r,t) is_free
forall(e in GUARANTEED) dual_9(e)>=0
forall(e in GUARANTEED) dual_10(e)>=0
forall(e in REG_EMP) dual_19(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30(r,t)<=0
```

```
forall(r in ALL_ROLES, t in TIME) dual_31(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40(e,v,t)<=0


status(prob):= STAT_READY
end-do
end-procedure

! Process dual solution data
procedure save_dualsolution(prob:mpproblem)
if(ct=0)then
forall(e in REG_EMP, v in VESSELS)do
sol_dual_3(ct,e,v):=getsol(dual_3(e,v))
end-do


forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_4(ct,e,v,t):=getsol(dual_4(e,v,t))
end-do

forall(e in REG_EMP, v in VESSELS, t in TIME)do
sol_dual_40(ct,e,v,t):=getsol(dual_40(e,v,t))
end-do

forall(r in ALL_ROLES)do
sol_dual_7(ct,r):=getsol(dual_7(r))
end-do

forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_8(ct,r,t):= getsol(dual_8(r,t))
end-do

forall(e in GUARANTEED)do
sol_dual_9(ct,e):=getsol(dual_9(e))
end-do

forall(e in GUARANTEED) do
```

```
sol_dual_10(ct,e):=getsol(dual_10(e))
end-do


forall(r in ALL_ROLES, t in TIME)do
sol_dual_30(ct,r,t):= getsol(dual_30(r,t))
end-do

forall(r in ALL_ROLES, t in TIME)do
sol_dual_31(ct,r,t):= getsol(dual_31(r,t))
end-do

forall(e in REG_EMP)do
sol_dual_19(ct,e):=getsol(dual_19(e))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_20(ct,e,t):= getsol(dual_20(e,t))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_23(ct,e,t):= getsol(dual_23(e,t))
end-do

forall(e in REG_EMP, t in TIME)do
sol_dual_21(ct,e,t):= getsol(dual_21(e,t))
end-do

end-if
status(prob):= STAT_SOLVED

fflush
end-procedure

! (Re)solve the dual problem
function solve_dualprob(prob:mpproblem): real
with prob do
```

```
status(prob):= STAT_READY

Dual_first:=(sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40(e,v,t))+
sum(e in REG_EMP, v in VESSELS)(dual_3(e,v)*(sum(r in ROLES(v))(allocate(e
    ,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4(e,v,t)*(sum(
    r in ROLES(v))(allocate(e,r,t)) - sum(r in ROLES(v))(allocate(e,r,(t-1)
    ))))+
sum(r in ALL_ROLES)(dual_7(r)*( allocate("AGENCY",r,1) - ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8(r,t)*(allocate("AGENCY",
    r,t) - allocate("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate(e,r,t)))))+
sum(e in GUARANTEED)(dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES, t
    in TIME)(allocate(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30(r,t)*(1-allocate("AGENCY",r,t)))+
sum(r in ALL_ROLES)(dual_31(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31(r,t)*(1-allocate("
    AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23(e,t)*(- min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart(e,v,t))))))
sethidden(Dual_first,false)



if (ct=0) then               ! Produce an initial solution for the
! dual problem using a dummy objective
maximize(XPRS_BAR,Dual_first)

if(getprobstat = XPRS_INF) then
writeln("Problem is infeasible")
```

```
send(STAT_INFEAS,0)           ! Problem is infeasible
else
write("**** Start solution: ")
write("**** Problem is not infeasible: ")
if(getprobstat = XPRS_UNB) then
writeln("Problem is unbounded")

cut_type:=0
writeln("Cut_type",cut_type)

Obj_mwp:=Obj_mwp_value=0

Dual_cost_mwp:=(sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40(e,v,t))
    +
sum(e in REG_EMP, v in VESSELS)(dual_3(e,v)*(sum(r in ROLES(v))(allocate(e
    ,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4(e,v,t)*(sum(
    r in ROLES(v))(allocate(e,r,t)) - sum(r in ROLES(v))(allocate(e,r,(t-1)
    ))))+
sum(r in ALL_ROLES)(dual_7(r)*( allocate("AGENCY",r,1) - ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8(r,t)*(allocate("AGENCY",
    r,t) - allocate("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate(e,r,t)))))+
sum(e in GUARANTEED)(dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES, t
    in TIME)(allocate(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30(r,t)*(1-allocate("AGENCY",r,t)))+
sum(r in ALL_ROLES)(dual_31(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31(r,t)*(1-allocate("
    AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23(e,t)*(- min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate(e,r,t))))))+
```

```
sum(e in REG_EMP, t in TIME)(dual_21(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart(e,v,t))))))=1


!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board(e,v,1):=dual_3(e,v)+
    dual_40(e,v,1)<=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_cons_board(
    e,v,t):=dual_4(e,v,t)+dual_40(e,v,t)<=0

!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard(r,1):= dual_7(r)+dual_31(r,1)
    <=0
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard(r,t) :=
    dual_8(r,t)+dual_31(r,t)<=0
!ag_rdepart

forall(r in ALL_ROLES) dual_cons_ag_rdepart(r,1):= -dual_7(r)+dual_30(r,1)
    <=0
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart(r,t) :=
    -dual_8(r,t)+dual_30(r,t)<=0


!undertime
forall(e in GUARANTEED) dual_cons_undertime(e):= dual_9(e)<=0

!overtime
forall(e in GUARANTEED) dual_cons_overtime(e):= dual_10(e)<=0

!rest_total

forall(e in REG_EMP) dual_cons_rest_total(e,1):= dual_19(e)-dual_20(e,2)+
    dual_21(e,1)-dual_23(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total(e,t):=
    dual_20(e,t)-dual_20(e,t+1)+dual_21(e,t)-dual_23(e,t+1)<=0
```

```
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
   dual_cons_rest_total(e,t):= dual_20(e,t)+dual_21(e,t)<=0


forall(e in REG_EMP, v in VESSELS) dual_3(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4(e,v,t)>=0
forall(r in ALL_ROLES) dual_7(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8(r,t) is_free
forall(e in GUARANTEED) dual_9(e)>=0
forall(e in GUARANTEED) dual_10(e)>=0
forall(e in REG_EMP) dual_19(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30(r,t)<=0
forall(r in ALL_ROLES, t in TIME) dual_31(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40(e,v,t)<=0

maximize(XPRS_BAR, Obj_mwp_value)

writeln("UB_first: +infinity")
writeln("UB_first:", UB_first)
UB:=UB_first
accept:=0
Dual_cost:=UB_first
else
writeln("Problem is feasible")
cut_type:=1
writeln("Cut_type:",cut_type)
accept:=1
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_best(l,r,t):=allocate(l,r,t)
end-do
Dual_cost:=getobjval

(!  UB:=(Dual_cost+
```

```
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
   allocate(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))))+
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
   depart(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t)))))+fixed)
!)
UB:=UB_first
writeln("UB_first:", UB)

end-if

writeln("accept: ", accept)
save_dualsolution(prob)
UB1:=(Dual_cost+
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
   allocate(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))))+
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
   depart(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t)))))+fixed)
writeln("UB1: ",UB1)
writeln("Dual_cost_first: ",Dual_cost)

BigM:= 0

end-if
end-if

end-do
end-function

!-------------------------------------------------------------

solve_cont

!-------------1st iteration of dual finished---------------
initializations to "bin:shmem:sol"
InstanceName sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
   sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
```

```
    accept allocate_best
end-initializations

!-------------------------------------------------------



! Define the integer problem
procedure define_intprob(prob:mpproblem)
with prob do

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_master(e,r,t))
end-if
end-do



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_master(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_master(e,r,t) <= 1


forall(e in REG_EMP, v in VESSELS) Depart_constr(e,v,1) := depart_master(e
    ,v,1) >= starting(e,v) - sum(r in ROLES(v))(allocate_master(e,r,1))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) Depart_constr(e,
    v,t) := depart_master(e,v,t) >= sum(r in ROLES(v))(allocate_master(e,r
    ,(t-1))) - sum(r in ROLES(v))(allocate_master(e,r,t))


!-------------depart linking-------------------------------------
```

```
forall(e in REG_EMP, t in TIME) Depart_linking(e,t):=sum(v in VESSELS)
   depart_master(e,v,t)+sum(r in ALL_ROLES) allocate_master(e,r,t)<=1
!-------------depart linking-----------------------------------


forall(e in REG_EMP|work_zero(e)+WEEKS_TO_PLAN>max_work(e))
   Work_count_start(e) := max_work(e) >= work_zero(e) + sum(r in ALL_ROLES
   , t in 0..max_work(e)-work_zero(e))(allocate_master(e,r,t+1))
!forall(e in REG_EMP| work_zero(e)+WEEKS_TO_PLAN>max_work(e))sethidden(
   Work_count_start(e) ,true)
!master11
forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-max_work(e)| work_zero(e)+
   WEEKS_TO_PLAN>max_work(e)) Work_count(e,t) := max_work(e) >= sum(r in
   ALL_ROLES, k in 0..max_work(e))(allocate_master(e,r,t+k))
!forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-max_work(e)| work_zero(e)+
   WEEKS_TO_PLAN>max_work(e))sethidden(Work_count(e,t) ,true)

!master12

!forall(e in REG_EMP| max_work(e)<=WEEKS_TO_PLAN) extra(e):=sum(r in
   ALL_ROLES,t in 1..max_work(e)+1, y in 0..WEEKS_TO_PLAN-max_work(e)-1)
   allocate_master(e,r,t+y) <= max_work(e)*(WEEKS_TO_PLAN-max_work(e))
!master12_ekstra

forall(r in ALL_ROLES|ag_work_zero(r)+WEEKS_TO_PLAN>ag_max_work(r))
   AG_work_count_start(r) := ag_max_work(r) >= ag_work_zero(r) + sum( t in
    0..ag_max_work(r)-ag_work_zero(r))(allocate_master("AGENCY",r,t+1))
!forall(r in ALL_ROLES|ag_work_zero(r)+WEEKS_TO_PLAN>ag_max_work(r))
   sethidden(AG_work_count_start(r),true)
!master13

forall(r in ALL_ROLES, t in 1..WEEKS_TO_PLAN-ag_max_work(r)|ag_work_zero(r
   )+WEEKS_TO_PLAN>ag_max_work(r)) AG_work_count(r,t) := ag_max_work(r) >=
    sum( k in 0..ag_max_work(r))(allocate_master("AGENCY",r,t+k))
!forall(r in ALL_ROLES, t in 1..WEEKS_TO_PLAN-ag_max_work(r)|ag_work_zero(
   r)+WEEKS_TO_PLAN>ag_max_work(r)) sethidden(AG_work_count(r,t) ,true)
!master14
```

```
! forall( e in REG_EMP | rest_zero(e)>=1) Rest_new(e):= sum(r in ALL_ROLES
    ,t in 1..rest_zero(e)) allocate_master(e,r,t)=0
forall( e in REG_EMP ,r in ALL_ROLES,t in 1..rest_zero(e)) do
if(rest_zero(e)>=1)then
Rest_new(e,r,t):= allocate_master(e,r,t)=0
end-if
end-do

forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)
if(b<=a) then
c:=b
else
c:=a
end-if
end-if
Rest_new2(e,t) := sum(r in ALL_ROLES, y in 0..c)(allocate_master(e,r,t+y))
    <=c*(1-sum (v in VESSELS) depart_master(e,v,t))
sethidden(Rest_new2(e,t),true)
end-do

forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)
if(b<=a) then
c:=b
else
c:=a
end-if
end-if
forall(y in 0..c)do
if(c>=1)then
```

```
Rest_new1(e,t,y) := sum(r in ALL_ROLES)(allocate_master(e,r,t+y))+sum (v
    in VESSELS) depart_master(e,v,t)<=1
end-if
end-do
end-do




forall(e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(allocate_master(e,
    r,t))) allocate_master(e,r,t) is_binary
forall(e in REG_EMP, v in VESSELS, t in TIME) depart_master(e,v,t)
    is_binary

z is_free
forall(e in REG_EMP, v in VESSELS) z1(e,v) is_free
forall(r in ALL_ROLES) z2(r) is_free
forall(e in REG_EMP) z4(e) is_free
forall(e in GUARANTEED) z3(e) is_free

end-do



status(prob):= STAT_READY
end-procedure

! Solve the integer problem
function solve_intprob(prob:mpproblem, ct:integer): real
with prob do
status(prob):= STAT_READY
Primal_cost:=(z+
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
    allocate_master(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))
    ))+
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
    depart_master(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t)))))+
```

```
sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*(
    cur_board(e,v,t)/((2*cur_board(e,v,t))-1))))+
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e))))

if(cut_type=1)then
writeln("Cut_type",cut_type)

(!MC(ax):= z >= (sum(e in REG_EMP, v in VESSELS, t in TIME)(sol_dual_40(ax
    ,e,v,t))+
sum(e in REG_EMP, v in VESSELS)(sol_dual_3(ax,e,v)*(sum(r in ROLES(v))(
    allocate_master(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_4(ax,e,v,t
    )*(sum(r in ROLES(v))(allocate_master(e,r,t)) - sum(r in ROLES(v))(
    allocate_master(e,r,(t-1)))))+
sum(r in ALL_ROLES)(sol_dual_7(ax,r)*( allocate_master("AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_8(ax,r,t)*(
    allocate_master("AGENCY",r,t) - allocate_master("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(sol_dual_9(ax,e)*(g_weeks(e) - (exp_worktime(e) + sum
    (r in ALL_ROLES, t in TIME)(allocate_master(e,r,t)))))+
sum(e in GUARANTEED)(sol_dual_10(ax,e)*((exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate_master(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(sol_dual_30(ax,r,t)*(1-allocate_master("
    AGENCY",r,t)))+
sum(r in ALL_ROLES)(sol_dual_31(ax,r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_31(ax,r,t)*(1-
    allocate_master("AGENCY",r,(t-1))))+
sum(e in REG_EMP)(sol_dual_19(ax,e)*( rest_zero(e) - (1-(sum(r in
    ALL_ROLES)(allocate_master(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_20(ax,e,t)*(- (1-(sum(r
    in ALL_ROLES)(allocate_master(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_23(ax,e,t)*(- min_rest(e
    )*(1-(sum(r in ALL_ROLES)(allocate_master(e,r,t))))))+
```

```
sum(e in REG_EMP, t in TIME)(sol_dual_21(ax,e,t)*((min_rest(e)-1)*(sum(v
    in VESSELS)(depart_master(e,v,t))))))
end-do
!)
forall(ax in 0..ct)do
forall(e in REG_EMP, v in VESSELS)do
MC10(ax,e,v):=z1(e,v)>= (sum( t in TIME)(sol_dual_40(ax,e,v,t))+
(sol_dual_3(ax,e,v)*(sum(r in ROLES(v))(allocate_master(e,r,1)) - starting
    (e,v)))+
sum(t in 2..WEEKS_TO_PLAN)(sol_dual_4(ax,e,v,t)*(sum(r in ROLES(v))(
    allocate_master(e,r,t)) - sum(r in ROLES(v))(allocate_master(e,r,(t-1))
    ))))
end-do

forall(r in ALL_ROLES)do
MC20(ax,r):=z2(r) >= ((sol_dual_7(ax,r)*( allocate_master("AGENCY",r,1) -
    ag_starting(r)))+
sum(t in 2..WEEKS_TO_PLAN)(sol_dual_8(ax,r,t)*(allocate_master("AGENCY",r,
    t) - allocate_master("AGENCY",r,(t-1))))+
sum(t in TIME)(sol_dual_30(ax,r,t)*(1-allocate_master("AGENCY",r,t)))+
(sol_dual_31(ax,r,1)*(1- ag_starting(r)))+
sum( t in 2..WEEKS_TO_PLAN)(sol_dual_31(ax,r,t)*(1-allocate_master("AGENCY
    ",r,(t-1)))))
end-do

forall(e in GUARANTEED)do
MC30(ax,e):= z3(e) >= ((sol_dual_9(ax,e)*(g_weeks(e) - (exp_worktime(e) +
    sum(r in ALL_ROLES, t in TIME)(allocate_master(e,r,t)))))+
(sol_dual_10(ax,e)*((exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(
    allocate_master(e,r,t)))- g_weeks(e))))
end-do

forall(e in REG_EMP)do
MC40(ax,e):= z4(e) >=(sol_dual_19(ax,e)*( rest_zero(e) - (1-(sum(r in
    ALL_ROLES)(allocate_master(e,r,1))))))+
sum( t in 2..WEEKS_TO_PLAN)(sol_dual_20(ax,e,t)*(- (1-(sum(r in ALL_ROLES)
    (allocate_master(e,r,t))))))+
```

```
sum( t in 2..WEEKS_TO_PLAN)(sol_dual_23(ax,e,t)*(- min_rest(e)*(1-(sum(r
    in ALL_ROLES)(allocate_master(e,r,t))))))+
sum( t in TIME)(sol_dual_21(ax,e,t)*((min_rest(e)-1)*(sum(v in VESSELS)(
    depart_master(e,v,t))))))
end-do

MC(ax):= z >= (sum(e in REG_EMP, v in VESSELS) (z1(e,v))+ sum (e in
    GUARANTEED) z3(e)+sum(r in ALL_ROLES)z2(r)+sum (e in REG_EMP) z4(e))
end-do
end-if

if(cut_type=0)then
writeln("Cut_type",cut_type)

forall(ax in 0..ct)do
MC(ax):= (sum(e in REG_EMP, v in VESSELS, t in TIME)(sol_dual_40(ax,e,v,t)
    )+
sum(e in REG_EMP, v in VESSELS)(sol_dual_3(ax,e,v)*(sum(r in ROLES(v))(
    allocate_master(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_4(ax,e,v,t
    )*(sum(r in ROLES(v))(allocate_master(e,r,t)) - sum(r in ROLES(v))(
    allocate_master(e,r,(t-1)))))+
sum(r in ALL_ROLES)(sol_dual_7(ax,r)*( allocate_master("AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_8(ax,r,t)*(
    allocate_master("AGENCY",r,t) - allocate_master("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(sol_dual_9(ax,e)*(g_weeks(e) - (exp_worktime(e) + sum
    (r in ALL_ROLES, t in TIME)(allocate_master(e,r,t)))))+
sum(e in GUARANTEED)(sol_dual_10(ax,e)*((exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate_master(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(sol_dual_30(ax,r,t)*(1-allocate_master("
    AGENCY",r,t)))+
sum(r in ALL_ROLES)(sol_dual_31(ax,r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_31(ax,r,t)*(1-
    allocate_master("AGENCY",r,(t-1))))+
sum(e in REG_EMP)(sol_dual_19(ax,e)*( rest_zero(e) - (1-(sum(r in
    ALL_ROLES)(allocate_master(e,r,1))))))+
```

```
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_20(ax,e,t)*(- (1-(sum(r
    in ALL_ROLES)(allocate_master(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_23(ax,e,t)*(- min_rest(e
    )*(1-(sum(r in ALL_ROLES)(allocate_master(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(sol_dual_21(ax,e,t)*((min_rest(e)-1)*(sum(v
    in VESSELS)(depart_master(e,v,t))))))<=0
end-do
end-if


setparam("XPRS_heurstrategy",3)

if(cut_type=0)then
setparam("XPRS_maxtime",-30)
else
if(x>=2)then
if(50*(x+1)>=3600-(finish-begin))then
setparam("XPRS_maxtime",-400)
else
setparam("XPRS_maxtime",-(50*(x+1)))
end-if
else
setparam("XPRS_maxtime",-(50*(x+1)))
end-if
end-if

!setparam("XPRS_miprelstop",(0.8-(0.1*x)))
setcallback(XPRS_CB_INTSOL, "printsol")

setparam("XPRS_verbose",true)
minimize( Primal_cost)

z_best(ct):=getparam("XPRS_BESTBOUND")
best_bound:=max(i in 0..ct)z_best(ct)

end-do
```

```
end-function

!-------------------------------------------------------------




int_started:=1
begin:=gettime

if(int_started=1)then


cuts_added(0):=0
forall(y in 1..200)do

x:=y

!writeln("x:",x)

! writeln(" best_bound:", best_bound)
solve_primal_int(ct)

finish:=gettime
writeln("FINISh_BEGIN: ", finish-begin )

ct:=ct+1


if(finish-begin>=3250)then

res2:= compile("subdualcombcut_updated-acmw.mos") ! Compile the knapsack
   model
load(Dualmodel, "subdualcombcut_updated-acmw.bim") ! Load the knapsack
   model
initializations to "bin:shmem:sol"
allocate_dual depart_dual ct UB Dual_cost InstanceName allocate_best
```

```
end-initializations
run(Dualmodel) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
   sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
   accept allocate_best
end-initializations



fopen(SUMMARYFILE, F_APPEND)
write(InstanceName)
write("\t",finish-begin)
write("\t",UB1,"\t",UB,"\t", z_best(ct))
write("\n")
fclose(F_APPEND)

fopen(LOGFILE, F_APPEND)
writeln("allocate: [")

forall(e in ALL_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) write(allocate_best(e,r,t),"\t")
end-do
write("\n")
end-do

write("]\n")

fclose(F_APPEND)

break
else
```

```
res2:= compile("subdualcombcut_updated-acmw.mos") ! Compile the knapsack
    model
load(Dualmodel, "subdualcombcut_updated-acmw.bim") ! Load the knapsack
    model
initializations to "bin:shmem:sol"
allocate_dual depart_dual ct UB Dual_cost InstanceName allocate_best
end-initializations
run(Dualmodel) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
    sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
    accept allocate_best
end-initializations
cuts_added(x):=ct
fopen(LOGFILE,F_APPEND)
write(InstanceName)
write("\t",finish-begin)
write("\t",UB1,"\t",UB,"\t", z_best(ct))
write("\n")
fclose(F_APPEND)
end-if

evaluate_solution

cuts_added(x):=ct

end-do
end-if


!-------------------------------------------------------------------

procedure printsol
```

```
declarations
objval:real
mip:integer
time_passed:real
end-declarations


time_passed:=gettime

objval:= getparam("XPRS_MIPOBJVAL")
mip:=getparam("XPRS_MIPSOLS")

mip_x:=mip


forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_dual(l,r,t):=round(getsol(allocate_master(l,r,t)))
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_dual(r,v,t):=round(getsol(depart_master(r,v,t)))
end-do

forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_iteration(ct,l,r,t):=round(getsol(allocate_master(l,r,t)))
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_iteration(ct,r,v,t):=round(getsol(depart_master(r,v,t)))
end-do

writeln("INTEGER PROBLEM")


end-procedure

!-------------------------------------------------------------------
```

```
procedure save_solution_dual
! Store values of u and x
if(ct=0)then
forall(e in REG_EMP, v in VESSELS)do
sol_dual_3(ct,e,v):=getsol(dual_3(e,v))
end-do


forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_4(ct,e,v,t):=getsol(dual_4(e,v,t))
end-do

forall(e in REG_EMP, v in VESSELS, t in TIME)do
sol_dual_40(ct,e,v,t):=getsol(dual_40(e,v,t))
end-do

forall(r in ALL_ROLES)do
sol_dual_7(ct,r):=getsol(dual_7(r))
end-do

forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_8(ct,r,t):= getsol(dual_8(r,t))
end-do

forall(e in GUARANTEED)do
sol_dual_9(ct,e):=getsol(dual_9(e))
end-do

forall(e in GUARANTEED) do
sol_dual_10(ct,e):=getsol(dual_10(e))
end-do


forall(r in ALL_ROLES, t in TIME)do
sol_dual_30(ct,r,t):= getsol(dual_30(r,t))
end-do
```

```
forall(r in ALL_ROLES, t in TIME)do
sol_dual_31(ct,r,t):= getsol(dual_31(r,t))
end-do

forall(e in REG_EMP)do
sol_dual_19(ct,e):=getsol(dual_19(e))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_20(ct,e,t):= getsol(dual_20(e,t))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_23(ct,e,t):= getsol(dual_23(e,t))
end-do

forall(e in REG_EMP, t in TIME)do
sol_dual_21(ct,e,t):= getsol(dual_21(e,t))
end-do

end-if



end-procedure
!-------------------------------------------------------------------
procedure mypreintsol(isheur:boolean,cutoff:real)

declarations
tamam:integer
end-declarations



tamam:=1
```

```
forall( e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)
if(b<=a) then
c:=b
else
c:=a
end-if
end-if
forall(y in 0..c)do
if(c>=1)then
if(sum(r in ALL_ROLES)(getsol(allocate_master(e,r,t+y)))+sum (v in VESSELS
    ) (getsol(depart_master(e,v,t)))>=2)then
sethidden(Rest_new1(e,t,y),false)
tamam:=0
end-if
end-if
end-do
end-do

if(tamam=0)then

rejectintsol

else
writeln("no reject")
ct:=ct+1

forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_iteration(ct,l,r,t):=round(getsol(allocate_master(l,r,t)))
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_iteration(ct,r,v,t):=round(getsol(depart_master(r,v,t)))
end-do
tamam:=1
```

```
end-if

end-procedure

!-------------------------------------------------------------------
procedure evaluate_solution


if(ct=0)then
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_iteration(ct,l,r,t):=allocate(l,r,t)
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_iteration(ct,r,v,t):=depart(r,v,t)
end-do
end-if

forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_dual(l,r,t):=allocate_iteration(ct,l,r,t)
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_dual(r,v,t):=depart_iteration(ct,r,v,t)
end-do



initializations to "bin:shmem:sol"
allocate_dual depart_dual ct
end-initializations

end-procedure


!-------------------------------------------------------------------
function subprob : real
```

```
initializations to "bin:shmem:sol"
allocate_dual depart_dual ct InstanceName allocate_best
end-initializations
run(Dualmodel) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
   sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
   accept allocate_best
end-initializations
end-function

!-------------------------------------------------------------------

function heuristic_solve : real

initializations to "bin:shmem:sol"
allocate_dual depart_dual InstanceName
end-initializations
run(Heuristic) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
allocate_dual depart_dual InstanceName
end-initializations
end-function

