INTEGRATION OF OPERATIONS RESEARCH AND ARTIFICIAL INTELLIGENCE APPROACHES TO SOLVE THE NURSE ROSTERING PROBLEM

by

Erfan Rahimian

Department of Management Science & Computer and Information Sciences

University of Strathclyde

A dissertation presented in fulfilment of the requirement for the degree of

Doctor of Philosophy

2016

Acknowledgements

First and foremost, I would like to thank my supervisors Kerem Akartunali and John Levine for all their support and guidance throughout my PhD study. I would like to give special thanks to Kerem for giving me the opportunity to come to the UK and to start a new chapter in my life.

I would also like to thank Greet Vanden Berghe and Ashwin Arulselvan for kindly agreeing to serve as examiners of the examining committee, and for providing valuable feedback which helped shape this dissertation. It was also a privilege for me to meet Greet and her team in Ghent, Belgium and to have her presence in my PhD viva.

I would also like to acknowledge my friends and colleagues in the department of Management Science for making the last three years of my life a truly joyful experience.

Finally, I would also like to thank my parents and wife for supporting me through every stage of my life. Without their unconditional and sincere love, pray, and encouragement, none of this would have been possible. You are the best and I truly appreciate having you all in my life. Thanks God.

Abstract

Nurse Rostering can be defined as assigning a series of shift sequences (schedules) to several nurses over a planning horizon according to some limitations and preferences. The inherent benefits of generating higher-quality rosters are a reduction in outsourcing costs and an increase in job satisfaction of employees. This problem is often very difficult to solve in practice, particularly by applying a sole approach. This dissertation discusses two hybrid solution methods to solve the Nurse Rostering Problem which are designed based on Integer Programming, Constraint Programming, and Meta-heuristics. The current research contributes to the scientific and practical aspects of the state of the art of nurse rostering.

The present dissertation tries to address two research questions. First, we study the extension of the reach of exact method through hybridisation. That said, we hybridise Integer and Constraint Programming to exploit their complementary strengths in finding optimal and feasible solutions, respectively. Second, we introduce a new solution evaluation mechanism designed based on the problem structure. That said, we hybridise Integer Programming and Variable Neighbourhood Search reinforced with the new solution evaluation method to efficiently deal with the problem. To benchmark the hybrid algorithms, three different datasets with different characteristics are used. Computational experiments illustrate the effectiveness and versatility of the proposed approaches on a large variety of benchmark instances.

Keywords: Nurse Rostering, Integer Programming, Constraint Programming, Variable Neighbourhood Search, Hybrid Algorithm

Publications

Some parts of this dissertation have been published in peer-reviewed journals and conference proceedings:

- Rahimian E., Akartunalı, K., and Levine J., A Hybrid Integer Programming and Variable Neighbourhood Search Algorithm to Solve Nurse Rostering Problems, *European Journal of Operational Research 258 (2017) pp. 411-423.*

- Rahimian E., Akartunalı, K., and Levine J., A Hybrid Integer and Constraint Programming Approach to Solve Nurse Rostering Problems, *Computers and Operations Research 82C (2017) pp. 83-94.*

- Rahimian E., Akartunalı, K., and Levine J., Integer Programming for Nurse Rostering: Modelling and Implementation Issues, *Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2016, pages 545–547, Udine, Italy, 2016.*

- Rahimian E., Akartunalı, K., and Levine J., A Hybrid Constraint Integer Programming Approach to Solve Nurse Scheduling Problems, *Proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2015, pages 429-442, Prague, Czech Republic, 2015.*

The following dataset is introduced in this dissertation:

- Rahimian E., Nurse Rostering Dataset, 2015, *doi: 10.15129/9664f00a-e2fd-4dbb-afef-f3c076e2c4f7.*

Contents

List of Algorithms

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
CP	Constraint Programming
CSP	Constraint Satisfaction Problem
IP	Integer Programming
NRP	Nurse Rostering Problem
OR	Operations Research
TS	Tabu Search
VND	Variable Neighbourhood Descent
VNS	Variable Neighbourhood Search
UTP	University Timetabling Problem

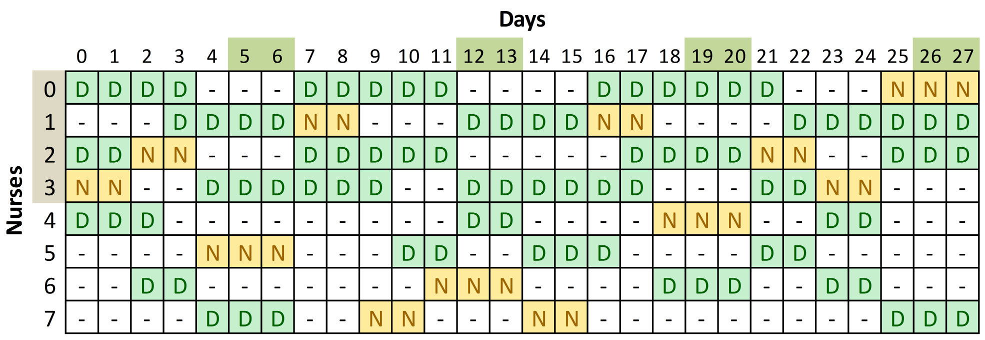
Chapter 1

Introduction

1.1 Nurse Rostering Problem

One of the most important success criteria for organisations to satisfy their customers' requirements and expectations is the ability to have the right staff on the right duty at the right time. Personnel rostering is defined as the process of constructing work timetables for staff, which has been extensively investigated by operations researchers, industrial engineers, computer scientists, and practitioners over the past 50 years [17, 30]. It basically involves specifying at what time (i.e. days, shifts, periods and so on) each employee should work over a specific planning period, which might vary from a day to several months, depending on the nature of the work environment.

Having a high-quality nurse roster enhances both the performance and the quality of a health care unit. Depending on organisational objectives, a roster has higher quality if, for example, it has less gaps which can reduce the outsourcing costs due to hiring fewer agency (bank) nurses to compensate gaps in rosters [47, 56, 85], or if nurses have less overwork, and therefore, more leisure time which can increase job satisfaction, and thus results in lowering staff turnover and absenteeism [17, 63].



	Days																											
Nurses	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	D	D	D	D	-	-	-	D	D	D	D	D	-	-	-	-	D	D	D	D	D	D	-	-	-	N	N	N
1	-	-	-	D	D	D	D	N	N	-	-	-	D	D	D	D	N	N	-	-	-	-	D	D	D	D	D	D
2	D	D	N	N	-	-	-	D	D	D	D	D	-	-	-	-	-	D	D	D	D	N	N	-	-	D	D	D
3	N	N	-	-	D	D	D	D	D	D	-	-	D	D	D	D	D	D	-	-	-	D	D	N	N	-	-	-
4	D	D	D	-	-	-	-	-	-	-	-	-	D	D	-	-	-	-	N	N	N	-	-	D	D	-	-	-
5	-	-	-	-	N	N	N	-	-	-	D	D	-	-	D	D	D	-	-	-	-	D	D	-	-	-	-	-
6	-	-	D	D	-	-	-	-	-	-	-	N	N	N	-	-	-	-	D	D	D	-	-	D	D	-	-	-
7	-	-	-	-	D	D	D	-	-	N	N	-	-	-	N	N	-	-	-	-	-	-	-	-	-	D	D	D

Figure 1.1 – A complete roster for 8 nurses over 4 weeks

In this dissertation, we focus on the *Nurse Rostering Problem* (NRP). This problem can be thought as building a schedule for each nurse. A schedule is a sequence of different types of shifts (e.g. early, day, night, days-off) spanning over the whole planning horizon. This pattern of shifts is generated according to a set of requirements such as hospital regulations, and a series of preferences like a fair distribution of shifts among nurses. These requirements and preferences vary drastically among different countries, institutions, and even wards within a hospital. The solution of this problem is a roster consisting of all schedules for all the nurses during the planning period. Figure 1.1 shows a complete roster, which consists of schedules of eight nurses, depicted in rows, over a 4-week scheduling period (totally 28 days), depicted in columns, where day, night, and off shifts are denoted as D, N, and $-$, respectively.

From theoretical perspectives, most variants of the NRP in real-world settings are known to be NP-hard problems [10, 46, 62], although some simplified versions can be solved in polynomial time [76]. Due to the inherent structure and complexity of the problem which arises from the huge number of often conflicting constraints that must be satisfied, fulfilment of all constraints is often impossible in real-world instances. That said, a distinction is made between *hard constraints* (often enforced by physical resource restrictions and legislation) and *soft constraints* (normally are referred to as staff preferences). Hard constraints must be satisfied (to have a feasible roster), but soft constraints may be violated. Each soft constraint is associated with a weight that represents its importance. Constraints with higher weights are more important to be satisfied, and thus cause a higher penalty if violated. The objective function used to determine the quality of a roster is the sum of all penalties incurred due to soft constraint violations. We will present a model of the NRP using Integer Programming and Constraint Programming in Chapter 2.

1.2 Research Questions

Due to the number of conflicting constraints which has to be satisfied in real-world instances of the NRP, standard Integer Programming (IP) and Constraint Programming (CP) solvers are not able to solve this problem on their own [69]. Therefore, researchers often prefer to hybridise these methods with other methods including meta-heuristics such as Tabu Search [53] and Variable Neighbourhood Search [82] algorithms to exploit their complementary strengths resulting in a competitive solution method for solving practical NRP instances.

Furthermore, some researchers try to extend the *reach*, i.e. the extent to which a method is able to generate an acceptable solution for real-world instances. The methods such as customised IP cuts [74] and propagation algorithms [79] were procedures that were capable of handling a wider range of problems that were able to solve specific practical instances. However, little attention has been paid to hybridise IP and CP together in order to extend the reach of exact methods, and therefore, to design an acceptable hybrid solution method. Since IP and CP solvers use different mechanisms to solve optimisation problems, i.e. IP solvers employ branch-and-cut algorithms in which corresponding LP relaxations of the problem are solved, and CP solvers employ a variety of propagation algorithms and heuristics to reduce domains of variables, hybridising these solvers is intuitive.

Nowadays, there has been an increasing progress in the performance of commercial IP and CP solvers such as Cplex [44] and Gurobi [35], which further motivates the study of hybridisation of such methods. Therefore, the present dissertation addresses the following research question.

Can the reach of exact methods be extended by hybridising IP and CP methods?

Recent academic research tends to focus on increasing the performance of existing methods or on introducing new hybrid algorithms [5, 21, 80, 66]. That said, researchers often overlook the importance of problem-specific information in evaluating solution quality and designing solution methods. Beyond doubt, comparing obtained solutions only based on the objective function value or the total number of soft constraints' violations is not the best and accurate way for evaluating solution quality. The other contribution made in this dissertation considers incorporating problem-specific information into different aspects of the solution method particularly through evaluation of solution quality, thereby addressing the following research question.

How can the particular problem structure of the NRP contribute to different aspects of designing a solution method?

In summary, the first research question tries to address whether hybridising IP and CP is able to improve the reach of those methods in solving the NRP. By extending the reach of such exact methods, one can benefit from the advantages of using IP and CP such as provision of optimality information. The second question tries to understand the impact of considering problem-specific information to the performance of a solution method. In effect, this question tries to redefine the evaluation of solution quality on the basis of problem-specific information.

As a final note, the main focus of this dissertation is on personnel rostering in the context of health care, however the results are applicable in other settings as well, such as services, logistics or manufacturing.

1.3 Background

This dissertation applies some of the established techniques in Operations Research and Artificial Intelligence for solving the NRP. Linear programming is a mathematical programming technique in which values are assigned to a set of decision variables so as to minimise or maximise an objective function, subject to a set of linear constraints. In (Mixed) IP, (a subset of) the decision variables are required to have integer values assigned. A complete theoretical treatment of these topics can be found in [86] and [60].

CP is a technique to solve constraint satisfaction problem which is defined by a finite sequence of variables with respective domains, together with a finite set of constraints, each on a subsequence of variables. In CP, the aim is to find a solution which satisfies all constraints or is optimal with respect to certain criteria using propagation algorithms and variable and value ordering heuristics. For more details, interested readers are referred to Apt [3] and Rossi et al. [72].

A meta-heuristic is a high-level heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimisation problem at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is [71]. Meta-heuristics often complete or replace a solution with the current solution by seeking neighbours of the current solution during the search process iteratively until some stopping criteria are met. Meta-heuristics have been extensively investigated during recent years where many instances of this method such as Tabu Search [52] and Variable Neighbourhood Search [66] are proposed. For more details, interested readers are referred to Burke and Kendall [14], Glover and Melian [33], Osman and Kelly [61].

1.4 Outline of the Dissertation

This dissertation is structured in four chapters. Chapter 1 briefly introduces the NRP along with research questions underlying this dissertation. Chapter 2 and 3 mainly focus on investigating the research questions established in the first chapter. Chapter 4 concludes the dissertation by presenting the summary of contributions and future research directions.

Chapter 2 presents a hybrid algorithm aiming to extend the reach of IP and CP methods. In academic research, IP or CP is often reinforced by meta-heuristics to solve the NRP (e.g. see Burke et al. [21] and M'Hallah and Alkhabbaz [56]). This chapter investigates the effect of hybridising these two methods in terms of computational performance and capability to solve a variety of problem instances existing in the literature. First, to have a comprehensive computational experimentation, two common models of the NRP are explained, and then several experiments are done using three different sets of problem instances existing in the literature.

Chapter 3 studies the impact of involving problem-specific information in solution evaluation and design of hybrid algorithms. That said, a hybrid algorithm, established upon VNS (as one of the most common solution methods applied to the NRP) and reinforced by IP, is elaborated in which the search process and solution evaluation are designed according to the problem structure and obtained problem-specific information. The resulted study experiments on the same models and instances introduced in chapter 2. Finally, the scope of computational tests becomes broaden by evaluating the performance of the presented methodology using a new rostering problem.

It should be noted that the level of integration/hybridisation of the algorithms presented in chapter 2 and 3 is different. According to Talbi [81], the level of hybridisation for the algorithm presented in chapter 3 is LRH (Low-level

Relay Hybrid) where an algorithm is embedded into another one. That said, in this chapter, a hybrid algorithm is proposed in which IP is embedded into a VNS algorithm. However, the level of hybridisation for the algorithm presented in chapter 2 is approximately HRH (High-level Relay Hybrid) where several algorithms are executed in sequence.

Chapter 4 concludes the dissertation by summarising the most important conclusions and contributions, and presents areas for future research.

Chapter 2

Integer and Constraint Programming: A Hybrid Algorithm

The academic literature tries design more efficient algorithms either by reinforcing these methods with a variety of approaches such as developing problem-specific cuts [74] or by employing heuristic methods [79]. However, little attention has been paid to reinforce such methods by hybridising them together which results in preserving the benefits of these methods such as provision of optimality information. This chapter tries to shed light on the hybridisation of CP and IP to investigate if their reach can be extended, and therefore, higher-quality solutions can be generated.

In this chapter, we exploit the strength of IP in obtaining lower-bounds and finding an optimal solution with the capability of CP in finding feasible solutions in a co-operative manner. To improve the performance of the hybrid algorithm, and therefore, to obtain high-quality solutions as well as strong lower-bounds for a relatively short time, we apply some innovative ways to extract useful information such as the computational difficulty of instances and constraints to adaptively set the search parameters. We test our algorithm using three different datasets consisting of various problem instances, and report results benchmarked with the state-of-the-art algorithms from the recent literature as well as standard IP and CP solvers.

The results presented in this chapter are published by Rahimian et al. [69].

2.1 Introduction

In this chapter, we propose a new hybrid algorithm integrating IP and CP to extend the reach of exact methods, utilising the strengths of IP in finding optimal solutions and of CP in finding feasible solutions while exploiting problem-specific information. Due to the exact nature of the proposed algorithm, it can also generate lower-bounds in contrast to most heuristic methods designed to solve the NRP in the literature (note that there are heuristics which are able to find lower bounds). The hybrid algorithm exploits problem-specific information such as the difficulty of constraints to reduce the search space (see CSP generation and Softness Adjustment components in Section 2.5), to fine tune search parameters (see Pre-solve component in Section 2.5), and to improve the efficiency of the whole search process. For instance, during the search process, we identify the potential constraints which are computationally expensive to predict the performance of the IP solver, and thus, setting search parameters adaptively.

In the literature, many exact methods have been investigated such as Integer Programming [32], Constraint Programming [79], and Goal Programming [4]. Furthermore, some academic research have focused on reinforcing the relevant IP and CP formulations [74, 55] or applied IP and CP solvers [13, 73]. However, little attention has been made to extend the reach of these methods in a hybrid setting, thereby preserving their computational benefits such as obtaining optimality information. He and Qu [40] employed CP in solving pricing sub-problems within a column generation method. The largest body of related research concerns hybridising IP and CP with heuristics. For example, Burke et al. [21] combined IP to solve the NRP considering all hard constraints, and a set of soft constraints based on their complexity and importance, and VNS to further improve the obtained solution, considering the rest of the soft constraints. Stølevik et al. [80] integrated CP and VNS to solve the NRP, in which a feasible initial solution is generated via CP, considering all hard constraints, and then, a VNS approach along with a CP destroy-and-repair strategy further improve attained solutions. Bunton et al. [11] employed IP within Ant Colony Optimization algorithm to generate feasible solutions since pure randomization often results in infeasible solutions. They also used IP to sub-optimize the generated rosters by searching neighbourhoods of past best solutions.

Our main purpose in this chapter is to extend the reach of exact methods through hybridisation of exact methods. Indeed, using an IP approach as the core solution method, we employ a CP approach and other algorithmic aids to improve the efficiency of the overall algorithm. This is based on our preliminary experiments where we observed that CP outperformed IP in finding feasible solutions for some hard-to-solve instances. That said, we decided to improve the reach of IP using CP through a hybrid setting. We do not intend to design a hybrid algorithm capable of generating the best result for challenging problem instances, particularly in comparison with advanced hybrid meta-heuristics [78]. Instead, we aim to develop a hybridisation of IP and CP which preserves benefits of exact methods, and is able to outperform each of which alone. Another motivation to hybridise IP and CP is due to the significant improvement of commercial solvers [35] in terms of performance and robustness which are able to solve instances to an extent that is often satisfactory according to the underlying business requirements. Aiming to ease the implementation process and to increase the applicability of the hybrid algorithm, we try to not apply any low-level or convoluted hybridisation settings.

The proposed algorithm is designed to obtain the best result in a pre-defined, relatively short computational time. In addition, the proposed algorithm does not depend on any specific settings regarding the importance of constraints, hence each constraint can be defined as hard or soft during the search process (i.e. the setting of constraints can be changed without altering the search strategy or its parameters). We formulate the problem according to a general model reported in the literature, and evaluate the proposed algorithm using three different datasets existing in the literature.

The rest of this chapter is organised as follows: problem definition and assumptions are presented in Section 2.2. The IP and CP formulations are presented in Sections 2.3 and 2.4, respectively. In Section 2.5, we elaborate on the proposed hybrid algorithm and the associated components. Computational results are reported in Section 2.6, and concluding remarks are briefly discussed in Section 2.7.

2.2 Problem Definition

In this section, we provide a brief description of the studied problem and the relevant constraints similar to two common models existing in the literature described by Burke et al. [19] and Curtois and Qu [26], which are called hereafter Model-I and Model-II, respectively. The first model represents one of the most common sets of benchmark instances in the literature, and the second one is relevant to the latest instances introduced in the literature. For further information regarding the problem, we refer interested readers to Burke et al. [19] and Curtois and Qu [26], where the detailed description of the problem as well as some instances are presented.

In these models, it is assumed that the current roster is modelled over a specified planning horizon in an isolated way, i.e. no information (history) from the previous roster is used to construct the current one. In addition, a day off is considered as a shift type for modelling purposes. We have observed in various health settings that nurse rostering is performed in each hospital ward separately, and therefore, a single-skill set applies well in practice compared with a multi-skill set. In case multiple skills (grades) are required within a ward, they can be accommodated into the current models by considering different personal contracts. For example, if there is a need for head nurses for some particular shift types, it can be modelled using coverage constraints (see Constraint *2*) to consider at least one place for those specific shift types within the planning period. Therefore, it is assumed that all nurses belong to the same skill category. For the sake of simplicity, it is also assumed that all rosters start from Monday and are made of a whole week (i.e. seven days with a two-day weekend).

In the following, the constraints of the problem are explained:

1. Maximum one assignment per shift type per day for each nurse,
2. The number of shift types for each day must be fulfilled [the penalty associated with this constraint is equal to the total amount of violated coverage multiplied by the specified relevant user-defined weight],
3. The minimum and maximum number of:
    (a) shift assignments within the scheduling period [the penalty associated with this constraint is equal to the total number of all violated shift assignments multiplied by the relevant user-defined weight],
    (b) consecutive working days over the planning horizon,
    (c) working hours within the scheduling period (and/or during a week),
    (d) shift assignments within a week,
    (e) shift assignments at the weekend,
    (f) consecutive shift types over the planning period,
4. Minimum number of days off after a night shift or a series of night shifts,

5. Over the weekends, there should be either an assignment to all days of weekends or no assignments at all,
6. No night shift before free weekends,
7. Maximum number of consecutive worked weekends, when there is at least one weekend assignment [the penalty associated with this constraint is equal to the total number of all violated consecutive worked weekends multiplied by the relevant user-defined weight],
8. Requested shifts (days) on or off, where some user-defined shifts (days) must (not) be allocated for a particular nurse within the planning horizon [the penalty associated with this constraint is equal to the total number of all violated assignments multiplied by the relevant user-defined weight],
9. Forbidden shift type patterns (e.g. the ND pattern, where the shift type D is not allowed to be assigned right after the shift type N).

