KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Technology

Nurse rostering: models and algorithms for theory, practice and integration with other problems

Pieter Smet

Supervisor:
Prof. dr. P. De Causmaecker
Prof. dr. ir. G. Vanden Berghe, co-supervisor

Dissertation presented in partial fulfilment of the requirements for the degree of PhD in Engineering Technology

July 2015

Nurse rostering: models and algorithms for theory, practice and integration with other problems

Pieter SMET

Dissertation presented in partial fulfilment of the requirements for the degree of PhD in Engineering Technology

July 2015

Preface

I define nothing. I take each thing as it is, without prior rules about what it should be.
— Robert Allen Zimmerman

Working on rostering problems has been a rewarding experience, both personally and professionally. Not only did it offer me the opportunity to explore various aspects of operational research; it also allowed me to meet many interesting people and to see different parts of the world. Of course, this dissertation would not be complete without acknowledging the many people who have contributed to it.

First and foremost, I would like to thank my supervisors Patrick De Causmaecker and Greet Vanden Berghe for giving me the opportunity to work in their research group, and for their expert guidance. Both Patrick and Greet have a high reputation in the rostering and timetabling community, so it was a privilege for me to continue the long-lasting line of research on nurse rostering in the CODeS research group.

I would like to thank Edmund Burke, Erik Demeulemeester, Luc De Raedt and Wim Van Doorsselaere for accepting to be member of my examination committee, and for providing valuable feedback which helped shape this dissertation. I thank Bert Van Bael for chairing my preliminary defence, and Toon Goedemé for chairing my public defence.

I am also thankful to IWT, the Flemish agency for Innovation by Science and Technology, for supporting the various projects I have worked on in the course of my doctoral programme, and to FWO, the Research Foundation - Flanders, for the grant awarded to do part of my research abroad.

I have been more than lucky to collaborate with many researchers from different areas who, undoubtedly, helped to create and evolve the research here presented. Therefore, I would like to thank Burak Bilgin, Peter Brucker, Mihail Mihaylov, Mustafa Mısır, Fabio Salassa and Wim Van Den Noortgate. A special word of thanks goes to Andreas Ernst, for hosting my stay at CSIRO, and for making me feel at home, far away from home.

Reading academic research papers on nurse rostering can teach you a lot, but, as the saying goes, practice makes perfect. In the last four years, I had several enlightening discussions with practitioners who helped me understand rostering from a practical point of view. For this, I would like to thank Karl Hendrickx, Jurgen Keymeulen and Wouter Kerkhofs from Tobania, and Annelies Feytens and Murielle Aendekerk from AZ Jan Portaels.

At least one paragraph should be dedicated to the people I have spent a lot of time with in the last years: my colleagues, past and present. Besides the entertaining office hours, I am thankful for the many non-computer scientific activities we have engaged in, such as the coffee breaks, pizza lunches and video game nights. I would like to thank Wim, Tony, Jannes, Jan C., Tulio, David, Thomas S., Evert-Jan, Eline, Thomas V.d.B., Joris K., Sam, Joris M., Tim, Katja, Vincent, Jan V., Koen, Laurens, Michiel, Faysal and Jorn. Special thanks go to Erik Van Achter for helping me with all my text-related issues, and for the amusing narrative on the vanity of academia.

Finally, I would like to thank my parents, Hans and Marleen, and my brother, Jasper, for supporting me in the last twenty-eight years. Their unwavering belief in my abilities to successfully complete my PhD and other engagements made all the difference. Without them, none of this would have been possible. And lastly, Sara, *you* are the best. Thank you for everything and more.

Pieter Smet Ghent, July 2015

Abstract

Nurse rostering is a personnel scheduling problem in health care in which shifts are assigned to nurses, subject to a large variety of constraints regarding personal preferences, organisational guidelines, and labour legislation. The present dissertation discusses models and algorithms for nurse rostering, treating three aspects: theory, practice and integration with other problems. This research contributes significantly to scientific, social and industrial aspects in the state of the art of nurse rostering.

By studying simplified nurse rostering problems, a basic understanding of the problem's complexity is established. These new insights identify a boundary between easy and hard problems, which strongly influences computational search approaches to the problem. Furthermore, issues regarding consistent constraint evaluation for long term rostering are exposed, and policies to address these issues are proposed. Computational experiments illustrate the importance of a consistent evaluation procedure and are employed to evaluate the presented policies.

Despite the many academic contributions, few results find their way into practice. The present dissertation therefore attempts to bridge this gap by offering two contributions that aim at facilitating the implementation of academic results. First, a general model for nurse rostering problems is introduced, which is capable of representing a large variety of personal, organisational and legislative constraints. Second, an approach is introduced to automatically order constraints according to their priority extracted from historical data. For practitioners, this is a complex and unintuitive task, which nevertheless strongly influences the outcome of any algorithm for nurse rostering. These two contributions have been implemented in a commercial software package for personnel rostering, and are currently used in hospitals and other organisations in Europe.

Finally, the scope of decision making is extended to include characteristics of related hard combinatorial optimisation problems. The focus lies on solving three different integrated task scheduling and personnel rostering problems: assigning tasks when shifts are predetermined and cannot be changed, assigning both tasks and shifts for a single isolated day, and assigning tasks and shifts for a longer scheduling period. Optimal and approximating decomposition algorithms are proposed which combine exact techniques and heuristic search. Computational experiments illustrate the effectiveness and versatility of the proposed approaches on a large variety of benchmark instances.