!-------------------------------------------------------------------
function cut_manage : boolean
loadcuts(-1,-1)
end-function

!-------------------------------------------------------------
```

```
end-model

!-------------------------------------------------------
(!*****************************************************
Mosel Example Problems
======================

file benders_main.mos
'''''''''''''''''''''
Benders decomposition for solving a simple MIP.
- Master model -

*** ATTENTION: This model will return an error if ***
*** no more than one Xpress licence is available. ***

(c) 2008 Fair Isaac Corporation
author: S. Heipcke, Jan. 2006, rev. Jan. 2013
*****************************************************!)

model "Benders (master model)"
uses "mmxprs", "mmjobs", "mmsystem"



parameters
ALG = 1
BIGM = 100000000

!PARAMETERFILE = "batch-input-parameters.dat"
end-parameters

declarations
InstanceName: string
end-declarations

initializations from "bin:shmem:sol"
```

```
InstanceName
end-initializations


DATAFILE :=InstanceName+"\\Time-Windows - Captains - "+InstanceName+".txt"
OUTPUTFILE_subdual:="Results - SubDual-Benders-mw - "+InstanceName+".txt"


forward procedure save_solution_dualsub


declarations

STEP_0=2                          ! Codes sent to subproblems
STEP_1=3
STEP_2=4
STAT_SOLVED=6                     ! Status codes returned by subproblems
STAT_INFEAS=7
STAT_READY=8

UB:real !upperbound
LB: real !lowerbound
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

sol_obj: real                     ! Objective function value (primal)
```

```
MC: array(range) of linctr      ! Constraints generated by alg.
RM: range                       ! Model indices
cut_type: real
accept: real
stepmod: array(RM) of Model     ! Submodels
end-declarations
!DECLARATION OF PARAMETERS AND DECISION VARIABLES
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
allocate_dual: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer !
   Variable for allocating employee to role during given time period
depart_dual: array(REG_EMP, VESSELS, TIME) of integer
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
allocate_best: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer
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 integer

mwp:linctr
end-declarations

!Reading from txt file
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

!Continue to define parameters and decision variables
```

```
declarations

!Added for recovery problem - detail of current roster, and change
    variable

!solution of dual sub problem
dual_3: array(REG_EMP, VESSELS) of mpvar
dual_4: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar
dual_7: array(ALL_ROLES) of mpvar
dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9: array(GUARANTEED) of mpvar
dual_10: array(GUARANTEED) of mpvar
dual_30: array(ALL_ROLES,TIME) of mpvar
dual_31: array(ALL_ROLES,TIME) of mpvar
dual_40: array(REG_EMP, VESSELS, TIME) of mpvar
dual_19:array(REG_EMP) of real
dual_21:array(REG_EMP, TIME) of real
dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of real
dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of real



dual_3_new: array(REG_EMP, VESSELS) of mpvar
dual_4_new: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar
dual_7_new: array(ALL_ROLES) of mpvar
dual_8_new: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9_new: array(GUARANTEED) of mpvar
dual_10_new: array(GUARANTEED) of mpvar
dual_30_new: array(ALL_ROLES,TIME) of mpvar
dual_31_new: array(ALL_ROLES,TIME) of mpvar
dual_40_new: array(REG_EMP, VESSELS, TIME) of mpvar
dual_19_new:array(REG_EMP) of mpvar
dual_21_new:array(REG_EMP, TIME) of mpvar
dual_20_new:array(REG_EMP, 2..WEEKS_TO_PLAN) of mpvar
dual_23_new:array(REG_EMP,2..WEEKS_TO_PLAN) of mpvar

sol_dual_3: array(range,REG_EMP, VESSELS) of real
```

```
sol_dual_4: array(range,REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of real

                                        ! or takes a non-negative integer
   value for agency crew
sol_dual_7: array(range,ALL_ROLES) of real
sol_dual_8: array(range,ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_9: array(range,GUARANTEED) of real
sol_dual_10: array(range,GUARANTEED) of real
sol_dual_19:array(range,REG_EMP) of real
sol_dual_21:array(range,REG_EMP, TIME) of real
sol_dual_20:array(range,REG_EMP, 2..WEEKS_TO_PLAN) of real
sol_dual_23:array(range,REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_30: array(range,ALL_ROLES,TIME) of real
sol_dual_31: array(range,ALL_ROLES,TIME) of real
sol_dual_40: array(range,REG_EMP, VESSELS, TIME) of real

                                  ! or takes a non-negative integer value
   for agency crew
dummy:linctr
! Objective Value of dual problem
Dual_cost:real
end-declarations

! reading the parameters

declarations
!constraints for dual problem
dual_cons_board: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart: array(REG_EMP, VESSELS, TIME) of linctr        ! =1 if
   employee boards / departs vessel in given time period, 0 otherwise

                                        ! or takes a non-negative integer
   value for agency crew
dual_cons_ag_rboard: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart: array(ALL_ROLES, TIME) of linctr    ! =1 if agency
   crew starts / ends working on a role in given time period, 0 otherwise
dual_cons_undertime: array(GUARANTEED) of linctr
```

```
dual_cons_overtime: array(GUARANTEED) of linctr                    !
   Variables to calculate the amount of under/overtime carried out by
   employee

dual_cons_board_new: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart_new: array(REG_EMP, VESSELS, TIME) of linctr    ! =1 if
   employee boards / departs vessel in given time period, 0 otherwise

                                          ! or takes a non-negative integer
   value for agency crew
dual_cons_ag_rboard_new: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart_new: array(ALL_ROLES, TIME) of linctr       ! =1 if
   agency crew starts / ends working on a role in given time period, 0
   otherwise
dual_cons_undertime_new: array(GUARANTEED) of linctr
dual_cons_overtime_new: array(GUARANTEED) of linctr                      !
   Variables to calculate the amount of under/overtime carried out by
   employee

dual_cons_rest_total:array(REG_EMP, TIME) of linctr
dual_cons_rest_total_new:array(REG_EMP, TIME) of linctr
!constraints for integer problem
All_covered: dynamic array(ALL_ROLES, TIME) of linctr
No_overlap: array(REG_EMP, TIME) of linctr
Work_count: array(REG_EMP, TIME) of linctr
Work_count_start: array(REG_EMP) of linctr
Depart_constr: array(REG_EMP, VESSELS, TIME) of linctr
Rest_vs_work: array(REG_EMP, TIME) of linctr
Rest_new: array(REG_EMP) of linctr
Rest_new1: array(REG_EMP, TIME) of linctr
ct:integer
int_started:integer
allocate_master: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
depart_master: array(REG_EMP, VESSELS, TIME) of mpvar     ! =1 if employee
   boards / departs vessel in given time period, 0 otherwise
Obj: linctr
Dual_first:linctr
```

```
z:mpvar
Primal_cost:linctr
stepprob: array(RM) of mpproblem ! Subproblems
status: array(mpproblem) of integer ! Subproblem status
AG_work_count_start:array(ALL_ROLES) of linctr
AG_work_count:array(ALL_ROLES, TIME) of linctr
Obj_mwp:linctr
Obj_mwp_value:mpvar
Dual_cost_mwp:linctr
allocate_mwp: dynamic array(1..1,ALL_EMP, ALL_ROLES, TIME) of real !
   Variable for allocating employee to role during given time period
depart_mwp: array(1..1, REG_EMP, VESSELS, TIME) of real
allocate: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer
depart: array(REG_EMP, VESSELS, TIME) of integer
end-declarations

initializations from DATAFILE
allocate depart
end-initializations

initializations from "bin:shmem:sol"
allocate_dual depart_dual ct UB Dual_cost allocate_best
end-initializations

writeln("SUB DUAL STARTED")
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_dual(e,r,t))
end-if
end-do

Obj:= (sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40_new(e,v,t))+
sum(e in REG_EMP, v in VESSELS)(dual_3_new(e,v)*(sum(r in ROLES(v))(
   allocate_dual(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4_new(e,v,t)*(
   sum(r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(
   allocate_dual(e,r,(t-1)))))+
```

```
sum(r in ALL_ROLES)(dual_7_new(r)*( allocate_dual("AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8_new(r,t)*(allocate_dual
    ("AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9_new(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
sum(e in GUARANTEED)(dual_10_new(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30_new(r,t)*(1-allocate_dual("AGENCY",
    r,t)))+
sum(r in ALL_ROLES)(dual_31_new(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31_new(r,t)*(1-
    allocate_dual("AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19_new(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20_new(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23_new(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21_new(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart_dual(e,v,t))))))
sethidden(Obj,false)


!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board_new(e,v,1):=dual_3_new(
    e,v)+dual_40_new(e,v,1)<=(board_chng_cost(e,v,1))/(1-(2*cur_board(e,v
    ,1)))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)
    dual_cons_board_new(e,v,t):=dual_4_new(e,v,t)+dual_40_new(e,v,t)<=(
    board_chng_cost(e,v,t))/(1-(2*cur_board(e,v,t)))


!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard_new(r,1):= dual_7_new(r)+
    dual_31_new(r,1)<=(ag_board_chng_cost(r,1)/(1-(2*cur_ag_rboard(r,1))))
```

```
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard_new(r,t)
    := dual_8_new(r,t)+dual_31_new(r,t)<=(ag_board_chng_cost(r,t)/(1-(2*
    cur_ag_rboard(r,t))))

!ag_rdepart

forall(r in ALL_ROLES) dual_cons_ag_rdepart_new(r,1):= -dual_7_new(r)+
    dual_30_new(r,1)<=(ag_depart_chng_cost(r,1)/(1-(2*cur_ag_rdepart(r,1)))
    )

forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart_new(r,t)
     := -dual_8_new(r,t)+dual_30_new(r,t)<=(ag_depart_chng_cost(r,t)/(1-(2*
    cur_ag_rdepart(r,t))))



!undertime
forall(e in GUARANTEED) dual_cons_undertime_new(e):= dual_9_new(e)<=
    under_rate(e)

!overtime
forall(e in GUARANTEED) dual_cons_overtime_new(e):= dual_10_new(e)<=
    over_rate(e)


!rest_total

forall(e in REG_EMP) dual_cons_rest_total_new(e,1):= dual_19_new(e)-
    dual_20_new(e,2)+dual_21_new(e,1)-dual_23_new(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total_new(e,t)
    := dual_20_new(e,t)-dual_20_new(e,t+1)+dual_21_new(e,t)-dual_23_new(e,t
    +1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_rest_total_new(e,t):= dual_20_new(e,t)+dual_21_new(e,t)<=0


forall(e in REG_EMP, v in VESSELS) dual_3_new(e,v)>=0
```

```
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4_new(e,v,t
    )>=0
forall(r in ALL_ROLES) dual_7_new(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8_new(r,t) is_free
forall(e in GUARANTEED) dual_9_new(e)>=0
forall(e in GUARANTEED) dual_10_new(e)>=0
forall(e in REG_EMP) dual_19_new(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20_new(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21_new(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23_new(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30_new(r,t)<=0
forall(r in ALL_ROLES, t in TIME) dual_31_new(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40_new(e,v,t)<=0

!fopen(OUTPUTFILE_subdual, F_OUTPUT)
!setparam("XPRS_verbose",true)
maximize(XPRS_BAR, Obj)

cut_type:=1

if(getprobstat=XPRS_UNB) then
write("Dual Unbounded ")
cut_type:=0
writeln("Cut_type",cut_type)

Obj_mwp:=Obj_mwp_value=0

Dual_cost_mwp:=(sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40_new(e,v
    ,t))+
sum(e in REG_EMP, v in VESSELS)(dual_3_new(e,v)*(sum(r in ROLES(v))(
    allocate_dual(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4_new(e,v,t)*(
    sum(r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(
    allocate_dual(e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7_new(r)*( allocate_dual("AGENCY",r,1) -
    ag_starting(r)))+
```

```
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8_new(r,t)*(allocate_dual
    ("AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9_new(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
sum(e in GUARANTEED)(dual_10_new(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30_new(r,t)*(1-allocate_dual("AGENCY",
    r,t)))+
sum(r in ALL_ROLES)(dual_31_new(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31_new(r,t)*(1-
    allocate_dual("AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19_new(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20_new(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23_new(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21_new(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart_dual(e,v,t))))))=1


!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board_new(e,v,1):=dual_3_new(
    e,v)+dual_40_new(e,v,1)<=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)
    dual_cons_board_new(e,v,t):=dual_4_new(e,v,t)+dual_40_new(e,v,t)<=0


!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard_new(r,1):= dual_7_new(r)+
    dual_31_new(r,1)<=0
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard_new(r,t)
    := dual_8_new(r,t)+dual_31_new(r,t)<=0

!ag_rdepart
```

```
forall(r in ALL_ROLES) dual_cons_ag_rdepart_new(r,1):= -dual_7_new(r)+
    dual_30_new(r,1)<=0

forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart_new(r,t)
     := -dual_8_new(r,t)+dual_30_new(r,t)<=0



!undertime
forall(e in GUARANTEED) dual_cons_undertime_new(e):= dual_9_new(e)<=0

!overtime
forall(e in GUARANTEED) dual_cons_overtime_new(e):= dual_10_new(e)<=0

!rest_total

forall(e in REG_EMP) dual_cons_rest_total_new(e,1):= dual_19_new(e)-
    dual_20_new(e,2)+dual_21_new(e,1)-dual_23_new(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total_new(e,t)
    := dual_20_new(e,t)-dual_20_new(e,t+1)+dual_21_new(e,t)-dual_23_new(e,t
    +1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_rest_total_new(e,t):= dual_20_new(e,t)+dual_21_new(e,t)<=0

forall(e in REG_EMP, v in VESSELS) dual_3_new(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4_new(e,v,t
    )>=0
forall(r in ALL_ROLES) dual_7_new(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8_new(r,t) is_free
forall(e in GUARANTEED) dual_9_new(e)>=0
forall(e in GUARANTEED) dual_10_new(e)>=0
forall(e in REG_EMP) dual_19_new(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20_new(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21_new(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23_new(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30_new(r,t)<=0
forall(r in ALL_ROLES, t in TIME) dual_31_new(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40_new(e,v,t)<=0
```

```
!setparam("XPRS_verbose",true)
maximize(XPRS_BAR, Obj_mwp_value)
end-if

writeln("Cut_type",cut_type)

if(cut_type=1) then

Dual_cost:=getobjval

writeln("Cut_type",cut_type)
writeln("Dual_cost: ",Dual_cost)
writeln("UB_once",UB)

if (UB>= (Dual_cost+
sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*(
    cur_board(e,v,t)/((2*cur_board(e,v,t))-1))))+
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e)))+
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
    allocate_dual(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))))
    +
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
    depart_dual(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t)))))))
then

UB:=(Dual_cost+
sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*(
    cur_board(e,v,t)/((2*cur_board(e,v,t))-1))))+
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e)))+
```

```
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
   allocate_dual(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))))
   +
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
   depart_dual(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t))))))

accept:=1


writeln("UB_sonra: ",UB)
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_best(l,r,t):=allocate_dual(l,r,t)
end-do

else

UB:=UB
writeln("UB_noimprove: ",UB)
accept:=0
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_best(l,r,t):=allocate_best(l,r,t)
end-do
end-if


save_solution_dualsub


y:=1
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) allocate_mwp(y,e,r,t):=(
   allocate(e,r,t)*0.3)+(allocate_dual(e,r,t)*0.7)
forall(e in REG_EMP, v in VESSELS, t in TIME) depart_mwp(y,e,v,t):=(depart
   (e,v,t)*0.3)+(depart_dual(e,v,t)*0.7)



Obj:= (sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40_new(e,v,t))+
```

```
sum(e in REG_EMP, v in VESSELS)(dual_3_new(e,v)*(sum(r in ROLES(v))(
    allocate_mwp(y,e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4_new(e,v,t)*(
    sum(r in ROLES(v))(allocate_mwp(y,e,r,t)) - sum(r in ROLES(v))(
    allocate_mwp(y,e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7_new(r)*( allocate_mwp(y,"AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8_new(r,t)*(allocate_mwp(y
    ,"AGENCY",r,t) - allocate_mwp(y,"AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9_new(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_mwp(y,e,r,t)))))+
sum(e in GUARANTEED)(dual_10_new(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_mwp(y,e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30_new(r,t)*(1-allocate_mwp(y,"AGENCY
    ",r,t)))+
sum(r in ALL_ROLES)(dual_31_new(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31_new(r,t)*(1-
    allocate_mwp(y,"AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19_new(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_mwp(y,e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20_new(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_mwp(y,e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23_new(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_mwp(y,e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21_new(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart_mwp(y,e,v,t))))))
sethidden(Obj,false)

!mwp constraints
mwp:=(sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40_new(e,v,t))+
sum(e in REG_EMP, v in VESSELS)(dual_3_new(e,v)*(sum(r in ROLES(v))(
    allocate_dual(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4_new(e,v,t)*(
    sum(r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(
    allocate_dual(e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7_new(r)*( allocate_dual("AGENCY",r,1) -
    ag_starting(r)))+
```

```
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8_new(r,t)*(allocate_dual
    ("AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9_new(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
sum(e in GUARANTEED)(dual_10_new(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30_new(r,t)*(1-allocate_dual("AGENCY",
    r,t)))+
sum(r in ALL_ROLES)(dual_31_new(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31_new(r,t)*(1-
    allocate_dual("AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19_new(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20_new(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23_new(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21_new(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart_dual(e,v,t))))))=Dual_cost



!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board_new(e,v,1):=dual_3_new(
    e,v)+dual_40_new(e,v,1)<=(board_chng_cost(e,v,1))/(1-(2*cur_board(e,v
    ,1)))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)
    dual_cons_board_new(e,v,t):=dual_4_new(e,v,t)+dual_40_new(e,v,t)<=(
    board_chng_cost(e,v,t))/(1-(2*cur_board(e,v,t)))



!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard_new(r,1):= dual_7_new(r)+
    dual_31_new(r,1)<=(ag_board_chng_cost(r,1)/(1-(2*cur_ag_rboard(r,1))))
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard_new(r,t)
    := dual_8_new(r,t)+dual_31_new(r,t)<=(ag_board_chng_cost(r,t)/(1-(2*
```

```
    cur_ag_rboard(r,t))))

!ag_rdepart

forall(r in ALL_ROLES) dual_cons_ag_rdepart_new(r,1):= -dual_7_new(r)+
    dual_30_new(r,1)<=(ag_depart_chng_cost(r,1)/(1-(2*cur_ag_rdepart(r,1)))
    )

forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart_new(r,t)
     := -dual_8_new(r,t)+dual_30_new(r,t)<=(ag_depart_chng_cost(r,t)/(1-(2*
    cur_ag_rdepart(r,t))))



!undertime
forall(e in GUARANTEED) dual_cons_undertime_new(e):= dual_9_new(e)<=
    under_rate(e)

!overtime
forall(e in GUARANTEED) dual_cons_overtime_new(e):= dual_10_new(e)<=
    over_rate(e)


!rest_total

forall(e in REG_EMP) dual_cons_rest_total_new(e,1):= dual_19_new(e)-
    dual_20_new(e,2)+dual_21_new(e,1)-dual_23_new(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total_new(e,t)
    := dual_20_new(e,t)-dual_20_new(e,t+1)+dual_21_new(e,t)-dual_23_new(e,t
    +1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_rest_total_new(e,t):= dual_20_new(e,t)+dual_21_new(e,t)<=0


forall(e in REG_EMP, v in VESSELS) dual_3_new(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4_new(e,v,t
    )>=0
```

```
forall(r in ALL_ROLES) dual_7_new(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8_new(r,t) is_free
forall(e in GUARANTEED) dual_9_new(e)>=0
forall(e in GUARANTEED) dual_10_new(e)>=0
forall(e in REG_EMP) dual_19_new(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20_new(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21_new(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23_new(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30_new(r,t)<=0
forall(r in ALL_ROLES, t in TIME) dual_31_new(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40_new(e,v,t)<=0


! setparam("XPRS_verbose",true)
maximize(XPRS_BAR, Obj)

if(getprobstat=XPRS_UNB) then
write("MWP Unbounded ")


end-if

if(getprobstat = XPRS_INF) then
writeln("MWP is infeasible")

end-if

if(getprobstat=XPRS_OPT) then
write("MWP Optimal ")
save_solution_dualsub
end-if



else

Dual_cost:=getobjval
```

```
UB:=UB

writeln("Cut_type",cut_type)
writeln("Dual_unb: ",Dual_cost)
writeln("UB_unb: ",UB)

forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_best(l,r,t):=allocate_best(l,r,t)
end-do
save_solution_dualsub
end-if


(!  writeln("allocate: [")

forall(e in ALL_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) write(allocate_best(e,r,t),"\t")
end-do
write("\n")
end-do

write("]\n")
!)
!  fclose(F_OUTPUT)
writeln("UB: ",UB)
fflush

!-------------------------------------------------------------------

procedure save_solution_dualsub
! Store values of u and x


forall(e in REG_EMP, v in VESSELS)do
sol_dual_3(ct,e,v):=getsol(dual_3_new(e,v))
end-do
```

```
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_4(ct,e,v,t):=getsol(dual_4_new(e,v,t))
end-do

forall(e in REG_EMP, v in VESSELS, t in TIME)do
sol_dual_40(ct,e,v,t):=getsol(dual_40_new(e,v,t))
end-do

forall(r in ALL_ROLES)do
sol_dual_7(ct,r):=getsol(dual_7_new(r))
end-do

forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_8(ct,r,t):= getsol(dual_8_new(r,t))
end-do

forall(e in GUARANTEED)do
sol_dual_9(ct,e):=getsol(dual_9_new(e))
end-do

forall(e in GUARANTEED) do
sol_dual_10(ct,e):=getsol(dual_10_new(e))
end-do



forall(r in ALL_ROLES, t in TIME)do
sol_dual_30(ct,r,t):= getsol(dual_30_new(r,t))
end-do

forall(r in ALL_ROLES, t in TIME)do
sol_dual_31(ct,r,t):= getsol(dual_31_new(r,t))
end-do



forall(e in REG_EMP)do
```

```
sol_dual_19(ct,e):=getsol(dual_19_new(e))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_20(ct,e,t):= getsol(dual_20_new(e,t))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_23(ct,e,t):= getsol(dual_23_new(e,t))
end-do

forall(e in REG_EMP, t in TIME)do
sol_dual_21(ct,e,t):= getsol(dual_21_new(e,t))
end-do


initializations to "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
   sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 UB Dual_cost cut_type
   accept allocate_best
end-initializations

end-procedure


end-model
```

B.1.4 Modern Benders Decomposition Algorithm

Hybrid method of Modern Benders Decomposition method with heuristic is also coded in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) and the related code is given in this section. There are three functions in this section, one is about to maintain the main algorithm and others are dual sub problem and heuristic implementations.