For constraints *3.f* and *9*, it is assumed that the last day of the previous planning period and the first day of the next planning horizon are days off. Furthermore, for Constraint *3.f*, it is assumed that there are an infinite number of consecutive shifts assigned at the end of the previous planning period and at the start of the next planning period. For Constraint *3.b*, a similar arrangement applies with days off. Indeed, this constraint limits the number of consecutive working days over the planning horizon for each nurse. According to this constraint, the number of consecutive working days assigned to each nurse should be within a particular range. The minimum limit could be provided to avoid unnecessary fragmentation in a schedule. The maximum restriction is often provided to avoid abnormal assignment of consecutive working days to a particular nurse, which may cause extreme fatigue, thereby lowering the quality of caring service.

Table 2.1 shows the availability of each constraint and whether they are soft for Model-I and Model-II. In this table, Y and N designate the presence of a

Table 2.1 – The setting of constraints for Model-I and Model-II

Constraint	Model I [19]	Model II [26]
1	Y	Y
2	Y	Y [S]
3.a	Y	Y(only max)
3.b	Y	Y(only max)
3.c	Y	Y
3.d	Y [S]	N
3.e	Y	Y(only max)
3.f	Y	Y
4	Y	N
5	Y	N
6	Y	N
7	Y [S]	N
8	Y [S]	Y(no day on) [S]
9	Y	Y

constraint within each model. All constraints are considered hard except for those which are tagged by *[S]*. It should be noted that as hardness and softness of constraints could be varied per instance, the configuration of models presented in Table 2.1 only represents the majority of instances. As it can be seen, both models are varied in terms of the availability of constraints and whether they are applied as hard or soft, which gives us a broader benchmark basis to evaluate the studied research questions.

In the next two sections, i.e. Section 2.3 and 2.4, we formulate this problem using IP and CP, respectively.

2.3 IP Formulation

Here, we present the mathematical formulation using IP based on the definitions and assumptions provided in Section 2.2 for all the constraints. For the sake of consistency, we also use the same numbering of constraints as in Section 2.2. For demonstration purposes, we provide here a combined formulation (of Model-I and Model-II) in which constraints 2.2, 2.3d, 2.7, and 2.8 are soft. In case one needs to consider any of these constraints as hard, they need to set the weight associated with those constraints in the objective function to a significantly big value or remove the associated auxiliary variables in those constraints and objective function.

Sets and parameters:

E	set of nurses.
D	set of days in the planning horizon.
A	set of shift types.
A'	set of all shift types except day off.
W	set of weeks in the planning horizon.
H_a	set of shift types that cannot be assigned immediately after shift type $a \in A$.
Q_{ad}	set of pre-assigned nurses to shift type $a \in A$ on day $d \in D$.
M_e^{min}, M_e^{max}	minimum and maximum number of shifts that can be assigned to nurse $e \in E$ within the planning period.
W_w^{min}, W_w^{max}	minimum and maximum number of shifts that can be assigned to a nurse within week $w \in W$.
V_d^{min}, V_d^{max}	minimum and maximum number of shifts that can be assigned to nurses on day $d \in D$.
A^{min}, A^{max}	minimum and maximum number of hours that can be assigned to each nurse during the planning period.
E_w^{min}, E_w^{max}	minimum and maximum number of hours that can be assigned to each nurse during week $w \in W$.
N^{min}, N^{max}	minimum and maximum number of consecutive working days over the planning period. b is the index of possible number of consecutive working days.
H_a^{min}, H_a^{max}	minimum and maximum number of consecutive shift type $a \in A$ over the planning period. c is the index of possible number of consecutive

	shifts.
K^{min}, K^{max}	minimum and maximum number of worked weekends over the planning horizon.
C^{max}	maximum number of consecutive worked weekends over the planning period.
U_a	total workloads (hours) of shift type $a \in A$ within the planning period.
U_{aw}	total workloads (hours) of shift type $a \in A$ during week $w \in W$.
B_{ew}^{wa}	the weight associated with the minimum and maximum number of shift assignments to nurse $e \in E$ within week $w \in W$.
B_{ew}^{cwx}	the weight associated with the maximum number of consecutive weekends if nurse $e \in E$ works at weekend of week $w \in W$.
B_{ead}^{rso}	the weight associated with requested shift $a \in A$ on day $d \in D$ for nurse $e \in E$.
B_{d}^{cv}	the weight associated with the minimum and maximum number of shifts that can be assigned to nurses on day $d \in D$.

Decision variables:

x_{ead}	$= 1$ if shift type $a \in A$ on day $d \in D$ is assigned to nurse $e \in E$, $= 0$ otherwise.
p_{ed}	$= 1$ if nurse $e \in E$ works on day $d \in D$, $= 0$ otherwise.
k_{ew}	$= 1$ if nurse $e \in E$ works at weekend of week $w \in W$, $= 0$ otherwise.
y_{ea}	total number of times that shift type $a \in A$ is assigned to nurse $e \in E$ over the planning period.
z_{ewa}	total number of shift type $a \in A$ assigned to nurse $e \in E$ within week $w \in W$.
$v_{ew}^{wam}, v_{ew}^{wax}$	total incurred penalty relevant to the minimum and maximum number of shift assignments to nurse $e \in E$ within week $w \in W$.

v_{ew}^{cwx}	the incurred penalty relevant to maximum number of consecutive weekends if nurse $e \in E$ works at weekend of week $w \in W$.
v_{ead}^{rso}	total incurred penalty relevant to requested shift $a \in A$ on day $d \in D$ assigned to nurse $e \in E$.
v_d^{cvm}, v_d^{cvx}	total incurred penalty relevant to the minimum and maximum number of shifts that can be assigned to nurses on day $d \in D$.

Constraints:

$$\sum_{a \in A} x_{ead} = 1, \ \ \forall e \in E, d \in D \tag{2.1}$$

$$\begin{cases} p_{ed} = \displaystyle\sum_{a \in A'} x_{ead}, & \forall e \in E, d \in D \\ V_d^{min} - v_d^{cvm} \le \displaystyle\sum_{e \in E} p_{ed} \le V_d^{max} + v_d^{cvx}, & \forall d \in D \end{cases} \tag{2.2}$$

$$\begin{cases} y_{ea} = \displaystyle\sum_{d \in D} x_{ead}, & \forall e \in E, a \in A \\ M_e^{min} \le \displaystyle\sum_{a \in A} y_{ea} \le M_e^{max}, & \forall e \in E \end{cases} \tag{2.3a}$$

$$\begin{cases} p_{ed} + \left(b - 1 - \displaystyle\sum_{g=d+1}^{d+b} p_{eg} \right) & \forall e \in E, d \in \{1 \dots |D| - (b+1)\}, \\ & b \in \{1 \dots N^{min} - 1\} \\ + p_{e(d+b+1)} \ge 0, & \\ \displaystyle\sum_{g=d}^{N^{max}+d} p_{eg} \le N^{max}, & \forall e \in E, d \in \{1 \dots |D| - N^{max}\} \end{cases} \tag{2.3b}$$

$$\begin{cases} A^{min} \le \displaystyle\sum_{a \in A} y_{ea} U_a \le A^{max}, & \forall e \in E \\ z_{ewa} = \displaystyle\sum_{d=7(w-1)+1}^{7w} x_{ead}, & \forall e \in E, a \in A, w \in W \\ E_w^{min} \le \displaystyle\sum_{a \in A} z_{ewa} U_{aw} \le E_w^{max}, & \forall e \in E, w \in W \end{cases} \tag{2.3c}$$

$$W_w^{min} - v_{ew}^{wam} \le \sum_{a \in A} z_{ewa} \le W_w^{max} + v_{ew}^{wax}, \ \ \forall e \in E, w \in W \tag{2.3d}$$

$$\begin{cases} k_{ew} \leq p_{e(7w-1)} + p_{e(7w)} \leq 2k_{ew}, & \forall e \in E, w \in W \\ K^{min} \leq \sum_{w \in W} k_{ew} \leq K^{max}, & \forall e \in E \end{cases} \tag{2.3e}$$

$$\begin{cases} x_{ead} + \left(c - 1 - \sum_{g=d+1}^{d+c} x_{eag} \right) & \forall e \in E, a \in A, d \in \{1 \dots |D| - (c+1)\}, \\ + x_{ea(d+c+1)} \geq 0, & c \in \{1 \dots H_a^{min} - 1\} \\ \sum_{g=d}^{H_a^{max}+d} x_{eag} \leq H_a^{max}, & \forall e \in E, a \in A, d \in \{1 \dots |D| - H_a^{max}\} \end{cases} \tag{2.3f}$$

$$\begin{cases} x_{end} \leq x_{en(d+1)} + 1 - p_{e(d+1)}, & \forall e \in E, d \in \{1 \dots |D| - 1\} \\ x_{end} - p_{e(d+1)} \leq 1 - p_{e(d+2)}, & \forall e \in E, d \in \{1 \dots |D| - 2\} \end{cases} \tag{2.4}$$

$$x_{er(7w-1)} = x_{er(7w)}, \;\; \forall e \in E, w \in W \tag{2.5}$$

$$x_{en(7w-2)} \leq p_{e(7w-1)} + p_{e(7w)}, \;\; \forall e \in E, w \in W \tag{2.6}$$

$$\sum_{i=0}^{C^{max}} k_{e(w+i)} \leq C^{max} + v_{ew}^{cwx}, \;\; \forall e \in E, w \in \{1 \dots |W| - C^{max}\} \tag{2.7}$$

$$x_{ead} = 1 - v_{ead}^{rso}, \;\; \forall e \in Q_{ad}, a \in A, d \in D \tag{2.8}$$

$$x_{ead} + x_{eh(d+1)} \leq 1, \;\; \forall e \in E, a \in A, h \in H_a, d \in \{1 \dots |D| - 1\} \tag{2.9}$$

$$x_{ead}, p_{ed}, k_{ew} \in \{0, 1\}, y_{ea}, z_{ewa} \in \mathbb{Z}, \;\; \forall e \in E, a \in A, d \in D, w \in W$$

Objective function:

$$\min \sum_{e\in E}\sum_{w\in W}\left(B^{wa}_{ew}\left(v^{wam}_{ew}+v^{wax}_{ew}\right)+B^{cv}_{d}\left(v^{cvm}_{d}+v^{cvx}_{d}\right)+B^{cwx}_{ew}v^{cwx}_{ew}\right)$$
$$+\sum_{e\in Q_{ad}}\sum_{a\in A}\sum_{d\in D}\left(B^{rso}_{ead}v^{rso}_{ead}\right)$$

Constraint 2.1 makes sure that only one shift type is assigned to a nurse during a day. Constraint 2.2 defines auxiliary variable p_{ed} which is restricted to the minimum and maximum values V^{min}_{d} and V^{max}_{d}, respectively. These constraints assure that the required demand for each day within the planning period would be satisfied. Variables v^{cvm}_{d} and v^{cvx}_{d} capture the incurred penalty of possible violations which is later weighted by the value B^{cv}_{d} in the objective function. Constraint 2.3a defines variable y_{ea} which counts the total number of shifts assigned to a nurse for a specific shift type. M^{min}_{e} and M^{max}_{e}, defined for each nurse, confine variable y_{ea}. In Constraint 2.3b, to take into account the number of consecutive working days below the minimum value N^{min}, we count all sequences individually up to the minimum exclusively. However, for the maximum part, we only need to count all the sequences which have a length of one day more than the maximum value N^{max}. Thus, all the sequences having a length of more than the maximum are counted accordingly. Constraint 2.3c ensures that the total number of working hours for each nurse is limited to the minimum and maximum value A^{min} and A^{max}, respectively. To restrict working hours within a week, variable z_{ewa} is defined which is restricted to take values between E^{min}_{w} and E^{max}_{w}. Working hours for each shift type is already given in parameters U_{a} and U_{aw}. Constraint 2.3d restricts the total number of shift assignments within a week to the minimum and maximum values W^{min}_{w} and W^{max}_{w}. The incurred penalty due to possible violations is measured by variables v^{wam}_{ew} and v^{wax}_{ew} which are weighted by the value B^{wa}_{ew} in the objective function. Constraint 2.3e limits the total number of shift assignments at weekends for each nurse by defining the auxiliary variable k_{ew}. This variable only takes value if at least one day within a two-day weekend is assigned to a nurse. Constraint 2.3f is similar to Constraint 2.3b, but only sequences of a particular shift type are counted and restricted to values H_a^{min} and H_a^{max}. In Constraint 2.4, we assume that there should be two days off after a night shift or a series of night shift types. Constraint 2.5 and 2.6 utilise weekends by restricting them to be assigned by either no assignment or a full two-day assignment, and also no night shift assignment on Fridays. In constraints 2.2, 2.4, 2.5, and 2.6, n and r indicate night and rest shift types, respectively. Constraint 2.7 adds more limitations to Constraint 2.3e by restricting the maximum consecutive sequences of worked weekends to the value C^{max}. Possible violations are captured in the variable v_{ew}^{cwx} and weighted by the parameter B_{ew}^{cwx} in the objective function. Requested shifts (days) on or off are taken into account in Constraint 2.8 where the incurred penalty is captured by the variable v_{ead}^{rso} and weighted by the parameter B_{ead}^{rso} in the objective function. Constraint 2.9 is dedicated to forbidden shift type patterns within the planning period. The objective function is the weighted sum of all the auxiliary variables associated with soft constraints 2.2, 2.3d, 2.7, and 2.8, and their relevant weights.

The proposed IP formulation is similar to most IP formulations reported in the literature [74, 13]. However, there are often some differences in the softness of constraints, and therefore, the associated objective function. The *softness* of constraint is defined as the proportional relationship between the number of soft and hard constraint which is directly relevant to defining which constraints are considered hard or soft by the user in a model. For example, Santos et al. [74] proposed an IP formulation for the problem instances introduced in the first International Nurse Rostering Competition (INRC-I) [39]. In their formulation, all constraints are considered soft, except Constraint 2.2. That said, the objective function is also different, which is the weighted sum of all penalties associated with the remaining soft constraints. In this dissertation, it is tried to not rely on a particular problem setting (e.g. softness of constraints) in the design of the proposed hybrid algorithms to better investigate the defined research questions. The main reason is to increase the capability and compatibility of the formulation and solution method to accommodate more computational instances that exist in the literature which often have different settings of constraints. Another reason is to broaden the practicality aspect of the research. In practice, settings of constraints could be changed based on various reasons such as adding new business requirements. Furthermore, IP and CP formulations are rather flexible (often using variants of mathematical programming language [35]) when it comes to softness and hardness of constraints. For some discussions regarding the presented formulation, we refer interested readers to Rahimian et al. [67].

2.4 CP Formulation

In this section, we present our CP formulation based on the Constraint Satisfaction Problem (CSP) model using the definitions and assumptions provided in Section 2.2. It is worth noting that we only utilise CP for generating feasible solutions in the configuration of the proposed hybrid algorithm, and thus, there is no need to model any soft constraints nor to define a Constraint Optimisation Problem. Nevertheless, all the soft constraints in Model-I and Model-II are defined as hard constraints in the following CP model. That said, first, we concisely explain two types of global constraints, which are required in the CP model: *Cardinality* and *Stretch* global constraints. For more information about global constraints in CP, we refer the interested readers to Laburthe and Jussien [48], Van Hoeve and Katriel [84], and Beldiceanu et al. [7].

Cardinality constraint (aka. *GCC* or *Generalised Cardinality*) bounds the number of times that variables take a certain set of domain values. It is written as:

$$cardinality\,(x, v, l, u)$$

where x is a set of variables $(x_1, \ldots, x_n)$; v is an m-tuple of domain values of the variables x; l and u are m-tuples of non-negative integers defining the lower and upper bounds of the times value v may be taken by variable x, respectively. The constraint defines that, for $j = 1, \ldots, m$, at least l_j and at most u_j of the variables x take value v_j.

Stretch constraint bounds the sequence of consecutive variables that take the same value (stretch), i.e. $x_{j-1} \neq 1, x_j, \ldots, x_k = v, x_{k+1} \neq v$. It is expressed as:

$$stretch\,(x, v, l, u, P)$$

where x is a set of variables $(x_1, \ldots, x_n)$; v is an m-tuple of possible domain values of x; l and u are m-tuples of lower and upper bounds for x, respectively. P is a set of patterns, i.e. pairs of values (v_j, v_k), requiring that when a stretch of value v_j immediately precedes a stretch of value v_k, the pair (v_j, v_k) must be in P.

As we use the same parameters as defined in Section 2.3, we define any additional parameters as well as the variables and constraints of the CP model next.

Sets and Parameters:

U	the vector of total workloads (hours) of all shift types within the planning period.
U_w	the vector of total workloads (hours) of all shift types during week $w \in W$.

Decision variable:

s_{ed}	integer variable indicating the shift type assigned to nurse $e \in E$ on day $d \in D$.

Constraints:

$$cardinality\left(\bigcup_{e\in E} s_{ed}, A, V_d^{min}, V_d^{max}\right),\ \ \forall d \in D \tag{2.2}$$

$$cardinality\left(\bigcup_{d\in D} s_{ed}, A, M_e^{min}, M_e^{max}\right),\ \ \forall e \in E \tag{2.3a}$$

$$stretch\left(\bigcup_{d\in D} s_{ed}, A', N^{min}, N^{max}, P\right),\ \ \forall e \in E, P = \{\} \tag{2.3b}$$

$$\begin{cases} A^{min} \leq prod(\bigcup_{d\in D} s_{ed}, U) \leq A^{max}, & \forall e \in E \\ E_w^{min} \leq prod(\bigcup_{d=7(w-1)+1}^{7w} s_{ed}, U_w) \leq E_w^{max}, & \forall e \in E, w \in W \end{cases} \tag{2.3c}$$

$$cardinality\left(\bigcup_{d=7(w-1)+1}^{7w} s_{ed}, A, W_w^{min}, W_w^{max}\right),\ \ \forall e \in E, w \in W \tag{2.3d}$$

$$cardinality\left(\bigcup_{w\in W} s_{e(7w)} + s_{e(7w-1)}, r, |W| - K^{max}, |W| - K^{min}\right),\ \ \forall e \in E \tag{2.3e}$$

$$stretch\left(\bigcup_{d\in D} s_{ed}, a, H_a^{min}, H_a^{max}, P\right),\ \ \forall e \in E, a \in A, P = \{\} \tag{2.3f}$$

$$if\ s_{ed} = n, then s_{e(d+1)} = r \wedge s_{e(d+2)} = r,\ \ \forall e \in E, d \in \{1, \ldots |D| - 2\} \tag{2.4}$$

$$s_{e(7w-1)} = s_{e(7w)},\ \ \forall e \in E, w \in W \tag{2.5}$$

$$if\ s_{e(7w-2)} = n, then s_{e(7w-1)} \neq r \vee s_{e(7w)} \neq r,\ \ \forall e \in E, w \in W \tag{2.6}$$

$$stretch\left(\bigcup_{w\in W} s_{e(7w)} + s_{e(7w-1)}, r, |W| - C^{max}, |W|, P\right),\ \ \forall e \in N, P = \{\} \quad (2.7)$$

$$s_{ed} = a,\ \ \forall e \in Q_{ad}, a \in A, d \in D \quad (2.8)$$

$$if\ s_{ed} = a, then s_{e(d+1)} \notin H_a,\ \ \forall e \in E, d \in \{1, \ldots |D| - 1\} \quad (2.9)$$

$$s_{ed} \in A,\ \ \forall e \in E, d \in D$$

Constraint 2.2 and 2.3a restrict the number of assigned shifts using cardinality constraint for all days and nurses, respectively. In Constraint 2.3b, sequences of working shifts ($a \in A'$) using stretch constraint are limited to N^{min} and N^{max}, where there is no need to any specific pattern. Constraint 2.3c is very similar to its relevant IP form, where function $prod(x, v)$ is defined as the product of all values of set x, and the relevant possible domain values in vector v. Constraints 2.3d and 2.3e resemble Constraint 2.2. Constraint 2.3f is the same as Constraint 2.3b, where sequences of a particular shift type are taken into account. In Constraint 2.4, we assume that there should be two days off after a night shift or a series of night shift types. Using stretch global constraint, Constraint 2.7 restricts the length of sequences of rest shifts (instead of working shifts) to $|W| - C^{max}$ and $|W|$, since it is easier to be counted in terms of modelling. Constraints 2.5, 2.6, 2.8, and 2.9 are similar to their IP forms, where we use some simple reified (if-then) relations. It should be noted that the first constraint is implicitly satisfied due to the inherent structure of the CP model, where only a single shift type can be assigned to each nurse at a day. In addition, there may exist some symmetries between nurses for a particular problem. For example, if there is a set of contractual constraints in which nurses are assigned based on different contracts (e.g. part-time and full-time), those nurses on a particular contract could have a number of permutations leading to the same solution. This sort of symmetry is common in the NRP. To resolve these symmetry issues and increase the performance of CP solvers over the CSP model, we apply the symmetry breaker switch (by setting parameter *Symmetry* in Gurobi) with default value in the IP solver and add some lexicographic ordering constraints [75, 31] to the main variable s_{ed} by taking into account the Constraint 2.8 for each group of nurses working on the same working contract in a similar way described by Smith [77].