```
model "Benders (master model)"
uses "mmxprs", "mmjobs", "mmsystem"
parameters
ALG = 1
BIGM = 100000000

DATE = "11-09-17"
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"
LOGFILE := InstanceName+"\\Logfile-1hr -Benders-onetree-heur-mw "+
    InstanceName+" - "+DATE+".txt"
SUMMARYFILE := "Results -Benders 1 hr- one tree-heur-mw"+DATE+".txt"

forward procedure solve_cont
forward procedure save_solution_dual
forward procedure evaluate_solution
forward function subprob : real
!forward function paretoprob : real
forward function heuristic_solve : real
forward procedure solve_primal_int(ct: integer)

forward procedure define_dualprob(prob:mpproblem)
forward function solve_dualprob(prob:mpproblem): real
forward procedure define_intprob(prob:mpproblem)
forward function solve_intprob(prob:mpproblem, ct:integer): real
```

```
declarations

STEP_0=2                          ! Codes sent to subproblems
STEP_1=3
STEP_2=4
STAT_SOLVED=6                     ! Status codes returned by subproblems
STAT_INFEAS=7
STAT_READY=8

UB:real !upperbound
LB: real !lowerbound
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
limited_time:real
exp_worktime, g_weeks, under_rate, over_rate: array(GUARANTEED) of real  !
    Data relating to over/undertime payments for employees
mip_x:integer
sol_obj: real                     ! Objective function value (primal)
MC: array(range) of linctr        ! Constraints generated by alg.
cuts_added:array(range) of integer
RM: range                         ! Model indices
cut_type:array(range) of real
accept: real
Dualmodel: Model
Heuristic: Model
!Pareto:Model
Primal_cost:linctr
mybasis:basis
```

```
stepprob: array(RM) of mpproblem
best_bound:real
z_best:array (range) of real
end-declarations
!DECLARATION OF PARAMETERS AND DECISION VARIABLES
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
allocate: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer ! Variable
   for allocating employee to role during given time period
allocate_dual: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer !
   Variable for allocating employee to role during given time period
depart_dual: array(REG_EMP, VESSELS, TIME) of integer
depart: array(REG_EMP, VESSELS, TIME) of integer
allocate_best: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer
allocate_iteration: dynamic array(range,ALL_EMP, ALL_ROLES, TIME) of
   integer ! Variable for allocating employee to role during given time
   period
depart_iteration: array(range,REG_EMP, VESSELS, TIME) of integer


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 integer


end-declarations

!Reading from txt file
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

!Continue to define parameters and decision variables
declarations

!Added for recovery problem - detail of current roster, and change
   variable

!solution of dual sub problem
dual_3: array(REG_EMP, VESSELS) of mpvar
dual_4: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar
dual_7: array(ALL_ROLES) of mpvar
dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9: array(GUARANTEED) of mpvar
dual_10: array(GUARANTEED) of mpvar
dual_30: array(ALL_ROLES,TIME) of mpvar
dual_31: array(ALL_ROLES,TIME) of mpvar
dual_40: array(REG_EMP, VESSELS, TIME) of mpvar
dual_19:array(REG_EMP) of mpvar
dual_21:array(REG_EMP, TIME) of mpvar
dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of mpvar
dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of mpvar



dual_3_new: array(REG_EMP, VESSELS) of mpvar
dual_4_new: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar
dual_7_new: array(ALL_ROLES) of mpvar
dual_8_new: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9_new: array(GUARANTEED) of mpvar
dual_10_new: array(GUARANTEED) of mpvar
dual_30_new: array(ALL_ROLES,TIME) of mpvar
```

```
dual_31_new: array(ALL_ROLES,TIME) of mpvar
dual_40_new: array(REG_EMP, VESSELS, TIME) of mpvar
dual_19_new:array(REG_EMP) of mpvar
dual_21_new:array(REG_EMP, TIME) of mpvar
dual_20_new:array(REG_EMP, 2..WEEKS_TO_PLAN) of mpvar
dual_23_new:array(REG_EMP,2..WEEKS_TO_PLAN) of mpvar

sol_dual_3: array(range,REG_EMP, VESSELS) of real
sol_dual_4: array(range,REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of real

                                        ! or takes a non-negative integer
   value for agency crew
sol_dual_7: array(range,ALL_ROLES) of real
sol_dual_8: array(range,ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_9: array(range,GUARANTEED) of real
sol_dual_10: array(range,GUARANTEED) of real
sol_dual_19:array(range,REG_EMP) of real
sol_dual_21:array(range,REG_EMP, TIME) of real
sol_dual_20:array(range,REG_EMP, 2..WEEKS_TO_PLAN) of real
sol_dual_23:array(range,REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_30: array(range,ALL_ROLES,TIME) of real
sol_dual_31: array(range,ALL_ROLES,TIME) of real
sol_dual_40: array(range,REG_EMP, VESSELS, TIME) of real
dummy:linctr
dummy2:array(range) of linctr
! Objective Value of dual problem
Dual_cost:real
end-declarations

! reading the parameters
initializations from DATAFILE
allocate depart
end-initializations
declarations
!constraints for dual problem
dual_cons_board: array(REG_EMP, VESSELS, TIME) of linctr
```

```
dual_cons_depart: array(REG_EMP, VESSELS, TIME) of linctr        ! =1 if
   employee boards / departs vessel in given time period, 0 otherwise

                                        ! or takes a non-negative integer
   value for agency crew
dual_cons_ag_rboard: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart: array(ALL_ROLES, TIME) of linctr    ! =1 if agency
   crew starts / ends working on a role in given time period, 0 otherwise
dual_cons_undertime: array(GUARANTEED) of linctr
dual_cons_overtime: array(GUARANTEED) of linctr                   !
   Variables to calculate the amount of under/overtime carried out by
   employee

dual_cons_board_new: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart_new: array(REG_EMP, VESSELS, TIME) of linctr    ! =1 if
   employee boards / departs vessel in given time period, 0 otherwise

                                        ! or takes a non-negative integer
   value for agency crew
dual_cons_ag_rboard_new: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart_new: array(ALL_ROLES, TIME) of linctr       ! =1 if
   agency crew starts / ends working on a role in given time period, 0
   otherwise
dual_cons_undertime_new: array(GUARANTEED) of linctr
dual_cons_overtime_new: array(GUARANTEED) of linctr                      !
   Variables to calculate the amount of under/overtime carried out by
   employee
dual_cons_rest_total:array(REG_EMP, TIME) of linctr

!constraints for integer problem
All_covered: dynamic array(ALL_ROLES, TIME) of linctr
No_overlap: array(REG_EMP, TIME) of linctr
Work_count: array(REG_EMP, TIME) of linctr
Work_count_start: array(REG_EMP) of linctr
Depart_constr: array(REG_EMP, VESSELS, TIME) of linctr
Rest_vs_work: array(REG_EMP, TIME) of linctr
!Rest_new: array(REG_EMP) of linctr
```

```
!Rest_new1: array(REG_EMP, TIME) of linctr
Rest_new: array(REG_EMP,ALL_ROLES,range) of linctr
Rest_new1: array(REG_EMP, 1..WEEKS_TO_PLAN-1,range) of linctr
Depart_linking: array(REG_EMP, TIME) of linctr
ct:integer
int_started:integer
allocate_master: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
depart_master: array(REG_EMP, VESSELS, TIME) of mpvar     ! =1 if employee
   boards / departs vessel in given time period, 0 otherwise
Obj: linctr
Dual_first:linctr
z:mpvar
status: array(mpproblem) of integer ! Subproblem status
AG_work_count_start:array(ALL_ROLES) of linctr
AG_work_count:array(ALL_ROLES, TIME) of linctr
extra:array(REG_EMP) of linctr
z1:array(REG_EMP, VESSELS) of mpvar
z2:array(ALL_ROLES)of mpvar
z4:array(REG_EMP)of mpvar
z3:array(GUARANTEED)of mpvar
MC10: array(range,REG_EMP, VESSELS) of linctr     ! Constraints generated
   by alg.
MC20: array(range,ALL_ROLES) of linctr
MC30: array(range,GUARANTEED) of linctr
MC40: array(range,REG_EMP) of linctr
Obj_mwp: linctr
Obj_mwp_value:mpvar
UB1:real
Cut:dynamic array(range)of linctr
type:dynamic array(range)of integer
cutid: array(range) of integer
finish:real
begin:real
end-declarations
```

```
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_master(e,r,t))
end-if
end-do

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_dual(e,r,t))
end-if
end-do

forall(a in 1..ct,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_iteration(a,e,r
    ,t))
end-if
end-do

!sharing data info with submodels
ct:=0
UB_first:=10000000000
x:=1
best_bound:=-10000000000
z_best(0):= best_bound
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_iteration(0,l,r,t):=allocate(l,r,t)
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_iteration(0,r,v,t):=depart(r,v,t)
end-do
```

```
create(stepprob(1)); define_intprob(stepprob(1))

create(stepprob(2)); define_dualprob(stepprob(2))


procedure solve_primal_int(ct: integer)
sol_obj:= solve_intprob(stepprob(1), ct)
end-procedure

procedure solve_cont
! Start the problem solving
res:= solve_dualprob(stepprob(2))

end-procedure




fixed:=(sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)
    *(cur_board(e,v,t)/((2*cur_board(e,v,t))-1))))+
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e))))

!-------------------------------------------------------------
! Define the dual problem
procedure define_dualprob(prob:mpproblem)
!board
with prob do

!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board(e,v,1):=dual_3(e,v)+
    dual_40(e,v,1)<=(board_chng_cost(e,v,1))/(1-(2*cur_board(e,v,1)))
```

```
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_cons_board(
    e,v,t):=dual_4(e,v,t)+dual_40(e,v,t)<=(board_chng_cost(e,v,t))/(1-(2*
    cur_board(e,v,t)))
forall(e in REG_EMP, v in VESSELS, t in TIME)sethidden(dual_cons_board(e,v
    ,t),false)

!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard(r,1):= dual_7(r)+dual_31(r,1)
    <=(ag_board_chng_cost(r,1)/(1-(2*cur_ag_rboard(r,1))))
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard(r,t) :=
    dual_8(r,t)+dual_31(r,t)<=(ag_board_chng_cost(r,t)/(1-(2*cur_ag_rboard(
    r,t))))
forall(r in ALL_ROLES,t in TIME) sethidden( dual_cons_ag_rboard(r,t),false
    )
!ag_rdepart

forall(r in ALL_ROLES) dual_cons_ag_rdepart(r,1):= -dual_7(r)+dual_30(r,1)
    <=(ag_depart_chng_cost(r,1)/(1-(2*cur_ag_rdepart(r,1))))

forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart(r,t) :=
    -dual_8(r,t)+dual_30(r,t)<=(ag_depart_chng_cost(r,t)/(1-(2*
    cur_ag_rdepart(r,t))))
forall(r in ALL_ROLES,t in TIME) sethidden( dual_cons_ag_rdepart(r,t),
    false)



!undertime
forall(e in GUARANTEED) dual_cons_undertime(e):= dual_9(e)<=under_rate(e)
forall(e in GUARANTEED) sethidden( dual_cons_undertime(e),false)
!overtime
forall(e in GUARANTEED) dual_cons_overtime(e):= dual_10(e)<=over_rate(e)
forall(e in GUARANTEED) sethidden( dual_cons_overtime(e),false)

!rest_total

forall(e in REG_EMP) dual_cons_rest_total(e,1):= dual_19(e)-dual_20(e,2)+
    dual_21(e,1)-dual_23(e,2)<=0
```

```
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total(e,t):=
   dual_20(e,t)-dual_20(e,t+1)+dual_21(e,t)-dual_23(e,t+1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
   dual_cons_rest_total(e,t):= dual_20(e,t)+dual_21(e,t)<=0
forall(e in REG_EMP,t in TIME) sethidden(dual_cons_rest_total(e,t),false)


forall(e in REG_EMP, v in VESSELS) dual_3(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4(e,v,t)>=0
forall(r in ALL_ROLES) dual_7(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8(r,t) is_free
forall(e in GUARANTEED) dual_9(e)>=0
forall(e in GUARANTEED) dual_10(e)>=0
forall(e in REG_EMP) dual_19(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30(r,t)<=0
forall(r in ALL_ROLES, t in TIME) dual_31(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40(e,v,t)<=0


status(prob):= STAT_READY
end-do
end-procedure

! Process dual solution data
procedure save_dualsolution(prob:mpproblem)
if(ct=0)then
forall(e in REG_EMP, v in VESSELS)do
sol_dual_3(ct,e,v):=getsol(dual_3(e,v))
end-do


forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_4(ct,e,v,t):=getsol(dual_4(e,v,t))
end-do
```

```
forall(e in REG_EMP, v in VESSELS, t in TIME)do
sol_dual_40(ct,e,v,t):=getsol(dual_40(e,v,t))
end-do

forall(r in ALL_ROLES)do
sol_dual_7(ct,r):=getsol(dual_7(r))
end-do

forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_8(ct,r,t):= getsol(dual_8(r,t))
end-do

forall(e in GUARANTEED)do
sol_dual_9(ct,e):=getsol(dual_9(e))
end-do

forall(e in GUARANTEED) do
sol_dual_10(ct,e):=getsol(dual_10(e))
end-do



forall(r in ALL_ROLES, t in TIME)do
sol_dual_30(ct,r,t):= getsol(dual_30(r,t))
end-do

forall(r in ALL_ROLES, t in TIME)do
sol_dual_31(ct,r,t):= getsol(dual_31(r,t))
end-do

forall(e in REG_EMP)do
sol_dual_19(ct,e):=getsol(dual_19(e))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_20(ct,e,t):= getsol(dual_20(e,t))
end-do
```

```
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_23(ct,e,t):= getsol(dual_23(e,t))
end-do

forall(e in REG_EMP, t in TIME)do
sol_dual_21(ct,e,t):= getsol(dual_21(e,t))
end-do

end-if
status(prob):= STAT_SOLVED

fflush
end-procedure

! (Re)solve the dual problem
function solve_dualprob(prob:mpproblem): real
with prob do
status(prob):= STAT_READY

Dual_first:=(sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40(e,v,t))+
sum(e in REG_EMP, v in VESSELS)(dual_3(e,v)*(sum(r in ROLES(v))(allocate(e
    ,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4(e,v,t)*(sum(
    r in ROLES(v))(allocate(e,r,t)) - sum(r in ROLES(v))(allocate(e,r,(t-1)
    ))))+
sum(r in ALL_ROLES)(dual_7(r)*( allocate("AGENCY",r,1) - ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8(r,t)*(allocate("AGENCY",
    r,t) - allocate("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate(e,r,t)))))+
sum(e in GUARANTEED)(dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES, t
    in TIME)(allocate(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30(r,t)*(1-allocate("AGENCY",r,t)))+
sum(r in ALL_ROLES)(dual_31(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31(r,t)*(1-allocate("
    AGENCY",r,(t-1))))+
```

```
sum(e in REG_EMP)(dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23(e,t)*(- min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart(e,v,t))))))
sethidden(Dual_first,false)



if (ct=0) then               ! Produce an initial solution for the
! dual problem using a dummy objective
maximize(XPRS_BAR,Dual_first)

if(getprobstat = XPRS_INF) then
writeln("Problem is infeasible")
send(STAT_INFEAS,0)           ! Problem is infeasible
else
write("**** Start solution: ")
write("**** Problem is not infeasible: ")
if(getprobstat = XPRS_UNB) then
writeln("Problem is unbounded")

cut_type(ct):=0
writeln("Cut_type",cut_type(ct))

Obj_mwp:=Obj_mwp_value=0

Dual_cost_mwp:=(sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40(e,v,t))
    +
sum(e in REG_EMP, v in VESSELS)(dual_3(e,v)*(sum(r in ROLES(v))(allocate(e
    ,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4(e,v,t)*(sum(
    r in ROLES(v))(allocate(e,r,t)) - sum(r in ROLES(v))(allocate(e,r,(t-1)
    ))))+
```

```
sum(r in ALL_ROLES)(dual_7(r)*( allocate("AGENCY",r,1) - ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8(r,t)*(allocate("AGENCY",
    r,t) - allocate("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9(e)*(g_weeks(e) - (exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate(e,r,t)))))+
sum(e in GUARANTEED)(dual_10(e)*((exp_worktime(e) + sum(r in ALL_ROLES, t
    in TIME)(allocate(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30(r,t)*(1-allocate("AGENCY",r,t)))+
sum(r in ALL_ROLES)(dual_31(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31(r,t)*(1-allocate("
    AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23(e,t)*(- min_rest(e)*(1-(
    sum(r in ALL_ROLES)(allocate(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart(e,v,t))))))=1


!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board(e,v,1):=dual_3(e,v)+
    dual_40(e,v,1)<=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_cons_board(
    e,v,t):=dual_4(e,v,t)+dual_40(e,v,t)<=0

!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard(r,1):= dual_7(r)+dual_31(r,1)
    <=0
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard(r,t) :=
    dual_8(r,t)+dual_31(r,t)<=0
!ag_rdepart

forall(r in ALL_ROLES) dual_cons_ag_rdepart(r,1):= -dual_7(r)+dual_30(r,1)
    <=0
```

```
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart(r,t) :=
    -dual_8(r,t)+dual_30(r,t)<=0


!undertime
forall(e in GUARANTEED) dual_cons_undertime(e):= dual_9(e)<=0

!overtime
forall(e in GUARANTEED) dual_cons_overtime(e):= dual_10(e)<=0

!rest_total

forall(e in REG_EMP) dual_cons_rest_total(e,1):= dual_19(e)-dual_20(e,2)+
    dual_21(e,1)-dual_23(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total(e,t):=
    dual_20(e,t)-dual_20(e,t+1)+dual_21(e,t)-dual_23(e,t+1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_rest_total(e,t):= dual_20(e,t)+dual_21(e,t)<=0


forall(e in REG_EMP, v in VESSELS) dual_3(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4(e,v,t)>=0
forall(r in ALL_ROLES) dual_7(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8(r,t) is_free
forall(e in GUARANTEED) dual_9(e)>=0
forall(e in GUARANTEED) dual_10(e)>=0
forall(e in REG_EMP) dual_19(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30(r,t)<=0
forall(r in ALL_ROLES, t in TIME) dual_31(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40(e,v,t)<=0

maximize(XPRS_BAR, Obj_mwp_value)

writeln("UB_first: +infinity")
```

```
writeln("UB_first:", UB_first)
UB:=UB_first
accept:=0
Dual_cost:=UB_first
else
writeln("Problem is feasible")
cut_type(ct):=1
writeln("Cut_type:",cut_type(ct))
accept:=1
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_best(l,r,t):=allocate(l,r,t)
end-do
Dual_cost:=getobjval

(! UB:=(Dual_cost+
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
   allocate(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))))+
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
   depart(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t)))))+fixed)
!)
UB:=UB_first
writeln("UB_first:", UB)

end-if

writeln("accept: ", accept)
save_dualsolution(prob)
UB1:=(Dual_cost+
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
   allocate(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))))+
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
   depart(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t)))))+fixed)
writeln("UB1: ",UB1)
writeln("Dual_cost_first: ",Dual_cost)

BigM:= 0
```

```
end-if
end-if

end-do
end-function

!-------------------------------------------------------------

solve_cont

!-----------------------------------1st iteration of dual finished
   ------------------------------------------------------
initializations to "bin:shmem:sol"
InstanceName sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
   sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
   accept allocate_best
end-initializations


!-----------------------------------1st iteration of dual finished
   ------------------------------------------------------

! Define the integer problem
procedure define_intprob(prob:mpproblem)
with prob do

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_master(e,r,t))
end-if
end-do



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_master(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_master(e,r,t) <= 1


forall(e in REG_EMP, v in VESSELS) Depart_constr(e,v,1) := depart_master(e
    ,v,1) >= starting(e,v) - sum(r in ROLES(v))(allocate_master(e,r,1))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) Depart_constr(e,
    v,t) := depart_master(e,v,t) >= sum(r in ROLES(v))(allocate_master(e,r
    ,(t-1))) - sum(r in ROLES(v))(allocate_master(e,r,t))


!-------------depart linking-------------------------------------
forall(e in REG_EMP, t in TIME) Depart_linking(e,t):=sum(v in VESSELS)
    depart_master(e,v,t)+sum(r in ALL_ROLES) allocate_master(e,r,t)<=1
!-------------depart linking-------------------------------------


forall(e in REG_EMP|work_zero(e)+WEEKS_TO_PLAN>max_work(e))
    Work_count_start(e) := max_work(e) >= work_zero(e) + sum(r in ALL_ROLES
    , t in 0..max_work(e)-work_zero(e))(allocate_master(e,r,t+1))
!forall(e in REG_EMP| work_zero(e)+WEEKS_TO_PLAN>max_work(e))sethidden(
    Work_count_start(e) ,true)
!master11
forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-max_work(e)| work_zero(e)+
    WEEKS_TO_PLAN>max_work(e)) Work_count(e,t) := max_work(e) >= sum(r in
    ALL_ROLES, k in 0..max_work(e))(allocate_master(e,r,t+k))
!forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-max_work(e)| work_zero(e)+
    WEEKS_TO_PLAN>max_work(e))sethidden(Work_count(e,t) ,true)

!master12

!forall(e in REG_EMP| max_work(e)<=WEEKS_TO_PLAN) extra(e):=sum(r in
    ALL_ROLES,t in 1..max_work(e)+1, y in 0..WEEKS_TO_PLAN-max_work(e)-1)
```

```
   allocate_master(e,r,t+y) <= max_work(e)*(WEEKS_TO_PLAN-max_work(e))
!master12_ekstra

forall(r in ALL_ROLES|ag_work_zero(r)+WEEKS_TO_PLAN>ag_max_work(r))
   AG_work_count_start(r) := ag_max_work(r) >= ag_work_zero(r) + sum( t in
    0..ag_max_work(r)-ag_work_zero(r))(allocate_master("AGENCY",r,t+1))
!forall(r in ALL_ROLES|ag_work_zero(r)+WEEKS_TO_PLAN>ag_max_work(r))
   sethidden(AG_work_count_start(r),true)
!master13

forall(r in ALL_ROLES, t in 1..WEEKS_TO_PLAN-ag_max_work(r)|ag_work_zero(r
   )+WEEKS_TO_PLAN>ag_max_work(r)) AG_work_count(r,t) := ag_max_work(r) >=
    sum( k in 0..ag_max_work(r))(allocate_master("AGENCY",r,t+k))
!forall(r in ALL_ROLES, t in 1..WEEKS_TO_PLAN-ag_max_work(r)|ag_work_zero(
   r)+WEEKS_TO_PLAN>ag_max_work(r)) sethidden(AG_work_count(r,t) ,true)
!master14

forall(e in REG_EMP) Rest_vs_work(e,1) := min_rest(e)*(1-(sum(r in
   ALL_ROLES)(allocate_master(e,r,1)))) >= rest_zero(e)



! forall( e in REG_EMP | rest_zero(e)>=1) Rest_new(e):= sum(r in ALL_ROLES
   ,t in 1..rest_zero(e)) allocate_master(e,r,t)=0
forall( e in REG_EMP ,r in ALL_ROLES,t in 1..rest_zero(e)) do
if(rest_zero(e)>=1)then
Rest_new(e,r,t):= allocate_master(e,r,t)=0
end-if
end-do

forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)
if(b<=a) then
c:=b
else
c:=a
```

```
end-if
end-if
Rest_new2(e,t) := sum(r in ALL_ROLES, y in 0..c)(allocate_master(e,r,t+y))
    <=c*(1-sum (v in VESSELS) depart_master(e,v,t))
sethidden(Rest_new2(e,t),true)
end-do

forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)
if(b<=a) then
c:=b
else
c:=a
end-if
end-if
forall(y in 0..c)do
if(c>=1)then
Rest_new1(e,t,y) := sum(r in ALL_ROLES)(allocate_master(e,r,t+y))+sum (v
    in VESSELS) depart_master(e,v,t)<=1
sethidden(Rest_new1(e,t,y),false)
end-if
end-do
end-do




forall(e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(allocate_master(e,
    r,t))) allocate_master(e,r,t) is_binary
forall(e in REG_EMP, v in VESSELS, t in TIME) depart_master(e,v,t)
    is_binary

z is_free
forall(e in REG_EMP, v in VESSELS) z1(e,v) is_free
forall(r in ALL_ROLES) z2(r) is_free
```

```
forall(e in REG_EMP) z4(e) is_free
forall(e in GUARANTEED) z3(e) is_free


end-do



status(prob):= STAT_READY
end-procedure

! Solve the integer problem
function solve_intprob(prob:mpproblem, ct:integer): real
with prob do
status(prob):= STAT_READY
Primal_cost:=(z+
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
    allocate_master(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))
    ))+
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
    depart_master(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t)))))+
sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*(
    cur_board(e,v,t)/((2*cur_board(e,v,t))-1))))+
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e))))

forall(ax in 0..ct)do
if(cut_type(ax)=1)then
!writeln("Cut_type",cut_type(ct))
forall(e in REG_EMP, v in VESSELS)do
MC10(ax,e,v):=z1(e,v)>= (sum( t in TIME)(sol_dual_40(ax,e,v,t))+
(sol_dual_3(ax,e,v)*(sum(r in ROLES(v))(allocate_master(e,r,1)) - starting
    (e,v)))+
```

```
sum(t in 2..WEEKS_TO_PLAN)(sol_dual_4(ax,e,v,t)*(sum(r in ROLES(v))(
   allocate_master(e,r,t)) - sum(r in ROLES(v))(allocate_master(e,r,(t-1))
   ))))
end-do

forall(r in ALL_ROLES)do
MC20(ax,r):=z2(r) >= ((sol_dual_7(ax,r)*( allocate_master("AGENCY",r,1) -
   ag_starting(r)))+
sum(t in 2..WEEKS_TO_PLAN)(sol_dual_8(ax,r,t)*(allocate_master("AGENCY",r,
   t) - allocate_master("AGENCY",r,(t-1))))+
sum(t in TIME)(sol_dual_30(ax,r,t)*(1-allocate_master("AGENCY",r,t)))+
(sol_dual_31(ax,r,1)*(1- ag_starting(r)))+
sum( t in 2..WEEKS_TO_PLAN)(sol_dual_31(ax,r,t)*(1-allocate_master("AGENCY
   ",r,(t-1)))))
end-do

forall(e in GUARANTEED)do
MC30(ax,e):= z3(e) >= ((sol_dual_9(ax,e)*(g_weeks(e) - (exp_worktime(e) +
   sum(r in ALL_ROLES, t in TIME)(allocate_master(e,r,t)))))+
(sol_dual_10(ax,e)*((exp_worktime(e) + sum(r in ALL_ROLES, t in TIME)(
   allocate_master(e,r,t)))- g_weeks(e))))
end-do

forall(e in REG_EMP)do
MC40(ax,e):= z4(e) >=(sol_dual_19(ax,e)*( rest_zero(e) - (1-(sum(r in
   ALL_ROLES)(allocate_master(e,r,1)))))+
sum( t in 2..WEEKS_TO_PLAN)(sol_dual_20(ax,e,t)*(- (1-(sum(r in ALL_ROLES)
   (allocate_master(e,r,t))))))+
sum( t in 2..WEEKS_TO_PLAN)(sol_dual_23(ax,e,t)*(- min_rest(e)*(1-(sum(r
   in ALL_ROLES)(allocate_master(e,r,t))))))+
sum( t in TIME)(sol_dual_21(ax,e,t)*((min_rest(e)-1)*(sum(v in VESSELS)(
   depart_master(e,v,t))))))
end-do

MC(ax):= z >= (sum(e in REG_EMP, v in VESSELS) (z1(e,v))+ sum (e in
   GUARANTEED) z3(e)+sum(r in ALL_ROLES)z2(r)+sum (e in REG_EMP) z4(e))
```

```
end-if

if(cut_type(ax)=0)then
!writeln("Cut_type",cut_type(ct))

MC(ax):= (sum(e in REG_EMP, v in VESSELS, t in TIME)(sol_dual_40(ax,e,v,t)
    )+
sum(e in REG_EMP, v in VESSELS)(sol_dual_3(ax,e,v)*(sum(r in ROLES(v))(
    allocate_master(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(sol_dual_4(ax,e,v,t
    )*(sum(r in ROLES(v))(allocate_master(e,r,t)) - sum(r in ROLES(v))(
    allocate_master(e,r,(t-1)))))+
sum(r in ALL_ROLES)(sol_dual_7(ax,r)*( allocate_master("AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_8(ax,r,t)*(
    allocate_master("AGENCY",r,t) - allocate_master("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(sol_dual_9(ax,e)*(g_weeks(e) - (exp_worktime(e) + sum
    (r in ALL_ROLES, t in TIME)(allocate_master(e,r,t)))))+
sum(e in GUARANTEED)(sol_dual_10(ax,e)*((exp_worktime(e) + sum(r in
    ALL_ROLES, t in TIME)(allocate_master(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(sol_dual_30(ax,r,t)*(1-allocate_master("
    AGENCY",r,t)))+
sum(r in ALL_ROLES)(sol_dual_31(ax,r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(sol_dual_31(ax,r,t)*(1-
    allocate_master("AGENCY",r,(t-1))))+
sum(e in REG_EMP)(sol_dual_19(ax,e)*( rest_zero(e) - (1-(sum(r in
    ALL_ROLES)(allocate_master(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_20(ax,e,t)*(- (1-(sum(r
    in ALL_ROLES)(allocate_master(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(sol_dual_23(ax,e,t)*(- min_rest(e
    )*(1-(sum(r in ALL_ROLES)(allocate_master(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(sol_dual_21(ax,e,t)*((min_rest(e)-1)*(sum(v
    in VESSELS)(depart_master(e,v,t))))))<=0

end-if
end-do
```

```
setparam("XPRS_heurstrategy",3)

if(cut_type(ct)=0)then
setparam("XPRS_MAXTIME",-30)
else
if(x>=2)then
if(50*(x+1)>=3600-(finish-begin))then
setparam("XPRS_MAXTIME",-800)
else
setparam("XPRS_MAXTIME",-(50*(x+1)))
end-if
else
setparam("XPRS_MAXTIME",-(50*(x+1)))
end-if
end-if

!setparam("XPRS_miprelstop",(0.8-(0.1*x)))

!setparam("XPRS_maxmipsol",x)
setcallback(XPRS_CB_INTSOL, "printsol")

setparam("XPRS_verbose",true)
minimize( Primal_cost)

z_best(ct):=getparam("XPRS_BESTBOUND")
best_bound:=max(i in 0..ct)z_best(ct)

end-do

end-function

!-------------------------------------------------------------




int_started:=1
```

```
begin:=gettime

if(int_started=1)then


cuts_added(0):=0
forall(y in 1..2000)do

x:=y

!writeln("x:",x)

! writeln(" best_bound:", best_bound)
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_dual(l,r,t):=allocate(l,r,t)
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_dual(r,v,t):=depart(r,v,t)
end-do
solve_primal_int(ct)

finish:=gettime
writeln("FINISh_BEGIN: ", finish-begin )
if(finish-begin>=3598)then
res3:= compile("heuristic_updated.mos") ! Compile the knapsack model
load(Heuristic, "heuristic_updated.bim") ! Load the knapsack model
initializations to "bin:shmem:sol"
allocate_dual depart_dual InstanceName
end-initializations
run(Heuristic) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
allocate_dual depart_dual
end-initializations
```

```
res2:= compile("subdualcombcut_updated-otheur.mos") ! Compile the knapsack
     model
load(Dualmodel, "subdualcombcut_updated-otheur.bim") ! Load the knapsack
    model
initializations to "bin:shmem:sol"
allocate_dual depart_dual ct UB Dual_cost InstanceName allocate_best
end-initializations
run(Dualmodel) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
    sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
    accept allocate_best
end-initializations

fopen(SUMMARYFILE, F_APPEND)
write(InstanceName)
write("\t",finish-begin)
write("\t",UB1,"\t",UB,"\t", z_best(ct))
write("\n")
fclose(F_APPEND)

fopen(LOGFILE, F_APPEND)
writeln("allocate: [")

forall(e in ALL_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) write(allocate_best(e,r,t),"\t")
end-do
write("\n")
end-do

write("]\n")

fclose(F_APPEND)
```

```
break
else


cuts_added(x):=ct
fopen(LOGFILE,F_APPEND)
write(InstanceName)
write("\t",finish-begin)
write("\t",UB1,"\t",UB,"\t", z_best(ct))
write("\n")
fclose(F_APPEND)

res2:= compile("subdualcombcut_updated-otheur.mos") ! Compile the knapsack
     model
load(Dualmodel, "subdualcombcut_updated-otheur.bim") ! Load the knapsack
    model
initializations to "bin:shmem:sol"
allocate_dual depart_dual ct UB Dual_cost InstanceName allocate_best
end-initializations
run(Dualmodel) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
    sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
    accept allocate_best
end-initializations


res3:= compile("heuristic_updated.mos") ! Compile the knapsack model
load(Heuristic, "heuristic_updated.bim") ! Load the knapsack model
initializations to "bin:shmem:sol"
allocate_dual depart_dual InstanceName
end-initializations
run(Heuristic) ! Start solving knapsack subproblem
```

```
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
allocate_dual depart_dual
end-initializations
ct:=ct+1
res2:= compile("subdualcombcut_updated-otheur.mos") ! Compile the knapsack
     model
load(Dualmodel, "subdualcombcut_updated-otheur.bim") ! Load the knapsack
    model
initializations to "bin:shmem:sol"
allocate_dual depart_dual ct UB Dual_cost InstanceName allocate_best
end-initializations
run(Dualmodel) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
    sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
    accept allocate_best
end-initializations

end-if

evaluate_solution

cuts_added(x):=ct
end-do
end-if


!-------------------------------------------------------------------

procedure printsol
declarations
objval:real
```

```
mip:integer
time_passed:real
end-declarations


time_passed:=gettime

objval:= getparam("XPRS_MIPOBJVAL")
mip:=getparam("XPRS_MIPSOLS")

mip_x:=mip

ct:=ct+1

forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_dual(l,r,t):=round(getsol(allocate_master(l,r,t)))
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_dual(r,v,t):=round(getsol(depart_master(r,v,t)))
end-do

forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_iteration(ct,l,r,t):=round(getsol(allocate_master(l,r,t)))
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_iteration(ct,r,v,t):=round(getsol(depart_master(r,v,t)))
end-do

res2:= compile("subdualcombcut_updated-otheur.mos") ! Compile the knapsack
    model
load(Dualmodel, "subdualcombcut_updated-otheur.bim") ! Load the knapsack
   model
initializations to "bin:shmem:sol"
allocate_dual depart_dual ct UB Dual_cost InstanceName allocate_best
end-initializations
```

```
run(Dualmodel) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
   sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
   accept allocate_best
end-initializations


writeln("INTEGER PROBLEM")


end-procedure

!-------------------------------------------------------------------

procedure save_solution_dual
! Store values of u and x
if(ct=0)then
forall(e in REG_EMP, v in VESSELS)do
sol_dual_3(ct,e,v):=getsol(dual_3(e,v))
end-do


forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_4(ct,e,v,t):=getsol(dual_4(e,v,t))
end-do

forall(e in REG_EMP, v in VESSELS, t in TIME)do
sol_dual_40(ct,e,v,t):=getsol(dual_40(e,v,t))
end-do

forall(r in ALL_ROLES)do
sol_dual_7(ct,r):=getsol(dual_7(r))
end-do
```

```
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_8(ct,r,t):= getsol(dual_8(r,t))
end-do

forall(e in GUARANTEED)do
sol_dual_9(ct,e):=getsol(dual_9(e))
end-do

forall(e in GUARANTEED) do
sol_dual_10(ct,e):=getsol(dual_10(e))
end-do



forall(r in ALL_ROLES, t in TIME)do
sol_dual_30(ct,r,t):= getsol(dual_30(r,t))
end-do

forall(r in ALL_ROLES, t in TIME)do
sol_dual_31(ct,r,t):= getsol(dual_31(r,t))
end-do

forall(e in REG_EMP)do
sol_dual_19(ct,e):=getsol(dual_19(e))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_20(ct,e,t):= getsol(dual_20(e,t))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_23(ct,e,t):= getsol(dual_23(e,t))
end-do

forall(e in REG_EMP, t in TIME)do
sol_dual_21(ct,e,t):= getsol(dual_21(e,t))
end-do
```

```
end-if



end-procedure
!-------------------------------------------------------------------
procedure mypreintsol(isheur:boolean,cutoff:real)

declarations
tamam:integer
end-declarations



tamam:=1

forall( e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)
if(b<=a) then
c:=b
else
c:=a
end-if
end-if
forall(y in 0..c)do
if(c>=1)then
if(sum(r in ALL_ROLES)(getsol(allocate_master(e,r,t+y)))+sum (v in VESSELS
    ) (getsol(depart_master(e,v,t)))>=2)then
sethidden(Rest_new1(e,t,y),false)
tamam:=0
end-if
end-if
end-do
end-do
```

```
if(tamam=0)then

rejectintsol

else
writeln("no reject")
ct:=ct+1

forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_iteration(ct,l,r,t):=round(getsol(allocate_master(l,r,t)))
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_iteration(ct,r,v,t):=round(getsol(depart_master(r,v,t)))
end-do
tamam:=1
end-if

end-procedure

!-------------------------------------------------------------------
procedure evaluate_solution


if(ct=0)then
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_iteration(ct,l,r,t):=allocate(l,r,t)
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_iteration(ct,r,v,t):=depart(r,v,t)
end-do
end-if

forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_dual(l,r,t):=allocate_iteration(ct,l,r,t)
```

```
end-do

forall( r in REG_EMP, v in VESSELS, t in TIME)do
depart_dual(r,v,t):=depart_iteration(ct,r,v,t)
end-do



initializations to "bin:shmem:sol"
allocate_dual depart_dual ct
end-initializations

end-procedure


!-------------------------------------------------------------------
function subprob : real

initializations to "bin:shmem:sol"
allocate_dual depart_dual ct InstanceName allocate_best
end-initializations
run(Dualmodel) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
   sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 Dual_cost UB cut_type
   accept allocate_best
end-initializations
end-function

!-------------------------------------------------------------------

function heuristic_solve : real

initializations to "bin:shmem:sol"
```

```
allocate_dual depart_dual InstanceName
end-initializations
run(Heuristic) ! Start solving knapsack subproblem
wait ! Wait until subproblem finishes
dropnextevent ! Ignore termination message
initializations from "bin:shmem:sol"
allocate_dual depart_dual InstanceName
end-initializations
end-function

!-------------------------------------------------------------------
function cut_manage : boolean
loadcuts(-1,-1)
end-function

!-------------------------------------------------------------
end-model
!---------------------Dual Sub Problem
   ------------------------------------

model "Benders (master model)"
uses "mmxprs", "mmjobs", "mmsystem"

parameters
ALG = 1
BIGM = 100000000
end-parameters

declarations
InstanceName: string
end-declarations

initializations from "bin:shmem:sol"
InstanceName
end-initializations
```

```
DATAFILE :=InstanceName+"\\Time-Windows - Captains - "+InstanceName+".txt"
OUTPUTFILE_subdual:=InstanceName+"\\Results - SubDual-Benders - "+
    InstanceName+".txt"


forward procedure save_solution_dualsub


declarations

STEP_0=2                          ! Codes sent to subproblems
STEP_1=3
STEP_2=4
STAT_SOLVED=6                     ! Status codes returned by subproblems
STAT_INFEAS=7
STAT_READY=8

UB:real !upperbound
LB: real !lowerbound
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

sol_obj: real                     ! Objective function value (primal)
MC: array(range) of linctr        ! Constraints generated by alg.
RM: range                         ! Model indices
cut_type:array(range) of real
```

```
accept: real
stepmod: array(RM) of Model      ! Submodels
end-declarations
!DECLARATION OF PARAMETERS AND DECISION VARIABLES
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
allocate_dual: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer !
   Variable for allocating employee to role during given time period
depart_dual: array(REG_EMP, VESSELS, TIME) of integer
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
allocate_best: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer
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 integer

mwp:linctr
end-declarations

!Reading from txt file
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

!Continue to define parameters and decision variables
declarations
```

```
!Added for recovery problem - detail of current roster, and change
   variable

!solution of dual sub problem
dual_3: array(REG_EMP, VESSELS) of mpvar
dual_4: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar
dual_7: array(ALL_ROLES) of mpvar
dual_8: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9: array(GUARANTEED) of mpvar
dual_10: array(GUARANTEED) of mpvar
dual_30: array(ALL_ROLES,TIME) of mpvar
dual_31: array(ALL_ROLES,TIME) of mpvar
dual_40: array(REG_EMP, VESSELS, TIME) of mpvar
dual_19:array(REG_EMP) of real
dual_21:array(REG_EMP, TIME) of real
dual_20:array(REG_EMP, 2..WEEKS_TO_PLAN) of real
dual_23:array(REG_EMP,2..WEEKS_TO_PLAN) of real


dual_3_new: array(REG_EMP, VESSELS) of mpvar
dual_4_new: array(REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of mpvar
dual_7_new: array(ALL_ROLES) of mpvar
dual_8_new: array(ALL_ROLES, 2..WEEKS_TO_PLAN) of mpvar
dual_9_new: array(GUARANTEED) of mpvar
dual_10_new: array(GUARANTEED) of mpvar
dual_30_new: array(ALL_ROLES,TIME) of mpvar
dual_31_new: array(ALL_ROLES,TIME) of mpvar
dual_40_new: array(REG_EMP, VESSELS, TIME) of mpvar
dual_19_new:array(REG_EMP) of mpvar
dual_21_new:array(REG_EMP, TIME) of mpvar
dual_20_new:array(REG_EMP, 2..WEEKS_TO_PLAN) of mpvar
dual_23_new:array(REG_EMP,2..WEEKS_TO_PLAN) of mpvar

sol_dual_3: array(range,REG_EMP, VESSELS) of real
sol_dual_4: array(range,REG_EMP, VESSELS, 2..WEEKS_TO_PLAN) of real

                                        ! or takes a non-negative integer
```

```
    value for agency crew
sol_dual_7: array(range,ALL_ROLES) of real
sol_dual_8: array(range,ALL_ROLES, 2..WEEKS_TO_PLAN) of real
sol_dual_9: array(range,GUARANTEED) of real
sol_dual_10: array(range,GUARANTEED) of real
sol_dual_19:array(range,REG_EMP) of real
sol_dual_21:array(range,REG_EMP, TIME) of real
sol_dual_20:array(range,REG_EMP, 2..WEEKS_TO_PLAN) of real
sol_dual_23:array(range,REG_EMP,2..WEEKS_TO_PLAN) of real
sol_dual_30: array(range,ALL_ROLES,TIME) of real
sol_dual_31: array(range,ALL_ROLES,TIME) of real
sol_dual_40: array(range,REG_EMP, VESSELS, TIME) of real

                                ! or takes a non-negative integer value
    for agency crew
dummy:linctr
! Objective Value of dual problem
Dual_cost:real
end-declarations

! reading the parameters

declarations
!constraints for dual problem
dual_cons_board: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart: array(REG_EMP, VESSELS, TIME) of linctr        ! =1 if
    employee boards / departs vessel in given time period, 0 otherwise

                                        ! or takes a non-negative integer
    value for agency crew
dual_cons_ag_rboard: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart: array(ALL_ROLES, TIME) of linctr    ! =1 if agency
    crew starts / ends working on a role in given time period, 0 otherwise
dual_cons_undertime: array(GUARANTEED) of linctr
dual_cons_overtime: array(GUARANTEED) of linctr                   !
    Variables to calculate the amount of under/overtime carried out by
    employee
```

```
dual_cons_board_new: array(REG_EMP, VESSELS, TIME) of linctr
dual_cons_depart_new: array(REG_EMP, VESSELS, TIME) of linctr    ! =1 if
   employee boards / departs vessel in given time period, 0 otherwise

                                        ! or takes a non-negative integer
   value for agency crew
dual_cons_ag_rboard_new: array(ALL_ROLES, TIME) of linctr
dual_cons_ag_rdepart_new: array(ALL_ROLES, TIME) of linctr       ! =1 if
   agency crew starts / ends working on a role in given time period, 0
   otherwise
dual_cons_undertime_new: array(GUARANTEED) of linctr
dual_cons_overtime_new: array(GUARANTEED) of linctr                      !
   Variables to calculate the amount of under/overtime carried out by
   employee

dual_cons_rest_total:array(REG_EMP, TIME) of linctr
dual_cons_rest_total_new:array(REG_EMP, TIME) of linctr
!constraints for integer problem
All_covered: dynamic array(ALL_ROLES, TIME) of linctr
No_overlap: array(REG_EMP, TIME) of linctr
Work_count: array(REG_EMP, TIME) of linctr
Work_count_start: array(REG_EMP) of linctr
Depart_constr: array(REG_EMP, VESSELS, TIME) of linctr
Rest_vs_work: array(REG_EMP, TIME) of linctr
Rest_new: array(REG_EMP) of linctr
Rest_new1: array(REG_EMP, TIME) of linctr
ct:integer
int_started:integer
allocate_master: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
depart_master: array(REG_EMP, VESSELS, TIME) of mpvar     ! =1 if employee
   boards / departs vessel in given time period, 0 otherwise
Obj: linctr
Dual_first:linctr
z:mpvar
Primal_cost:linctr
stepprob: array(RM) of mpproblem ! Subproblems
```

```
status: array(mpproblem) of integer ! Subproblem status
AG_work_count_start:array(ALL_ROLES) of linctr
AG_work_count:array(ALL_ROLES, TIME) of linctr
Obj_mwp:linctr
Obj_mwp_value:mpvar
Dual_cost_mwp:linctr
allocate_mwp: dynamic array(1..1,ALL_EMP, ALL_ROLES, TIME) of real !
   Variable for allocating employee to role during given time period
depart_mwp: array(1..1, REG_EMP, VESSELS, TIME) of real
allocate: dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer
depart: array(REG_EMP, VESSELS, TIME) of integer
end-declarations

initializations from DATAFILE
allocate depart
end-initializations

initializations from "bin:shmem:sol"
allocate_dual depart_dual ct UB Dual_cost allocate_best
end-initializations

writeln("SUB DUAL STARTED")
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_dual(e,r,t))
end-if
end-do

Obj:= (sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40_new(e,v,t))+
sum(e in REG_EMP, v in VESSELS)(dual_3_new(e,v)*(sum(r in ROLES(v))(
   allocate_dual(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4_new(e,v,t)*(
   sum(r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(
   allocate_dual(e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7_new(r)*( allocate_dual("AGENCY",r,1) -
   ag_starting(r)))+
```

```
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8_new(r,t)*(allocate_dual
    ("AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9_new(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
sum(e in GUARANTEED)(dual_10_new(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30_new(r,t)*(1-allocate_dual("AGENCY",
    r,t)))+
sum(r in ALL_ROLES)(dual_31_new(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31_new(r,t)*(1-
    allocate_dual("AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19_new(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20_new(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23_new(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21_new(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart_dual(e,v,t))))))
sethidden(Obj,false)


!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board_new(e,v,1):=dual_3_new(
    e,v)+dual_40_new(e,v,1)<=(board_chng_cost(e,v,1))/(1-(2*cur_board(e,v
    ,1)))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)
    dual_cons_board_new(e,v,t):=dual_4_new(e,v,t)+dual_40_new(e,v,t)<=(
    board_chng_cost(e,v,t))/(1-(2*cur_board(e,v,t)))


!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard_new(r,1):= dual_7_new(r)+
    dual_31_new(r,1)<=(ag_board_chng_cost(r,1)/(1-(2*cur_ag_rboard(r,1))))
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard_new(r,t)
    := dual_8_new(r,t)+dual_31_new(r,t)<=(ag_board_chng_cost(r,t)/(1-(2*
```

```
    cur_ag_rboard(r,t))))

!ag_rdepart

forall(r in ALL_ROLES) dual_cons_ag_rdepart_new(r,1):= -dual_7_new(r)+
    dual_30_new(r,1)<=(ag_depart_chng_cost(r,1)/(1-(2*cur_ag_rdepart(r,1)))
    )

forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart_new(r,t)
     := -dual_8_new(r,t)+dual_30_new(r,t)<=(ag_depart_chng_cost(r,t)/(1-(2*
    cur_ag_rdepart(r,t))))



!undertime
forall(e in GUARANTEED) dual_cons_undertime_new(e):= dual_9_new(e)<=
    under_rate(e)

!overtime
forall(e in GUARANTEED) dual_cons_overtime_new(e):= dual_10_new(e)<=
    over_rate(e)


!rest_total

forall(e in REG_EMP) dual_cons_rest_total_new(e,1):= dual_19_new(e)-
    dual_20_new(e,2)+dual_21_new(e,1)-dual_23_new(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total_new(e,t)
    := dual_20_new(e,t)-dual_20_new(e,t+1)+dual_21_new(e,t)-dual_23_new(e,t
    +1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_rest_total_new(e,t):= dual_20_new(e,t)+dual_21_new(e,t)<=0


forall(e in REG_EMP, v in VESSELS) dual_3_new(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4_new(e,v,t
    )>=0
```

```
forall(r in ALL_ROLES) dual_7_new(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8_new(r,t) is_free
forall(e in GUARANTEED) dual_9_new(e)>=0
forall(e in GUARANTEED) dual_10_new(e)>=0
forall(e in REG_EMP) dual_19_new(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20_new(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21_new(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23_new(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30_new(r,t)<=0
forall(r in ALL_ROLES, t in TIME) dual_31_new(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40_new(e,v,t)<=0

!fopen(OUTPUTFILE_subdual, F_OUTPUT)
!setparam("XPRS_verbose",true)
maximize(XPRS_BAR, Obj)

cut_type(ct):=1

if(getprobstat=XPRS_UNB) then
write("Dual Unbounded ")
cut_type(ct):=0
writeln("Cut_type",cut_type(ct))

Obj_mwp:=Obj_mwp_value=0

Dual_cost_mwp:=(sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40_new(e,v
    ,t))+
sum(e in REG_EMP, v in VESSELS)(dual_3_new(e,v)*(sum(r in ROLES(v))(
    allocate_dual(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4_new(e,v,t)*(
    sum(r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(
    allocate_dual(e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7_new(r)*( allocate_dual("AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8_new(r,t)*(allocate_dual
    ("AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
```

```
sum(e in GUARANTEED)(dual_9_new(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
sum(e in GUARANTEED)(dual_10_new(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30_new(r,t)*(1-allocate_dual("AGENCY",
    r,t)))+
sum(r in ALL_ROLES)(dual_31_new(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31_new(r,t)*(1-
    allocate_dual("AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19_new(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20_new(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23_new(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21_new(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart_dual(e,v,t))))))=1


!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board_new(e,v,1):=dual_3_new(
    e,v)+dual_40_new(e,v,1)<=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)
    dual_cons_board_new(e,v,t):=dual_4_new(e,v,t)+dual_40_new(e,v,t)<=0


!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard_new(r,1):= dual_7_new(r)+
    dual_31_new(r,1)<=0
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard_new(r,t)
    := dual_8_new(r,t)+dual_31_new(r,t)<=0

!ag_rdepart

forall(r in ALL_ROLES) dual_cons_ag_rdepart_new(r,1):= -dual_7_new(r)+
    dual_30_new(r,1)<=0
```

```
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart_new(r,t)
     := -dual_8_new(r,t)+dual_30_new(r,t)<=0


!undertime
forall(e in GUARANTEED) dual_cons_undertime_new(e):= dual_9_new(e)<=0

!overtime
forall(e in GUARANTEED) dual_cons_overtime_new(e):= dual_10_new(e)<=0

!rest_total

forall(e in REG_EMP) dual_cons_rest_total_new(e,1):= dual_19_new(e)-
    dual_20_new(e,2)+dual_21_new(e,1)-dual_23_new(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total_new(e,t)
    := dual_20_new(e,t)-dual_20_new(e,t+1)+dual_21_new(e,t)-dual_23_new(e,t
    +1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_rest_total_new(e,t):= dual_20_new(e,t)+dual_21_new(e,t)<=0

forall(e in REG_EMP, v in VESSELS) dual_3_new(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4_new(e,v,t
    )>=0
forall(r in ALL_ROLES) dual_7_new(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8_new(r,t) is_free
forall(e in GUARANTEED) dual_9_new(e)>=0
forall(e in GUARANTEED) dual_10_new(e)>=0
forall(e in REG_EMP) dual_19_new(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20_new(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21_new(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23_new(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30_new(r,t)<=0
forall(r in ALL_ROLES, t in TIME) dual_31_new(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40_new(e,v,t)<=0
!setparam("XPRS_verbose",true)
maximize(XPRS_BAR, Obj_mwp_value)
```

```
end-if

writeln("Cut_type",cut_type(ct))

if(cut_type(ct)=1) then

Dual_cost:=getobjval

writeln("Cut_type",cut_type(ct))
writeln("Dual_cost: ",Dual_cost)
writeln("UB_once",UB)

if (UB>= (Dual_cost+
sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*(
    cur_board(e,v,t)/((2*cur_board(e,v,t))-1))))+
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e)))+
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
    allocate_dual(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))))
    +
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
    depart_dual(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t)))))))
then

UB:=(Dual_cost+
sum(e in REG_EMP, v in VESSELS, t in TIME)((board_chng_cost(e,v,t)*(
    cur_board(e,v,t)/((2*cur_board(e,v,t))-1))))+
sum(r in ALL_ROLES, t in TIME)((ag_board_chng_cost(r,t)*(cur_ag_rboard(r,t
    )/((2*cur_ag_rboard(r,t))-1))) + (ag_depart_chng_cost(r,t)*(
    cur_ag_rdepart(r,t)/((2*cur_ag_rdepart(r,t))-1)))) +
sum(e in GUARANTEED)((-under_rate(e)*cur_undertime(e))+ (-over_rate(e)*
    cur_overtime(e)))+
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(work_chng_cost(e,r,t)*((
    allocate_dual(e,r,t)-cur_allocate(e,r,t))/(1-2*(cur_allocate(e,r,t)))))
```

```
   +
sum(e in REG_EMP, v in VESSELS, t in TIME)(depart_chng_cost(e,v,t)*((
   depart_dual(e,v,t)-cur_depart(e,v,t))/(1-2*(cur_depart(e,v,t))))))

accept:=1


writeln("UB_sonra: ",UB)
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_best(l,r,t):=allocate_dual(l,r,t)
end-do

else

UB:=UB
writeln("UB_noimprove: ",UB)
accept:=0
forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_best(l,r,t):=allocate_best(l,r,t)
end-do
end-if


save_solution_dualsub

y:=1
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) allocate_mwp(y,e,r,t):=(
   allocate(e,r,t)*0.3)+(allocate_dual(e,r,t)*0.7)
forall(e in REG_EMP, v in VESSELS, t in TIME) depart_mwp(y,e,v,t):=(depart
   (e,v,t)*0.3)+(depart_dual(e,v,t)*0.7)