2.5 Integration of IP and CP

For relatively small- to medium-size problems (see Section 2.6 for more details), IP solvers are often efficient enough to find the optimal solution or near-optimal solutions, and to generate strong lower-bounds (e.g. see instances solved to optimality via IP in Section 2.6 or Santos et al. [74]). Similarly, CP solvers are capable of finding feasible solutions efficiently. However, using these approaches on their own for solving large-scale problems, or even small-scale problems with a highly-constrained structure often leads to poor performance. For example, solving most of the benchmark instances using the model presented in Section 2.3 with a standard IP solver, we were not able to obtain an optimal solution (and in some cases even a good-quality solution) in a reasonable amount of time, where some instances took more than 24 hours to solve (e.g. see Table 2.5). Similarly, a standard CP solver results in poor performance, since it often takes a long time to achieve an optimal solution. Therefore, it is intuitive to hybridise them in order to utilise their complimentary strengths for efficiently solving the NRP. That said, we investigate how we can extend the reach of IP and CP through a hybridisation setting. In fact, we combine the strength of IP in obtaining lower-bounds and finding an optimal solution with the capability of CP in finding feasible solutions. Applying the IP approach as our core solution component, we use a CP approach and other algorithmic modules to increase its efficiency. Moreover, IP and CP on their own are quite efficient in solving some specific problem structures such as network flow and bin packing problems [35, 45], which further motivates us to combine such strengths together in order to achieve better overall performance. To improve the efficiency of the hybrid algorithm, we also exploit the problem structure to provide valuable information such as difficulty of constraints, thereby setting the parameters of the proposed algorithm adaptively. Applying a high-level hybridisation scheme of IP and CP, our purpose is to extend the reach of IP and CP through a hybrid algorithm that is able to efficiently generate good-quality solutions for a wide range of problem instances, particularly within a relatively short computational time.

In the following, we provide a brief description of the performance of the hybrid algorithm, and then elaborate on each associated component individually. After a quick pre-processing in order to create appropriate data structures for the algorithm, at the first step, we employ an IP solver to pre-solve the problem in order to identify any valuable information, which can be used to adjust the parameters of other components accordingly. In the next step, we employ a CP solver to iteratively solve various CSP models in order to generate a good-quality solution, and to identify difficult constraints. Then, using the best-obtained solution provided within the CSP generation step, we further improve the attained solution by an IP solver in the remaining time and report the final solution to the user. We also add two more components to reinforce the search process using the exploited problem-specific information: *softness adjustment* and *lower-bound enhancement*. The former attempts to modify the softness of each constraint based on a pre-defined threshold in order to tighten the problem formulation, thereby aiming to help the IP solver to generate better bounds; and the latter aims to provide a stronger lower-bound for the IP solver by decomposing the problem.

It should be noted that the proposed hybrid algorithm runs in a pre-defined

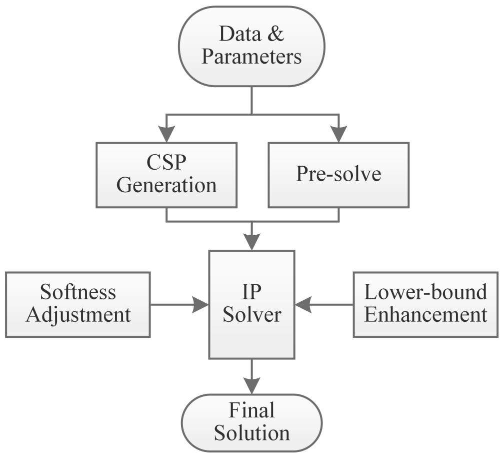


Figure 2.1 – Schematic diagram of the proposed hybrid algorithm

time limit to solve the problem. Thus, the user is able to determine the running time of each component by setting the relevant computational time parameter. It is noteworthy to mention that all components of the hybrid algorithm work with the IP model except the CSP generation step, for which the CSP model is employed. The schematic diagram of the proposed algorithm is depicted in Figure 2.1. Next, we explain each component individually in more details. It is also noted that the quality of solutions is evaluated by turning all hard constraints except Constraint *1* into soft constraints (by assigning a significantly large weight (e.g. 10,000)). That said, the quality of the obtained solutions is measured based on the total number of violations of soft constraints.

Pre-solve: In a nutshell, the duty of this step is to extract some information from the problem, and to regulate the parameters of the main IP solver adaptively. In fact, this component is the first step in most commercial solvers to analyse and simplify the problem structure, and also to identify any specific structures such as network flow or assignment problems [35]. If the IP solver can identify any particular structures, experimentally speaking, it is often led to better performance during the branch-and-bound algorithm. Here, we only call the pre-solve step of an IP solver from the hybrid algorithm as a black-box. We use the information obtained from this step including the obtained lower-bound and relaxed objective function to predict if there are any specific structures, thereby setting the parameters of the IP solver. According to our experiments, within the pre-solve step, if the IP solver provides a stronger (bigger in our problem setting) lower-bound than the relaxed objective function value (which is obtained by relaxing all integer constraints), the employed IP solver might be able to solve the problem in better performance due to the identification of a specific data structure. We switch on the relevant parameter for the pre-solve step of the main IP solver to the highest degree (aggressive mode) automatically (e.g. setting parameter *presolve* in Gurobi). Moreover, using the reported number of constraints and variables in this step, if they are more than user-defined threshold *psThr*, we call the problem *difficult*, thus, we set the search strategy of the main IP solver to spend more effort on obtaining a feasible solution rather than on proving optimality. We do not change the default search strategy in case a problem is not difficult to solve. In most modern solvers, the user can modify the search strategy by setting a specific parameter defined therein. For example, in Gurobi IP solver, the user can tailor the search strategy by setting the parameter *mipfocus*.

CSP generation: After we set some parameters of the main IP solver, a CP solver is employed to generate a good-quality initial solution and to identify difficult constraints. This solver solves the problem according to the CSP model presented in Section 2.4. However, in our preliminary experiments on the benchmark instances, the CP approach does not provide very good-quality or even feasible solutions within a limited computational time (e.g. see Table 2.5). To address this issue, we implement the following procedure: at each iteration i, we modify and then solve the CSP model p considering all those constraints that have a weight higher than the user-defined threshold $cspThr$. If modified problem p'_i is feasible, we generate a number of different solutions based on p'_i according to the user-defined parameter $numSols$. Otherwise, we decrease the threshold value by one unit, and go to the next iteration. Therefore, on each threshold

Algorithm 1: The pseudo code of CSP generation in the hybrid algorithm

```
1  Solutions, $p'_{i-1}$ = empty;
2  i = 1;
3  while true do
4      $p'_i$ = generateCSP(p, cspThr);
5      if $p'_i$ is feasible then
6          for j = 1 to numSols do
7              Solutions[$p'_i$].add(solve($p'_i$));
8          end
9          if $\left|1 - \frac{best(Solutions[p'_{i-1}])}{best(Solutions[p'_i])}\right| \leq q$ then
10             break
11         end
12     else
13         cspThr = cspThr - 1;
14         if $p'_i == p$ then
15             break
16         end
17     end
18     i++;
19 end
20 return [best(Solutions)]
```

level or iteration, there might be several feasible solutions, which are stored in data structure *Solutions*. This process continues until the quality gap between the best-obtained solutions in two consecutive levels is less than q percent, or p'_i is equal to p. Finally, all stored solutions are evaluated according to p and the best-quality solution is reported. Then, the reported solution is imported to the IP solver for further improvement. The pseudo code of this procedure is presented in Algorithm 1, where p, p'_i, $cspThr$, q, and $numSols$ indicate the original problem, the new generated problem in each iteration i, the user-defined threshold level to cut off the constraints, the maximum gap between the quality of the best-obtained solutions in two consecutive threshold levels, and the user-defined number of solutions required to be generated for each threshold level, respectively. In this algorithm, *Solutions* is a data structure which stores a CSP and all the associated solutions, and function *best* evaluates the quality of the input solution according to the original problem setting.

Indeed, the aim of this step is to construct a weak CSP and iteratively strengthen it until some stopping criteria are met. Note that solving a CSP and generating a number of solutions accordingly is done quite fast, often within a few seconds. Nevertheless, experimentally speaking, the time limit of the CP solver is set to 10 seconds. Furthermore, it should be noted that although a stronger CSP (i.e. a CSP with more constraints) is generated while progressing through lower threshold levels, and therefore, it is always expected to obtain better solutions afterwards, the quality of generated solutions is varied within each level. This is because the CP solver is indeed a satisfiability solver [3] rather than an optimisation solver which tries to find a solution that satisfies all included constraints in the CSP model. Therefore, the quality of generated solutions in each level can be different according to the defined objective function in the optimisation problem.

Using the information obtained from the threshold levels within the CSP generation step, we can also find out an estimate for the computational difficulty of each constraint: if the difference in solution times attained by removing a constraint from a problem in order to solve a new modified problem is significant, we count it as a *difficult* constraint. Indeed, we measure the time which takes to solve the problem before and after adding a constraint. Experimentally speaking, difficult constraints are hard to handle for IP and CP solvers and often adding them to a problem instance needs more time and effort. In our preliminary experiments, we realised that 10 seconds is sufficient for most benchmark instances. Identifying difficult constraints helps us in the softness adjustment step to tighten the formulation of the problem.

IP Solver: In this component, we use an IP solver to solve the problem within the remaining time, to which an initial solution generated from the CSP generation step is imported. Indeed, we use the solution obtained from the CP solver as a warm start for the IP solver. To improve the performance of the IP solver, we also apply two more techniques performed within the softness adjustment and lower-bound enhancement steps as described next.

Softness adjustment: In order to improve the efficiency of the IP solver during the search process, we modify the weights (e.g. see B^{wa}_{ew} and B^{rso}_{ead} in Section 2.3) in the objective function due to the difficulty degree of constraints obtained within the CSP generation step. According to this degree, if a constraint is not difficult, we impose it to the IP solver as a hard constraint. In fact, only difficult constraints are kept in the objective function, and the remaining ones are moved to the set of hard constraints. This is similar to modifying the associated weights with hard constraints in the objective function to a significantly large value. Theoretically, this process may lead to an infeasible problem due to possible conflicts between hard constraints. In this case, we undo the relevant change (i.e. declare the constraint as soft again) and continue the process for the rest of the constraints. Finally, we solve the new modified problem using the IP solver. This component helps to reduce the search space by tightening the problem formulation, which often results in better efficiency during the search process, and higher-quality feasible solutions.

Lower-bound enhancement: In this step, we try to find out a better lower-bound by decomposing the problem to different sub-problems, and then, to solve each by using an IP solver. For this purpose, we decompose the problem in three ways: first, we break down the problem to weekly rosters. For each weekly roster, all constraints are taken into account except those associated with the whole planning horizon. These constraints are not examined unless their imposed restriction is valid for a week. For example, if the total number of assignments for each nurse within the planning period is 5, it is added to the involved constraints. Second, we decompose the problem to personal schedules for each nurse. In this case, all constraints are considered excluding Constraint 2.2. Finally, we decompose the problem to groups of similar nurses. To identify similar nurses, we iterate through the set of nurses and constraints except Constraint 2.2 to find out nurses who are included in a similar set of constraints. In order to facilitate this process, we define data structure $LBE = \{(e, c_e)|e \in E, c_e = \{c \Lleftarrow e|c \in C\}\}$, where E and C show the set of nurses and constraints, c_e denotes the set of constraints which affects nurse e, and $\Lleftarrow$ indicates involvement. Then, we group all nurses who have the same set of c_e. Thus, each group shows a sub-problem. It should be noted that the identified groups do not necessarily partition the set of nurses or complement each other. Experimentally speaking, these groups are often associated with different working contracts in the problem data. Indeed, we try to find out whether there is an inevitable conflict in the model, which may be discovered before solving the problem by decomposing it into smaller sub-problems. Finally, the best lower-bound calculated in this step by solving different sub-problems is imposed to the IP solver by setting the relevant parameter (e.g. setting parameter *start* in Gurobi).

It should be noted that what is being done in this step is a semantic and heuristic decomposition rather than a mathematical decomposition. It is designed to obtain a better lower-bound by focusing on approximately independent elements of the problem such as weeks and contracts. In effect, it tries to find out elements/constraints of the problem which are not able to be satisfied even in a subset and rather independent part of the problem. For each part, those constraints affecting more than a single part of the problem become loose (apart from those that are valid for that single subset of the problem) to make sure that the obtained lower-bound is valid for the whole problem. The motivation behind this heuristic decomposition comes from the analysis conducted by Glass and Knight [32].

2.6 Computational Results

The algorithm presented in this chapter is tested on three different datasets: Dataset-I consists of 9 instances published by Burke et al. [19], which are made based on real-world cases. Dataset-II comprises 11 randomly generated instances introduced in this dissertation [65], which are made to extend our benchmarks according to the most difficult instance in Dataset-I, i.e, ORTEC01. Another reason to create this dataset is the unavailability of a commonly-used benchmark dataset in the literature on which about eight solution methods were evaluated (this dataset was the first dataset on which the solution methods presented in this thesis were tested). This dataset consists of 11 instances which are captured from a hospital within a year. That said, these instances are similar in many aspects except for caring coverages and pre-assignments which makes them computationally different. The process of generating this dataset is elaborated in Appendix A. Dataset-III consists of 24 instances that have been recently introduced by Curtois and Qu [26] to challenge state-of-the-art algorithms. Most instances in the benchmark datasets are known to be computationally intractable for exact and heuristic methods, thereby allowing us to study the extent of the hybrid algorithm comprehensively. The characteristic of benchmark instances (e.g. size and softness of constraints, number of variables and constraints) in these datasets are varied per instance, which makes them an appropriate benchmark for the proposed algorithm. Tables 2.2 and 2.3 show some characteristics of the benchmark instances in the three datasets including number of nurses, shift types, days, and number of pre-assignments as well as number of IP variables and constraints (before any reductions). In Table 2.2, instances belonging to dataset-II are prefixed by *NRP*. It should be noted that the number of variables and constraints are different for the same input size (number of fundamental elements such as nurses and days) because of different settings of constraints. Indeed, although the applied sets of

Table 2.2 – The characteristics of benchmark dataset-I and dataset-II

Instance	Nurses	Shift types	Days	Pre.	Var.	Const.	Instance	Nurses	Shift types	Days	Pre.	Var.	Const.
GPOST	8	3	28	8	5680	5504	NRP02	16	5	33	6	21,923	20,580
GPOSTB	8	3	28	0	5680	5496	NRP03	16	5	33	15	21,929	20,586
ORTEC01	16	5	33	32	19,096	19,170	NRP04	16	5	33	5	21,925	20,582
ORTEC02	16	5	33	42	19,101	19,175	NRP05	16	5	33	5	21,925	20,582
Valouxis-1	16	4	28	0	9776	9968	NRP06	16	5	33	4	21,924	20,581
SINTEF	24	6	21	96	8118	6927	NRP07	16	5	33	12	21,926	20,583
WHPP	30	4	14	0	6000	5842	NRP08	16	5	33	35	21,924	20,581
MILLAR-1	8	3	14	0	1956	1820	NRP09	16	5	33	1	21,920	20,577
LLR	27	4	7	107	1139	979	NRP10	16	5	33	9	21,926	20,583
NRP01	16	5	33	4	21,924	20,581	NRP11	16	5	33	3	21,923	20,580

Table 2.3 – The characteristics of benchmark dataset-III

Instance	Nurses	Shift types	Days	Pre.	Var.	Const.	Instance	Nurses	Shift types	Days	Pre.	Var.	Const.
Instance01	8	1	14	34	301	517	Instance13	120	18	28	1081	68,027	65,257
Instance02	14	2	14	76	748	1030	Instance14	32	4	42	487	7543	9081
Instance03	20	3	14	84	1366	1896	Instance15	45	6	42	670	14,521	18,736
Instance04	10	2	28	91	1052	1622	Instance16	20	3	56	400	5158	7498
Instance05	16	2	28	138	1620	2676	Instance17	32	4	56	640	9996	13,742
Instance06	18	3	28	171	2390	3458	Instance18	22	3	84	590	8391	12,289
Instance07	20	3	28	208	2639	3769	Instance19	40	5	84	1154	21,983	29,391
Instance08	30	4	28	285	4799	6443	Instance20	50	6	182	3218	68,144	78,434
Instance09	36	4	28	304	5704	7132	Instance21	100	8	182	6502	171,542	210,222
Instance10	40	5	28	364	7526	9806	Instance22	50	10	364	6438	211,720	214,270
Instance11	50	6	28	436	10,842	12,582	Instance23	100	16	364	13010	639,836	644,926
Instance12	60	10	28	542	20,154	20,574	Instance24	150	32	364	19209	1,842,080	1,733,557

constraints are the same, they could be configured differently for each problem instance. For example, for the constraint "minimum and maximum number of working days", the value of minimum and maximum are often different across the benchmark problem instances, thereby leading to different number of variables and constraints. For more details regarding the definition of constraints of each instance in datasets I, II, and III, we refer interested readers to Burke et al. [19], Rahimian [65], and Curtois and Qu [26], respectively.

According to our computational experiments in this chapter and chapter 3, Instance20 to Instance 24 are large-size and the most difficult instances in our benchmark datasets. Instance9, Instance11, and Instance19 in dataset-III and ORTEC01 and all the instances in dataset-II are medium-size and fairly difficult to solve. It should be also noted that to study the behaviour of the proposed hybrid algorithms in this dissertation, we run them for more than 60 minutes up to a few hours. However, only results for 10 and 60 minutes are reported and taken into consideration according to the literature and relevant practicalities. In general, for hard instances, for dataset-I, there is no improvement beyond 60 minutes. For dataset-II and dataset-III, there are some improvements but the proportional growth is low during the time.

To evaluate the proposed hybrid algorithm, we implemented it in Java 1.7, and used IBM CP solver 1.7 [45] and Gurobi IP solver 5.6 [35] for solving all CSP and IP models, respectively. The reason to use the aforementioned solvers is that they are easier to implement in terms of modelling, and also they suit our hybrid framework better than other software packages. In addition, we note that benchmarks reported by Mittelmann [57] show that Gurobi produces very similar results for most instances compared with other state-of-the-art IP solvers such as Cplex [44]. We ran our experiments on a PC with an Intel 3.4 GHz processor and 4 GB of RAM, and only on one CPU core, in order to have more accurate comparison. Running benchmarks on one CPU core ensures that the applied solver does not use concurrency techniques to boost its performance, which often makes the comparison more inaccurate [35]. Another reason for this is to minimise the interference of other processes while the applied solver is running. In addition, the CPU might use other techniques to boost the performance while more than one CPU unit is intensively involved which often is varied between different hardware brands and models. Note that running the algorithm multiple times increases the robustness of results not its accuracy.

For evaluation purposes, we ran our hybrid algorithm for 10 minutes. The reason is two-fold: first, the hybrid algorithm is primarily designed to perform well in short times, and second, the selected time is in line with the testing times used by most algorithms reported in the literature, including the time used in the INRC-I competition [39], so that a platform for a fair comparison is established. To have better insight into performance of the hybrid algorithm, it was also run for 60 minutes.

After extensive preliminary experimental testing of the algorithm using different settings, the following parameters are chosen: We dedicate 5%, 25%, and 50% of the computational time to the pre-solve, CSP generation, and IP solver steps, respectively. The remaining time is distributed equally to softness adjustment and lower-bound enhancement steps as they require significantly shorter times in comparison. Furthermore, we set threshold parameters $psThr$ and $cspThr$ of the pre-solve and CSP generation steps to 10,000 and 1000, and parameters q and $numSols$ of the CSP generation step to 5 and 1000, respectively. To set the value of $psThr$, we ran the algorithm on dataset-I using values 5000, 8000, 10000, and 15000. It was observed that the algorithm obtains better solutions with less variance when the threshold is set to 10000. Smaller and bigger values for this parameter could lower the efficiency of the hybrid algorithm. To set the value of q which is the maximum gap between the quality of best-obtained solutions in two consecutive threshold levels, two gaps, 5% and 10%, were tested along with other combinations of parameters. For each experiment in our computational tests, the algorithm is run 10 times per instance and average values are reported accordingly. It is worth mentioning for each instance the variance of obtained values within our tests is very small and almost negligible due to the inherent nature of the algorithm.

We conducted three experiments to test the proposed algorithm: first, we analysed the performance of different components of the hybrid algorithm, and how they affect the efficiency of the algorithm. Second, we compared the hybrid algorithm with standard IP and CP solvers and finally, we compared the performance of the hybrid algorithm with other state-of-the-art methods in the relevant literature for all the benchmark datasets.

The first experiment was designed to investigate the effectiveness of softness adjustment and lower-bound enhancement steps on the overall performance of the

Table 2.4 – Results of the hybrid algorithm by using different settings

Instance	CG solution	LE bound		WA + LE (default)			No WA			No LE			No WA + No CG		
		RR	LB	Obj.	LB	G(%)	Obj.	LB	G(%)	Obj.	LB	G(%)	Obj.	LB	G(%)
GPOST	21	1	1	5	5	**0.0**	5	5	**0.0**	5	5	**0.0**	9	5	44.4
GPOSTB	20	0	0	5	0	**100.0**	5	0	**100.0**	5	0	**100.0**	7	0	100.0
ORTEC01	1031	40	60	380	158	**58.4**	392	158	59.7	380	158	**58.4**	1968	158	92.9
ORTEC02	984	40	60	370	150	**59.5**	380	148	61.1	390	148	62.1	10,545	148	99.1
Valouxis-1	1651	0	0	20	10	**50.0**	20	10	**50.0**	20	10	**50.0**	1120	10	100.0
WHPP	728	0	0	5	0	**100.0**	5	0	**100.0**	5	0	**100.0**	24,061	0	**100.0**
NRP01	30,561	550	554	14,714	577	**96.1**	15,027	570	96.2	14,714	570	**96.1**	15,500	570	96.3
NRP02	45,125	180	180	19,025	227	**98.8**	21,731	227	99.0	21,731	200	99.1	31,741	227	99.4
NRP03	38,450	120	122	14,008	122	**99.1**	14,008	122	**99.1**	14,768	122	99.2	15,510	122	99.2
NRP04	41,890	1300	1300	13,960	1300	**90.7**	21,855	1300	94.1	17,220	1300	92.5	25,495	1300	94.9
NRP05	27,667	210	211	14,855	211	**98.6**	14,855	211	**98.6**	14,855	211	**98.6**	14,855	211	**98.6**
NRP06	47,295	130	136	2876	142	**95.1**	2911	138	95.3	2894	138	95.2	2911	138	95.3
NRP07	38,388	140	140	5520	141	**97.4**	5601	141	97.5	5588	141	97.5	5660	141	97.5
NRP08	16,515	110	125	371	201	**45.8**	380	201	47.1	371	201	**45.8**	385	201	47.8
NRP09	21,940	1130	1130	9850	1131	**88.5**	14,940	1131	92.4	14,940	1130	92.4	14,940	1131	92.4
NRP10	57,266	310	310	22,601	310	**98.6**	22,821	310	**98.6**	22,821	310	**98.6**	22,863	310	**98.6**
NRP11	60,011	110	110	6483	110	**98.3**	13,991	110	99.2	10,051	110	98.9	37,698	110	99.7

hybrid algorithm, where we ran the hybrid algorithm for 10 minutes and recorded the detailed results in Table 2.4. We do not report the results of instances SINTEF, MILLAR-1, and LLR, since they have very short solution time, hence giving us no insight into the impact of each component. In this table, the results of running the hybrid algorithm with default settings (i.e. keeping both softness adjustment (WA) and lower-bound enhancement (LE) steps), and without WA or LE are reported. We also add the result of the algorithm without WA and CSP generation (CG) step. For each setting, the obtained objective function value (*Obj.*), lower-bound (*LB*), and optimality gap (*G(%)*) were recorded. The optimality gap is defined as the discrepancy between the value of the best feasible solution (for the primal problem) and the value of the best lower-bound (feasible for the dual problem). When the optimality gap is zero, the best feasible solution is optimal. Furthermore, we report the initial solution generated from CG step, and the lower-bound generated from LE step in comparison with the Root Relaxation (RR) value.

As it can be seen, removing WA and LE from the main algorithm setting results in poorer performance for the majority of instances, while for some instances such as GPOST there is no change between the performance of different settings. This is because the expected impact of WA and LE steps is mostly realised when the problem instance contains a high number of conflicting constraints, and therefore, it is computationally difficult to solve. Thus, the impact of removing these two steps is only visible for some difficult instances such as ORTEC02 and NRP09. Indeed, when a problem instance is computationally easy to solve, even tightening the problem formulation or providing a better lower bound does not affect the whole performance of the search process significantly and the core components of the solution method are still able to achieve almost the same result.

Removing WA and CG from the main algorithm setting generates similar results to the results reported in Table 2.5 which confirms the importance of these components particularly CG in the performance of the algorithm (running this setting even for 60 minutes leads to similar results as running IP on its own). In general, according to the obtained results, including these components within the search process is preferable. It is worth mentioning that running the algorithm using default settings also generates better lower-bounds for 5 instances in total.

In order to compare the performance of the hybrid algorithm against standard IP and CP solvers, i.e. running the solvers in default settings without any tuning or extra reinforcing techniques, we ran the algorithm for 10 and 60 minutes (short and long runtimes), and reported the results in Table 2.5 and 2.6. On a side note, it should be noted that even if we tune the IP solver, it would not have any significant impact on the benchmarking results in comparison because the hybrid algorithm is made upon on the same IP solver. In Tables 2.5 and 2.6, *Obj.*, *LB*, and *G(%)* show the obtained solution, lower-bound, and optimality gap, respectively. The best results in each runtime are shown in bold style. It should be noted that for running the CP solver for benchmark instances, therefore incorporating the soft constraints into the CP model explained in Section 2.4, we apply a COP model in a similar way to Qu and He [64], who applied CP by using two different sub-models.

Running the IP solver for 10 minutes, one can see that the hybrid algorithm found better solutions for most instances particularly for difficult ones and improved lower-bounds for some such as ORTEC01. Overall, from 9, 11, and 19 benchmark instances included in all the datasets, the hybrid algorithm obtains better solutions for 56%, 82%, and 47% of the instances within 10 minutes.

We also run the IP solver for the longer time 60 minutes. It can be seen that the hybrid algorithm is able to find a similar result to the 10-minute experiment using dataset-I and dataset-II. Using dataset-III, the algorithm finds the same result as the standard IP solver. That said, a question might arise is that if it is worth using the algorithm on this benchmark dataset. To answer that, two aspects should be noted. First, the difficulty of the benchmark dataset should be

Table 2.5 – Benchmark results of the hybrid algorithm and standard IP and CP solvers within 10 and 60 minutes using dataset-I and dataset-II

Instance	Hybrid Algorithm						IP						CP
	10 min			60 min			10 min			60 min			60 min
	Obj.	LB	G(%)	Obj.	LB	G(%)	Obj.	LB	G(%)	Obj.	LB	G(%)	Obj.
GPOST	5	5	**0.0**	5	5	**0.0**	9	5	44.4	5	5	**0.0**	11
GPOSTB	5	0	**100.0**	3	0	**100.0**	7	0	**100.0**	5	0	**100.0**	8
ORTEC01	380	158	**58.4**	270	210	**22.2**	1970	140	92.9	720	210	70.8	-
ORTEC02	370	150	**59.5**	270	210	**22.2**	15,415	140	99.1	700	210	70.0	830
Valouxis-1	20	10	**50.0**	20	20	**0.0**	1090	0	100.0	90	0	100.0	180
SINTEF	0	0	**0.0**	0	0	**0.0**	8	0	100.0	0	0	**0.0**	0
MILLAR-1	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**	0	0	**0.0**	0
WHPP	5	0	**100.0**	5	5	**0.0**	24,061	0	**100.0**	1004	0	100.0	67,050
LLR	301	301	**0.0**	301	301	**0.0**	301	301	**0.0**	301	301	**0.0**	301
NRP01	14,714	577	**96.1**	9971	610	**93.9**	15,500	570	96.3	11,162	610	94.5	-
NRP02	19,025	227	**98.8**	11,850	230	**98.1**	31,741	200	99.4	12,850	230	98.2	-
NRP03	14,008	122	**99.1**	5366	181	**96.6**	15,510	122	99.2	8553	181	97.9	-
NRP04	13,960	1300	**90.7**	10,241	1360	**86.7**	25,495	1300	94.9	13,385	1360	89.8	-
NRP05	14,855	211	**98.6**	14,855	271	**98.2**	14,855	211	**98.6**	14,855	271	**98.2**	-
NRP06	2876	142	**95.1**	2459	200	**91.9**	2911	138	95.3	2521	200	92.1	-
NRP07	5520	141	**97.4**	5012	141	**97.2**	5660	141	97.5	5301	141	97.3	-
NRP08	371	201	**45.8**	239	201	**15.9**	385	201	47.8	241	201	16.6	-
NRP09	9850	1131	**88.5**	5851	1200	**79.5**	14,940	1130	92.4	5880	1200	79.6	-
NRP10	22,601	310	**98.6**	18,985	3508	**81.5**	22,863	310	**98.6**	22,551	3508	84.4	-
NRP11	6483	110	**98.3**	5409	160	**97.0**	37,698	110	99.7	5828	160	97.3	-

Table 2.6 – Benchmark results of the hybrid algorithm and standard IP and CP solvers within 10 and 60 minutes using dataset-III

Instance	Hybrid Algorithm						IP						CP
	10 min			60 min			10 min			60 min			60 min
	Obj.	LB	G(%)	Obj.	LB	G(%)	Obj.	LB	G(%)	Obj.	LB	G(%)	Obj.
Instance01	607	607	0.0	607	607	0.0	607	607	0.0	607	607	0.0	-
Instance02	828	828	0.0	828	828	0.0	828	828	0.0	828	828	0.0	-
Instance03	1001	1001	0.0	1001	1001	0.0	1001	1001	0.0	1001	1001	0.0	-
Instance04	1716	1716	0.0	1716	1716	0.0	1716	1716	0.0	1716	1716	0.0	-
Instance05	1143	1143	0.0	1143	1143	0.0	1143	1143	0.0	1143	1143	0.0	-
Instance06	1950	1950	0.0	1950	1950	0.0	1950	1950	0.0	1950	1950	0.0	-
Instance07	1056	1056	0.0	1056	1056	0.0	1056	1056	0.0	1056	1056	0.0	-
Instance08	5784	1290	77.7	1323	1300	1.7	8995	1280	85.8	1323	1300	1.7	-
Instance09	439	411	6.4	439	439	0.0	439	406	7.5	439	439	0.0	-
Instance10	4631	4631	0.0	4631	4631	0.0	4631	4631	0.0	4631	4631	0.0	-
Instance11	3443	3443	0.0	3443	3443	0.0	3443	3443	0.0	3443	3443	0.0	-
Instance12	4045	3636	10.1	4040	4040	0.0	4045	3636	10.1	4040	4040	0.0	-
Instance13	112,462	500	99.6	3109	519	83.3	500,410	500	99.9	3109	519	83.3	-
Instance14	1397	1276	8.7	1280	1278	0.2	1482	1276	13.9	1280	1278	0.2	-
Instance15	45,379	3811	91.6	4964	3740	24.7	78,144	3766	95.2	4964	3740	24.7	-
Instance16	3501	3213	8.2	3233	3224	0.3	3521	3213	8.8	3233	3224	0.3	-
Instance17	6044	5733	5.1	5851	5706	2.5	6149	5731	6.8	5851	5706	2.5	-
Instance18	7215	4175	42.1	4760	4291	9.9	7950	4245	46.6	4760	4291	9.9	-
Instance19	18,568	2673	85.6	5420	2689	50.4	29,968	2673	91.1	5420	2689	50.4	-

Table 2.7 – Benchmark results of the hybrid algorithm against VNS-1 [18], MA [12], VDS [24], HSA [36], SS [20], VNS-2 [55], BAP [13], and SVN [78] using dataset-I

Instance	BKS	HA	VNS-1	MA	VDS	HSA	SS	VNS-2	BAP	SVN
GPOST	**5**	**5**	-	915	-	-	9	8	**5**	-
GPOSTB	**3**	5	-	789	-	-	5	-	**3**	-
ORTEC01	**270**	380	541	535	360	310	365	-	**270**	-
ORTEC02	**270**	370	-	-	-	330	-	-	**270**	-
Valouxis-1	**20**	**20**	-	560	-	-	100	160	80	73
SINTEF	**0**	**0**	-	8	-	-	4	-	**0**	-
MILLAR-1	**0**	**0**	-	100	-	-	**0**	**0**	**0**	-
WHPP	**5**	**5**	-		-	-	-	-	**5**	**5**
LLR	**301**	**301**	-	305	-	-	**301**	314	**301**	**301**

taken into consideration. As we have seen the result using dataset-I and dataset-II in Table 2.5, the proposed algorithm generates better results rather than the standard IP solver. Second, the pertinent business requirements (e.g. required solution quality and time frame) should be evaluated to justify what solution method is required. It could be cases that only a standard IP solver is sufficient for satisfying underlying business expectations.

Running the CP solver within the long runtime using all the benchmark instances, it is not able to find even feasible solutions for most instances. In effect, the performance of the CP solver is inferior likely due to the complexity (e.g. number of conflicting constraints) and size (e.g. number of nurses and days) of the benchmark instances. Therefore, it can be seen that hybridising IP and CP leads to better performance for most instances in comparison with standard IP and CP solvers and indeed it is able to extend the reach of IP and CP methods.

To compare the performance of the hybrid algorithm (HA) with state-of-the-art algorithms reported in the literature by using dataset-I, we present in Table 2.7 the best-published results by a hybrid Variable Neighbourhood Search [18] (VNS-1), a Memetic Algorithm [12] (MA), a Variable Depth Search [24] (VDS), a Harmony Search Algorithm [36] (HSA), a Scatter Search [20] (SS), another variant of hybrid Variable Neighbourhood Search [55] (VNS-2), a hybrid Branch-and-Price algorithm [13] (BAP), and a stochastic VNS [78] (SVN). In this table, *BKS* shows the best-known solutions from the literature for the benchmark instances, which are obtained using column generation and relaxation techniques with an IP solver or other heuristic methods within a long runtime. "-" shows that there is no published result for the relevant instance using the benchmarking algorithm. All the algorithms were run for 10 minutes except VNS-1, which was executed for one hour.

Table 2.8 – Benchmark results of the hybrid algorithm in comparison with the branch-and-price and ejection chain heuristic [13] running for 10 and 60 minutes using dataset-III

Instance	Hybrid algorithm	Ejection chain	Hybrid algorithm	Ejection chain	B&P
	10 min		60 min		
Instance01	607	607	607	607	607
Instance02	828	923	828	837	828
Instance03	1001	1003	1001	1003	1001
Instance04	1716	1719	1716	1718	1716
Instance05	1143	1439	1143	1358	1160
Instance06	1950	2344	1950	2258	1952
Instance07	1056	1284	1056	1269	1058
Instance08	1364	2529	1344	2260	1308
Instance09	439	474	439	463	439
Instance10	4631	4999	4631	4797	4631
Instance11	3443	3967	3443	3661	3443
Instance12	4042	5611	4040	5211	4046
Instance13	3109	8707	1905	3037	-
Instance14	1281	2542	1279	1847	-
Instance15	4144	6049	3928	5935	-
Instance16	3306	4343	3225	4048	3323
Instance17	5760	7835	5750	7835	-
Instance18	5049	6404	4662	6404	-
Instance19	3974	6522	3224	5531	-
Instance20	5242	23,531	4913	9750	-
Instance21	26,977	38,294	23,191	36,688	-
Instance22	130,107	-	32,126	516,686	-
Instance23	40,543	-	3794	54,384	-
Instance24	2,829,680	-	2,281,440	156,858	-

As it can be observed, the proposed algorithm is able to generate competitive solutions using dataset-I. Looking at the results for instances ORTEC01 and ORTEC02, the limitations of using IP-based approaches versus (meta-)heuristic algorithms become clear.

Table 2.8 shows the results obtained by an ejection chain method [13] running for 10 and 60 minutes using dataset-III. The results of a branch-and-price (B&P) algorithm [13] without any time limits are also presented.

It can be seen that the results are promising for the short runtime except for large instances where limitations of IP become involved. For Instance20 to Instance24 the algorithm was not able to generate any results due to huge sizes of these instances for short and long runtimes.

Overall, hybridising IP and CP is able to extend the reach of IP and CP methods individually, though the limitations of such approaches are persistent. That said, using IP/CP-based approaches such as the hybrid algorithm presented in this chapter is beneficial when an IP/CP solver is available (e.g. a commercial IP solver has been already bought and in use) and an increase of the computational power/reach is sought, or when having optimality information is essential for business purposes (indeed in such cases, the company is seeking to increase the value of the money that has been spent for purchasing an IP/CP solver). Since the hybrid algorithm uses IP and CP models for modeling the NRP, applying it to other problem instances with the similar structure as we have seen for different datasets here in this chapter is expected to generate similar results. Nevertheless, limitations of IP and CP methods on which the hybrid algorithm is based are still in place. That said, we expect a high consumption of computer memory and a considerable decline in the performance of the hybrid algorithm when large-size instances (e.g. more than 50 nurses and/or more than 6 months scheduling duration) are dealt with.

2.7 Concluding Remarks

In this chapter, we proposed a hybrid algorithm to extend the computational reach of IP and CP. The algorithm utilised strengths of CP to aid the IP solver in achieving higher-quality solutions. We also developed some components to provide valuable problem-specific information such as the computational difficulty of instances and constraints to both IP and CP solvers so that better performance can be achieved in solving problem instances. We also attempted to design a hybrid method for generating a high-quality solution as well as a strong lower-bound in order to guarantee the solution quality. Due to the high-level hybridisation of IP and CP, an important aspect of the proposed hybrid algorithm is its straightforward adaptability to practical circumstances in terms of implementation, particularly where an IP/CP solver is already being used.

We benchmarked the hybrid algorithm with three different datasets consisting of a variety of instances. In comparison with standard IP and CP solvers, the obtained results showed better performance, indicating the effectiveness of the proposed hybridisation. Considering the limitations of IP and CP, we also obtained acceptable results compared with state-of-the-art algorithms for a diverse range of instances. In summary, depending on the underlying business requirements and the difficulty of the problem in hand, the proposed algorithm could be used to extend the reach of IP and CP to a promising level.

2.8 Contributions

This chapter extends the computational reach of IP and CP through hybridisation, where the strength of CP in generating feasible solutions, and the capability of IP in obtaining the optimal solution and providing optimality information are exploited. To aid the search process, some algorithmic components exploiting problem-specific information are incorporated into the hybrid algorithm to reduce the search space and fine tune search parameters. The algorithm is benchmarked against state-of-the-art algorithms, and standard IP and CP solvers. The tests show competitive results despite combining two exact methods together, which are often not capable of solving practical complicated instances solely.

Modelling the problem as integer program and constraint satisfaction problem makes the hybrid algorithm flexible enough for applying to a variety of problem instances having different levels of softness of constraints. It means the hybrid algorithm is less dependent on a particular setting (e.g. softness) of constraints in comparison with heuristics mostly relying on instance-specific information. Moreover, the hybrid algorithm is capable of obtaining optimality information in comparison with heuristics, which mostly focuses on obtaining higher-quality solutions.

Due to the special design of the hybrid algorithm which can be seen as a reinforced IP solver, it is capable of generating good-quality solutions in relatively short computational times. This characteristic is beneficial particularly in practical settings, where often a good-quality roster (which is defined based on the underlying business requirements) is needed in rather short times (it varies in different business areas. For example, according to our experience, it is about 1 to 3 hours in some hospitals and wards in Scotland, Glasgow).

Chapter 3

A Hybrid Variable Neighbourhood Search And Integer Programming Algorithm

One of the important aspects of designing solution methods for optimisation problems is how to evaluate the quality of obtained solutions. In the literature, the quality of attained rosters is often assessed based on the sum of all penalties occurred due to the violations of soft constraints [74, 87, 28, 13]. Although this approach is fast and straightforward and also fits the way many heuristics work (i.e. a roster is evaluated and if it has less penalty it is chosen as the best current roster, otherwise it is ignored), it has some disadvantages: first, there might be cases that few rosters are assigned the same penalty, though they are indeed different. That said, the performance of the search algorithm gets weaker in finding the best-quality roster. Second, considering a specific constraint, there might be cases that few nurses are assigned the same penalty, though they violate the constraint in different ways. In other words, there is no clear and transparent picture of how violations are accumulated which defines the perception of quality in evaluation of rosters. Since most solution methods operate based on the quality of obtained solutions and decide how to move through the solution space accordingly, a poor-quality solution evaluation mechanism leads to unwanted inefficiencies and redundancies. To reduce the resulting flaws, this chapter introduces a new solution evaluation mechanism, on which a new hybrid algorithm is designed.

In this chapter, we combine the strengths of IP and VNS algorithm to design a hybrid method for solving the NRP. Both IP and VNS are common within the NRP literature and many works [78, 28, 74] have been published as per these methods. After generating the initial solution using a greedy heuristic, the solution is further improved by employing a Variable Neighbourhood Descent (VND) algorithm (a particular type of VNS). Then IP, deeply embedded in the VNS algorithm, is employed within a ruin-and-recreate framework to assist the search process. Finally, IP is called again to further refine the solution during the remaining time. We utilise the strength of IP not only to diversify the search process, but also to intensify the search efforts. To identify the quality of the current solution, we use a new generic scoring scheme to mark the low-penalty parts of the solution. Based on the computational tests with some benchmark instances, promising results are obtained.

The results presented in this chapter are published by Rahimian et al. [66].

3.1 Introduction

In this chapter, we propose a hybrid IP and VNS algorithm to solve the NRP, where a new solution evaluation mechanism is employed within a ruin-and-recreate framework to focus the search process on most promising areas, i.e. areas that might have a higher-quality solution if get searched. First, a greedy heuristic is used to generate an initial solution, and then the generated solution is further improved using a VNS algorithm until a stopping criterion is met. To further enhance the efficiency of the VNS algorithm, IP is employed iteratively during the running of the algorithm as a neighbourhood structure to improve the quality of the incumbent solution using a ruin-and-recreate framework. In this framework, high-penalty components of the solution are destroyed according to a generic scoring scheme, and then they are created again by an IP solver. Finally, IP is applied once more to the best-found solution to improve it globally as much as possible until the overall time limit is reached. The proposed algorithm is designed to perform efficiently when only short computational times are available, so that many practical problems can be tackled.