Obj:= (sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40_new(e,v,t))+
sum(e in REG_EMP, v in VESSELS)(dual_3_new(e,v)*(sum(r in ROLES(v))(
   allocate_mwp(y,e,r,1)) - starting(e,v)))+
```

```
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4_new(e,v,t)*(
    sum(r in ROLES(v))(allocate_mwp(y,e,r,t)) - sum(r in ROLES(v))(
    allocate_mwp(y,e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7_new(r)*( allocate_mwp(y,"AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8_new(r,t)*(allocate_mwp(y
    ,"AGENCY",r,t) - allocate_mwp(y,"AGENCY",r,(t-1))))+
sum(e in GUARANTEED)(dual_9_new(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_mwp(y,e,r,t)))))+
sum(e in GUARANTEED)(dual_10_new(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_mwp(y,e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30_new(r,t)*(1-allocate_mwp(y,"AGENCY
    ",r,t)))+
sum(r in ALL_ROLES)(dual_31_new(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31_new(r,t)*(1-
    allocate_mwp(y,"AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19_new(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_mwp(y,e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20_new(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_mwp(y,e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23_new(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_mwp(y,e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21_new(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart_mwp(y,e,v,t))))))
sethidden(Obj,false)

!mwp constraints
mwp:=(sum(e in REG_EMP, v in VESSELS, t in TIME)(dual_40_new(e,v,t))+
sum(e in REG_EMP, v in VESSELS)(dual_3_new(e,v)*(sum(r in ROLES(v))(
    allocate_dual(e,r,1)) - starting(e,v)))+
sum(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)(dual_4_new(e,v,t)*(
    sum(r in ROLES(v))(allocate_dual(e,r,t)) - sum(r in ROLES(v))(
    allocate_dual(e,r,(t-1)))))+
sum(r in ALL_ROLES)(dual_7_new(r)*( allocate_dual("AGENCY",r,1) -
    ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_8_new(r,t)*(allocate_dual
    ("AGENCY",r,t) - allocate_dual("AGENCY",r,(t-1))))+
```

```
sum(e in GUARANTEED)(dual_9_new(e)*(g_weeks(e) - (exp_worktime(e) + sum(r
    in ALL_ROLES, t in TIME)(allocate_dual(e,r,t)))))+
sum(e in GUARANTEED)(dual_10_new(e)*((exp_worktime(e) + sum(r in ALL_ROLES
    , t in TIME)(allocate_dual(e,r,t)))- g_weeks(e)))+
sum(r in ALL_ROLES, t in TIME)(dual_30_new(r,t)*(1-allocate_dual("AGENCY",
    r,t)))+
sum(r in ALL_ROLES)(dual_31_new(r,1)*(1- ag_starting(r)))+
sum(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)(dual_31_new(r,t)*(1-
    allocate_dual("AGENCY",r,(t-1))))+
sum(e in REG_EMP)(dual_19_new(e)*( rest_zero(e) - (1-(sum(r in ALL_ROLES)(
    allocate_dual(e,r,1))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_20_new(e,t)*(- (1-(sum(r in
    ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in 2..WEEKS_TO_PLAN)(dual_23_new(e,t)*(- min_rest(e)
    *(1-(sum(r in ALL_ROLES)(allocate_dual(e,r,t))))))+
sum(e in REG_EMP, t in TIME)(dual_21_new(e,t)*((min_rest(e)-1)*(sum(v in
    VESSELS)(depart_dual(e,v,t))))))=Dual_cost

!board

forall(e in REG_EMP, v in VESSELS) dual_cons_board_new(e,v,1):=dual_3_new(
    e,v)+dual_40_new(e,v,1)<=(board_chng_cost(e,v,1))/(1-(2*cur_board(e,v
    ,1)))
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)
    dual_cons_board_new(e,v,t):=dual_4_new(e,v,t)+dual_40_new(e,v,t)<=(
    board_chng_cost(e,v,t))/(1-(2*cur_board(e,v,t)))



!ag_rboard
forall(r in ALL_ROLES) dual_cons_ag_rboard_new(r,1):= dual_7_new(r)+
    dual_31_new(r,1)<=(ag_board_chng_cost(r,1)/(1-(2*cur_ag_rboard(r,1))))
forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rboard_new(r,t)
    := dual_8_new(r,t)+dual_31_new(r,t)<=(ag_board_chng_cost(r,t)/(1-(2*
    cur_ag_rboard(r,t))))

!ag_rdepart
```

```
forall(r in ALL_ROLES) dual_cons_ag_rdepart_new(r,1):= -dual_7_new(r)+
    dual_30_new(r,1)<=(ag_depart_chng_cost(r,1)/(1-(2*cur_ag_rdepart(r,1)))
    )

forall(r in ALL_ROLES,t in 2..WEEKS_TO_PLAN) dual_cons_ag_rdepart_new(r,t)
     := -dual_8_new(r,t)+dual_30_new(r,t)<=(ag_depart_chng_cost(r,t)/(1-(2*
    cur_ag_rdepart(r,t))))



!undertime
forall(e in GUARANTEED) dual_cons_undertime_new(e):= dual_9_new(e)<=
    under_rate(e)

!overtime
forall(e in GUARANTEED) dual_cons_overtime_new(e):= dual_10_new(e)<=
    over_rate(e)



!rest_total

forall(e in REG_EMP) dual_cons_rest_total_new(e,1):= dual_19_new(e)-
    dual_20_new(e,2)+dual_21_new(e,1)-dual_23_new(e,2)<=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN-1) dual_cons_rest_total_new(e,t)
    := dual_20_new(e,t)-dual_20_new(e,t+1)+dual_21_new(e,t)-dual_23_new(e,t
    +1)<=0
forall(e in REG_EMP,t in WEEKS_TO_PLAN..WEEKS_TO_PLAN)
    dual_cons_rest_total_new(e,t):= dual_20_new(e,t)+dual_21_new(e,t)<=0



forall(e in REG_EMP, v in VESSELS) dual_3_new(e,v)>=0
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN) dual_4_new(e,v,t
    )>=0
forall(r in ALL_ROLES) dual_7_new(r) is_free
forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN) dual_8_new(r,t) is_free
forall(e in GUARANTEED) dual_9_new(e)>=0
forall(e in GUARANTEED) dual_10_new(e)>=0
```

```
forall(e in REG_EMP) dual_19_new(e)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_20_new(e,t)>=0
forall(e in REG_EMP,t in TIME) dual_21_new(e,t)>=0
forall(e in REG_EMP,t in 2..WEEKS_TO_PLAN) dual_23_new(e,t)>=0
forall(r in ALL_ROLES, t in TIME) dual_30_new(r,t)<=0
forall(r in ALL_ROLES, t in TIME) dual_31_new(r,t)<=0
forall(e in REG_EMP, v in VESSELS, t in TIME) dual_40_new(e,v,t)<=0


!  setparam("XPRS_verbose",true)
maximize(XPRS_BAR, Obj)

if(getprobstat=XPRS_UNB) then
write("MWP Unbounded ")


end-if

if(getprobstat = XPRS_INF) then
writeln("MWP is infeasible")

end-if

if(getprobstat=XPRS_OPT) then
write("MWP Optimal ")
save_solution_dualsub
end-if



else

Dual_cost:=getobjval
UB:=UB

writeln("Cut_type",cut_type)
writeln("Dual_unb: ",Dual_cost)
```

```
writeln("UB_unb: ",UB)

forall(l in ALL_EMP, r in ALL_ROLES, t in TIME)do
allocate_best(l,r,t):=allocate_best(l,r,t)
end-do
save_solution_dualsub
end-if




(!  writeln("allocate: [")

forall(e in ALL_EMP) do
forall(r in ALL_ROLES) do
forall(t in TIME) write(allocate_best(e,r,t),"\t")
end-do
write("\n")
end-do

write("]\n")
!)
!  fclose(F_OUTPUT)
writeln("UB: ",UB)
fflush

!-------------------------------------------------------------------

procedure save_solution_dualsub
! Store values of u and x


forall(e in REG_EMP, v in VESSELS)do
sol_dual_3(ct,e,v):=getsol(dual_3_new(e,v))
end-do
```

```
forall(e in REG_EMP, v in VESSELS, t in 2..WEEKS_TO_PLAN)do
sol_dual_4(ct,e,v,t):=getsol(dual_4_new(e,v,t))
end-do

forall(e in REG_EMP, v in VESSELS, t in TIME)do
sol_dual_40(ct,e,v,t):=getsol(dual_40_new(e,v,t))
end-do

forall(r in ALL_ROLES)do
sol_dual_7(ct,r):=getsol(dual_7_new(r))
end-do

forall(r in ALL_ROLES, t in 2..WEEKS_TO_PLAN)do
sol_dual_8(ct,r,t):= getsol(dual_8_new(r,t))
end-do

forall(e in GUARANTEED)do
sol_dual_9(ct,e):=getsol(dual_9_new(e))
end-do

forall(e in GUARANTEED) do
sol_dual_10(ct,e):=getsol(dual_10_new(e))
end-do


forall(r in ALL_ROLES, t in TIME)do
sol_dual_30(ct,r,t):= getsol(dual_30_new(r,t))
end-do

forall(r in ALL_ROLES, t in TIME)do
sol_dual_31(ct,r,t):= getsol(dual_31_new(r,t))
end-do


forall(e in REG_EMP)do
sol_dual_19(ct,e):=getsol(dual_19_new(e))
end-do
```

```
forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_20(ct,e,t):= getsol(dual_20_new(e,t))
end-do

forall(e in REG_EMP, t in 2..WEEKS_TO_PLAN)do
sol_dual_23(ct,e,t):= getsol(dual_23_new(e,t))
end-do

forall(e in REG_EMP, t in TIME)do
sol_dual_21(ct,e,t):= getsol(dual_21_new(e,t))
end-do



initializations to "bin:shmem:sol"
sol_dual_3 sol_dual_4 sol_dual_7
sol_dual_8 sol_dual_9 sol_dual_10 sol_dual_19 sol_dual_20 sol_dual_21
   sol_dual_23 sol_dual_30 sol_dual_31 sol_dual_40 UB Dual_cost cut_type
   accept allocate_best
end-initializations

end-procedure
end-model
!-------------------------Heuristic
   ------------------------------------------------------

model "Benders (master model)"
uses "mmxprs", "mmjobs", "mmsystem"

parameters
!cost

max_iteration = 3000
max_runtime = 60
short_list_fraction = 1
accept_rule = "current"      ! the solution to be compared to when
   deciding whether to accept a change automatically
```

```
!PARAMETERFILE = "batch-input-parameters.dat"
end-parameters

declarations
InstanceName: string
end-declarations

initializations from "bin:shmem:sol"
InstanceName
end-initializations

DATAFILE :=InstanceName+"\\Time-Windows - Captains - "+InstanceName+".txt"
OUTPUTFILE_heuristic:=InstanceName+"\\Results - Heuristics-Benders- - "+
    InstanceName+".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
FEAS_CHECK: set of string
```

```
end-declarations

initializations from DATAFILE
REG_EMP GUARANTEED VESSELS WEEKS_TO_PLAN ROLES
end-initializations

SOL_TYPE := {"current", "best", "eval", "backward", "forward", "swap", "
    candidate", "kick"}
FEAS_CHECK := {"backward", "forward", "swap", "kick"}
ORD_RULES := {"earliest", "latest", "random"}


! 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_INDEX = 1..48!EMP_NO
!       ROLE_INDEX = 1..25!ROLE_NO
!       emp_array = array(1..48) of string !array(EMP_INDEX) of string
!       role_array = array(1..25) of string !array(ROLE_INDEX) of string
emp_count, role_count: 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 integer
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 real   !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
best_sol_time: integer
no_fwd, no_bkwd, no_swap, no_cand, no_kick: integer
no_tabu, no_infeas: integer
no_nonreduce: integer
initial_cost: real

terminate: boolean
update_done: boolean
candidate_exist: boolean

last_kick_time: integer


! Relating to the various solutions which must be recorded:
number_best: integer
best_index = 1..(max_iteration+1)
best_sols: array(best_index, REG_EMP) of list of string
ag_best_sols: array(best_index, ALL_ROLES) of set of integer
same_best: array(best_index) of boolean
new_best: boolean
```

```
tabu_sol: array(REG_EMP) of list of string
ag_tabu_sol: array(ALL_ROLES) of set of integer
to_check_tabu: string

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

list_sol: array(SOL_TYPE, REG_EMP) of list of string
ag_list_sol: array(SOL_TYPE, ALL_ROLES) of set of integer
ag_crewchange: array(SOL_TYPE, ALL_ROLES) of set of integer

allocate_sol:dynamic 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 real
allocate_dual:dynamic array(ALL_EMP, ALL_ROLES, TIME) of integer
depart_dual: array(REG_EMP, VESSELS, TIME) of integer


! Relating to calculating the costs:
emps_changed: set of string
ag_roles_changed: set of string
to_calculate: string

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
chng_long_work: array(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 real


! Relating to the identification and manipulation of the working blocks:
max_bkwd_extend, max_fwd_extend: integer
extend_len_bkwd, extend_len_fwd: integer

current_list, new_list: list of string
reverse_current_list, reverse_new_list: list of string

emp_extend, swap_emp: string
swappable_emp: array(REG_EMP) of boolean
ag_swappable: boolean
task_extend, swap_task: string
vessel_extend, swap_vessel: string

block_found, swap_block_found: boolean
new_block, swap_new_block: boolean

time_count, find_time, swap_find_time: integer
block_start, swap_block_start, swap_block_earliest: integer
block_end, swap_block_end, swap_block_latest: integer
block_len, swap_block_len: integer
too_early, swap_allowed: boolean
all_rest: boolean
```

```
length_count: integer
add_to_emp: string
removed: set of integer
prev_work: boolean
in_reserve_bkwd, in_reserve_fwd: integer
reserve_list_bkwd, reserve_list_fwd: array(ALL_ROLES) of set of integer
conflict_found_bkwd, conflict_found_fwd: integer

feasible, tabu: boolean
do_extend_bkwd, do_extend_fwd, do_swap: boolean
extend_cost_bkwd, extend_cost_fwd, swapping_cost, candidate_cost: real

swaps_examined: array(REG_EMP) of set of string
emps_to_update: set of string
candidate_emps: set of string


! Used for selecting a random assignment with which to alter the current
   solution
random_emp, random_task, random_length, random_time: integer
kick_emp, kick_task: string
kick_start, kick_end: integer
kick_feas: boolean
kick_count: integer
y: integer


! Relating to determining the order in which employees are examined:
order_rule: string
order_number, last_changed: array(REG_EMP) of real
ordered_list, short_ordered_list: list of string
added_set: set of string
min_no, change_no: real
min_emp: string

end-declarations
```

```
initializations from "bin:shmem:sol"
allocate_dual depart_dual
end-initializations

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_dual(e,r,t))
end-if
end-do

forall(a in SOL_TYPE, 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_sol(a,e,r,t))
end-if
end-do
!-------------------------------------------------------------------------------

!-------------------------------------------------------------------------------



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)
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)
end-do
forall(e in REG_EMP) do
emp_cost(transfer_sol_to,e) := emp_cost(transfer_sol_from,e)
list_sol(transfer_sol_to,e) := list_sol(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)
```

```
depart_sol(transfer_sol_to,e,v,t) := depart_sol(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)
overtime_sol(transfer_sol_to,e) := overtime_sol(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_list_sol(transfer_sol_to,r) := ag_list_sol(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)
ag_rdepart_sol(transfer_sol_to,r,t) := ag_rdepart_sol(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 compare_to_best

if(total_cost("current") = total_cost("best")) then
new_best := true
x := 0
```

```
while(new_best = true and x < number_best) do
x := x + 1
same_best(x) := true
emp_count := 0
while(same_best(x) = true and emp_count < EMP_NO) do
emp_count := emp_count + 1
if(list_sol("current",emp_array(emp_count)) <> best_sols(x,emp_array(
   emp_count))) then same_best(x) := false; end-if
end-do
role_count := 0
while(same_best(x) = true and role_count < ROLE_NO) do
role_count := role_count + 1
if(ag_list_sol("current",role_array(role_count)) <> ag_best_sols(x,
   role_array(role_count))) then same_best(x) := false; end-if
end-do
if(same_best(x) = true) then new_best := false; end-if
end-do
if(new_best = true) then
writeln("\t(This solution is a new equal 'best' solution)")
number_best := number_best + 1
forall(e in REG_EMP) best_sols(number_best,e) := list_sol("current",e)
forall(r in ALL_ROLES) ag_best_sols(number_best,r) := ag_list_sol("current
   ",r)
else
writeln("\t(This solution is identical to a previously found 'best'
   solution)")
end-if
end-if


if(total_cost("current") < total_cost("best")) then
transfer_sol_from := "current"
transfer_sol_to := "best"
transfer_solution

number_best := 1
forall(e in REG_EMP) best_sols(1,e) := list_sol("best",e)
```

```
forall(r in ALL_ROLES) ag_best_sols(1,r) := ag_list_sol("best",r)

writeln("New best solution found, with cost ",total_cost("best"))
best_sol_time := iteration
end-if

end-procedure


!-------------------------------------------------------------------------------

procedure check_tabu

tabu := true

if(to_check_tabu in SOL_TYPE) then
emp_count := 0
while(tabu = true and emp_count < EMP_NO) do
emp_count := emp_count + 1
if(list_sol(to_check_tabu,emp_array(emp_count)) <> tabu_sol(emp_array(
   emp_count))) then tabu := false; end-if
end-do
role_count := 0
while(tabu = true and role_count < ROLE_NO) do
role_count := role_count + 1
if(ag_list_sol(to_check_tabu,role_array(role_count)) <> ag_tabu_sol(
   role_array(role_count))) then tabu := false; end-if
end-do

else
writeln("ERROR - incorrect option selected for checking")
end-if

end-procedure
```

```
!-------------------------------------------------------------------------------


procedure update_swaps_and_changes

forall(e in emps_to_update) do
swaps_examined(e) := {}
forall(f in REG_EMP | f not in emps_to_update) do
if(e in swaps_examined(f)) then swaps_examined(f) -= {e}; end-if
end-do
end-do

if(last_kick_time = iteration) then
forall(e in REG_EMP) last_change(e) := iteration
else
forall(e in emps_to_update) last_changed(e) := iteration
end-if

end-procedure


!-------------------------------------------------------------------------------


procedure check_feasibility

feasible := TRUE


if(feasible = true) then
JCfeas := TRUE        ! Job Cover constraints
forall(r in ALL_ROLES, t in TIME | required(r,t) > 0) do
if(sum(e in ALL_EMP | eligable(e,r,t)>=1)(eligable(e,r,t)*allocate_sol("
    eval",e,r,t)) <> required(r,t)) then
JCfeas := FALSE
writeln("JC inf: ", r," ", t)
end-if
```

```
end-do
if(JCfeas = FALSE) then
feasible := FALSE
end-if
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("OL inf")
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("Board inf")
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("Board inf")
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("depart inf")
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("depart inf")
end-if
end-do
end-do

forall(e in REG_EMP, t in TIME) do
if(sum(v in VESSELS)depart_sol("eval",e,v,t) + sum(r in ALL_ROLES)(
   allocate_sol("eval",e,r,(t)))>=2) then
Dprtfeas := FALSE
writeln("depart inf")
end-if
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("agboard inf")
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("agboard inf")
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("undertime inf")
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("overtime inf")
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:
LWfeas := TRUE

forall(e in emps_changed|work_zero(e)>=1) do
if(work_zero(e) + sum(r in ALL_ROLES, t in 0..max_work(e)-work_zero(e))(
    allocate_sol("eval",e,r,t+1))>max_work(e)) then
LWfeas := FALSE
writeln("long work inf")
end-if
end-do

forall(e in emps_changed,t in 1..WEEKS_TO_PLAN-max_work(e)) do
if(sum(r in ALL_ROLES, k in 0..max_work(e))(allocate_sol("eval",e,r,t+k))
    > max_work(e)) then
LWfeas := FALSE
writeln("lw inf")
end-if
end-do




if(LWfeas = FALSE) then feasible := FALSE; end-if
end-if

! Agency Long Work constraints
if(feasible = true) then
AGLWfeas := TRUE
! First, calculate Agency work resource values
forall(r in ag_roles_changed|ag_work_zero(r)>=1) do
if(ag_max_work(r) < ag_work_zero(r) + sum( t in 0..ag_max_work(r)-
    ag_work_zero(r))(allocate_sol("eval","AGENCY",r,t+1)) ) then
AGLWfeas := FALSE
```

```
writeln("aglw inf")
end-if
end-do

forall(r in ag_roles_changed,t in 1..WEEKS_TO_PLAN-ag_max_work(r)) do
if(ag_max_work(r) < sum( k in 0..ag_max_work(r))(allocate_sol("eval","
    AGENCY",r,t+k)) ) then
AGLWfeas := FALSE
writeln("aglw inf")
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
forall(r in ALL_ROLES,t in 1..rest_zero(e)) do
if(rest_zero(e)>=1)then
if(allocate_sol("eval",e,r,t)>0) then
RvWfeas := FALSE
end-if
end-if
end-do
end-do

forall(e in emps_changed) do
forall(t in 1..WEEKS_TO_PLAN-1) do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)
if(b<=a) then
c:=b
else
c:=a
end-if
end-if
forall(ax in 0..c)do
```

```
if( sum(r in ALL_ROLES)(allocate_sol("eval",e,r,t+ax))+sum (v in VESSELS)
    depart_sol("eval",e,v,t)>=2)then
RvWfeas := FALSE
end-if
end-do
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(e,r,t) <> allocate_sol("eval",e,r,t)) then
Linkfeas := FALSE
end-if
else
if(chng_allocate(e,r,t) <> cur_allocate(e,r,t) - allocate_sol("eval",e,r,t
    )) then
Linkfeas := FALSE
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(e,v,t) <> board_sol("eval",e,v,t)) then
Linkfeas := FALSE
end-if
```

```
else
if(chng_board(e,v,t) <> cur_board(e,v,t) - board_sol("eval",e,v,t)) then
Linkfeas := FALSE
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(e,v,t) <> depart_sol("eval",e,v,t)) then
Linkfeas := FALSE
end-if
else
if(chng_depart(e,v,t) <> cur_depart(e,v,t) - depart_sol("eval",e,v,t))
   then
Linkfeas := FALSE
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(r,t) <> ag_rboard_sol("eval",r,t)) then
Linkfeas := FALSE
end-if
else
if(chng_ag_rboard(r,t) <> cur_ag_rboard(r,t) - ag_rboard_sol("eval",r,t))
   then
Linkfeas := FALSE
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(r,t) <> ag_rdepart_sol("eval",r,t)) then
Linkfeas := FALSE
end-if
else
```

```
if(chng_ag_rdepart(r,t) <> cur_ag_rdepart(r,t) - ag_rdepart_sol("eval",r,t
   )) then
Linkfeas := FALSE
end-if
end-if
end-do

forall(e in emps_changed | e in GUARANTEED) do
if(chng_undertime(e) <> undertime_sol("eval",e) - cur_undertime(e)) then
Linkfeas := FALSE
end-if
if(chng_overtime(e) <> overtime_sol("eval",e) - cur_overtime(e)) then
Linkfeas := FALSE
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(e,r,t) <> 0 and chng_allocate(e,r,t) <> 1) then Statfeas
   := FALSE; end-if
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(e,v,t) <> 0 and chng_board(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(e,v,t) <> 0 and chng_depart(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(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_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
feasible := FALSE
end-if
end-if


if(feasible = FALSE) then
no_infeas := no_infeas +1
end-if

end-procedure




!-------------------------------------------------------------------------------
```

```
procedure calculate_cost

!      writeln("Costs may have changed for employees:  ",emps_changed)
!      writeln("Costs may have changed for agency in roles: ",
   ag_roles_changed)

if(to_calculate in SOL_TYPE) then
forall(e in REG_EMP) list_sol("eval",e) := list_sol(to_calculate,e)
forall(r in ALL_ROLES) ag_list_sol("eval",r) := ag_list_sol(to_calculate,r
   )

total_cost("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))
forall(t in TIME) do
forall(r in ALL_ROLES) do
allocate_sol("eval",e,r,t) := allocate_sol("current",e,r,t)
end-do
forall(v in VESSELS) board_sol("eval",e,v,t) := board_sol("current",e,v,t)
forall(v in VESSELS) depart_sol("eval",e,v,t) := depart_sol("current",e,v,
   t)
end-do
if(e in GUARANTEED) then
undertime_sol("eval",e) := undertime_sol("current",e)
overtime_sol("eval",e) := overtime_sol("current",e)
end-if

else

emp_cost("eval",e) := 0
                        !; writeln("Emp ",e," cost must be recalculated,
   now = ",emp_cost("eval",e))
calc_time_count := 0
```

```
consec_work := work_zero(e)

forall(j in list_sol("eval",e)) do
calc_time_count := calc_time_count +1

forall(r in ALL_ROLES) do
allocate_sol("eval",e,r,calc_time_count) := 0

if(j = r) then
allocate_sol("eval",e,r,calc_time_count) := 1
consec_work := consec_work +1
end-if
end-do
if(j = "rest") then
consec_work := 0
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

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(e) := integer(undertime_sol("eval",e) - cur_undertime(e))
chng_overtime(e) := integer(overtime_sol("eval",e) - cur_overtime(e))
emp_cost("eval",e) := emp_cost("eval",e) + (under_rate(e)*chng_undertime(e
    )) + (over_rate(e)*chng_overtime(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(e,r,t) := allocate_sol("
    eval",e,r,t)
else chng_allocate(e,r,t) := cur_allocate(e,r,t) - allocate_sol("eval",e,r
    ,t)
end-if
emp_cost("eval",e) := emp_cost("eval",e) + (work_chng_cost(e,r,t)*
    chng_allocate(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
end-do

forall(v in VESSELS, t in TIME) do
if(cur_board(e,v,t) = 0) then chng_board(e,v,t) := board_sol("eval",e,v,t)
else chng_board(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(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(e,v,t) := depart_sol("eval",e,v
    ,t)
else chng_depart(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(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

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_crewchange("eval",r) := ag_crewchange("current",r)
```

```
forall(t in TIME) do
allocate_sol("eval","AGENCY",r,t) := allocate_sol("current","AGENCY",r,t)
ag_rboard_sol("eval",r,t) := ag_rboard_sol("current",r,t)
ag_rdepart_sol("eval",r,t) := ag_rdepart_sol("current",r,t)
end-do
else

ag_cost("eval",r) := 0

                                                                    !; writeln("
    Ag Role ",r," cost must be recalculated, now = ",ag_cost("eval",r))

if(ag_starting(r) = 1) then
onboard := true
else
onboard := false
end-if

possible_crewchange := {}
definite_crewchange := {}
forall(t in TIME) do
if(t in ag_list_sol("eval",r)) 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
```

```
if(ag_list_sol("eval",r) = ag_list_sol("best",r) and iteration > 0) then
ag_crewchange("eval",r) := ag_crewchange("best",r)
                                !; writeln("Evaluating solution = best
   solution (and iteration = ",iteration,") for agency for role ",r,"\t=>
   crewchange set = ",ag_evaluating_crewchange(r))
elif(to_calculate <> "current" and ag_list_sol("eval",r) = ag_list_sol("
   current",r)) then
ag_crewchange("eval",r) := ag_crewchange("current",r)
                          !; writeln("Evaluating solution = current
   solution (but 'to_calculate <> current ) for agency for role ",r,"\t=>
   crewchange set = ",ag_evaluating_crewchange(r))
else
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




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
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

if(cur_allocate("AGENCY",r,t) = 0) then chng_allocate("AGENCY",r,t) :=
    allocate_sol("eval","AGENCY",r,t)
else chng_allocate("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("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

if(cur_ag_rboard(r,t) = 0) then chng_ag_rboard(r,t) := ag_rboard_sol("eval
    ",r,t)
else chng_ag_rboard(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(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(r,t) := ag_rdepart_sol("
    eval",r,t)
else chng_ag_rdepart(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(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



end-do

end-if
end-do


total_cost("eval") := (sum(e in REG_EMP) emp_cost("eval",e)) + (sum(r in
    ALL_ROLES) ag_cost("eval",r))

transfer_sol_from := "eval"
```

```
transfer_sol_to := to_calculate
transfer_solution

else
writeln("ERROR - incorrect option selected for evaluation")
end-if


if(to_calculate in FEAS_CHECK) then
check_feasibility
end-if


end-procedure




!-------------------------------------------------------------------------


procedure evaluate_backwards

extend_len_bkwd := max_bkwd_extend
while(do_extend_bkwd = false and extend_len_bkwd > 0) do

emps_changed := {}
ag_roles_changed := {}

forall(e in REG_EMP) list_sol("backward",e) := list_sol("current",e)
forall(r in ALL_ROLES) ag_list_sol("backward",r) := ag_list_sol("current",
    r)

list_sol("backward",emp_extend) := []
emps_changed += {emp_extend}
time_count := 0
rest_count := 0
```

```
length_count := work_zero(emp_extend)
forall(r in ALL_ROLES) reserve_list_bkwd(r) := {}
in_reserve_bkwd := 0


forall(j in list_sol("current",emp_extend)) do
time_count := time_count +1

if(time_count <= block_end) then
list_sol("backward",emp_extend) += [j]
if(j = "rest") then length_count := 0
else length_count := length_count + 1
end-if
elif(time_count <= block_end + extend_len_bkwd) then
list_sol("backward",emp_extend) += [task_extend]
                                                    !; if(j <>
   task_extend) then writeln("\t Add role ",task_extend," at time ",
   time_count," for extending emp"); end-if
length_count := length_count + 1
if(j <> task_extend and j <> "rest") then
reserve_list_bkwd(j) += {time_count}
                                                    !; writeln("\tRemove
    role ",j," at time ",time_count," for extending emp")
in_reserve_bkwd := in_reserve_bkwd + 1
end-if
else
if(j <> "rest" and (rest_count < min_rest(emp_extend) or length_count >
   max_work(emp_extend))) then
list_sol("backward",emp_extend) += ["rest"]
reserve_list_bkwd(j) += {time_count}
                                                    !; writeln("\tRemove
    role ",j," at time ",time_count," for extending emp")
in_reserve_bkwd := in_reserve_bkwd + 1
rest_count := rest_count +1
length_count := 0
else
list_sol("backward",emp_extend) += [j]
```

```
if(j = "rest") then
rest_count := rest_count +1
length_count := 0
else
length_count := length_count + 1
end-if
end-if
end-if
end-do



conflict_found_bkwd := 0
forall(i in 1..extend_len_bkwd) do
if((block_end+i) in ag_list_sol("backward",task_extend)) then
!                            writeln("\tFor i = ",i,", conflicting task is
     currently assigned to agency crew")
conflict_found_bkwd := conflict_found_bkwd + 1
ag_list_sol("backward",task_extend) -= {block_end+i}
                                                  !; writeln("\tRemove role ",
   task_extend," at time ",block_end+i," for agency crew")
ag_roles_changed += {task_extend}
end-if
end-do
forall(r in ALL_ROLES | reserve_list_bkwd(r) <> {}) do
removed := {}
forall(t in reserve_list_bkwd(r)) do
if(t-1 in ag_list_sol("backward",r)) then
ag_list_sol("backward",r) += {t}
                                                                  !; writeln("\
   t  Add role ",r," at time ",t," for agency crew")
ag_roles_changed += {r}
removed += {t}
in_reserve_bkwd := in_reserve_bkwd - 1
end-if
end-do
forall(t in removed) reserve_list_bkwd(r) -= {t}
end-do
```

```
forall(e in REG_EMP | e <> emp_extend) do
if(conflict_found_bkwd < extend_len_bkwd or in_reserve_bkwd > 0) then
list_sol("backward",e) := []
time_count := 0
rest_count := 0
length_count := work_zero(e)
add_to_emp := ""
if(work_zero(e) > 0) then
prev_work := true
rest_count := 0
else
prev_work := false
rest_count := min_rest(e) - rest_zero(e)
end-if

forall(j in list_sol("current",e)) do
time_count := time_count +1

if(time_count <= block_end) then
list_sol("backward",e) += [j]
if(j <> "rest") then
length_count := length_count + 1
prev_work := true
rest_count := 0
else
length_count := 0
prev_work := false
rest_count := rest_count + 1
end-if
else
if(j = "rest") then
if(add_to_emp <> "") then
list_sol("backward",e) += [add_to_emp]
                                !; writeln("\t Add role ",add_to_emp," at
    time ",time_count," for employee ",e)
```

```
emps_changed += {e}
length_count := length_count + 1
reserve_list_bkwd(add_to_emp) -= {time_count}
in_reserve_bkwd := in_reserve_bkwd - 1
rest_count := 0
prev_work := true

if(time_count < WEEKS_TO_PLAN) then
if(length_count >= max_work(e) or eligable(e,add_to_emp,time_count+1) < 1
   or time_count+1 not in reserve_list_bkwd(add_to_emp)) then
add_to_emp := ""
end-if
else
add_to_emp := ""
end-if
else
list_sol("backward",e) += [j]
length_count := 0
rest_count := rest_count + 1
prev_work := false
end-if
else
if(j = task_extend and time_count <= block_end + extend_len_bkwd) then
!                                                          writeln("\tFor i
   = ",block_start-time_count,", conflicting task is currently assigned to
    ",e)
conflict_found_bkwd := conflict_found_bkwd + 1
list_sol("backward",e) += ["rest"]
                                          !; writeln("\tRemove role ",j," at
   time ",time_count," for employee ",e)
emps_changed += {e}
length_count := 0
rest_count := rest_count + 1
prev_work := false
else
if(prev_work = false and rest_count < min_rest(e)) then
```

```
list_sol("backward",e) += ["rest"]
                                !; writeln("\tRemove role ",j," at time ",
   time_count," for employee ",e)
ag_list_sol("backward",j) += {time_count}
                          !; writeln("\t Add role ",j," at time ",
   time_count," for agency crew")
ag_roles_changed += {j}
rest_count := rest_count + 1
else
list_sol("backward",e) += [j]
length_count := length_count + 1
rest_count := 0
prev_work := true
if(time_count < WEEKS_TO_PLAN) then
if(length_count < max_work(e) and eligable(e,j,time_count+1) > 0 and
   time_count+1 in reserve_list_bkwd(j)) then
add_to_emp := j
else
add_to_emp := ""
end-if
else
add_to_emp := ""
end-if
end-if
end-if
end-if
end-if
end-do
end-if
end-do
forall(r in ALL_ROLES | reserve_list_bkwd(r) <> {}) do
!                     writeln("\tTimes still 'in reserve' for role ",r,":
   ",reserve_list_bkwd(r))
forall(t in reserve_list_bkwd(r)) ag_list_sol("backward",r) += {t}
                                              !; forall(t in
   reserve_list_bkwd(r)) writeln("\t Add role ",r," at time ",t," for
   agency crew")
```

```
ag_roles_changed += {r}
end-do


! Check if this solution is "tabu":
to_check_tabu := "backward"
check_tabu

if(tabu = true) then
no_tabu := no_tabu + 1
else

! Now, evaluate cost and decide whether to accept these changes...
!      If the extending cost is negative we should accept the change.
!      It it is zero we should probably accept the change.
!      If it is positive perhaps we should:
!             Evaluate the cost of the alternative forward extension.
!             If the forward extension yields a negative (or non-positive
   ?) cost, accept this.
!             Otherwise identify whichever is the cheapest...
!             ... and carry it out so long as it is within a certain cost
   range?
!      Or perhaps evaluate based on another property? E.g.
!             If it is taking a task away from agency crew?
!             Or not removing another working task (from the extending
   schedule)
!             Then is it a more promising move?

to_calculate := "backward"
calculate_cost
if(feasible = true) then
extend_cost_bkwd := total_cost("backward") - total_cost("current")
if(total_cost("backward") < total_cost(accept_rule)) then !!!!!!!!
   Decision rule for accepting
do_extend_bkwd := true
          !!!!!!!!             backward extension
no_nonreduce := 0
```

```
emps_to_update := emps_changed
update_swaps_and_changes

else
if(candidate_exist = false or total_cost("backward") < total_cost("
    candidate")) then
writeln("New candidate solution using backward extension (change in cost =
     ",extend_cost_bkwd,")")
writeln("Details - Emp: ",emp_extend,"\tRole: ",task_extend,"\tOrig start:
     ",block_start,"\tOrig End: ",block_end,"\tExt length: ",
    extend_len_bkwd)
writeln

candidate_exist := true
transfer_sol_from := "backward"
transfer_sol_to := "candidate"
transfer_solution
candidate_cost := extend_cost_bkwd
forall(e in emps_changed) candidate_emps += {e}
end-if
end-if
end-if
end-if

if(do_extend_bkwd = false) then extend_len_bkwd := extend_len_bkwd - 1 ;
    end-if
end-do

end-procedure

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

procedure evaluate_forwards

extend_len_fwd := max_fwd_extend
while(do_extend_fwd = false and extend_len_fwd > 0) do
```

```
emps_changed := {}
ag_roles_changed := {}

forall(e in REG_EMP) list_sol("forward",e) := list_sol("current",e)
forall(r in ALL_ROLES) ag_list_sol("forward",r) := ag_list_sol("current",r
    )

reverse_current_list := getreverse(list_sol("current",emp_extend))
reverse_new_list := []
time_count := WEEKS_TO_PLAN+1
rest_count := 0
forall(r in ALL_ROLES) reserve_list_fwd(r) := {}
in_reserve_fwd := 0

forall(j in reverse_current_list) do
time_count := time_count -1

if(time_count >= block_start) then
reverse_new_list += [j]

          !; writeln("\t Keep role ",j," at time ",time_count," for
    extending emp")
elif(time_count >= block_start - extend_len_fwd) then
if(j <> task_extend) then
reverse_new_list += [task_extend]
                                                                    !;
    writeln("\t Add role ",task_extend," at time ",time_count," for
    extending emp")
if(j <> "rest") then
reserve_list_fwd(j) += {time_count}
                                                        !; writeln("\
    tRemove role ",j," at time ",time_count," for extending emp")
in_reserve_fwd := in_reserve_fwd + 1
end-if
end-if
else
```

```
if(j <> "rest" and rest_count < min_rest(emp_extend)) then
reverse_new_list += ["rest"]
reserve_list_fwd(j) += {time_count}
                                                                    !; writeln("\
   tRemove role ",j," at time ",time_count," for extending emp")
in_reserve_fwd := in_reserve_fwd + 1
ag_roles_changed += {j}
rest_count := rest_count +1
else
reverse_new_list += [j]
                                                                          !;
   writeln("\t Keep role ",j," at time ",time_count," for extending emp")
if(j = "rest") then   rest_count := rest_count +1; end-if
end-if
end-if
end-do
list_sol("forward",emp_extend) := getreverse(reverse_new_list)
emps_changed += {emp_extend}


conflict_found_fwd := 0
forall(i in 1..extend_len_fwd) do
if((block_start-i) in ag_list_sol("forward",task_extend)) then
!                            writeln("\tFor i = ",i,", conflicting task is
     currently assigned to agency crew")
conflict_found_fwd := conflict_found_fwd + 1
ag_list_sol("forward",task_extend) -= {block_start-i}
                                              !; writeln("\tRemove role ",
   task_extend," at time ",block_end+i," for agency crew")
ag_roles_changed += {task_extend}
end-if
end-do
forall(r in ALL_ROLES | reserve_list_fwd(r) <> {}) do
removed := {}
forall(t in reserve_list_fwd(r)) do
if(t+1 in ag_list_sol("forward",r)) then
```

```
ag_list_sol("forward",r) += {t}
                                                                    !; writeln("\
   t Add role ",r," at time ",t," for agency crew")
ag_roles_changed += {r}
removed += {t}
in_reserve_fwd := in_reserve_fwd - 1
end-if
end-do
forall(t in removed) reserve_list_fwd(r) -= {t}
end-do



forall(e in REG_EMP | e <> emp_extend) do
if(conflict_found_fwd < extend_len_fwd or in_reserve_fwd > 0) then
reverse_current_list := getreverse(list_sol("current",e))
reverse_new_list := []
time_count := WEEKS_TO_PLAN + 1
rest_count := min_rest(e)
length_count := 0
add_to_emp := ""
prev_work := false

forall(j in reverse_current_list) do
time_count := time_count -1

if(time_count >= block_start) then
reverse_new_list += [j]
                                                                    !; writeln("\
   t Keep role ",j," at time ",time_count," for employee ",e)
if(j <> "rest") then
length_count := length_count + 1
prev_work := true
else
length_count := 0
prev_work := false
end-if
else
```

```
if(j = "rest") then
if(add_to_emp <> "") then
reverse_new_list += [add_to_emp]
                                          !; writeln("\t Add role ",
    add_to_emp," at time ",time_count," for employee ",e)
emps_changed += {e}
length_count := length_count + 1
reserve_list_fwd(add_to_emp) -= {time_count}
in_reserve_fwd := in_reserve_fwd - 1
prev_work := true
rest_count := 0

if(time_count > 1) then
if(length_count >= max_work(e) or eligable(e,add_to_emp,time_count-1) < 1
    or time_count-1 not in reserve_list_fwd(add_to_emp) or time_count-1 <=
    rest_zero(e)) then
add_to_emp := ""
else
if(((sum(v in VESSELS | add_to_emp not in ROLES(v)) starting(e,v)) > 0 and
     time_count-1 <= min_rest(e)) or ((sum(v in VESSELS | add_to_emp in
    ROLES(v)) starting(e,v)) > 0 and time_count-1 >= 2 and time_count-1 <=
    min_rest(e))) then
add_to_emp := ""
end-if
end-if
else
add_to_emp := ""
end-if
else
reverse_new_list += [j]
                                                !; writeln("\t Keep role ",j
    ," at time ",time_count," for employee ",e)
length_count := 0
rest_count := rest_count + 1
prev_work := false
end-if
else
```

```
if(j = task_extend and time_count >= block_start - extend_len_fwd) then
!                                                          writeln("\tFor i
   = ",block_start-time_count,", conflicting task is currently assigned to
    ",e)
conflict_found_fwd := conflict_found_fwd + 1
reverse_new_list += ["rest"]
                                          !; writeln("\tRemove role ",j," at
   time ",time_count," for employee ",e)
emps_changed += {e}
length_count := 0
rest_count := rest_count + 1
prev_work := false
else
if(prev_work = false and rest_count < min_rest(e)) then
reverse_new_list += ["rest"]
                                  !; writeln("\tRemove role ",j," at time ",
   time_count," for employee ",e)
ag_list_sol("forward",j) += {time_count}
                          !; writeln("\t Add role ",j," at time ",
   time_count," for agency crew")
ag_roles_changed += {j}
rest_count := rest_count + 1
else
reverse_new_list += [j]
                                          !; writeln("\t Keep role ",j," at
   time ",time_count," for employee ",e)
length_count := length_count + 1
rest_count := 0
prev_work := true
if(time_count > 1) then
if(length_count < max_work(e) and eligable(e,j,time_count-1) > 0 and
   time_count-1 in reserve_list_fwd(j) and time_count-1 > rest_zero(e))
   then
if(((sum(v in VESSELS | add_to_emp not in ROLES(v)) starting(e,v)) < 1 or
   time_count-1 > min_rest(e)) and ((sum(v in VESSELS | add_to_emp in
   ROLES(v)) starting(e,v)) < 1 or time_count-1 = 1 or time_count-1 >
   min_rest(e))) then
```

```
add_to_emp := j
else
add_to_emp := ""
end-if
else
add_to_emp := ""
end-if
else
add_to_emp := ""
end-if
end-if
end-if
end-if
end-if
end-do

list_sol("forward",e) := getreverse(reverse_new_list)

end-if
end-do
forall(r in ALL_ROLES | reserve_list_fwd(r) <> {}) do
forall(t in reserve_list_fwd(r)) ag_list_sol("forward",r) += {t}
                                                    !; forall(t in
    reserve_list_fwd(r)) writeln("\t Add role ",r," at time ",t," for
    agency crew")
ag_roles_changed += {r}
end-do


! Check if this solution is "tabu":
to_check_tabu := "forward"
check_tabu

if(tabu = true) then
no_tabu := no_tabu + 1
else
```

```
! Now, evaluate cost and decide whether to accept these changes...
!      If the extending cost is negative we should accept the change.
!      It it is zero we should probably accept the change.
!      If it is positive perhaps we should:
!             Evaluate the cost of the alternative forward extension.
!             If the forward extension yields a negative (or non-positive
   ?) cost, accept this.
!             Otherwise identify whichever is the cheapest...
!             ... and carry it out so long as it is within a certain cost
   range?
!      Or perhaps evaluate based on another property? E.g.
!             If it is taking a task away from agency crew?
!             Or not removing another working task (from the extending
   schedule)
!             Then is it a more promising move?

to_calculate := "forward"
calculate_cost
if(feasible = true) then
extend_cost_fwd := total_cost("forward") - total_cost("current")
if(total_cost("forward") < total_cost(accept_rule)) then  !!!!!!!!
   Decision rule for accepting
do_extend_fwd := true
          !!!!!!!!             forward extension
no_nonreduce := 0

emps_to_update := emps_changed
update_swaps_and_changes

else
if(candidate_exist = false or total_cost("forward") < total_cost("
   candidate")) then
candidate_exist := true
writeln("New candidate solution using forward extension (change in cost =
   ",extend_cost_fwd,")")
writeln("Details - Emp: ",emp_extend,"\tRole: ",task_extend,"\tOrig start:
    ",block_start,"\tOrig End: ",block_end,"\tExt length: ",extend_len_fwd
```

```
    )
writeln

transfer_sol_from := "forward"
transfer_sol_to := "candidate"
transfer_solution
candidate_cost := extend_cost_fwd
forall(e in emps_changed) candidate_emps += {e}
end-if
end-if
end-if
end-if

if(do_extend_fwd = false) then extend_len_fwd := extend_len_fwd - 1; end-
    if
end-do


end-procedure




! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

procedure swap_calculation

emps_changed := {}
ag_roles_changed := {}

forall(e in REG_EMP) list_sol("swap",e) := list_sol("current",e)
forall(r in ALL_ROLES) ag_list_sol("swap",r) := ag_list_sol("current",r)

list_sol("swap",emp_extend) := []
emps_changed += {emp_extend}
```

```
time_count := 0

forall(j in list_sol("current",emp_extend)) do
time_count := time_count +1

if(time_count < block_start and time_count < swap_block_start) then
if(swap_block_start < block_start and time_count >= (swap_block_start -
   min_rest(emp_extend)) and j <> "rest") then
list_sol("swap",emp_extend) += ["rest"]
ag_list_sol("swap",j) += {time_count}
ag_roles_changed += {j}
else
list_sol("swap",emp_extend) += [j]
end-if
elif(time_count >= swap_block_start and time_count <= swap_block_end) then
list_sol("swap",emp_extend) += [swap_task]
if(j <> "rest" and (time_count < block_start or time_count > block_end))
   then
ag_list_sol("swap",j) += {time_count}
ag_roles_changed += {j}
end-if
elif(time_count >= block_start and time_count <= block_end) then
list_sol("swap",emp_extend) += ["rest"]
else
if(swap_block_end > block_end and time_count <= (swap_block_end + min_rest
   (emp_extend)) and j <> "rest") then
list_sol("swap",emp_extend) += ["rest"]
ag_list_sol("swap",j) += {time_count}
ag_roles_changed += {j}
else
list_sol("swap",emp_extend) += [j]
end-if
end-if
end-do
```

```
if(swap_emp <> "AGENCY") then
list_sol("swap",swap_emp) := []
emps_changed += {swap_emp}
time_count := 0
!if(swap_task = "rest") then writeln("Original block starts / ends: ",
   block_start," / ",block_end,"\tSwap block starts / ends: ",
   swap_block_start," / ",swap_block_end); end-if
forall(j in list_sol("current",swap_emp)) do
time_count := time_count +1

if(time_count < block_start and time_count < swap_block_start) then
if((block_start < swap_block_start or swap_task = "rest") and time_count
   >= (block_start - min_rest(swap_emp)) and j <> "rest") then
list_sol("swap",swap_emp) += ["rest"]

                        !; if(swap_task = "rest") then writeln("At time
   ",time_count," replace task ",j," with 'rest'"); end-if
ag_list_sol("swap",j) += {time_count}
ag_roles_changed += {j}
else
list_sol("swap",swap_emp) += [j]

                                !; if(swap_task = "rest") then writeln("At
    time ",time_count," item ",j," remains unchanged (time is before block
    starts)"); end-if
end-if
elif(time_count >= block_start and time_count <= block_end) then
list_sol("swap",swap_emp) += [task_extend]

                                !; if(swap_task = "rest") then writeln("At
    time ",time_count," insert the extending task ",task_extend); end-if
if(j <> "rest" and (time_count < swap_block_start or time_count >
   swap_block_end)) then
ag_list_sol("swap",j) += {time_count}

                        !; if(swap_task = "rest") then writeln("\t(This
   is done in place of task ",j); end-if
```

```
ag_roles_changed += {j}
end-if
elif(time_count >= swap_block_start and time_count <= swap_block_end) then
list_sol("swap",swap_emp) += ["rest"]

                                !; if(swap_task = "rest") then writeln("At
    time ",time_count," we insert a 'rest' task"); end-if
else
if((block_end > swap_block_end or swap_task = "rest") and time_count <= (
   block_end + min_rest(swap_emp)) and j <> "rest") then
list_sol("swap",swap_emp) += ["rest"]

                          !; if(swap_task = "rest") then writeln("At time
   ",time_count," replace task ",j," with 'rest'"); end-if
ag_list_sol("swap",j) += {time_count}
ag_roles_changed += {j}
else
list_sol("swap",swap_emp) += [j]

                                !; if(swap_task = "rest") then writeln("At
    time ",time_count," item ",j," remains unchanged (time is after block
   ends)"); end-if
end-if
end-if
end-do

else
forall(t in block_start..block_end) do
if(t > 0) then ag_list_sol("swap",task_extend) += {t}; end-if
end-do
ag_roles_changed += {task_extend}

if(swap_task <> "rest") then
forall(t in swap_block_start..swap_block_end) ag_list_sol("swap",swap_task
   ) -= {t}
ag_roles_changed += {swap_task}
end-if
```

```
end-if


! Check if this solution is "tabu":
to_check_tabu := "swap"
check_tabu

if(tabu = true) then
no_tabu := no_tabu + 1
else

to_calculate := "swap"
calculate_cost
if(feasible = true) then
swapping_cost := total_cost("swap") - total_cost("current")
if(total_cost("swap") < total_cost(accept_rule)) then !!!!!!!!   Decision
    rule for accepting
do_swap := true
           !!!!!!!!              swap procedure
no_nonreduce := 0

emps_to_update := emps_changed
update_swaps_and_changes

else
if(candidate_exist = false or total_cost("swap") < total_cost("candidate")
    ) then
candidate_exist := true
writeln("New candidate solution using swap procedure (change in cost = ",
    swapping_cost,")")
writeln(" Details - Emp: ",emp_extend," \tRole: ",task_extend,"\tStart: ",
    block_start,"\tEnd: ",block_end)
writeln("Swap with - Emp: ",swap_emp," \tRole: ",swap_task,"\tStart: ",
    swap_block_start,"\tEnd: ",swap_block_end)
writeln

transfer_sol_from := "swap"
```

```
transfer_sol_to := "candidate"
transfer_solution
candidate_cost := swapping_cost
candidate_emps := {emp_extend}
if(swap_emp <> "AGENCY") then candidate_emps += {swap_emp}; end-if
end-if
end-if
end-if
end-if


end-procedure




! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

procedure evaluate_swap

ag_swappable := true
forall(t in block_start..block_end | t > 0) do
if(eligable("AGENCY",task_extend,t) < 1) then ag_swappable := false; end-
    if
end-do

if(ag_swappable = true) then
forall(r in ALL_ROLES | r <> task_extend) do

if(do_swap = false) then
swap_emp := ""
swap_task := ""
swap_block_start := 0
swap_block_end := 0
swap_block_len := 0
```