Most academic research have considered the evaluation of obtained solutions according to soft constraints. In this approach, the penalty associated with a solution is calculated via multiplying violations of soft constraints (or even hard constraints) by relevant user-defined weights. For example, Valouxis et al. [83] utilised a two-phase constructive approach in which an IP solver solves series of problems by considering different soft constraints in each step. In particular, they assign nurses to working days in the first step (one problem per week) and then nurses are assigned to shift types in the second step (one problem per shift type per day). Lü and Hao [53] also presented an adaptive neighbourhood search algorithm in which obtained solutions are evaluated according to violations of all soft constraints with each iteration. Burke et al. [16] and Cowling et al. [25] applied a tabu search hyper-heuristic in which infeasibility (regarding hard constraints) is considered in evaluation of solution quality to balance under-covered and over-covered shifts in a set of days. One of the potential weaknesses of this approach is that there could be many solutions whose calculated penalties are the same but have different allocations of nurses and days (different structure). This could lead to a completely different search path in most heuristic methods. Because, in such methods, often the best solution which has the lowest penalty is kept and all other solutions which have the same or higher penalty are discarded while a neighbourhood is being searched. Therefore, a solution which could lead to a better solution in following neighbourhood searches is already discarded. In general, most heuristics (or even exact methods such as IP or CP) operate based on the solution quality which is measured by a function given by the user. If the measured penalty is not generated accurately and genuinely and is the same for a bigger population of solutions, the search method is not able to track the best solution easily. Another downside is that the contribution of violated soft constraints into the calculated penalty is not clear. For example, considering the total penalty 1000, there can be a solution in which the contribution of constraint A and B is 200 and 800, respectively, while there can be another solution having the same penalty to which constraints A and B contribute 600 and 400. This could lead to difficulties when the user needs to decide which solution is the best solution (among a population of solutions which have the same penalty) according to the underlying business requirements at the end of the search process. That said, knowing the contribution of each constraint to the total penalty helps the user to identify most important constraints and choose the most proper solution according to his needs. These weaknesses mean provision of a poor picture of search space, which then lessen the performance and effectiveness of the search algorithm.

In the literature, depending on what solution method is employed, other approaches are also used to evaluate attained solutions. Burke et al. [23] applied a similar approach to Li et al. [50], in which authors proposed a pareto-based search technique by modelling the NRP as a multi-objective optimisation problem. For evaluating solution quality, they considered the contribution of each nurse assignment towards the solution feasibility. Brucker et al. [9] applied a constructive and decomposition-based solution method in which constraints of the problem were categorised as schedule- and roster-related. Obtained solutions were evaluated by ranking schedules of nurses. In this chapter, we introduce a scoring scheme which evaluates the quality of obtained solutions according to the associated underlying elements such as nurses or days, thereby empowering the algorithm to genuinely focus on most promising parts of the solution. The strength of the proposed scoring scheme is that the contribution of violated soft constraints is captured which is then used to destroy high-penalty parts of the solution.

Another feature of our approach is to embed IP as a neighbourhood structure through a ruin-and-recreate framework in the VNS algorithm to improve the quality of the obtained solution and diversify the search process at the same time. Our method of hybridisation is different from similar algorithms (sometimes called Matheuristics [28, 2]) reported in the literature [64, 21, 80]. In fact, there are various hybridisation schemes in order to combine different approaches together [70]. For example, Qu and He [64] applied CP to generate an initial solution by decomposing the problem into various sub-problems, and then applied VNS to improve the generated solution. Stølevik et al. [80] applied an Iterated Local Search framework for generating an initial solution and employed VNS and CP in order to improve the solution and diversify the search process, respectively. Burke et al. [21] employed IP to generate a solution satisfying all hard constraints, and then improve it using VNS to satisfy the remaining soft constraints. Gomes et al. [34] applied a VNS to generate rosters using the column generation method and then implemented a relax-and-fix heuristic to improve the feasibility and optimality of the generated rosters. In these approaches, IP or CP is used to generate a solution satisfying some constraints of the problem (or parts of the problem), and then a meta-heuristic algorithm is applied to further improve the generated solution. However, in our approach, we employ VNS as the main local search framework and then embed IP as a neighbourhood structure to intensify and diversify the search process in an iterative manner considering all constraints. That said, incorporating IP into our hybrid algorithm, we also allow the search process to traverse the infeasible space by allowing all the constraints to be violated in order to find out latent feasible solutions.

The rest of this chapter is organised as follows. We first elaborate on the solution method and different components of the proposed hybrid algorithm in Section 3.2. In Section 3.3 and 3.5, we present our computational results and concluding remarks, respectively. It should be noted that throughout this chapter we use the same models and benchmark datasets presented in Chapter 2.

3.2 Hybrid Approach

In this section, we describe a hybrid method combining VNS and IP techniques (aka. matheuristic [28, 2]) to solve the NRP. The schematic overview of the proposed hybrid algorithm is demonstrated in Algorithm 2.

Algorithm 2: The complete pseudo code of the hybrid algorithm

```
1  $x^* \leftarrow x \leftarrow GreedyHeuristic()$;
2  repeat
3      $x \leftarrow VNDSearch(x)$;
4      $x \leftarrow IPRuinAndRecreate(x)$;
5      if $x < x^*$ then
6          $x^* \leftarrow x$;
7      end
8  until some stopping criteria are met;
9  $x^* \leftarrow IPImprove(x^*)$;
10 return $[x^*]$
```

After generating an initial solution using a greedy heuristic (*GreedyHeuristic()*), a VND algorithm (*VNDSearch()*) using a set of distinct neighbourhoods tries to improve the generated initial solution until no more improvements can be obtained by cycling through all the neighbourhoods. Then the best solution obtained from the VND algorithm is employed by an IP solver by fixing low-penalty parts of the solution, where it tries to generate a better-quality (exploitation) and a different-structured (exploration) solution. In fact, in this step, the best solution obtained so far is partially destroyed and again recreated aiming to have a higher-quality, and at the same time, a different solution in terms of the underpinning structure (a ruin-and-recreate strategy). In other words, the IP solver is applied as a neighbourhood within the VNS, aiming to change the structure and at the same time to improve the quality of the obtained solution. All this process is accomplished in the *IPRuinAndRecreate()* block. To ensure a sufficient diversification within the search process, and therefore, not getting stuck in local optima, some low-penalty parts of the solution might also be destroyed and recreated randomly. The final obtained solution is again imported to the VND search algorithm and this process continues until some stopping criteria are met. Ultimately, the attained solution is further improved by applying IP to the whole problem instance to ensure a holistic search until the overall time limit is reached (*IPImprove()).*

Next, we elaborate on each of the main components of the hybrid algorithm, i.e. initial solution construction in *GreedyHeuristic()* block, VND search algorithm in *VNDSearch()* block, and IP ruin-and-recreate framework in *IPRuinAndRecreate()* block.

3.2.1 Initial Solution Construction

In this block, a greedy heuristic search is employed to generate an initial solution for the VND algorithm. Empirically, by trying different initial solutions, we have observed that having a high-quality initial solution reduces the efficiency of the VNS algorithm subsequently. For the same reason, a random initialisation often results in poor performance due to the very low-quality of the generated solution. This sensitivity is mostly because of the IP solver employed within the algorithm. In effect, if the quality of the initial solution is too high, it disrupts the performance of the IP solver as it needs historical information during the search process which gets cut if a high-quality initial solution is provided. Therefore, we decide to apply a simple greedy heuristic algorithm which is able to generate a different solution per experiment. The pseudo-code of this algorithm is depicted in Algorithm 3. After creating an empty solution (roster), at the first step, we randomly select a nurse whose all pre-defined days off are set in *SetDaysOff()* block. In the next step, we randomly mark all the working days to which a specific shift needs to be assigned later (*AssignWorkDays()*), and then we assign a randomly selected shift to those days accordingly (*AssignShifts()*). Assigning shifts to the nurses within two different levels, i.e. first assigning working and non-working days, and then assigning shifts to the working days, helps us to only check the constraints related to each level independently, and hence, reducing the complexity of the constraint conflict resolution process. That said, in the first level, only the maximum number of working days, and the minimum and maximum number of consecutive shift types including day off shifts are checked. Therefore, in the next level, we only need to check the maximum number of shift types and avoid assigning shifts where a forbidden pattern is matched.

Algorithm 3: The pseudo code of the greedy heuristic algorithm to generate initial solutions

```
1  Create an empty solution $x = \{x_i, i \in I\}$;
2  foreach $i \in I$ do
3      while $x_i$ is not feasible do
4          $x_i \leftarrow SetDaysOff(x_i)$;
5          $x_i \leftarrow AssignWorkDays(x_i)$;
6          $x_i \leftarrow AssignShifts(x_i)$;
7          $p_1 = EvaluateWorkload(x_i)$;
8          $p_2 = EvaluateWeekend(x_i)$;
9          if $p_1 + p_2 > 0$ then
10             $x_i \leftarrow Destroy(x_i)$;
11         end
12     end
13 end
14 return [x]
```

Finally, to ensure satisfying the remaining constraints, i.e. the maximum number of worked weekends, and the minimum and maximum total times of assigned shifts, we calculate the associated incurred penalties (indicated as p_1

and p_2, computed within the *EvaluateWeekend()* and *EvaluateWorkload()* blocks, respectively). If there is any associated penalty, we destroy the current schedule for the current nurse by unassigning all the allocated shifts using the *Destroy()* block, and repeat the process until a feasible solution is obtained. This process is iterated for all the nurses until all the required shifts are assigned to all the days within a schedule, while satisfying all the hard constraints. According to our experiments, the greedy heuristic is able to produce a feasible solution for all the problem instances in up to 100 cycles per nurse. Finally, the generated feasible solution is returned to the VND algorithm for further improvement.

3.2.2 Variable Neighbourhood Descent

When there is an initial solution, either generated using the greedy heuristic algorithm in the *GreedyHeuristic()* block or passed from the previous cycle of the hybrid algorithm to the current one (see Algorithm 2), the VND algorithm is applied to refine the solution locally according to a set of distinct neighbourhoods [38]. In the VND search algorithm, a best-improvement descent local search algorithm is applied through cycling over a set of neighbourhoods (full search per neighbourhood) until no improvement can be found in all the neighbourhood structures, or when the total number of iterations is reached a certain maximum value.

VNS as a generalized VND approach is a meta-heuristic approach based on the simple idea of systematically changing neighbourhoods both to escape from the areas which contain local optima and within a local search to identify better local optima [37, 38]. It has been applied to many $\mathcal{NP}$-hard problems including the NRP [21, 80]. In a simple VNS scheme, a local search is applied to the incumbent solution using a neighbourhood structure until certain criteria such as the total number of iterations are met. Then the local search is restarted using a different neighbourhood structure, trying to improve the best solution obtained from the previous iterations. This process continues until there is no more improvement gained from any of the neighbourhood structures. The neighbourhood structures in a VNS are often selected to drive the search process towards different desired objectives, or to investigate different structures of the obtained solution in order to diversify the search process, and therefore, to avoid getting stuck in local optima. This capability of exploring a variety of neighbourhoods aiming different structures suits very well with the applied ruin-and-recreate framework designed upon our specially-designed scoring scheme. Furthermore, VNS (and its varieties) is one of the common solution methods to solve the NRP in the relevant literature [21, 80, 88].

The following neighbourhoods are applied to the VND block of the hybrid algorithm (*VNDSearch()*):

1. **2-Exchange:** this neighbourhood consists of all moves, where two shifts are swapped between two different nurses on the same day.
2. **3-Exchange:** it includes all moves, where three (or more) shifts are exchanged between three (or more) different nurses on the same day.
3. **Double-Exchange:** it includes all moves that swap two shifts between two different nurses on two different days. In fact, this neighbourhood is made from two different *2-Exchange* neighbourhoods applying on two consecutive days.
4. **Multi-Exchange:** this neighbourhood is very similar to *Double-Exchange* but three (or more) shifts are swapped between two different nurses on three (or more) different days. Indeed, this neighbourhood is made from three (or more) different *2-Exchange* neighbourhoods, which are not necessarily applied on consecutive days.
5. **Block-Exchange:** this neighbourhood includes all moves where a specific

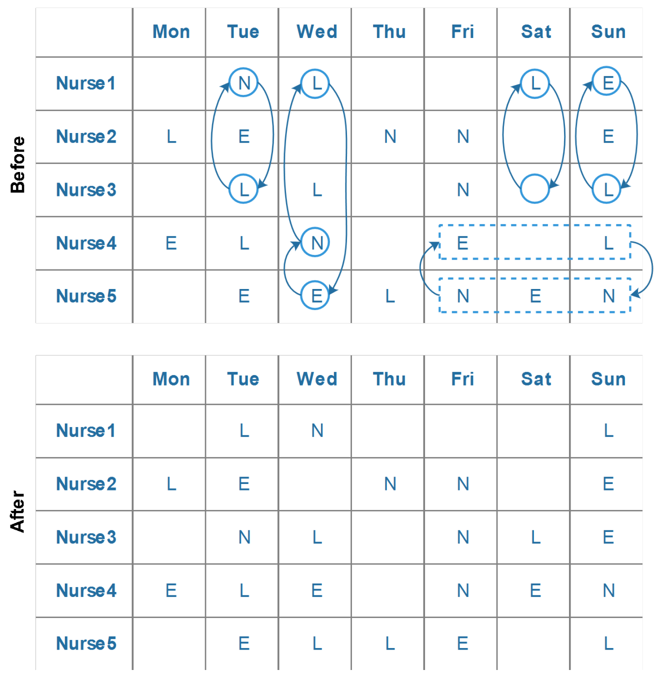
Table 3.1 – Improvement percentages of the VNS algorithm by applying 2-Exchange, Double-Exchange, and Multi-Exchange neighbourhoods using Dataset-I

Instance	Two	Double	Multi3	Multi4	Multi5
GPOST	51%	97%	98%	98%	97%
GPOSTB	56%	94%	94%	93%	95%
Valouxis-1	99%	99%	99%	99%	99%
WHPP	45%	54%	85%	84%	85%
ORTEC01	91%	91%	96%	97%	96%
ORTEC02	93%	93%	94%	95%	92%
Ave.	73%	87%		94%	

number of consecutive shifts is swapped between two different nurses within the planning period.

Apart from *Multi-Exchange* neighbourhood, which is our new neighbourhood structure, the rest of the defined neighbourhoods are used in many local search algorithms in the literature (for example see [21, 80]). The *Multi-Exchange* neighbourhood is defined to overcome the complex structure of the problem, and therefore, to overcome the potential complicated local optima by applying some simple *2-Exchange* moves simultaneously. In fact, on the one hand, this neighbourhood helps to break complicated structures for a number of constraints such as *3.f* and *3.b*. On the other hand, it is helpful to move good shift patterns from one nurse to another. Experimentally speaking, this neighbourhood structure gives better performance rather than a simple *2-Exchange* or *Double-Exchange* as we can see in Table 3.1. In this table, the improvement percentages of the VNS algorithm by using three neighbourhoods 2-Exchange, Double-Exchange, and Multi-Exchange with lengths 3, 4, and 5 within 2 minutes runtime are shown.

The defined neighbourhoods are illustrated on a weekly roster for five nurses in Figure 3.1, where E, L, and N indicate early, late, and night shifts, respectively, and all blank shifts are days off. In this figure, the swaps of shifts between nurses 1 and 3 are examples of *2-Exchange* (Tue), *Multi-Exchange* (Tue, Sat, and Sun), and *Double-Exchange* (Sat and Sun) neighbourhoods. As an example



	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Nurse1		N	L			L	E
Nurse2	L	E		N	N		E
Nurse3		L	L		N		L
Nurse4	E	L	N		E		L
Nurse5		E	E	L	N	E	N

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Nurse1		L	N				L
Nurse2	L	E		N	N		E
Nurse3		N	L		N	L	E
Nurse4	E	L	E		N	E	N
Nurse5		E	L	L	E		L

Figure 3.1 – Examples of 3-Exchange (Nurses 1, 4, & 5 on Wed), Block-Exchange (Nurses 4 & 5 from Fri to Sun), Multi-Exchange (Nurses 1 & 3 on Tue, Sat, and Sun), 2-Exchange (Nurses 1 & 3 on Tue), and Double-Exchange (Nurses 1 & 3 on Sat and Sun) neighbourhoods applied in the VND algorithm

of *3-Exchange* neighbourhood, the three shifts between nurses 1, 4, and 5 on Wednesday can be sequentially swapped. Moreover, swapping the blocks of shifts from Friday to Sunday between nurses 4 and 5 can be an instance of a *Block-Exchange* neighbourhood.

3.2.3 IP Ruin-and-Recreate Framework

If the VND search algorithm could not find any better solutions by cycling through the set of neighbourhoods or reaches a maximum number of iterations, the incumbent solution is passed to an IP solver as a perturbation neighbourhood structure within the VNS. The IP solver searches for a better alternative solution based on the IP model of the problem introduced in Section 2.3, by fixing the low-penalty parts of the solution and exploring all the remaining possibilities to find a higher-quality solution in an iterative manner (*ruin-and-recreate* framework).

Throughout this process, two possible outcomes may happen: 1) the IP solver can find a better solution, which can be a different solution in terms of the underlying structure in comparison with the last one. In this case, the IP solver helps the VND algorithm both in terms of intensification and diversification. 2) the IP solver cannot produce a better-quality solution due to the timeout criterion or due to the non-existence of a better alternative. In this case, the IP solver may produce a solution with a different structure, and hence, helps the search process only in terms of diversification. In either case, the role of the embedded IP component in the hybrid algorithm is essential. It is noteworthy to mention that by destroying parts of the solution and repairing them again, there is no guarantee of the quality and structure of the new solution.

In terms of diversification, generating a solution with a different structure is crucial. The structure of the solution is different, if it cannot be obtained by searching through the defined neighbourhoods of the current solution and some nearby solutions. Indeed, a solution is different from the other solutions in terms of the underlying structure, if we cannot generate it by iteratively applying the defined neighbourhoods for a sufficient number of iterations which is varied per instance. In our experiments on Dataset-I, IP generates different solutions which cannot be obtained after at least 50 iterations. In the literature, a ruin-and-recreate strategy either by IP or CP is mostly employed in order to diversify the search process and to perturb the obtained solution. For example, Stølevik et al. [80] have applied this strategy by destroying parts of the solution and then using CP to rebuild it. Li et al. [50] have also used this strategy by the evolutionary elimination of parts of the solution and subsequently repairing it by using a greedy heuristic. A more advanced ruin-and-recreate based algorithm is also reported by Li et al. [51], who applied a stochastic modelling and Markov chain analysis. Nonetheless, in the proposed hybrid algorithm, we apply this strategy not only for the diversification purpose but also for improving the quality of the obtained solution, i.e. intensification. This is the main reason that we select IP for the ruin-and-recreate framework compared with CP and other heuristics (which are often powerful for either intensification or diversification), which is able to improve the current solution, and at the same time, to investigate many areas within the search space.

Another novel aspect of our ruin-and-recreate framework is to apply a flexible generic scoring scheme for evaluating different parts of the solution, which allows us to adaptively focus on most promising areas of the solution. In order to fix some parts of the solution, we apply a scoring scheme by assigning a value to each cell within the roster, where each cell is an intersection of one particular day and one particular nurse. In fact, using the scoring scheme, the total penalty associated with the current solution can be broken down to fundamental elements of the problem. It means a shift can be assigned to each cell and if so, there is an associated penalty according to the objective function and the constraints that are involved. In other words, each cell can be designated by a value, which is the proportion of (an assignment to) that cell of the total number of violations respecting to the current solution. We call this estimated value *cell penalty.* Cell penalties can be easily aggregated to different dimensions, therefore providing an insightful tool to analyse and discriminate different parts of the solution. For example, in *Random* configuration which we explain later in this section, this value is used as a weight in a simple linear weighted random function, where a random cell is selected in order to be destroyed later.

Next, we demonstrate how to calculate a cell penalty, which is calculated based on the total incurred violations of the involved constraints. Here, we consider all the constraints either hard or soft, which might be violated throughout the search process, and hence, we also define the violations associated with hard constraints. To determine the relevant weights of hard constraints, a significant value (e.g., 10000) is selected to ensure that the final solution is feasible by directing the

Table 3.2 – The required information to calculate cell penalties for Model-I and Model-II

Ct.	Weight (w_c)	Violation	Cell share (s_c)	Affected cells
2	B_d^{cv}	number of deviations from the minimum and maximum values	$\sum_{a \in A} {}^{\lvert V\rvert}/_{\lvert E\rvert}$	all the involved cells (for all nurses)
3.a	$10,000$	number of shifts more than the maximum value	${}^{1}/_{\lvert D\rvert}$	all cells
3.b	$10,000$	number of days off less than the minimum value in an isolated sequence of shifts	${}^{\lvert V\rvert}/_{1-N^{max}+\lvert V\rvert}$	all the cells on two sides of the isolated sequence of days off
3.c	$10,000$	number of minutes more/less than the maximum value	${}^{1}/_{\lvert D\rvert}$	all cells
3.d	W_w^{min}, W_w^{max}	number of shifts more than the maximum value	${}^{\lvert V\rvert}/_{7}$	all cells within a week
3.e	$10,000$	number of weekends more than the maximum value	${}^{\lvert V\rvert}/_{2\lvert W\rvert}$	all the involved cells
3.f max	$10,000$	number of shifts more than the maximum value in an isolated sequence of shifts	${}^{\lvert V\rvert}/_{H_a^{max}+\lvert V\rvert}$	all the involved cells
3.f min	$10,000$	number of shifts less than the minimum value in an isolated sequence of shifts	${}^{\lvert V\rvert}/_{H_a^{min}+\lvert V\rvert}$	all the cells on two sides of the isolated sequence of shifts
4	$10,000$	number of shifts less than the minimum value	${}^{\lvert V\rvert}/_{2}$	all the involved cells
5	$10,000$	a shift	${}^{1}/_{2}$	all the involved cells
6	$10,000$	a shift	1	the involved cell
7	B_{ew}^{cwx}	number of weekends more than the maximum value in an an isolated sequence of weekends	${}^{\lvert V\rvert}/_{C^{max}+\lvert V\rvert}$	all the cells of weekends on two sides of the isolated sequence of weekends
8	B_{ead}^{rso}	one shift (un-)assignment	1	the involved cell
9	$10,000$	number of forbidden shift rotations	${}^{1}/_{2\lvert V\rvert}$	all the involved cells

algorithm to feasible regions. Table 3.2 shows for each constraint the assigned weight (w_c), the relevant violation of the constraint, cell share, i.e. the relevant proportion of the constraint violation for all the affected cells (s_c), and the affected cells associated with the violation, respectively. The assigned weights are given by the user as input values per instance and often are different for each family of soft constraints and are set to 10000 for hard constraints. In this table, $|D|$, $|V|$, $|W|$ and $|E|$ denote the total number of days within the planning period, the total number of violations relevant to a particular constraint, the total number of weeks, and the total number of nurses, respectively. All the other parameters are already defined in Section 2.3.

To calculate the cell penalty (p_{cell}) for a particular cell, we need to multiply the amount of cell share (s_c) with the relevant weight (w_c) for each constraint ($c \in C$) for a particular day and nurse. Hence, the total penalty allocated to each cell can be calculated using the following equation:

$$p_{cell} = \sum_{c \in C} s_c w_c$$

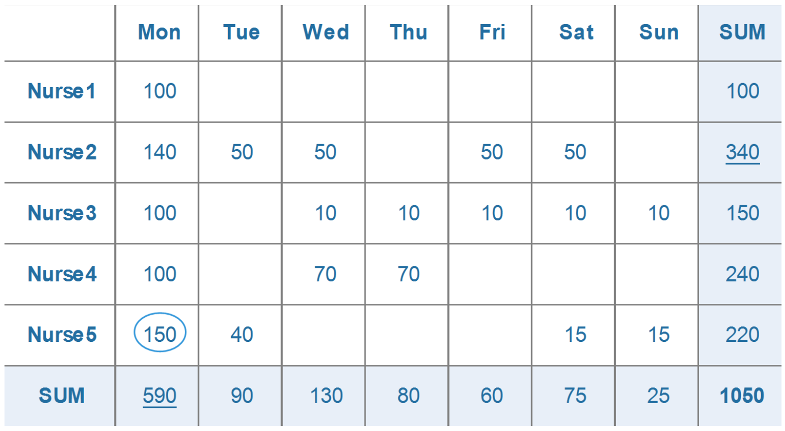
For example, for Constraint *9*, assuming that we have only one violated rotation for a specific nurse occurred on Tuesday and Wednesday in the first week of a roster, the calculated penalty for each of the two cells involved in the violated rotation is equal to 5000. It should be noted that for Constraint *2*, because the minimum and maximum number of shifts per day are given, we should sum up the total number of violations for all shifts ($a \in A$) over a day in order to calculate cell penalties over days.

As it can be seen, cell penalties are distributed evenly over affected cells. Although this calculation is not accurate in general, it is sufficient for our purpose. It is noteworthy to mention that the penalty evaluation is quite fast since it is done using a delta function (i.e., we only calculate the difference of total penalties between two solutions according to the violated constraints) and highly optimised data structures (e.g. in Java, *LinkedHashMap* is very efficient for accessing and iterating over elements of a collection in comparison. This hashmap is further improved by reimplementing custom hash algorithms which are available in Boost C++ libraries [8])

Calculating cell penalties in a schedule, we are also able to accumulate them over different dimensions such as nurses and days in the schedule. In fact, by accumulating the cell penalties, we can elicit more information, which gives us more insight into how to destroy and recreate rosters as we see later. That said, we use the following *aggregation settings* to configure the IP solver in order to fix different parts of the solution during the search process. In Section 3.3, we will test the hybrid algorithm by combining the following settings together within different configurations in order to identify the most efficient one.

1. **Nurses:** by accumulating cell penalties within the planning horizon for each nurse, we are able to identify the contribution of each nurse to the total penalty respecting to the current solution. Therefore, we can identify nurses who have the most contributed penalties among the others.
2. **Days:** in this setting, cell penalties are accumulated for all the nurses within each day. Therefore, similar to *Nurses* setting, we can identify the days having the most contributed penalties.
3. **Weeks:** analogous to the other settings, here cell penalties are accumulated for all nurses and all days within each week.
4. **Random:** in this setting, there is no accumulation indeed. Instead, cells are selected randomly according to their relevant cell penalty. For this purpose, a simple linear weighted random function is used, where cells with a higher penalty have more chance to be selected.

For illustration purposes, Figure 3.2 shows the first week of a roster, in which cell



	Mon	Tue	Wed	Thu	Fri	Sat	Sun	SUM
Nurse1	100							100
Nurse2	140	50	50		50	50		340
Nurse3	100		10	10	10	10	10	150
Nurse4	100		70	70				240
Nurse5	150	40				15	15	220
SUM	590	90	130	80	60	75	25	**1050**

Figure 3.2 – The associated cell penalties calculated for the first week of a roster

penalties are calculated for all the involved constraints. It can be seen that for some cells, there are no associated penalties (blank cells), which means they do not contribute to the total incurred penalty of the current solution. For the cell at the intersection of *Monday* and *Nurse5*, the calculated cell penalty is 150, which is the highest value among all the cells. Therefore, in the Random aggregation setting, this cell is very likely to be selected and unassigned afterwards. If we aggregate cell penalties for all the nurses and days as calculated in the last column and row of the weekly roster, it is realised that *Monday* and *Nurse2* (shown in underlined style) have the greatest contributions to the total penalty associated with the roster. Therefore, using the Nurses and Days aggregation settings, *Monday* and *Nurse2* are selected to be destroyed. Consequently, it can be seen that much useful information can be extracted from a solution after calculating the relevant cell penalties.

When the candidate cells required to be fixed (or to be destroyed) are identified, the IP solver solves the problem using the incumbent solution, where the integer variables associated with the fixed cells (x_{ead}) are set before starting the search process. Next, considering all constraints as soft except Constraint *1*, the IP solver produces another solution which can be different from the current solution in terms of quality and the underlying structure. In either case, the generated solution is passed to the next iteration of the VND algorithm as an initial solution. It is noteworthy to mention that because a significant number of variables in the IP model is fixed, particularly those which are involved in more constraints, the IP solver can easily solve the problem even when the scale of the problem instance is relatively large. Therefore, according to our experiments, in most cases, the IP solver can produce a solution even in a very short timeout condition.

Ultimately, after running the VND search algorithm and the IP ruin-and-recreate blocks in order to find a better solution, there is still a chance of getting stuck in a local optimum. In fact, due to not having any global picture throughout the search space, there can be another solution close to the current local optimum, but not detectable due to the complex structure of the problem. To resolve this issue, another IP solver improves the best-found solution at the end of the hybrid algorithm to solve the problem in the remaining time, and then the final solution is reported to the user. This IP solver employs the same IP model introduced in Section 2.3 including all the constraints, but it starts from the best-found solution thus far. To configure the IP solver to start from the current best solution rather than a randomly generated roster, the appropriate parameter (e.g. *MIPFocus* parameter in Gurobi) is set before starting the search process. It should be noted that the final IP solver is also useful to provide some insight to the optimality and quality of the current solution.

3.3 Computational Results

We tested the proposed hybrid algorithm on the same benchmark datasets and within the same computational environment as introduced in Chapter 2. We made a concerted effort to optimise the implemented code using the latest software technologies and code optimisation practices. For example, we used efficient hash algorithms and appropriate data structures which were available in Boost

C++ libraries [8], and generic programming to minimise performance overheads. Moreover, we run all our experiments on one CPU core to have more accurate comparison. After extensive testing of the algorithm using different settings, the following parameters were set. We dedicate 70% of the total allowed runtime to the VNS algorithm (VND and IP ruin-and-recreate components), and the rest to the final IP block. For VND stopping criteria, we set the maximum number of iterations to 50,000 and the maximum number of non-improvement iterations to 5. In our preliminary experiments, the algorithm mostly stopped due to the maximum number of non-improvement iterations. Setting a small value for this parameter allows the algorithm to try a higher number of neighbourhoods and configurations and to avoid getting stuck in local optima. We also employed the neighbourhoods 2-exchange, double-exchange, multi-exchange with the length of 3, block-exchange with the length of 4, and 3-exchange with the length of 3 in order. In our preliminary experiments, keeping the length of neighbourhoods on 3 or 4 was the best option which is also in line with the experiments reported in the literature [24].

Two experiments were carried out to test the proposed algorithm: first, we evaluated different aggregation settings through the ruin-and-recreate framework to understand how they impact the performance of the algorithm, and then we analysed the performance of different components of the hybrid algorithm. Second, we compared the hybrid algorithm with other state-of-the-art algorithms, and the Gurobi IP solver. For each experiment, the algorithm was run 10 times per instance and average values are reported accordingly.

In the first experiment, in order to examine the best combination of aggregation settings defined in Section 3.2 for the IP ruin-and-recreate component, we ran the algorithm with a variety of combinations. For each combination, we defined a *selection range*, i.e. the total percentage of available candidates in each aggregation setting which was selected in decreasing order to be destroyed and recreated. Ac-

Table 3.3 – Results of the hybrid algorithm by applying different configurations of aggregation settings

Instance	Config1		Config2		Config3	
	Obj.	Gap (%)	Obj.	Gap (%)	Obj.	Gap (%)
Instance08	1958	33.76	1695	23.48	1364	4.91
Instance09	439	7.52	439	7.52	439	7.52
Instance10	4631	0.00	4631	0.00	4631	0.00
Instance11	3443	0.00	3443	0.00	3443	0.00
Instance12	4045	0.12	4045	0.12	4042	0.05
Instance13	3109	56.71	3109	56.71	3109	56.71
Instance14	1361	6.17	1342	4.84	1281	0.31
Instance15	4463	14.72	4588	17.04	4144	8.16
Instance16	3384	4.73	3306	2.48	3306	2.48
Instance17	5956	3.86	6043	5.25	5760	0.59
Instance18	5158	15.65	5158	15.65	5049	13.82
Instance19	4365	32.53	4145	28.95	3974	25.89
Instance20	5451	12.99	5603	15.35	5242	9.52
Instance21	27,281	23.51	28,356	26.41	24,977	16.45
Instance22	176,652	86.38	173,371	86.12	130,107	81.50
Instance23	57,210	95.17	97,893	97.18	40,543	93.18
Instance24	3,173,810	97.74	3,160,760	97.73	2,829,680	97.46
Average		28.91		28.51		24.62

cording to our non-exhaustive preliminary tests on a variety of combinations using dataset-III, in the following, we present the best three identified *configurations* and the relevant selection ranges:

- *[Config1]*: Use only Nurses aggregation setting with the selection range at least 20%.
- *[Config2]*: Apply Nurses, Days, and Weeks aggregation settings in order, with the selection ranges at least 20%, 50%, and 60%, respectively.
- *[Config3]*: Apply Nurses, Days, Weeks, and Random aggregation settings in order, with the selection ranges [10%, 30%], [10%, 40%], [10%, 50%], and [30%, 50%], respectively. For each setting, the minimum value in the relevant range is increased by 10% (the increment rate) after each VNS iteration (e.g. [10%, 30%] is changed to [20%, 30%]).

Table 3.3 shows the results of the benchmarked configurations for instances 8 to 24 of dataset-III, where the algorithm was run for 10 minutes. We do not report the results of the first seven instances since the algorithm returns the same results for all the mentioned configurations. In this table, the objective function value and its difference to the best-known lower bound in percentage (denoted as Gap (%)) according to Curtois and Qu [26] are shown for each configuration and instance. One can see that running the algorithm with the third configuration generally results in better solutions in average. The reason for the superiority of the third configuration is due to the comprehensive investigation of the solution space by using different ruin-and-recreate strategies, and as a result, facilitating the hybrid algorithm to escape from a variety of local optima. In fact, in this configuration, we re-evaluate the current solution through four different dimensions (i.e. Nurses, Days, Weeks, and Random) if it gets stuck in a local optimum. Moreover, changing the selection ranges incrementally, equips the hybrid algorithm to behave adaptively during the search progress. It means, the more the hybrid algorithm advances within the search process, more parts of the solution are selected to be changed, i.e. the diversification rate is being increased. Nonetheless, by looking at the average results of the three configurations, it can be seen that the performance of the algorithm using Config1 and Config2 is only 4% lower than the best configuration. This means that the hybrid algorithm is not too sensitive to the particular setting of parameters which is beneficial in practical circumstances.

Applying the third configuration (i.e. Config3), we ran the algorithm using dataset-II and dataset-III for 10 minutes computational time. The detailed results of these tests are shown in Table 3.4 and 3.5, where the initial solution generated by the greedy heuristic algorithm, the improved solution by the VNS, and the final solution further improved by the IP solver are reported, respectively. In these tables, $\Delta_{iv}\%$, and $\Delta_{vo}\%$ denote the percentage of improvement achieved using the VNS component and the final IP block, respectively. Furthermore, the number of

Table 3.4 – Detailed results of the hybrid algorithm using dataset-III for 10 minutes

Instance	Initial	$\Delta_{iv}\%$	VNS	$\Delta_{vo}\%$	Obj.	Cycle	$\frac{\Delta}{C}\%$
Instance01	11,525	94.73	607	0.00	607	18	5.26
Instance02	14,827	94.42	828	0.00	828	30	3.15
Instance03	22,480	95.55	1001	0.00	1001	16	5.97
Instance04	20,775	91.74	1716	0.00	1716	56	1.64
Instance05	31,947	96.41	1147	0.35	1143	48	2.01
Instance06	30,944	93.40	2041	4.46	1950	52	1.80
Instance07	42,625	97.49	1070	1.31	1056	52	1.88
Instance08	63,153	95.98	2538	46.26	1364	81	1.21
Instance09	55,320	99.21	439	0.00	439	42	2.36
Instance10	102,073	94.97	5133	9.78	4631	19	5.02
Instance11	120,287	97.13	3450	0.20	3443	25	3.89
Instance12	146,970	96.05	5801	30.32	4042	77	1.26
Instance13	289,121	98.88	3231	3.78	3109	49	2.02
Instance14	117,166	98.19	2116	39.46	1281	79	1.25
Instance15	144,631	96.37	5245	20.99	4144	60	1.62
Instance16	105,714	95.40	4861	31.99	3306	81	1.20
Instance17	174,308	95.99	6986	17.55	5760	67	1.44
Instance18	181,068	96.79	5815	13.17	5049	50	1.94
Instance19	322,730	98.59	4564	12.93	3974	47	2.10
Instance20	910,083	99.42	5242	0.00	5242	36	2.76
Instance21	197,130,000	99.99	26,989	0.04	26,977	44	2.27
Instance22	168,433,000	99.92	130,107	0.00	130,107	42	2.38
Instance23	15,542,000	99.74	40,543	0.00	40,543	45	2.22
Instance24	201,119,000	98.55	2,925,411	3.27	2,829,680	17	5.80
Average		96.87		9.83			2.60

Table 3.5 – Detailed results of the hybrid algorithm using dataset-II for 10 minutes

Instance	Initial	$\Delta_{iv}\%$	VNS	$\Delta_{vo}\%$	LB	Obj.	Cycle	$\frac{\Delta}{C}\%$
NRP01	70,863	89.12	7711	0.27	570	7690	50	1.78
NRP02	76,169	87.02	9886	0.86	200	9801	50	1.74
NRP03	127,870	95.01	6380	0.63	122	6340	41	2.32
NRP04	87,018	95.35	4050	0.74	1300	4020	53	1.80
NRP05	89,815	89.35	9561	1.16	211	9450	53	1.69
NRP06	67,417	96.44	2401	0.87	130	2380	54	1.79
NRP07	91,082	96.23	3431	1.49	140	3380	58	1.66
NRP08	70,700	98.07	1368	81.29	133	256	33	3.02
NRP09	82,975	95.63	3625	1.79	1130	3560	63	1.52
NRP10	111,571	92.08	8840	0.23	310	8820	51	1.81
NRP11	88,373	96.04	3501	1.46	110	3450	57	1.69
Average		94.05		14.16				1.89

cycles and the average improvement obtained throughout each cycle (denoted as $\frac{\Delta}{C}\%$) are reported in the last two columns of Table 3.4 and 3.5.

As it can be seen in Table 3.4, the VNS algorithm is able to improve the generated initial solution via the greedy heuristic by 97%, which is then further optimised by the final IP block by 10%. Moreover, it is observed that the final IP solver is not able to improve the generated solution for a number of instances. For example, for *Instance02*, the IP solver is only employed to prove the optimality of the obtained solution by the VNS algorithm. However, for some instances such as *Instance23*, the IP solver does not manage to produce any better solutions due to the limited computational time. Similarly, the results in Table 3.5 show that the VNS algorithm is able to improve the initial solution by 94%, which is further enhanced by the final IP block by 14%. It should be noted that for generating the initial solution for dataset-II, a similar IP solver was run for 20 seconds, since employing a greedy heuristic similar to the one explained in Section 3.2.1 often resulted in infeasible solutions. Nevertheless, the role of the final IP solver as the last component of the hybrid algorithm in order to improve the output of the VNS algorithm is crucial, where the attained improvement can even reach more than 30% for some instances.

The number of cycles performed on each instance shows an estimation of the structure of each instance where instances with more complex structure often require more cycles. Observing the average improvement obtained through cycles, it is realised that how changing aggregations of settings along with neighbourhoods allows the hybrid algorithm to escape from local optima and to further improve obtained solutions.

In the second experiment, we benchmarked the efficiency of the proposed algorithm against current state-of-the-art algorithms for all the benchmark datasets. For dataset-III, we compared it with the results published by Curtois and Qu [26], who reported the results of two algorithms from Burke and Curtois [13], i.e. a branch-and-price and an ejection chain heuristic. For dataset-I, we compared it with the results of a hybrid VNS [18], denoted as VNS-1, a Memetic Algorithm [12], denoted as MA, a Variable Depth Search [24], denoted as VDS, a Harmony Search Algorithm [36], denoted as HSA, a Scatter Search [20], denoted as SS, another variant of hybrid VNS [55], denoted as VNS-2, a hybrid branch-and-price algorithm [13], denoted as BAP, and a stochastic VNS [78], denoted as SVN.

All our experiments are given 10 minutes computational time, since the hybrid algorithm is particularly designed to perform well in short computational times, and also it is common to use short times, as seen in the relevant literature and the nurse rostering competition [39]. Experimentally speaking, we have also seen that in practice a significant number of rosters should be generated to achieve the desired result, and therefore, the generation of rosters should be done in relatively short runtimes. However, to have a comprehensive comparison with the available results and the benchmark algorithms, we also run the proposed algorithm for a longer time, i.e. 60 minutes.

Table 3.6 presents the best results of the ejection chain method, Gurobi IP solver with default settings, and our hybrid algorithm using the third configuration running for the limited computational times 10 and 60 minutes. The results of the branch-and-price (B&P) algorithm without any time limits are also presented. In this table, "-" indicates that the algorithm does not generate any feasible solutions within the allocated time limit.

As we can see in Table 3.6, within the 10 minutes computational time, from the total of 24 instances, the hybrid algorithm outperforms the ejection chain method for 23 instances, and produces the same results for *Instance01*. In comparison with the Gurobi IP solver, the algorithm performs better for 14 instances and generates the same results for the remaining 10 instances, where 9 of which are optimal solutions.

Overall, the proposed algorithm outperforms the ejection chain method and

Table 3.6 – Benchmark results of the hybrid algorithm in comparison with a branch-and-price and ejection chain heuristic [13], and Gurobi IP solver [35] running for 10 and 60 minutes

Instance	Hybrid Algorithm	Ejection Chain	Gurobi	Hybrid Algorithm	Ejection Chain	Gurobi	B&P
	10 min			60 min			
Instance01	607	607	607	607	607	607	607
Instance02	828	923	828	828	837	828	828
Instance03	1001	1003	1001	1001	1003	1001	1001
Instance04	1716	1719	1716	1716	1718	1716	1716
Instance05	1143	1439	1143	1143	1358	1143	1160
Instance06	1950	2344	1950	1950	2258	1950	1952
Instance07	1056	1284	1056	1056	1269	1056	1058
Instance08	1364	2529	8995	1344	2260	1323	1308
Instance09	439	474	439	439	463	439	439
Instance10	4631	4999	4631	4631	4797	4631	4631
Instance11	3443	3967	3443	3443	3661	3443	3443
Instance12	4042	5611	4045	4040	5211	4040	4046
Instance13	3109	8707	500,410	1905	3037	3109	-
Instance14	1281	2542	1482	1279	1847	1280	-
Instance15	4144	6049	78,144	3928	5935	4964	-
Instance16	3306	4343	3521	3225	4048	3233	3323
Instance17	5760	7835	6149	5750	7835	5851	-
Instance18	5049	6404	7950	4662	6404	4760	-
Instance19	3974	6522	29,968	3224	5531	5420	-
Instance20	5242	23,531	-	4913	9750	-	-
Instance21	26,977	38,294	-	23,191	36,688	-	-
Instance22	130,107	-	-	32,126	516,686	-	-
Instance23	40,543	-	-	3794	54,384	-	-
Instance24	2,829,680	-	-	2,281,440	156,858	-	-

Gurobi IP solver within 10 minutes computational time for 14 instances, and produces the same or better results for the rest of the instances. It should be noted that the ejection chain method and Gurobi IP solver could not solve the last 3 and 5 instances, respectively. That said, obtaining the reported solutions for these instances, which are hard to solve and huge in size, makes the hybrid algorithm an appropriate candidate to tackle such instances even in a short runtime.