```
swap_find_time := 0
swap_block_found := false
too_early := false
swap_allowed := true
end-if

forall(t in TIME) do

if(do_swap = false) then

swap_new_block := FALSE
if(swap_block_found = FALSE and t in ag_list_sol("current",r) and t <=
   swap_block_latest) then
if(t < swap_block_earliest) then too_early := true; end-if
if((sum(v in VESSELS | r in ROLES(v)) starting(emp_extend,v)) > 0 and t >=
    2 and t <= min_rest(emp_extend)) then too_early := true; end-if
if((sum(v in VESSELS | r not in ROLES(v)) starting(emp_extend,v)) > 0 and
   t <= min_rest(emp_extend)) then too_early := true; end-if
if(eligable(emp_extend,r,t) < 1) then
swap_allowed := false
end-if
swap_new_block := TRUE
swap_emp := "AGENCY"
swap_task := r
swap_block_start := t


elif(swap_block_found = TRUE and t in ag_list_sol("current",r)) then

if(t in ag_crewchange("current",r)) then
swap_block_end := t-1

swap_block_len := (swap_block_end - swap_block_start) +1
if(too_early = false and swap_allowed = true) then
if(swap_block_len <= max_work(emp_extend)) then
swap_calculation
end-if
```

```
end-if


if(t <= swap_block_latest and do_swap = false) then
! also, new block is starting...
swap_task := r
swap_block_start := t

if(t < swap_block_earliest) then too_early := true
elif((sum(v in VESSELS | r in ROLES(v)) starting(emp_extend,v)) > 0 and t
   >= 2 and t <= min_rest(emp_extend)) then too_early := true
elif((sum(v in VESSELS | r not in ROLES(v)) starting(emp_extend,v)) > 0
   and t <= min_rest(emp_extend)) then too_early := true
else too_early := false
end-if
if(eligable(emp_extend,r,t) < 1) then swap_allowed := false
else swap_allowed := true
end-if

swap_block_end := 0
swap_block_len := 0

else
swap_block_found := false
end-if

else
if(t > swap_block_latest) then
swap_block_found := false
else
if(eligable(emp_extend,r,t) < 1) then
swap_allowed := false
end-if
end-if
end-if

elif(swap_block_found = TRUE and t not in ag_list_sol("current",r)) then
```

```
swap_block_end := t-1

swap_block_len := (swap_block_end - swap_block_start) +1
if(too_early = false and swap_allowed = true) then
if(swap_block_len <= max_work(emp_extend)) then
swap_calculation
end-if
end-if

swap_block_found := FALSE
if(do_swap = false) then     ! and reset...
too_early := false
swap_allowed := true
swap_block_start := 0
swap_block_end := 0
swap_block_len := 0
swap_emp := ""
swap_task := ""
end-if
end-if

if(swap_new_block = TRUE) then
swap_block_found := TRUE
swap_new_block := FALSE
end-if

! if we are at the end of the planning period, conclude any remaining
   blocks
if(t = WEEKS_TO_PLAN and swap_block_found = TRUE) then

if(eligable(emp_extend,r,t) < 1) then
swap_allowed := false
end-if

swap_block_end := t
swap_block_len := (swap_block_end - swap_block_start) +1
```

```
if(too_early = false and swap_allowed = true) then
if(swap_block_len <= max_work(emp_extend)) then
swap_calculation
end-if
end-if

end-if
end-if

end-do
end-do

! could also potentially "swap" with an un-used Agency employee
if(do_swap = false) then
swap_emp := "AGENCY"
swap_task := "rest"
swap_block_start := block_start
swap_block_end := block_end
swap_calculation
end-if

end-if
```

```
forall(e in REG_EMP) do
swappable_emp(e) := true

if(e = emp_extend) then swappable_emp(e) := false; end-if
if(block_len > max_work(e)) then swappable_emp(e) := false; end-if
if(block_start <= rest_zero(e)) then swappable_emp(e) := false; end-if

forall(i in block_start..block_end | i > 0) do
if(eligable(e,task_extend,i) < 1) then swappable_emp(e) := false; end-if
end-do
```

```
if((sum(v in VESSELS | task_extend not in ROLES(v)) starting(e,v)) > 0 and
     block_start <= min_rest(e)) then swappable_emp(e) := false; end-if
if((sum(v in VESSELS | task_extend in ROLES(v)) starting(e,v)) > 0) then
if(block_start >= 2 and block_start <= min_rest(e)) then swappable_emp(e)
    := false; end-if
if(block_start <= 1 and (block_len + work_zero(e)) > max_work(e)) then
    swappable_emp(e) := false; end-if
end-if
end-do


!       forall(e in ordered_list | swappable_emp(e) = true and (e not in
    swaps_examined(emp_extend) or do_kick = true)) do
forall(e in ordered_list | swappable_emp(e) = true and (e not in
    swaps_examined(emp_extend)) ) do


if(do_swap = false) then

swap_emp := ""
swap_task := ""
swap_vessel := ""
swap_block_start := 0
swap_block_end := 0
swap_block_len := 0

swap_find_time := 0
swap_block_found := false
too_early := false
swap_allowed := true
all_rest := true
end-if

new_list := list_sol("current",e)

while(do_swap = false and getsize(new_list) > 0) do
```

```
!              forall(j in new_list) do
!                     if(do_swap = false) then
j := getfirst(new_list)
cuthead(new_list,1)

swap_find_time := swap_find_time + 1                        !; writeln("\
   tTime: ",swap_find_time,", j = ",j)

if(j <> "rest" and swap_find_time >= swap_block_earliest and
   swap_find_time <= swap_block_latest) then
all_rest := false
end-if

swap_new_block := FALSE
if(swap_block_found = FALSE and j <> "rest" and swap_find_time <=
   swap_block_latest) then
if(swap_find_time < swap_block_earliest) then too_early := true; end-if
if((sum(v in VESSELS | j in ROLES(v)) starting(emp_extend,v)) > 0 and
   swap_find_time >= 2 and swap_find_time <= min_rest(emp_extend)) then
   too_early := true; end-if
if((sum(v in VESSELS | j not in ROLES(v)) starting(emp_extend,v)) > 0 and
   swap_find_time <= min_rest(emp_extend)) then too_early := true; end-if
if(eligable(emp_extend,j,swap_find_time) < 1) then
swap_allowed := false
end-if

swap_new_block := TRUE
swap_emp := e
swap_task := j
swap_block_start := swap_find_time

forall(v in VESSELS, k in ROLES(v)) do
if(j = k) then
swap_vessel := v
end-if
end-do
```

```
elif(swap_block_found = TRUE and j <> "rest") then

if(j <> swap_task) then
link_to_vessel := FALSE

if(swap_vessel <> "") then
forall(k in ROLES(vessel_extend)) do
if(j = k) then
link_to_vessel := TRUE
end-if
end-do
end-if

if(link_to_vessel) = TRUE then
if(swap_find_time > swap_block_latest) then
swap_block_found := false
else
if(eligable(emp_extend,j,swap_find_time) < 1) then
swap_allowed := false
else
swap_task := j
end-if
end-if
else
swap_block_end := swap_find_time-1

swap_block_len := (swap_block_end - swap_block_start) +1
if(too_early = false and swap_allowed = true) then
if(swap_block_len <= max_work(emp_extend)) then
swap_calculation
end-if
end-if

if(swap_find_time <= swap_block_latest and do_swap = false) then
! also, new block is starting...
swap_task_extend := j
```

```
swap_block_start := swap_find_time

if(swap_find_time < swap_block_earliest) then too_early := true
elif((sum(v in VESSELS | j in ROLES(v)) starting(emp_extend,v)) > 0 and
    swap_find_time >= 2 and swap_find_time <= min_rest(emp_extend)) then
    too_early := true
elif((sum(v in VESSELS | j not in ROLES(v)) starting(emp_extend,v)) > 0
    and swap_find_time <= min_rest(emp_extend)) then too_early := true
else too_early := false
end-if
if(eligable(emp_extend,j,swap_find_time) < 1) then swap_allowed := false
else swap_allowed := true
end-if

forall(v in VESSELS, k in ROLES(v)) do
if(j = k) then
swap_vessel := v
end-if
end-do

swap_block_end := 0
swap_block_len := 0

else
swap_block_found := false
end-if

end-if
else
if(swap_find_time > swap_block_latest) then
swap_block_found := false
else
if(swap_allowed = true and eligable(emp_extend,j,swap_find_time) < 1) then
swap_allowed := false
end-if

end-if
```

```
end-if


elif(swap_block_found = TRUE and j = "rest") then
swap_block_end := swap_find_time-1

swap_block_len := (swap_block_end - swap_block_start) +1
if(too_early = false and swap_allowed = true) then
if(swap_block_len <= max_work(emp_extend)) then
swap_calculation
end-if
end-if

swap_block_found := FALSE
if(do_swap = false) then
! and reset...
too_early := false
swap_allowed := true
swap_block_start := 0
swap_block_end := 0
swap_block_len := 0
swap_emp := ""
swap_task := ""
swap_vessel := ""
end-if
end-if

if(swap_new_block = TRUE) then
swap_block_found := TRUE
swap_new_block := FALSE
end-if

! if we are at the end of the planning period, conclude any remaining
   blocks
if(swap_find_time = WEEKS_TO_PLAN and swap_block_found = TRUE) then

if(eligable(emp_extend,j,swap_find_time) < 1) then
```

```
swap_allowed := false
end-if

swap_block_end := swap_find_time
swap_block_len := (swap_block_end - swap_block_start) +1

if(too_early = false and swap_allowed = true) then
if(swap_block_len <= max_work(emp_extend)) then
swap_calculation
end-if
end-if

end-if

!      end-if

end-do

if(do_swap = false and all_rest = true) then
swap_emp := e
swap_task := "rest"
swap_block_start := block_start
swap_block_end := block_end
swap_calculation
end-if

end-do


end-procedure




!-------------------------------------------------------------------------------
```

```
procedure evaluate_block
! Examine the block which has been identified as being useable


extend_cost_bkwd := 0
extend_cost_fwd := 0
swapping_cost := 0
do_extend_bkwd := FALSE
do_extend_fwd := FALSE
do_swap := FALSE

if(max_work(emp_extend) - block_len > 0) then
max_bkwd_extend := max_work(emp_extend) - block_len
max_fwd_extend := max_work(emp_extend) - block_len
else
max_bkwd_extend := 0
max_fwd_extend := 0
end-if

! Can the block be extended backwards?
if(max_bkwd_extend > WEEKS_TO_PLAN - block_end) then max_bkwd_extend :=
    WEEKS_TO_PLAN - block_end; end-if
if(max_bkwd_extend > 0) then
x := max_bkwd_extend
forall(i in 1..x) do
if(eligable(emp_extend, task_extend, block_end+i) < 1 or required(
    task_extend, block_end+i) < 1) then
if(max_bkwd_extend >= i) then max_bkwd_extend := i-1; end-if
end-if
end-do
end-if
if(block_end = 0 and rest_zero(emp_extend) > 0) then max_bkwd_extend := 0;
     end-if

if(max_bkwd_extend > 0) then
evaluate_backwards
end-if
```

```
! Can the block be extended forwards?
if(do_extend_bkwd = false) then
if(max_fwd_extend > block_start - rest_zero(emp_extend) - 1) then
   max_fwd_extend := block_start - rest_zero(emp_extend) - 1; end-if
if((sum(v in VESSELS | task_extend not in ROLES(v)) starting(emp_extend,v)
   ) > 0) then
if(max_fwd_extend > block_start - min_rest(emp_extend) - 1) then
   max_fwd_extend := block_start - min_rest(emp_extend) - 1; end-if
end-if
if(max_fwd_extend > 0) then
x := max_fwd_extend
forall(i in 1..x) do
if(max_fwd_extend >= i) then
if((sum(v in VESSELS | task_extend in ROLES(v)) starting(emp_extend,v)) >
   0 and block_start - i > 1 and block_start - i - 1 < min_rest(emp_extend
   )) then max_fwd_extend := i-1; end-if
if(eligable(emp_extend, task_extend, block_start-i) < 1 or required(
   task_extend, block_start-i) < 1) then max_fwd_extend := i-1; end-if
end-if
end-do
end-if

if(max_fwd_extend > 0) then
evaluate_forwards
end-if
end-if


! Can the block be swapped with another?
if(do_extend_bkwd = false and do_extend_fwd = false) then
if(block_start > 0) then
evaluate_swap
end-if
end-if
```

```
! Now, assuming a decision has been made to extend (forward or backward),
    we can deal with the actual extension:
if(do_extend_bkwd = true) then
writeln("Backward extension has been selected (change in cost = ",
    extend_cost_bkwd,")")
writeln("Details - Emp: ",emp_extend,"\tRole: ",task_extend,"\tOrig start:
     ",block_start,"\tOrig End: ",block_end,"\tExt length: ",
    extend_len_bkwd)
transfer_sol_from := "backward"
no_bkwd := no_bkwd + 1
elif(do_extend_fwd = true) then
writeln("Forward extension has been selected (change in cost = ",
    extend_cost_fwd,")")
writeln("Details - Emp: ",emp_extend,"\tRole: ",task_extend,"\tOrig start:
     ",block_start,"\tOrig End: ",block_end,"\tExt length: ",extend_len_fwd
    )
transfer_sol_from := "forward"
no_fwd := no_fwd + 1
elif(do_swap = true) then
writeln("Swap has been selected (change in cost = ",swapping_cost,")")
writeln(" Details - Emp: ",emp_extend," \tRole: ",task_extend,"\tStart: ",
    block_start,"\tEnd: ",block_end)
writeln("Swap with - Emp: ",swap_emp," \tRole: ",swap_task,"\tStart: ",
    swap_block_start,"\tEnd: ",swap_block_end)
transfer_sol_from := "swap"
no_swap := no_swap + 1
end-if


if(do_extend_bkwd = true or do_extend_fwd = true or do_swap = true) then

forall(e in REG_EMP) tabu_sol(e) := list_sol("current",e)
forall(r in ALL_ROLES) ag_tabu_sol(r) := ag_list_sol("current",r)
transfer_sol_to := "current"
transfer_solution
```

```
update_done := true
writeln("Current solution cost is ",total_cost("current"))

compare_to_best

end-if

end-procedure




!-------------------------------------------------------------------------------


procedure find_useable_block
! Looking for blocks which can be extended or swapped:

update_done := FALSE
candidate_exist := false
candidate_emps := {}

forall(e in short_ordered_list) do

if(update_done = false) then

emp_extend := ""
task_extend := ""
block_start := 0
block_end := 0
block_len := 0


block_found := FALSE
forall(v in VESSELS) do
if(starting(e,v) = 1) then
block_found := TRUE
```

```
emp_extend := e
block_start := 1-work_zero(e)
swap_block_earliest := 1
vessel_extend := v
end-if
if(block_found = FALSE) then
vessel_extend := ""
end-if
end-do
end-if

find_time := 0
current_list := list_sol("current",e)
while(update_done = false and getsize(current_list) > 0) do
!              forall(j in current_list) do
!                     if(update_done = false) then
j := getfirst(current_list)
cuthead(current_list,1)



find_time := find_time + 1

new_block := FALSE
if(block_found = FALSE and j <> "rest") then
new_block := TRUE
emp_extend := e
task_extend := j
block_start := find_time
if(find_time = 1) then swap_block_earliest := 1
elif(find_time-1 <= rest_zero(e)) then swap_block_earliest := find_time
else swap_block_earliest := find_time -1
end-if

forall(v in VESSELS, k in ROLES(v)) do
if(j = k) then
vessel_extend := v
end-if
```

```
end-do

elif(block_found = TRUE and j <> "rest") then
if(j <> task_extend) then
link_to_vessel := FALSE

if(vessel_extend <> "") then
forall(k in ROLES(vessel_extend)) do
if(j = k) then
link_to_vessel := TRUE
end-if
end-do
end-if

if(link_to_vessel) = TRUE then
task_extend := j
else
block_end := find_time-1
swap_block_latest := find_time-1

block_len := (block_end - block_start) +1
if(task_extend <> "") then
evaluate_block
else
if(block_len < max_work(e)) then
forall(k in ROLES(vessel_extend)) do
task_extend := k
evaluate_block
end-do
end-if
end-if


! also, new block is starting...
task_extend := j
block_start := find_time
swap_block_earliest := find_time
```

```
forall(v in VESSELS, k in ROLES(v)) do
if(j = k) then
vessel_extend := v
end-if
end-do

block_end := 0
block_len := 0

end-if

end-if

elif(block_found = TRUE and j = "rest") then
block_end := find_time-1
swap_block_latest := find_time

block_len := (block_end - block_start) +1
if(task_extend <> "") then
evaluate_block
else
if(block_len < max_work(e)) then
forall(k in ROLES(vessel_extend)) do
task_extend := k
evaluate_block
end-do
end-if
end-if


! and reset...
block_found := FALSE
block_start := 0
block_end := 0
block_len := 0
emp_extend := ""
```

```
task_extend := ""
vessel_extend := ""

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(find_time = WEEKS_TO_PLAN and block_found = TRUE) then

block_end := find_time
swap_block_latest := find_time

block_len := (block_end - block_start) +1
if(task_extend <> "") then
evaluate_block
else
if(block_len < max_work(e)) then
forall(k in ROLES(vessel_extend)) do
task_extend := k
evaluate_block
end-do
end-if
end-if

end-if

!                     end-if

end-do


if(update_done = false) then
```

```
forall(f in REG_EMP | f <> e) swaps_examined(e) += {f}
end-if

end-do


! If no improving solution has been found for any of the examined
   employees,
!      implement the best non-improving solution (if one exists):
if(update_done = false and candidate_exist = true) then

writeln("Best non-improving candidate has been selected")

forall(e in REG_EMP) tabu_sol(e) := list_sol("current",e)
forall(r in ALL_ROLES) ag_tabu_sol(r) := ag_list_sol("current",r)

if(candidate_cost < 0) then no_nonreduce := 0
else no_nonreduce := no_nonreduce + 1
end-if

transfer_sol_from := "candidate"
transfer_sol_to := "current"
transfer_solution
update_done := true
no_cand := no_cand + 1

compare_to_best

emps_to_update := candidate_emps
update_swaps_and_changes

writeln("Current solution cost is ",total_cost("current"))
end-if


end-procedure
```

```
!-------------------------------------------------------------------------------------

procedure random_kick

writeln
    ("-------------------------------------------------------------------------------

writeln("As no improvement has been made recently, a 'kick' will be
    carried out:")

update_done := false
while(update_done = false) do

emps_changed := {}
ag_roles_changed := {}

random_emp := integer( (getsize(REG_EMP)*random) + 1)
y := 0
forall(e in REG_EMP) do
y := y +1
if(y = random_emp) then kick_emp := e; end-if
end-do

random_task := integer( (getsize(ALL_ROLES)*random) + 1)
y := 0
forall(r in ALL_ROLES) do
y := y +1
if(y = random_task) then kick_task := r; end-if
end-do

random_length := integer( (5*random) + 1)

random_time := integer( ((WEEKS_TO_PLAN - (random_length-1))*random) + 1)

kick_start := random_time
```

```
kick_end := random_time + random_length - 1

kick_feas := true
if(kick_start <= rest_zero(kick_emp)) then kick_feas := false; end-if
if((sum(v in VESSELS | kick_task not in ROLES(v)) starting(kick_emp,v)) >
    0 and kick_start <= min_rest(kick_emp)) then kick_feas := false; end-if
if((sum(v in VESSELS | kick_task in ROLES(v)) starting(kick_emp,v)) > 0)
    then
if(kick_start >= 2 and kick_start <= min_rest(kick_emp)) then kick_feas :=
     false; end-if
if(kick_start <= 1 and (random_length + work_zero(kick_emp)) > max_work(
    kick_emp)) then kick_feas := false; end-if
end-if
forall(t in kick_start..kick_end) do
if(eligable(kick_emp, kick_task, t) < 1) then kick_feas := false; end-if
end-do


if(kick_feas = true) then

forall(r in ALL_ROLES) ag_list_sol("kick",r) := ag_list_sol("current",r)
forall(t in kick_start..kick_end) do
if(t in ag_list_sol("kick",kick_task)) then
ag_list_sol("kick",kick_task) -= {t}
ag_roles_changed += {kick_task}
end-if
end-do

forall(e in REG_EMP) do
list_sol("kick",e) := []

if(e <> kick_emp) then
kick_count := 0
rest_count := 0
if((sum(v in VESSELS | kick_task in ROLES(v)) starting(e,v)) > 0) then
if(kick_start = 1) then rest_count := min_rest(e); end-if
end-if
```

```
forall(j in list_sol("current",e)) do
kick_count := kick_count + 1
if(kick_count = kick_start - 1 and j = kick_task) then
list_sol("kick",e) += [j]
rest_count := min_rest(e)
elif(kick_count = kick_start) then
if(j = kick_task) then
list_sol("kick",e) += ["rest"]
emps_changed += {e}
rest_count := rest_count - 1
else
list_sol("kick",e) += [j]
rest_count := 0
end-if
elif(kick_count > kick_start and kick_count <= kick_end) then
if(j = kick_task) then
list_sol("kick",e) += ["rest"]
emps_changed += {e}
rest_count := rest_count - 1
elif(j <> "rest") then
if(rest_count > 0) then
list_sol("kick",e) += ["rest"]
ag_list_sol("kick",j) += {kick_count}
ag_roles_changed += {j}
rest_count := rest_count - 1
else
list_sol("kick",e) += [j]
end-if
else
list_sol("kick",e) += [j]
rest_count := rest_count - 1
end-if
elif(kick_count > kick_end and j <> "rest" and rest_count > 0) then
list_sol("kick",e) += ["rest"]
ag_list_sol("kick",j) += {kick_count}
ag_roles_changed += {j}
```

```
rest_count := rest_count - 1
else
list_sol("kick",e) += [j]
end-if
end-do
else
emps_changed += {e}
kick_count := 0
forall(j in list_sol("current",e)) do
kick_count := kick_count + 1
if(kick_count < kick_start - min_rest(e)) then
list_sol("kick",e) += [j]
elif(kick_count < kick_start) then
list_sol("kick",e) += ["rest"]
if(j <> "rest") then
ag_list_sol("kick",j) += {kick_count}
ag_roles_changed += {j}
end-if
elif(kick_count <= kick_end) then
list_sol("kick",e) += [kick_task]
if(j <> "rest" and j <> kick_task) then
ag_list_sol("kick",j) += {kick_count}
ag_roles_changed += {j}
end-if
elif(kick_count <= kick_end + min_rest(e)) then
list_sol("kick",e) += ["rest"]
if(j <> "rest") then
ag_list_sol("kick",j) += {kick_count}
ag_roles_changed += {j}
end-if
else
list_sol("kick",e) += [j]
end-if
end-do
end-if
end-do
```

```
! ensure that this kick is not 'tabu':
to_check_tabu := "kick"
check_tabu

if(tabu = true) then
no_tabu := no_tabu + 1
else

to_calculate := "kick"
calculate_cost
if(feasible = true) then
writeln("Kick selected - place ",kick_emp," in role ",kick_task," from
    time ",kick_start," to ",kick_end)

forall(e in REG_EMP) tabu_sol(e) := list_sol("current",e)
forall(r in ALL_ROLES) ag_tabu_sol(r) := ag_list_sol("current",r)
transfer_sol_from := "kick"
transfer_sol_to := "current"
transfer_solution

last_kick_time := iteration
update_done := true
no_nonreduce := 0

compare_to_best

emps_to_update := emps_changed
update_swaps_and_changes

writeln("Current solution cost is ",total_cost("current"))

end-if
end-if
end-if

end-do
```

```
end-procedure




!-------------------------------------------------------------------------------


procedure sort_employee_list

if(order_rule not in ORD_RULES) then
writeln("ERROR - incorrect option selected for order rule. List will be
    RANDOMIZED.")
order_rule := "random"
end-if


forall(e in REG_EMP) order_number(e) := random
ordered_list := []
short_ordered_list := []
added_set := {}

forall(x in REG_EMP) do
min_no := 1
if(order_rule = "earliest") then change_no := iteration
else change_no := 0
end-if
min_emp := ""

forall(e in REG_EMP | e not in added_set) do
if((order_rule = "earliest" and last_changed(e) < change_no) or (
    order_rule = "latest" and last_changed(e) > change_no)) then
change_no := last_changed(e)
min_no := order_number(e)
```

```
min_emp := e
elif((order_rule = "earliest" and last_changed(e) = change_no) or (
    order_rule = "latest" and last_changed(e) = change_no) or (order_rule =
     "random")) then
if(order_number(e) < min_no) then
min_no := order_number(e)
min_emp := e
end-if
end-if
end-do

if(getsize(short_ordered_list) < short_list_fraction*getsize(REG_EMP))
    then short_ordered_list += [min_emp]; end-if
ordered_list += [min_emp]
added_set += {min_emp}
end-do

writeln("Employee list sorted using '",order_rule,"' ordering rule.")
writeln("    List is: ",ordered_list)
writeln("Short list is: ",short_ordered_list)
writeln

end-procedure




!-------------------------------------------------------------------------------------


procedure initialise

! calculate the list representation

forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) taskbased_sol(e,r,t):=
    allocate_dual(e,r,t)
```