Running our benchmarks for 60 minutes, the hybrid algorithm outperforms the ejection chain method for 22 instances and does not generate a better result only for *Instance24*. The reason for obtaining a poor-quality solution for *Instance24* (and in general the last four instances) could be the inherent nature of the hybrid algorithm as a matheuristic (based on IP and CP) which is not efficient in solving rather large-size instances. Since this instance is huge in size, it is a challenge for the algorithm to solve it in comparison with a meta-heuristic approach like the ejection chain method. Another reason is the limitation of computer memory and hardware. The IP component of the hybrid algorithm needs a massive amount of memory to solve the relevant LP problems during the search process. Moreover, it could have a particular structure which cannot be exploited using the current setting of the proposed algorithm, but easy to be identified by the ejection chain method.

Similarly, in comparison with Gurobi IP solver, the hybrid algorithm is able to generate better results for 13 instances and obtains the same results for the remaining 10 instances. For *Instance08*, the IP solver outperforms the hybrid algorithm for only a slight difference. Overall, the hybrid algorithm attains better solutions for half of the instances, which makes it an acceptable candidate even for longer computational times.

Comparing with B&P, the hybrid algorithm is outperformed only for *Instance08*, where B&P takes more than 197 minutes to generate the solution. Apart from the 11 instances, for which B&P cannot produce any results, the hybrid algorithm generates better solutions for 5 instances and achieves the same results for the remaining instances.

Table 3.7 shows similar results using dataset-I for 10 minutes computational time, where the hybrid algorithm generates the best-known solutions [13] for all instances except ORTEC01.

We also tried to compare the hybrid algorithm with a similar approach reported by Burke et al. [21], in which IP and VNS are hybridised in a pipeline fashion, i.e. running sequentially. Burke et al. [21] also developed a decomposition technique for handling constraints, and evaluated their hybrid VNS using the studied decomposition. Unfortunately, after making inquiries from the relevant

Table 3.7 – Benchmark results of the hybrid algorithm against VNS-1 [18], MA [12], VDS [24], HSA [36], SS [20], VNS-2 [55], BAP [13], and SVN [78] for dataset-I

Instance	BKS	HA	VNS-1	MA	VDS	HSA	SS	VNS-2	BAP	SVN
GPOST	**5**	**5**	-	915	-	-	9	8	**5**	-
GPOSTB	**3**	**3**	-	789	-	-	5	-	**3**	-
ORTEC01	**270**	290	541	535	360	310	365	-	**270**	-
ORTEC02	**270**	**270**	-	-	-	330	-	-	**270**	-
Valouxis-1	**20**	**20**	-	560	-	-	100	160	80	73
SINTEF	**0**	**0**	-	8	-	-	4	-	**0**	-
MILLAR-1	**0**	**0**	-	100	-	-	**0**	**0**	**0**	-
WHPP	**5**	**5**	-		-	-	-	-	**5**	**5**
LLR	**301**	**301**	-	305	-	-	**301**	314	**301**	**301**

authors, it is found out that the benchmarked dataset including 12 instances is lost except one of them, i.e. ORTEC01 (see dataset-I). In effect, being unsuccessful in obtaining the benchmarked instances and willing to further evaluate the efficiency of the hybrid algorithm are the reasons for creating dataset-II, where we attempted to make it similar to the sole existing instance ORTEC01. This instance is also the hardest instance in dataset-I. In Appendix A, we describe how this dataset is generated.

Table 3.8 shows results of the hybrid algorithm and IP solver running for 10 and 60 minutes, and the hybrid VNS (shown as IPVNS) running for 60 minutes computational time. Better solutions for each computational runtime are marked bold.

To have more accurate comparison, we simulate the computational environment of the IPVNS algorithm (Pentium 2.0 GHz PC) by running the hybrid algorithm on a different PC with an Intel Core-i7 1.6 GHz CPU but only using one core of the CPU. That said, the first reported value for instance *ORTEC01* is the one similar to the other instances by running on our regular benchmark PC, and the second one is relevant to the less-powerful PC used only for comparing with IPVNS algorithm. As we can see in Table 3.8, compared with the results obtained by the IP solver, the hybrid algorithm finds better solutions for all the instances.

Table 3.8 – Benchmark results of the hybrid algorithm in comparison with Gurobi IP solver [35] and the hybrid VNS algorithm (IPVNS) reported by Burke et al. [21] for dataset-II

Instance	Hybrid Algorithm		Gurobi		IPVNS
	10 min	60 min	10 min	60 min	60 min
ORTEC01	$\underline{\textit{270}}$*,* ***315***	***270***	*1410*	*405*	*460*
NRP01	**7690**	**7620**	15,500	11,162	-
NRP02	**9801**	**9638**	31,741	12,850	-
NRP03	**6340**	**5230**	15,510	8553	-
NRP04	**4020**	**3700**	25,495	13,385	-
NRP05	**9450**	**9400**	14,855	14,855	-
NRP06	**2380**	**2320**	2911	2521	-
NRP07	**3380**	**3220**	5660	5301	-
NRP08	**256**	**230**	385	241	-
NRP09	**3560**	**3360**	14,940	5880	-
NRP10	**8820**	**8530**	22,863	22,551	-
NRP11	**3450**	**3290**	37,698	5828	-

In particular, for instance *ORTEC01*, when we compare the results with IPVNS, the algorithm reaches the objective value of 315, which is 31% better than the one obtained by IPVNS, i.e. 460 on a similar computational environment. Running the algorithm on our regular benchmark PC, the objective value is slightly improved and reaches the value of 270 known as the optimal solution, which could be due to using a more powerful PC.

Considering instance *ORTEC01*, we also benchmark the algorithm against the winner of personnel scheduling track of CHeSC hyper-heuristic competition [43, 41], who developed a VNS-based hyper-heuristic (VNS-TW) consisting of two steps, i.e. shaking and local search, which is able to dynamically adjust to various problems using different techniques. Running the hybrid algorithm within the standardised time limit using the benchmark tool provided by the competition organisers, it obtains the objective value 270 in comparison with the result 320 obtained by VNS-TW.

Comparing our hybrid algorithm and Gurobi IP solver for 60 minutes, the hybrid algorithm obtains better results. It is worth noting that the results generated by the hybrid algorithm for 10 minutes are superior to the solutions produced by the Gurobi IP solver for 60 minutes except for instance *NRP08*, where there is only a slight difference.

In summary, it is observed that the algorithm performs well for short computational time and is able to produce acceptable solutions in long computational runtimes. In benchmarks, it is also seen that the hybrid algorithm is able to compete with state-of-the-art algorithms. However, since the core component of the algorithm is IP, it has still the limitations of IP solvers and is not able to handle relatively large-size instances. For example, we expect to see a high consumption of computer memory and a considerable decline in the performance of the algorithm when large-size instances (e.g. more than 50 nurses and/or more than 6 months scheduling duration) are dealt with. That said, the hybrid algorithm is suitable when an IP solver is available and better performance is expected to achieve without spending significant efforts and investments. Due to the nature of the algorithm in using mathematical models and rather general neighbourhood structures, it is also expected to perform well on other sets of instances with similar sizes perhaps with some parameter tunings.

3.4 Extended Computational Results

In this section, we go beyond the scope of the NRP to investigate the extendability and scalability of the proposed solution evaluation mechanism, on which the hybrid algorithm is designed. That said, we benchmark the hybrid algorithm based on a similar rostering problem called University Timetabling Problem (UTP) [22]. The UTP is defined as the process of scheduling some courses within a given number of periods and rooms, considering some requirements such as university regulations, and some preferences such as maximum utilisation of rooms. For our benchmarks, we use the problem description and IP formulation explained in Appendix B.

Table 3.9 – The characteristics of the benchmark instances for the UTP

Instance	Periods	Days	Courses	Curricula	Rooms
Comp1	30	5	30	14	6
Comp2	25	5	82	70	16
Comp3	25	5	72	68	16
Comp4	25	5	79	57	18
Comp5	36	6	54	139	9
Comp6	25	5	108	70	18
Comp7	25	5	131	77	20
Comp8	25	5	86	61	18
Comp9	25	5	76	75	18
Comp10	25	5	115	67	18
Comp11	45	5	30	13	5
Comp12	36	6	88	150	11
Comp13	25	5	82	66	19
Comp14	25	5	85	60	17

We tested the algorithm on 14 instances introduced in the third track of ITC-2007 competition [29], whose characteristics are reported in Table 3.9. For benchmarking purposes, we use the same environment as well as parameter settings for the NRP mentioned in Section 3.3, unless otherwise stated. Setting up the algorithm based on the UTP, courses, period, and rooms resemble nurses, days, and shifts, respectively. That said, same neighbourhood structures and aggregation settings (*Courses*, *Periods*, *Days*, and *Random)* are applied. We also use Config3 as the best-identified configuration and employ a simple greedy heuristic similar to the algorithm reported by Lü and Hao [52] for generating initial solutions.

Table 3.10 shows the required information for calculating cell penalties according to the IP formulation presented in Appendix B, which are calculated in a similar way to the NRP. In this table, $|AC|$ denotes the total number of affected cells relevant to a violation. For example, for Constraint *1*, the number of unassigned lectures for a course, the total number of violations (unassigned lectures) $|V|$ are divided to all periods $|P|$. That said, each cell contributes equally to the total number of violations as much as the value of cell share, ${}^{|V|}/_{|P|}$.

Table 3.10 – The required information to calculate cell penalties for the UTP

Ct.	Weight (w_c)	Violation	Cell share (s_c)	Affected cells
1	$10,000$	number of unassigned lectures	$^{\|V\|}/_{\|P\|}$	all cells
2	$10,000$	number of extra lectures in the same period and room	$^{\|V\|}/_{\|AC\|}$	all the involved cells (assigned to the same room and period)
3	$10,000$	a pair of conflicted lectures	$^1/_2$	all the involved cells
4	$10,000$	one lecture assigned at a forbidden period	1	the involved cell
5	w_{cr}^{cap}	number of students exceeded the assigned room capacity	$^{\|V\|}/_{\|AC\|}$	all the involved cells (assigned to the same room)
6	w_c^{spr}	number of days less than the minimum value	$^{\|V\|}/_{(d_c^{min}-\|V\|)\|P\|}$	all the involved cells
7	w_{udp}^{com}	number of isolated lectures in a day	$^1/_3$	the involved cell and the immediate next cells on two sides of the isolated lecture
8	w_c^{stb}	number of extra rooms for a course	$^{\|V\|}/_{\|AC\|}$	all the involved cells

Table 3.11 – Benchmark results of the algorithm (HA) in comparison with the results of Lach and Lübbecke [49] (BF), Burke et al. [22] (MA), and Müller [59] (WN)

Instance	1U			10U			WN	
	HA	MA	BF	HA	MA	BF	Avg. (IU)	Best (10U)
Comp1	**5**	168	12	**5**	10	12	**5**	**5**
Comp2	105	114	239	68	101	93	61	**51**
Comp3	126	158	194	**74**	144	86	95	84
Comp4	53	153	44	**37**	36	41	43	**37**
Comp5	764	1447	965	**308**	649	468	344	330
Comp6	161	277	395	132	317	79	57	**48**
Comp7	161	-	525	62	857	28	34	**20**
Comp8	80	173	78	44	53	48	47	**41**
Comp9	137	112	115	110	115	**106**	113	109
Comp10	60	70	235	58	49	44	21	**16**
Comp11	**0**	288	7	**0**	12	7	**0**	**0**
Comp12	649	-	1122	626	889	657	352	**333**
Comp13	98	556	98	78	92	67	74	**66**
Comp14	100	123	113	65	72	**54**	62	59

To benchmark the efficiency of the proposed algorithm against current state-of-the-art algorithms, we compared it with the results of a strong IP formulation composed of two stages (BF) [49], a matheuristic using a decomposed formulation (*MA*) [22], and the winner of the third track of ITC-2007 competition (*WN*) [59]. We ran the algorithm for 1 and 10 CPU unit(s). Using the benchmark tool provided by the competition organisers to normalise time measurement, one CPU unit corresponds to 210 seconds on our benchmark PC. The results are reported in Table 3.11, where *"-"* indicates that the algorithm does not generate any feasible solutions within the allocated time limit. The results shown in bold indicate best obtained solutions.

As we can see in Table 3.11, from the total of 14 instances, HA is able to generate better results for 11 and 7 instances within 1 and 10 CPU unit(s), respectively. Comparing with the results of MA, HA produces better solutions for all the benchmark instances except instances *Comp4, Comp9,* and *Comp10* within 1 and 10 CPU unit(s). Moreover, although a strengthened formulation is employed in BF, it outperforms HA only for 3 and 6 instances within 1 and 10 CPU unit(s), respectively. It is also worth mentioning that for those instances that the proposed algorithm does not generate better solutions, the difference in the quality of the obtained solutions is not significant. In comparison with WN, whether according to the average or best results within 10 and 1 CPU unit(s), the obtained solutions of HA are inferior, though it is capable of generating optimality information.

Overall, in terms of performance, it can be observed that using the same parameters as we set for the NRP, the hybrid algorithm is able to obtain competitive results for the UTP using the new solution evaluation mechanism when it is compared against similar IP-based approaches. However, in comparison with meta-heuristic methods such as WN, the performance of the hybrid algorithm was not promising which again confirms the limitations of exact methods such as IP. In terms of implementation, we applied identical values for all parameters except the ones associated with penalty calculation which have to be set according to individual constraints of a problem. That said, no significant tuning is required to apply the algorithm to a new problem. Therefore, there is potential to extend the applicability of the proposed solution evaluation mechanism on which the hybrid algorithm is designed to similar optimisation problems which have similar characteristics as the NRP.

3.5 Concluding Remarks

We have presented a hybrid algorithm employing VNS and IP which is designed based on a new solution evaluation mechanism. At the first step, after generating an initial solution using a greedy heuristic, the solution is improved using a VND algorithm. To increase the exploitation and exploration in the VNS, IP within a ruin-and-recreate framework is employed, where parts of the solution are kept fixed by applying a new scoring scheme. In order to ensure the investigation of the whole search space, IP is applied again to improve the obtained solution in the remaining time.

The proposed scoring scheme is able to break down the total penalty associated with a solution into fundamental elements of the problem, and to guide the search process adaptively towards high-potential parts of the solution. Moreover, incorporating an IP approach into a meta-heuristic algorithm confirms the applicability of exact methods for practical instances in a hybrid setting. We evaluated the proposed algorithm with three different datasets consisting of a variety of instances. The benchmark results showed competitive performance in comparison with state-of-the-art algorithms and a standard IP solver. The algorithmic concepts of the proposed algorithm particularly the embedded solution evaluation mechanism are general enough to be applied to other similar rostering problems such as the UTP as it has been observed in the extended computational results. For further information regarding generalisation of the proposed algorithm, we refer the interested readers to our technical report [68]. In summary, it is observed that using the new solution evaluation mechanism, thereby exploiting problem-specific information during the search process directs the algorithm to the most promising areas of the search space and results in generating solutions which are competitive with those obtained by state-of-the-art algorithms.

3.6 Contributions

This chapter proposes a hybrid algorithm in which a new solution evaluation mechanism is applied. The algorithm is a hybridisation of IP and VNS, where IP is employed to intensify and diversify the search process. IP is integrated within a ruin-and-recreate framework and is employed as a neighbourhood structure throughout VNS cycles. In the ruin-and-recreate framework, elements of the solution having a high penalty are destroyed and recreated using IP aiming to obtain a better-quality and different-structured solution. Identifying high-penalty parts of the solution is done using a novel scoring scheme, in which the total penalty is broken down into fundamental elements of the solution. The proposed algorithm is benchmarked against state-of-the-art algorithms and a standard IP solver using three different datasets. The tests show promising results for most instances in short and long computational times. Furthermore, computational results are extended by using a different optimisation problem to show the applicability and extendability of the algorithm.

The proposed algorithm is able to traverse the infeasible space through the ruin-and-recreate framework by allowing all constraints to be violated. This characteristic allows the search process to find latent feasible solutions and escape from complicated local optima.

The new scoring scheme is applied to bring violations of soft constraints into play, and to empower the hybrid algorithm to assess the quality of solutions through different perspectives and dimensions. That said, a solution which is not good-quality according to a dimension, could have better-quality based on other dimensions.

Chapter 4

Conclusions and Future Research

In this chapter, we mainly address the contributions of this dissertation, and discuss some future research directions.

4.1 Conclusion

Over the last few decades, a wide variety of approaches and solution methods have been developed to solve the NRP. This dissertation aimed for contributing to the theory and practice of the NRP in two ways. First, to extend the reach of exact methods such as IP and CP, a hybridisation of IP and CP is proposed whose computational strengths are exploited to solve practical problem instances. That said, one can apply such methods to a wider set of problem instances while preserving their computational benefits such as accessing optimality information. Second, a new solution evaluation mechanism is introduced to assess the quality of obtained rosters within the search process which is able to provide useful information by breaking down the solution penalty. That said, a hybrid algorithm was designed to accommodate the proposed evaluation method enclosed within a ruin-and-recreate framework.

In Chapter 2, we proposed a hybrid algorithm to extend the computational reach of IP and CP, where the strength of CP in generating feasible solutions is utilised to aid the IP solver to attain better results. To achieve better performance, some algorithmic components were provided to extract valuable problem-specific information such as the computational difficulty of instances and constraints. The hybrid algorithm was able to provide optimality information by decomposing the problem into various sub-problems. Competitive results were obtained for a diverse range of instances classified in three different datasets, indicating the effectiveness of the algorithm for practical use.

In summary, it was observed that the reach of IP and CP approaches can be extended through a hybridisation scheme which makes them competitive enough in comparison with other state-of-the-art algorithms. However, computational limitations of such exact methods, and therefore, the proposed hybrid algorithm are still in place.

In Chapter 3, two common and successful solution approaches existing in the relevant literature, i.e. IP and VNS algorithms were combined based on a new solution evaluation mechanism. A ruin-and-recreate framework using IP was employed in the VNS algorithm to increase the exploitation and exploration throughout the search process, where parts of the solution are kept fixed by applying a new scoring scheme. The scoring scheme allowed us to identify the high-penalty parts of the solution from a variety of perspectives and dimensions within VNS cycles, thereby directing the search process adaptively towards most promising parts of the solution. The hybrid algorithm was evaluated using three different datasets and the benchmark results showed promising performance in comparison with state-of-the-art algorithms and a standard IP solver. The computational results were also extended to study the applicability of the proposed evaluation mechanism to a similar optimisation problem, where the hybrid algorithm showed acceptable performance.

In summary, it was observed that using a new solution evaluation mechanism embedded within a hybrid algorithm, thereby exploiting problem-specific information during the search process empowers the proposed hybrid algorithm to obtain solutions which are competitive with those obtained by state-of-the-art algorithms.

Overall, the results of extensive computational tests confirmed the validity of research contributions explained in this dissertation. In particular, it was shown that applying exact methods such as IP and CP in a hybrid setting is still beneficial to deal with practical problem instances, though it has still its computational limitations. The significant improvement in computational power of off-the-shelf IP and CP solvers during recent years (e.g. see the change history in [35]) and their increasing applications in bespoke business solutions (e.g. SAP [54]) further highlights the importance of this class of methods to solve real-world problem instances. It was also shown that extracting problem-specific information during the search process using an efficient solution evaluation method is beneficial to the overall performance of the hybrid algorithm.

4.2 Future Research

The present dissertation has made several new contributions to personnel rostering. Nevertheless, a large variety of topics remains open for future research. Furthermore, based on the contributions, several new research questions have become relevant. In what follows, possible further extensions are given.

In Chapters 2 and 3, we formulated the NRP in IP and CP considering that nurses have only one skill. Although our practical experience indicates that this assumption is not too restrictive (as we explained in Section 2.2), there could be circumstances that the problem is required to be modelled as a multi-skill problem. To deal with this new assumption, the algorithms presented in this dissertation need to be manipulated. This manipulation is mostly done to the problem formulation by defining new variables, but some further tweaks might be required to increase the performance of hybrid algorithms such as defining new neighbourhood structures considering skills.

The IP formulations presented in Chapter 2 is one of the common formulations of the NRP. This formulation suited the proposed algorithm very well due to its inherent transparency in modelling of constraints and better capturing of problem information during the search process. However, by using this formulation, the size of the resulted integer program exponentially increases when the scale of the problem becomes larger. Another alternative is to use pattern-based formulation [6, 40], in which modelling of complex constraints and incorporating changes into the model are easier, though it is very difficult to capture the interaction between constraints. To exploit such benefits, using pattern-based formulation within the proposed hybrid algorithm, which is often solved by applying column generation technique is open for future studies. To tackle the opaque interaction between constraints in a column generation formulation, using two models in parallel during the search process would be helpful. That said, the information throughout the search process can be shared and exchanged between two models.

Due to the rather general definition of the hybrid algorithm presented in Chapter 2 and using IP and CP models therein, there are possibilities to investigate the extension of this algorithm to other similar problems in terms of the underlying structure such as the course timetabling problem (as we have seen similarly in Section 3.4), thereby exploiting practical benefits in more general settings.

For the hybrid algorithm presented in Chapter 3, the VNS algorithm is chosen due to the flexible, simple, and rather general nature of this meta-heuristic and its suitability for incorporating complex neighbourhoods. Nonetheless, other heuristic algorithms such as population-based meta-heuristics which are popular according to the relevant literature [58, 20, 1] can be accommodated. Using population-based meta-heuristics, applying parallel computing becomes relevant to distribute the computational burdens over several computational units, or to perform different stages of the search process simultaneously. For example, different configuration settings could be used in parallel. Another interesting research direction is to investigate more sophisticated neighbourhood structures in order to improve the efficiency of the VNS algorithm and exploitation of the problem-specific information.

For easier configuration of the hybrid algorithm presented in Chapter 3, it would be interesting to employ a parameter tuning tool (e.g. see Hutter et al. [42]) to precisely select best configurations for the proposed ruin-and-recreate strategy, though it makes the implementation of the algorithm more difficult.

Although there were attempts to implement the algorithms presented in this dissertation in a real hospital environment, there has been no real case study based on the theories studied in this dissertation. Therefore, future case studies might provide more insights into the applicability, usability, and performance of the presented models and algorithms in this dissertation.

References

[1] U. Aickelin and K. A. Dowsland. An indirect genetic algorithm for a nurse-scheduling problem. *Computers and Operations Research*, 31(5):761–778, 2004.

[2] H. Allaoua, S. Borne, L. Letocart, and R. Wolfler Calvo. A matheuristic approach for solving a home health care problem. *Electronic Notes in Discrete Mathematics*, 41:471–478, 2013.

[3] K. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.

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

[5] R. Bai, E. K. Burke, G. Kendall, J. Li, and B. McCollum. A hybrid evolutionary approach to the nurse rostering problem. *IEEE Transactions on Evolutionary Computation*, 14(4):580–590, 2010.

[6] J. F. Bard and H. W. Purnomo. A column generation-based approach to solve the preference scheduling problem for nurses with downgrading. *Socio-Economic Planning Sciences*, 39(3):193–213, 2005.

[7] N. Beldiceanu, M. Carlsson, and J. Rampon. Global constraint catalog. Technical report, SICS, 2005. URL http://eprints.sics.se/2366.

[8] Boost.org. Boost C++ Libraries, 2015. URL http://www.boost.org/.

[9] P. Brucker, E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A shift sequence based approach for nurse scheduling and a new benchmark dataset. *Journal of Heuristics*, 16(4):559–573, 2010.

[10] P. Brucker, R. Qu, and E. Burke. Personnel scheduling: Models and complexity. *European Journal of Operational Research*, 210(3):467–473, 2011.

[11] J. Bunton, A. Ernst, and M. Krishnamoorthy. An integer programming based ant colony optimisation method for nurse rostering. In *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017*, pages 407–414, 2017.

[12] E. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence*, 15(3):199–214, 2001.

[13] E. K. Burke and T. Curtois. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, 237(1):71–81, 2014.

[14] E. K. Burke and G. Kendall. *Search methodologies: Introductory tutorials in optimization and decision support techniques.* Springer, 2005.

[15] E. K. Burke and S. Petrovic. Recent research directions in automated

timetabling. In *European Journal of Operational Research*, volume 140, pages 266–280, 2002.

[16] E. K. Burke, G. Kendall, and E. Soubeiga. A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[17] E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–449, 2004.

[18] E. K. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2):330–341, 2008.

[19] E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. Problem model for nurse rostering benchmark instances. Technical report, ASAP, School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham, UK, 2008. URL http://www.cs.nott.ac.uk/{~}tec.

[20] E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A scatter search methodology for the nurse rostering problem. *Journal of the Operational Research Society*, 61(11):1667–1679, 2010.

[21] E. K. Burke, J. Li, and R. Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484–493, 2010.

[22] E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová. Decomposition, reformulation, and diving in university course timetabling. *Computers and Operations Research*, 37(3):582–597, 2010.

[23] E. K. Burke, J. Li, and R. Qu. A Pareto-based search methodology for multi-objective nurse scheduling. *Annals of Operations Research*, 196(1): 91–109, 2012.

[24] E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A time predefined variable depth search for nurse rostering. *INFORMS Journal on Computing*, 25(3):411–419, 2013.

[25] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: a robust optimisation method applied to nurse scheduling. In *Lecture Notes in Computer Science. Parallel Problem Solving from Nature (PPSN02)*, volume 2439, pages 851–860. Springer-Verlag, 2002.

[26] T. Curtois and R. Qu. Computational results on new staff scheduling benchmark instances. Technical Report 06-Oct-2014, ASAP Research Group, School of Computer Science, University of Nottingham, 2014.

[27] H. De Beukelaer, G. F. Davenport, G. De Meyer, and V. Fack. JAMES: a modern object-oriented Java framework for discrete optimization using local search metaheuristics. In *4th International symposium and 26th National conference on Operational Research*, pages 134–138. Hellenic Operational Research Society, 2015. URL www.jamesframework.org/index.html.

[28] F. Della Croce and F. Salassa. A variable neighborhood search based matheuristic for nurse rostering problems. *Annals of Operations Research*, 218(1): 185–199, 2014.

[29] L. Di Gaspero, B. McCollum, and A. Schaerf. The Second International

Timetabling Competition (ITC-2007): Curriculum-based Course Timetabling (Track 3). Technical report, University of Udine, 2007.

[30] A. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, 2004.

[31] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. *Lecture Notes in Computer Science*, 2470:462–476, 2002.

[32] C. A. Glass and R. A. Knight. The nurse rostering problem: A critical appraisal of the problem structure. *European Journal of Operational Research*, 202(2):379–389, 2010.

[33] F. Glover and B. Melian. *Tabu Search*, volume 7. Springer, 2003.

[34] R. A. Gomes, T. A. Toffolo, and H. G. Santos. Variable neighborhood search accelerated column generation for the nurse rostering problem. *Electronic Notes in Discrete Mathematics*, 58:31–38, 2017.

[35] I. Gurobi Optimization. Gurobi IP Solver, 2015. URL http://www.gurobi.com.

[36] M. Hadwan, M. Ayob, N. R. Sabar, and R. Qu. A harmony search algorithm for nurse rostering problems. *Information Sciences*, 233(0):126–140, 2013.

[37] P. Hansen and N. Mladenovic. *An Introduction to Variable Neighborhood Search*. Springer, 1999.

[38] P. Hansen and N. Mladenovic. Variable Neighborhood Search: Principles and Applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[39] S. Haspeslagh, P. De Causmaecker, A. Schaerf, and M. Stølevik. The first international nurse rostering competition 2010. *Annals of Operations Research*, 218(1):221–236, 2014.

[40] F. He and R. Qu. A constraint programming based column generation approach to nurse rostering problems. *Computers and Operations Research*, 39(12):3331–3343, 2012.

[41] P.-C. Hsiao, T.-C. Chiang, and L.-C. Fu. CHeSC 2011: A variable neighborhood search-based hyperheuristic for cross-domain optimization problems. Technical report, CHeSC 2011, 2011. URL http://www.asap.cs.nott.ac.uk/external/chesc2011/.

[42] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[43] M. Hyde and G. Ochoa. CHeSC 2011-The First Cross-domain Heuristic Search Challenge, 2011. URL http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html.

[44] IBM. IBM ILOG CPLEX Optimizer, 2015. URL http://www.ibm.com/software/integration/optimization/cplex-optimizer/.

[45] IBM. IBM ILOG CPLEX CP Optimizer, 2015. URL http://www.ibm.com/software/integration/optimization/cplex-cp-optimizer/.

[46] R. M. Karp. *Reducibility among Combinatorial Problems: Complexity of*

Computer Computations. Springer, 1972.

[47] G. Kazahaya. Harnessing technology to redesign labor cost management reports. *Healthcare financial management : journal of the Healthcare Financial Management Association*, 59(4):94–100, 2005.

[48] F. Laburthe and N. Jussien. CHOCO solver - Documentation, 2011. URL http://choco.mines-nantes.fr/.

[49] G. Lach and M. E. Lübbecke. Curriculum based course timetabling: New solutions to Udine benchmark instances. *Annals of Operations Research*, 194 (1):255–272, 2012.

[50] J. Li, U. Aickelin, and E. K. Burke. A component-based heuristic search method with evolutionary eliminations for hospital personnel scheduling. *INFORMS Journal on Computing*, 21(3):468–479, 2009.

[51] J. Li, R. Bai, Y. Shen, and R. Qu. Search with evolutionary ruin and stochastic rebuild: A theoretic framework and a case study on exam timetabling. *European Journal of Operational Research*, 242(3):798–806, 2015.

[52] Z. Lü and J. K. Hao. Adaptive Tabu Search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010.

[53] Z. Lü and J. K. Hao. Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, 218(3):865–876, 2012.

[54] S. Markandeya and K. Roy. ERP and SAP Overview. In *SAP ABAP*, pages 1–20. 2014.

[55] J. P. Metivier, P. Boizumault, and S. Loudni. Solving nurse rostering problems using soft global constraints. In I. Gent, editor, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5732 LNCS, chapter 9, pages 73–87. Springer Berlin Heidelberg, 2009.

[56] R. M'Hallah and A. Alkhabbaz. Scheduling of nurses: A case study of a Kuwaiti health care unit. *Operations Research for Health Care*, 2(1-2):1–19, 2013.

[57] H. D. Mittelmann. Decison Tree for Optimization Software, 2008.

[58] M. Moz and M. Vaz Pato. A genetic algorithm approach to a nurse rerostering problem. *Computers and Operations Research*, 34(3):667–691, 2007.

[59] T. Müller. ITC2007 solver description: A hybrid approach. *Annals of Operations Research*, 172(1):429–446, 2009.

[60] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. New York : Wiley, 1999.

[61] I. H. Osman and J. P. Kelly. *Meta-heuristics: theory and applications*. Springer, 1996.

[62] T. Osogami and H. Imai. Classification of various neighborhood operations for the nurse scheduling problem. In *Lecture Notes in Computer Science*, volume 1969, pages 72–83. Springer, 2000.

[63] Y. Ozcan. *Quantitative methods in health care management: techniques and applications*, volume 4. John Wiley & Sons, 2005.

[64] R. Qu and F. He. A hybrid constraint programming approach for nurse rostering problems. In T. Allen, R. Ellis, and M. Petridis, editors, *Applications*

and Innovations in Intelligent Systems XVI - Proceedings of AI 2008, the 28th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, chapter 16, pages 211–224. Springer London, 2009.

[65] E. Rahimian. Nurse Rostering Dataset, 2015. URL http://dx.doi.org/10.15129/9664f00a-e2fd-4dbb-afef-f3c076e2c4f7.

[66] E. Rahimian, K. Akartunali, and J. Levine. A Hybrid Integer Programming and Variable Neighbourhood Search Algorithm to Solve Nurse Rostering Problems. *European Journal of Operational Research*, 2016.

[67] E. Rahimian, K. Akartunali, and J. Levine. Integer Programming for Nurse Rostering: Modelling and Implementation Issues. In *Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling, PATAT 2016*, pages 545–547, Udine, Italy, 2016.

[68] E. Rahimian, K. Akartunali, and J. Levine. A Structure-Based Matheuristic Framework for Timetabling and Rostering Problems. Technical report, University of Strathclyde, 2016.

[69] E. Rahimian, K. Akartunali, and J. Levine. A Hybrid Integer and Constraint Programming Approach to Solve Nurse Rostering Problems. *Computers and Operations Research*, 2017.

[70] G. R. Raidl, J. Puchinger, and C. Blum. Metaheuristic Hybrids. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146, chapter 16, pages 469–496. Springer US, 2010.

[71] C. R. Reeves. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, 1993.

[72] F. Rossi, P. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science, 2006.

[73] D. Santos, P. Fernandes, H. L. Cardoso, and E. Oliveira. A weighted constraint optimization approach to the nurse scheduling problem. In *IEEE 18th International Conference on Computational Science and Engineering*, pages 233–239. Institute of Electrical and Electronics Engineers Inc., 2016.

[74] H. G. Santos, T. A. M. Toffolo, R. A. M. Gomes, and S. Ribas. Integer programming techniques for the nurse rostering problem. *Annals of Operations Research*, 239(1):225–251, 2016.

[75] I. Shlyakhter. Generating effective symmetry-breaking predicates for search problems. *Discrete Applied Mathematics*, 155(12):1539–1548, 2007.

[76] P. Smet, P. Brucker, P. De Causmaecker, and G. Vanden Berghe. Polynomially solvable personnel rostering problems. *European Journal of Operational Research*, 249(1):67–75, 2016.

[77] B. Smith. Modelling for Constraint Programming. In F. Rossi, P. Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 11, pages 377–406. Elsevier, 2006.

[78] I. Solos, I. Tassopoulos, and G. Beligiannis. A Generic Two-Phase Stochastic Variable Neighborhood Approach for Effectively Solving the Nurse Rostering Problem. *Algorithms*, 6(2):278–308, 2013.

[79] R. Soto, B. Crawford, R. Bertrand, and E. Monfroy. Modeling NRPs with Soft and Reified Constraints. *AASRI Procedia*, 4(0):202–205, 2013.

[80] M. Stølevik, T. E. Nordlander, A. Riise, and H. Frøyseth. A hybrid approach for solving real-world nurse rostering problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 85–99. Springer Berlin Heidelberg, 2011.

[81] E. G. Talbi. *Metaheuristics: From Design to Implementation.* Wiley, 2009.

[82] I. X. Tassopoulos, I. P. Solos, and G. N. Beligiannis. A two-phase adaptive variable neighborhood approach for nurse rostering. *Computers and Operations Research*, 60:150–169, 2015.

[83] C. Valouxis, C. Gogos, G. Goulas, P. Alefragis, and E. Housos. A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, 219(2):425–433, 2012.

[84] W. Van Hoeve and I. Katriel. Global Constraints. In F. Rossi, P. Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 6, pages 169–208. Elsevier B.V., first edition, 2006.

[85] J. Welton. Paying for nursing care in hospitals. *The American journal of nursing*, 106(11):67–69, 2006.

[86] L. A. Wolsey. *Integer programming.* Wiley, 1998.

[87] T.-H. Wu, J.-Y. Yeh, and Y.-M. Lee. A particle swarm optimization approach with refinement procedure for nurse rostering problem. *Computers and Operations Research*, 54:52–63, 2015.

[88] Z. Zheng, X. Liu, and X. Gong. A simple randomized variable neighbourhood search for nurse rostering. *Computers and Industrial Engineering*, 110:165–174, 2017.

Appendix A

Dataset-II

This dataset is generated to resemble the only available instance ORTEC01 (in dataset-I) as closely as possible, according to the following assumptions:

1. Since the lost instances belong to a yearly dataset extracted from a hospital over 12 months, we assumed that all staff contracts are not changed and fixed during the planning year.
2. Considering the yearly nature of the dataset, we assumed that there is no change in the hospital regulations, number of shifts, and number of nurses (i.e. no hiring or firing occurs).
3. Considering the yearly nature of the dataset, we assumed that there should not be any major changes in the coverage and shift on/off requests.
4. We assumed that the coverage data for the weekend days follows a similar pattern to the coverage data of the available instance.

According to these assumptions, we only generate random instances by changing the coverage and shift on/off requests constraints. For generating coverage data, for all weekdays and for all shifts except the night shift, we use a weighted uniform random function within the range [2, 4], by considering the associated weights 0.25, 0.5, 0.25 for the included numbers within the range. For night shifts, we

Table A.1 – Results of standard Gurobi IP solver using dataset-II for 10 minutes

Instance	Obj.	LB	Gap (%)	Instance	Obj.	LB	Gap (%)
ORTEC01	1410	145	89.72	NRP06	2911	138	95.26
NRP01	15,500	570	96.32	NRP07	5660	141	97.51
NRP02	31,741	200	99.37	NRP08	385	201	47.79
NRP03	15,510	122	99.21	NRP09	14,940	1130	92.44
NRP04	25,495	1300	94.90	NRP10	22,863	310	98.64
NRP05	14,855	211	98.58	NRP11	37,698	110	99.71

use a uniform random function to generate the coverage data within the range [1, 2]. Thus, we ensure that the generated coverage data are similar to the available instance with only very slight perturbation, and at the same time, ensure that a reasonable level of randomness is reserved.

For generating shift on/off requests, first, we use a uniform random function to generate some request data including involved employees, requested shifts consisting of days off, requested days, and associated weights, while considering the ranges [0, total number of employees], [0, total number of shifts + 1], [0, total number of days], and the set {100, 1000, 10,000}, respectively. Then we use a uniform random function again to generate the required number of shift on/off requests within the range [0, 5] independently. Finally, knowing the total number of shift on/off requests, first we pick the number of shift on requests and then the number of shift off requests from the relevant data, if any.

We use an identical random seed for the whole generation of instances, and we repeat the process until we obtain a feasible problem instance. Table A.1 shows the obtained results of the standard Gurobi IP solver for 10 minutes runtime using the generated random instances with prefix *NRP* (to differentiate with original instances which are prefixed with *ORTEC*). As one can see in this table, all the generated instances are solved to a gap greater than 90% except instance *NRP08* with the gap of 48%. Therefore, it can be speculated that the generated new instances are at least as challenging as *ORTEC01*.

Appendix B

University Timetabling Problem

There are a few versions of the UTP in the literature [15], however, here we focus on the *Curriculum-based Course Timetabling Problem*, which is presented in the third track of Second International Timetabling Competition (ITC-2007) [29]. For further information regarding the problem, we refer the interested reader to Di Gaspero et al. [29], where the detailed description of the problem and the benchmark instances are presented. Next, constraints of the problem are explained.

1. All lectures of a course must be allocated.
2. Two lectures of a course cannot take place in the same room at the same period.
3. Lectures of courses which are in the same curriculum or taught by the same teacher must be all timetabled in different periods.
4. No lecture of a course can be timetabled at a period which the teacher of that course is not available to teach.
5. The number of students who attend to each course must be less or equal than the number of seats of all the rooms that host its lectures. Each student above the capacity is counted as one violation.
6. All lectures of a course must be spread into a given minimum number of

days. Each day below the minimum is counted as one violation.

7. All lectures belonging to a curriculum should be adjacent to each other within the same day. Each isolated lecture is counted as one violation.
8. All lectures of a course should be held in the same room. Each distinct room except the first is counted as one violation.

The objective is to reduce the number of violations associated with the soft constraints *5* to *8* (to increase the quality of the timetable) as much as possible. The IP formulation of the problem is presented in the following.

Sets and parameters:

C	set of courses.
U	set of curricula.
T	set of teachers.
R	set of rooms.
D, P	set of days and periods.
C_u	set of courses in curriculum $u \in U$.
C_t	set of courses taught by teacher $t \in T$.
P_c	set of forbidden periods for course $c \in C$, when the teacher of that course is not available to teach.
l_c	number of lectures that course $c \in C$ has.
s_c	number of students enrolled in course $c \in C$.
a_r	capacity (number of seats) of room $r \in R$.
d_c^{min}	minimum number of days in which the lectures of course $c \in C$ must spread.
w_{cr}^{cap}	weight relevant to room capacities for course $c \in C$ in room $r \in R$.
w_c^{spr}	weight relevant to minimum working days of course $c \in C$.
w_{udp}^{com}	weight relevant to curriculum compactness of curriculum $u \in U$ within day $d \in D$ at period $p \in P$.
w_c^{stb}	weight relevant to room stability of course $c \in C$.

Decision variables:

x_{cpr}	$= 1$ if course $c \in C$ is taught at period $p \in P$ in room $r \in R$, $= 0$ otherwise.
k_{cd}	$= 1$ if course $c \in C$ is held on day $d \in D$, $= 0$ otherwise.
q_{cr}	$= 1$ if course $c \in C$ is held in room $r \in R$, $= 0$ otherwise.
h_{up}	$= 1$ if any course belongs to curriculum $u \in U$ is taught at period $p \in P$, $= 0$ otherwise.
e_{cr}	the incurred penalty if course $c \in C$ is taught in room $r \in R$ whose capacity is less than the number of enrolled students in that course.
b_c	the incurred penalty if course $c \in C$ spreads in less than the given minimum number of days for that course.
y_{udp}	the incurred penalty if some lectures belonging to curriculum $u \in U$ are not adjacent to each other for a sequence of lectures starting from period $p \in P$ within day $d \in D$.
z_c	the incurred penalty if lectures of course $c \in C$ are not held in the same room.

Constraints:

$$\sum_{p \in P} \sum_{r \in R} x_{cpr} = l_c, \;\; \forall c \in C \tag{B.1}$$

$$\sum_{c \in C} x_{cpr} \leq 1, \;\; \forall p \in P, r \in R \tag{B.2}$$

$$\begin{cases} \sum_{r \in R} x_{cpr} \leq 1, & \forall c \in C, p \in P \\ \sum_{c \in C_u} \sum_{r \in R} x_{cpr} \leq 1, & \forall p \in P, u \in U \\ \sum_{c \in C_t} \sum_{r \in R} x_{cpr} \leq 1, & \forall p \in P, t \in T \end{cases} \tag{B.3}$$

$$\sum_{r \in R} x_{cpr} = 0, \;\; \forall c \in C, p \in P_c \tag{B.4}$$

$$\sum_{p \in P, a_r < s_c} x_{cpr}(s_c - a_r) - e_{cr} = 0, \;\; \forall c \in C, r \in R \tag{B.5}$$

$$\sum_{d\in D} k_{cd} + b_c \geq d_c^{min}, \ \ \forall c \in C \tag{B.6}$$

$$h_{up+1} - h_{up} - h_{up+2} - y_{udp} \leq 0, \ \ \forall u \in U, d \in D, p \in \left\{1 \ldots d\frac{|P|}{|D|} - 2\right\} \tag{B.7}$$

$$\sum_{r\in R} q_{cr} = 1 + z_c, \ \ \forall c \in C \tag{B.8}$$

$$x_{cpr}, k_{cd}, q_{cr}, h_{up} \in \{0,1\}, e_{cr}, b_c, y_{udp}, z_c \in \mathbb{Z}, \ \ \forall c \in C, p \in P, r \in R, d \in D, u \in U$$

Objective function:

$$\min \sum_{c\in C}\sum_{r\in R} e_{cr} w_{cr}^{cap} + \sum_{c\in C} b_c w_c^{spr} + \sum_{u\in U}\sum_{d\in D}\sum_{p\in P} y_{udp} w_{udp}^{com} + \sum_{c\in C} z_c w_c^{stb}$$