```
emps_changed := {}
forall(e in REG_EMP) do
list_sol("current",e) := []
list_sol("best",e) := []
forall(x in best_index) best_sols(x,e) := []
forall(t in TIME) do
added_t := FALSE
forall(r in ALL_ROLES) do
if(added_t = FALSE and taskbased_sol(e,r,t) = 1) then
list_sol("current",e) += [r]
list_sol("best",e) += [r]
added_t := TRUE
end-if
end-do
if(added_t = FALSE) then
list_sol("current",e) += ["rest"]
list_sol("best",e) += ["rest"]
end-if
end-do
emps_changed += {e}
swaps_examined(e) := {}
end-do

ag_roles_changed := {}
forall(r in ALL_ROLES) do
ag_list_sol("current",r) := {}
ag_list_sol("best",r) := {}
forall(x in best_index) ag_best_sols(x,r) := {}
forall(t in TIME | required(r,t) > 0) do
if(sum(e in REG_EMP) taskbased_sol(e,r,t) < 1) then
ag_list_sol("current",r) += {t}
ag_list_sol("best",r) += {t}
end-if
end-do
ag_roles_changed += {r}
end-do
```

```
! and calculate the initial cost

to_calculate := "current"
calculate_cost
initial_cost := total_cost("current")

transfer_sol_from := "current"
transfer_sol_to := "best"
transfer_solution
best_sol_time := 0

number_best := 1
forall(e in REG_EMP) best_sols(1,e) := list_sol("best",e)
forall(r in ALL_ROLES) ag_best_sols(1,r) := ag_list_sol("best",r)

writeln
writeln("Initial solution:")
writeln
forall(e in REG_EMP) writeln(e,"\t",list_sol("current",e),"\tCost: ",
    emp_cost("current",e))
writeln
write("AGENCY")
forall(r in ALL_ROLES) do
write("\t",r,"\t",ag_list_sol("current",r),"\tCost: ",ag_cost("current",r)
    )
if(ag_list_sol("current",r) <> {}) then
write("\t\t(Crew changes take place ahead of the following weeks: ",
    ag_crewchange("current",r),")")
end-if
write("\n")
end-do
writeln
writeln
writeln("Cost for initial solution is:\t",initial_cost)
```

```
! and set initial values:

no_fwd := 0
no_bkwd := 0
no_swap := 0
no_cand := 0
no_kick := 0
no_tabu := 0
no_infeas := 0

terminate := false
iteration := 0
last_kick_time := 0
no_nonreduce := 0
forall(e in REG_EMP) last_changed(e) := 0

end-procedure



!-------------------------------------------------------------------------------


! MAIN PROGRAMME!!

writeln("Heuristic Started")



fopen(OUTPUTFILE_heuristic, F_OUTPUT)
initialise

! Apply heuristics...
writeln
writeln
    ("-------------------------------------------------------------------------------
```

```
writeln("Applying heuristics...")
writeln

while(terminate = false) do
iteration := iteration +1

if(no_nonreduce >= 1) then
order_rule := "earliest"
                          ! prioritize employees who have been examined
   less recently
elif((iteration - last_kick_time > 1) or (iteration > 1 and last_kick_time
    = 0)) then
order_rule := "latest"
                          ! prioritize employees who have recently been
   examined
else
order_rule := "random"
                          ! if a kick has just been carried out or the
   algorithm is just starting, randomize the list
end-if
sort_employee_list

find_useable_block

writeln
writeln("End of iteration ",iteration)
writeln("\t(Best solution value so far: ",total_cost("best"),")")
writeln("\t(There is/are ",number_best," solution(s) found with this value
   )")
writeln
if(update_done = false) then
writeln("No useable block was found - terminate programme")
terminate := true
else
current_time := gettime
if(iteration = max_iteration) then
writeln("Iteration limit now reached - terminate programme")
```

```
terminate := true
else
writeln("Running time so far: ",current_time - prog_starttime)
if(current_time - prog_starttime >= max_runtime) then
writeln("Time limit now exceeded - terminate programme")
terminate := true
end-if
end-if
end-if

if((iteration - best_sol_time) >= 4 and terminate = false) then
if(last_kick_time = 0 or (last_kick_time > 0 and (iteration -
    last_kick_time) >= 20) or (total_cost("current") = total_cost("best")
    and no_nonreduce >= 1) or no_nonreduce >= 4) then
random_kick
no_kick := no_kick + 1
writeln("Running time so far: ",current_time - prog_starttime)
end-if
end-if

writeln
    ("-------------------------------------------------------------------------------

end-do


writeln
writeln("Best available solution:")
writeln
forall(e in REG_EMP) writeln(e,"\t",list_sol("best",e))
writeln
write("AGENCY")
forall(r in ALL_ROLES) do
write("\t",r,"\t",ag_list_sol("best",r))
if(ag_list_sol("best",r) <> {}) then
write("\t\t(with crew changes ahead of the following weeks: ",
    ag_crewchange("best",r),")")
```

```
end-if
write("\n")
end-do
writeln
writeln("Solution cost is:\t",total_cost("best"))
if(number_best > 1) then writeln("NOTE: in total there were ",number_best
    ," solutions found with this cost"); end-if




prog_endtime := gettime
writeln
writeln("-----------------------------------------------")
writeln("Total running time: ", prog_endtime - prog_starttime)
writeln
writeln("Number of iterations: ",iteration)
writeln("which were of the following types:")
writeln("\tbackward: ",no_bkwd)
writeln("\tforward: ",no_fwd)
writeln("\tswapping: ",no_swap)
writeln("\tcandidate: ",no_cand)
writeln
writeln("Best solution found at iteration: ",best_sol_time)
writeln
writeln("No of kick procedures carried out: ",no_kick)
writeln("No of tabu solutions proposed:   ",no_tabu)
writeln("No of infeasible solutions proposed: ",no_infeas)




! Calculate some stats...
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(e in REG_EMP) do
final_count := 0
forall(j in list_sol("best",e)) do
final_count := final_count + 1
forall(r in ALL_ROLES) do
if(r = j and cur_allocate(e,r,final_count) = 0) then changes_to_reg :=
   changes_to_reg +1; end-if
if(r <> j and cur_allocate(e,r,final_count) = 1) then changes_to_reg :=
   changes_to_reg +1; end-if
end-do
end-do
end-do
forall(r in ALL_ROLES, t in TIME) do
if(t in ag_list_sol("best",r)) then
number_of_AG := number_of_AG +1
if(cur_allocate("AGENCY",r,t) = 0) then changes_to_AG := changes_to_AG +1;
    end-if
else
if(cur_allocate("AGENCY",r,t) = 1) then changes_to_AG := changes_to_AG +1;
    end-if
end-if
end-do

fclose(F_OUTPUT)

forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) allocate_dual(e,r,t):=
   allocate_sol("best",e,r,t)
forall(e in REG_EMP, v in VESSELS, t in TIME) depart_dual(e,v,t):=
   depart_sol("best",e,v,t)
writeln("Solution cost is:\t",total_cost("best"))
initializations to "bin:shmem:sol"
allocate_dual depart_dual InstanceName
```

```
end-initializations

writeln("Heuristic Finished")

end-model
```

B.2 Robust Formulations

The robust model ,which is explained by equation (6.9), is applied in FICO® Xpress-MP (Mosel v3.6.0, Xpress-MP v7.7) with the code given under Appendix B.2.1.

B.2.1 Robust Formulation Against Uncertainty of Crew Availability

This sections shows the implementation suggested model for dealing with one type of uncertainty.

```
model ModelName
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver

DATAFILE := "Real time-window data - Captains.txt"
LOGFILE := "Logfile-Robust-Time-window real data.txt"
SUMMARYFILE := "Results-Robust-Time-window real data.txt"

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
Robust_eligable:dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar !gamma
linear_dv: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar !z_ijt
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_cost, depart_cost: array(REG_EMP, VESSELS, TIME) of real ! Costs of
   CHANGES TO employees boarding / leaving vessel
ag_board_cost, ag_depart_cost: array(ALL_ROLES, TIME) of real ! Costs of
   CHANGES TO agency employees boarding / leaving for a given role
work_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
end-declarations

initializations from DATAFILE
board_cost depart_cost work_cost
ag_board_cost ag_depart_cost
required eligable starting ag_starting
work_zero rest_zero
max_work min_rest ag_max_work
```

```
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

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(linear_dv(e,r,t))
end-if
end-do

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(Robust_eligable(e,r,t))
end-if
end-do


initializations from DATAFILE
extension_cost
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
Work_count_start: array(REG_EMP) of linctr
AG_work_count: array(ALL_ROLES, TIME) of linctr
AG_work_count_start: array(ALL_ROLES) of linctr
AG_work_reset: array(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
Rest_new: array(REG_EMP,ALL_ROLES,range) of linctr
Rest_new1: array(REG_EMP, 1..WEEKS_TO_PLAN-1,range) of linctr ! Added for
    recovery problem - constraints to link change, current and new values:
linear1: array(ALL_EMP, ALL_ROLES, TIME) of linctr
linear2: array(ALL_EMP, ALL_ROLES, TIME) of linctr
linear3: array(ALL_EMP, ALL_ROLES, TIME) of linctr
Depart_linking:array(REG_EMP, TIME) of linctr
Robust_criteria:array(TIME) of linctr
a:integer
b:integer
c:integer
end-declarations
!objective function- cost calculation
prog_setup_time := gettime

Total_cost := (sum(e in REG_EMP, v in VESSELS, t in TIME)((board_cost(e,v,
    t)*board(e,v,t)) + (depart_cost(e,v,t)*depart(e,v,t))) +
sum(r in ALL_ROLES, t in TIME)((ag_board_cost(r,t)*ag_rboard(r,t)) + (
    ag_depart_cost(r,t)*ag_rdepart(r,t))) +
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((work_cost(e,r,t)*allocate(e,
    r,t)))+
```

```
sum(e in GUARANTEED)((under_rate(e)*undertime(e))+ (over_rate(e)*overtime(
    e))))

!covering tasks
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))-
    sum(e in ALL_EMP)(linear_dv(e,r,t))= required(r,t)
end-do

!level of uncertainty
forall( r in ALL_ROLES, t in TIME)Robust_eligable("AGENCY",r,t)=0
forall( t in TIME ) Robust_criteria(t) := sum(e in REG_EMP,r in ALL_ROLES|
    eligable(e,r,t)>0)(Robust_eligable(e,r,t)) >=32

!linearization
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME)linear1(e,r,t):=linear_dv(e
    ,r,t)<=Robust_eligable(e,r,t)
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME)linear2(e,r,t):=linear_dv(e
    ,r,t)<=allocate(e,r,t)
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME)linear3(e,r,t):=linear_dv(e
    ,r,t)>=Robust_eligable(e,r,t)+allocate(e,r,t)-1

!overlap
forall(e in REG_EMP, t in TIME) No_overlap(e,t) := sum(r in ALL_ROLES)
    allocate(e,r,t) <= 1

!board
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)))

!depart
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))

!-------------depart linking-------------------------------------
forall(e in REG_EMP, t in TIME) Depart_linking(e,t):=sum(v in VESSELS)
   depart(e,v,t)+sum(r in ALL_ROLES) allocate(e,r,t)<=1
!-------------depart linking-------------------------------------

!agency boarding and departing
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))

!under and over time
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)

!max consecutive work
forall(e in REG_EMP|work_zero(e)>=1) Work_count_start(e) := max_work(e) >=
    work_zero(e) + sum(r in ALL_ROLES, t in 0..max_work(e)-work_zero(e))(
   allocate(e,r,t+1))
forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-max_work(e)) Work_count(e,t) :=
    max_work(e) >= sum(r in ALL_ROLES, k in 0..max_work(e))(allocate(e,r,t
   +k))

forall(r in ALL_ROLES|ag_work_zero(r)>=1) AG_work_count_start(r) :=
   ag_max_work(r) >= ag_work_zero(r) + sum( t in 0..ag_max_work(r)-
   ag_work_zero(r))(allocate("AGENCY",r,t+1))
forall(r in ALL_ROLES, t in 1..WEEKS_TO_PLAN-ag_max_work(r)) AG_work_count
   (r,t) := ag_max_work(r) >= sum( k in 0..ag_max_work(r))(allocate("
   AGENCY",r,t+k))
```

```
!rest
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))


!--------comb cut-----------------------------------------------
forall( e in REG_EMP ,r in ALL_ROLES,t in 1..rest_zero(e)) do
if(rest_zero(e)>=1)then
Rest_new(e,r,t):= allocate(e,r,t)=0
end-if
end-do


forall( e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)

if(b<=a) then
c:=b
else
c:=a
end-if
end-if
forall(y in 1..c)do
Rest_new1(e,t,y) := sum(r in ALL_ROLES)(allocate(e,r,t+y))+sum (v in
    VESSELS) depart(e,v,t)<=1
end-do
end-do
```

```
!---------comb cut---------------------


! finally, DVs 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 | exists(linear_dv(e,r,t)))
    linear_dv(e,r,t) is_binary
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(Robust_eligable(e,
   r,t))) Robust_eligable(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) 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) ag_rdepart(r,t) is_binary

forall(e in GUARANTEED) undertime(e) >= 0
forall(e in GUARANTEED) overtime(e) >= 0

setparam("XPRS_verbose",true)
minimize(Total_cost)

end-model
```

B.2.2 Robust Formulation Against Uncertainty of Crew Availability and Demand

```
model ModelName
uses "mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer solver

DATAFILE := "Real time-window data - Captains.txt"
LOGFILE := "Logfile-Robust-Time-window real data.txt"
SUMMARYFILE := "Results-Robust-Time-window real data.txt"

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
```

```
Robust_eligable:dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar  !z_ijt
linear_dv: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
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
Robust_demand:dynamic array(ALL_ROLES, TIME) of mpvar  !z_ijt

board_cost, depart_cost: array(REG_EMP, VESSELS, TIME) of real ! Costs of
    CHANGES TO employees boarding / leaving vessel
ag_board_cost, ag_depart_cost: array(ALL_ROLES, TIME) of real ! Costs of
    CHANGES TO agency employees boarding / leaving for a given role
work_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
end-declarations

initializations from DATAFILE
board_cost depart_cost work_cost
ag_board_cost ag_depart_cost
required eligable starting ag_starting
work_zero rest_zero
max_work min_rest ag_max_work

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

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(linear_dv(e,r,t))
end-if
end-do
```

```
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(Robust_eligable(e,r,t))
end-if
end-do


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
Work_count_start: array(REG_EMP) of linctr
AG_work_count: array(ALL_ROLES, TIME) of linctr
AG_work_count_start: array(ALL_ROLES) of linctr
AG_work_reset: array(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
Rest_new: array(REG_EMP,ALL_ROLES,range) of linctr
Rest_new1: array(REG_EMP, 1..WEEKS_TO_PLAN-1,range) of linctr ! Added for
   recovery problem - constraints to link change, current and new values:
linear1: array(ALL_EMP, ALL_ROLES, TIME) of linctr
linear2: array(ALL_EMP, ALL_ROLES, TIME) of linctr
linear3: array(ALL_EMP, ALL_ROLES, TIME) of linctr
Depart_linking:array(REG_EMP, TIME) of linctr
Robust_criteria:array(TIME) of linctr
a:integer
b:integer
c:integer
end-declarations
!objective function- cost calculation
```

```
prog_setup_time := gettime

Total_cost := (sum(e in REG_EMP, v in VESSELS, t in TIME)((board_cost(e,v,
    t)*board(e,v,t)) + (depart_cost(e,v,t)*depart(e,v,t))) +
sum(r in ALL_ROLES, t in TIME)((ag_board_cost(r,t)*ag_rboard(r,t)) + (
    ag_depart_cost(r,t)*ag_rdepart(r,t))) +
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)((work_cost(e,r,t)*allocate(e,
    r,t)))+
sum(e in GUARANTEED)((under_rate(e)*undertime(e))+ (over_rate(e)*overtime(
    e))))

!covering tasks
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))-
    sum(e in ALL_EMP)(linear_dv(e,r,t))= required(r,t)*(1-Robust_demand(r,t
    ))
end-do

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))-
    sum(e in ALL_EMP)(linear_dv(e,r,t))= required(r,t)+(Robust_demand(r,t))
end-do



!level of uncertainty
forall( r in ALL_ROLES, t in TIME)Robust_eligable("AGENCY",r,t)=0
forall( t in TIME ) Robust_criteria(t) := sum(e in REG_EMP,r in ALL_ROLES|
    eligable(e,r,t)>0)(Robust_eligable(e,r,t)) >=32

!upper and lower should be decided
Robust_criteria2 := sum(r in ALL_ROLES, t in TIME|required(r,t)=0)(
    Robust_demand(r,t))>=0
Robust_criteria3 := sum(r in ALL_ROLES, t in TIME|required(r,t)=0)(
    Robust_demand(r,t))<=13*25
```

```
!linearization
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME)linear1(e,r,t):=linear_dv(e
    ,r,t)<=Robust_eligable(e,r,t)
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME)linear2(e,r,t):=linear_dv(e
    ,r,t)<=allocate(e,r,t)
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME)linear3(e,r,t):=linear_dv(e
    ,r,t)>=Robust_eligable(e,r,t)+allocate(e,r,t)-1

!overlap
forall(e in REG_EMP, t in TIME) No_overlap(e,t) := sum(r in ALL_ROLES)
    allocate(e,r,t) <= 1

!board
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)))

!depart
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))

!-------------depart linking-------------------------------------
forall(e in REG_EMP, t in TIME) Depart_linking(e,t):=sum(v in VESSELS)
    depart(e,v,t)+sum(r in ALL_ROLES) allocate(e,r,t)<=1
!-------------depart linking-------------------------------------

!agency boarding and departing
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))

!under and over time
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)

!max consecutive work
forall(e in REG_EMP|work_zero(e)>=1) Work_count_start(e) := max_work(e) >=
     work_zero(e) + sum(r in ALL_ROLES, t in 0..max_work(e)-work_zero(e))(
    allocate(e,r,t+1))
forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-max_work(e)) Work_count(e,t) :=
     max_work(e) >= sum(r in ALL_ROLES, k in 0..max_work(e))(allocate(e,r,t
    +k))

forall(r in ALL_ROLES|ag_work_zero(r)>=1) AG_work_count_start(r) :=
    ag_max_work(r) >= ag_work_zero(r) + sum( t in 0..ag_max_work(r)-
    ag_work_zero(r))(allocate("AGENCY",r,t+1))
forall(r in ALL_ROLES, t in 1..WEEKS_TO_PLAN-ag_max_work(r)) AG_work_count
    (r,t) := ag_max_work(r) >= sum( k in 0..ag_max_work(r))(allocate("
    AGENCY",r,t+k))

!rest
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))


!--------comb cut-----------------------------------------------
```

```
forall( e in REG_EMP ,r in ALL_ROLES,t in 1..rest_zero(e)) do
if(rest_zero(e)>=1)then
Rest_new(e,r,t):= allocate(e,r,t)=0
end-if
end-do


forall( e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)

if(b<=a) then
c:=b
else
c:=a
end-if
end-if
forall(y in 1..c)do
Rest_new1(e,t,y) := sum(r in ALL_ROLES)(allocate(e,r,t+y))+sum (v in
    VESSELS) depart(e,v,t)<=1
end-do
end-do
!---------comb cut---------------------


! finally, DVs 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 | exists(linear_dv(e,r,t)))
     linear_dv(e,r,t) is_binary
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(Robust_eligable(e,
    r,t))) Robust_eligable(e,r,t) is_binary

forall(r in ALL_ROLES, t in TIME) Robust_demand(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) 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) ag_rdepart(r,t) is_binary

forall(e in GUARANTEED) undertime(e) >= 0
forall(e in GUARANTEED) overtime(e) >= 0

setparam("XPRS_verbose",true)
minimize(Total_cost)

end-model
```

B.2.3 Robust Formulation with Cost Uncertainty

```
model ModelName
uses "mmrobust","mmxprs", "mmsystem"; !gain access to the Xpress-Optimizer
    solver

parameters
TC=0.3
end-parameters
! 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
DATAFILE := "Real time-window data - Captains.txt"
LOGFILE := "Logfile-Robust Cost-Time-window real data.txt"
SUMMARYFILE := "Results-Robust-Cost-Time-window real data.txt"



! Set parameters relating to running time and acceptable optimality gap:
setparam("XPRS_maxtime", -3600)

!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

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
Robust_eligable:dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
linear_dv: dynamic array(ALL_EMP, ALL_ROLES, TIME) of mpvar
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_cost, depart_cost: array(REG_EMP, VESSELS, TIME) of real ! Costs of
    CHANGES TO employees boarding / leaving vessel
ag_board_cost, ag_depart_cost: array(ALL_ROLES, TIME) of real ! Costs of
    CHANGES TO agency employees boarding / leaving for a given role
work_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
end-declarations

initializations from DATAFILE
board_cost depart_cost work_cost
ag_board_cost ag_depart_cost
required eligable starting ag_starting
work_zero rest_zero
max_work min_rest ag_max_work

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
```

```
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(linear_dv(e,r,t))
end-if
end-do

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(Robust_eligable(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_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
end-declarations

initializations from DATAFILE
extension_cost
end-initializations



declarations
Total_cost: robustctr
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
Work_count_start: array(REG_EMP) 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_count_start: array(ALL_ROLES) 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
Rest_new: array(REG_EMP,ALL_ROLES,range) of linctr
Rest_new1: array(REG_EMP, 1..WEEKS_TO_PLAN-1,range) of linctr ! Added for
   recovery problem - constraints to link change, current and new values:
linear1: array(ALL_EMP, ALL_ROLES, TIME) of linctr
linear2: array(ALL_EMP, ALL_ROLES, TIME) of linctr
linear3: array(ALL_EMP, ALL_ROLES, TIME) of linctr
Depart_linking:array(REG_EMP, TIME) of linctr
Robust_criteria:array(TIME) of linctr
a:integer
b:integer
c:integer
transportation_cost: array(ALL_EMP, ALL_ROLES, TIME) of uncertain

end-declarations
!objective function- cost calculation
prog_setup_time := gettime



forall(e in ALL_EMP, r in ALL_ROLES, t in TIME) do
transportation_cost(e,r,t)<=work_cost(e,r,t)*TC
transportation_cost(e,r,t)>=0
end-do


Total_cost := (sum(e in REG_EMP, v in VESSELS, t in TIME)((board_cost(e,v,
   t)*board(e,v,t)) + (depart_cost(e,v,t)*depart(e,v,t))) +
```

```
sum(r in ALL_ROLES, t in TIME)((ag_board_cost(r,t)*ag_rboard(r,t)) + (
    ag_depart_cost(r,t)*ag_rdepart(r,t))) +
sum(e in ALL_EMP, r in ALL_ROLES, t in TIME)(((work_cost(e,r,t)+
    transportation_cost(e,r,t))*allocate(e,r,t)))+
sum(e in GUARANTEED)((under_rate(e)*undertime(e))+ (over_rate(e)*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))-
    sum(e in ALL_EMP)(linear_dv(e,r,t))= required(r,t)
end-do

forall( r in ALL_ROLES, t in TIME)Robust_eligable("AGENCY",r,t)=0
forall( t in TIME ) Robust_criteria(t) := sum(e in REG_EMP,r in ALL_ROLES|
    eligable(e,r,t)>0)(Robust_eligable(e,r,t)) >=32
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME)linear1(e,r,t):=linear_dv(e
    ,r,t)<=Robust_eligable(e,r,t)
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME)linear2(e,r,t):=linear_dv(e
    ,r,t)<=allocate(e,r,t)
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME)linear3(e,r,t):=linear_dv(e
    ,r,t)>=Robust_eligable(e,r,t)+allocate(e,r,t)-1

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))

!-------------depart linking-------------------------------------
forall(e in REG_EMP, t in TIME) Depart_linking(e,t):=sum(v in VESSELS)
    depart(e,v,t)+sum(r in ALL_ROLES) allocate(e,r,t)<=1
!-------------depart linking-------------------------------------

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_zero(e)>=1) Work_count_start(e) := max_work(e) >=
     work_zero(e) + sum(r in ALL_ROLES, t in 0..max_work(e)-work_zero(e))(
    allocate(e,r,t+1))
forall(e in REG_EMP, t in 1..WEEKS_TO_PLAN-max_work(e)) Work_count(e,t) :=
     max_work(e) >= sum(r in ALL_ROLES, k in 0..max_work(e))(allocate(e,r,t
    +k))

forall(r in ALL_ROLES|ag_work_zero(r)>=1) AG_work_count_start(r) :=
    ag_max_work(r) >= ag_work_zero(r) + sum( t in 0..ag_max_work(r)-
    ag_work_zero(r))(allocate("AGENCY",r,t+1))
forall(r in ALL_ROLES, t in 1..WEEKS_TO_PLAN-ag_max_work(r)) AG_work_count
    (r,t) := ag_max_work(r) >= sum( k in 0..ag_max_work(r))(allocate("
    AGENCY",r,t+k))


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))


!----------------comb cut-------------------------------
forall( e in REG_EMP ,r in ALL_ROLES,t in 1..rest_zero(e)) do
if(rest_zero(e)>=1)then
Rest_new(e,r,t):= allocate(e,r,t)=0
end-if
end-do


forall( e in REG_EMP, t in 1..WEEKS_TO_PLAN-1)do
if(min_rest(e)>=1)then
b:=(min_rest(e)-1)
a:=(WEEKS_TO_PLAN-t)
if(b<=a) then
c:=b
else
c:=a
end-if
end-if
forall(y in 1..c)do
Rest_new1(e,t,y) := sum(r in ALL_ROLES)(allocate(e,r,t+y))+sum (v in
    VESSELS) depart(e,v,t)<=1
end-do
end-do
!---------------------------comb cut------------------


! 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 | exists(linear_dv(e,r,t)))
    linear_dv(e,r,t) is_binary
forall(e in ALL_EMP, r in ALL_ROLES, t in TIME | exists(Robust_eligable(e,
   r,t))) Robust_eligable(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) 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) ag_rdepart(r,t) is_binary

forall(e in GUARANTEED) undertime(e) >= 0
forall(e in GUARANTEED) overtime(e) >= 0


setparam("XPRS_verbose",true)
minimize(Total_cost)

end-model
```