Beknopte samenvatting

Het opstellen van werkschema's in de gezondheidszorg kan gedefinieerd worden als een personeelsplanningsprobleem waarin shifts worden toegewezen aan verpleegkundigen, rekening houdend met verschillende beperkingen betreffende voorkeuren van het personeel, richtlijnen opgelegd door de organisatie en arbeidswetgeving. Deze verhandeling beschrijft modellen en algoritmen voor personeelsplanningsproblemen in de gezondheidszorg, en belicht drie aspecten: theorie, praktijk en de integratie met andere problemen. In elk van deze domeinen worden contributies gemaakt met een significante wetenschappelijke, sociale en industriële impact.

Door vereenvoudigde personeelsplanningsproblemen te beschouwen, wordt een eerste grondslag gelegd voor het begrijpen van de complexiteit van het probleem. Deze nieuwe inzichten laten toe een scheidingslijn te identificeren tussen eenvoudige en moeilijke problemen, dewelke een directe impact heeft op oplossingsmethodes voor het probleem. Verder worden ook tekortkomingen blootgelegd met betrekking tot de consistente evaluatie van beperkingen. Om deze problemen aan te pakken worden een aantal maatregelen voorgesteld en geëvalueerd. Computationele experimenten illustreren het belang van een consistente evaluatieprocedure en tonen de voordelen aan van de voorgestelde maatregelen.

Ondanks de vele academische bijdragen tot het domein van personeelsplanning vinden slechts enkele resultaten hun weg naar de praktijk. Deze verhandeling probeert de kloof tussen academisch onderzoek en praktijk te dichten met twee contributies die de implementatie van academische resultaten vereenvoudigen. Eerst wordt een algemeen model voor personeelsplanningsproblemen voorgesteld dat erin slaagt de grote verscheidenheid aan beperkingen in de gezondheidszorg te modelleren. Daarna wordt een techniek geïntroduceerd die automatisch de prioriteit van dergelijke beperkingen bepaalt op basis van historische data. In de praktijk is dit een complexe en weinig intuïtieve taak met een sterke invloed op de automatisch gegenereerde planning. Deze twee contributies werden reeds geïmplementeerd in een softwarepakket voor personeelsplanning dat momenteel gebruikt wordt in ziekenhuizen en andere instellingen in Europa.

Ten slotte wordt de scope van beslissingsniveau uitgebreid om eigenschappen van gerelateerde combinatorische optimalisatieproblemen in beschouwing te nemen. De nadruk ligt op het oplossen van drie geïntegreerde taak- en personeelsplanningsproblemen: taken toewijzen indien shifts reeds toegekend en bevestigd zijn, zowel taken als shifts toewijzen voor één geïsoleerde dag, en taken en shifts toewijzen voor een planningsperiode van meerdere dagen. Verschillende optimale en benaderende decompositie-algoritmen worden geïntroduceerd die exacte technieken combineren met heuristische zoekprocessen. Computationele experimenten tonen de doeltreffendheid en veelzijdigheid aan van deze algoritmen met behulp van een gevarieerde set van benchmarkinstanties.

Abbreviations

AWE	Automated weight extraction
CFC	Circulation feasibility condition
CMH	Constructive matheuristic
DN	Day nodes
GM	Geriatric medicine
GS	General surgery
INRC	International nurse rostering competition
KAHO	KAHO dataset
LBIH	Local branching improvement heuristic
MDTSS	Multi-day task and shift scheduling problem
MT	Maternity
NN	Nurse nodes
NOTT	Nottingham dataset
OT	Operating theatre
SDTSS	Single day task and shift scheduling problem
SMPTSP	Shift minimisation personnel task scheduling problem
TSS	Task and shift scheduling problem

WN	Work nodes

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and motivation

In many organisations, the workforce represents a vital resource which is typically both costly and scarce. As personnel costs account for a large part of the operational expenses, it is important to organise a given workforce as efficiently as possible. This is especially true in health care, where organisations suffer from a severe shortage of nurses and other staff [97]. In a recent study, 306 residential care facilities in Belgium were asked whether they experienced issues caused by a structural shortage of nursing personnel; almost 60% of the replies indicated that this was indeed the case [2].

The ageing population requires an increasing amount of care, while the relative size of the working population able to deliver such care, is gradually decreasing. High turnover rates in nursing further reinforce this structural understaffing. In 2007, Kovner et al. [75] reported that 13% of the newly licensed registered nurses left their job after one year, and 37% had the intention to leave. Several studies have pointed out that overall job (dis)satisfaction is strongly influenced by work-life balance, caused by the shift roster of nurses [62]. Unsurprisingly, staff scheduling, which is concerned with the construction of the nurses', and employees' in general, rosters, has received a considerable amount of attention in the academic literature.

Figure 1.1 shows an overview of staff scheduling as a collection of sequential steps, grouped in three main processes: demand modelling, rostering, and disruption handling [49]. In practice, it is common for several steps to be performed simultaneously, or in a different order. The main processes are

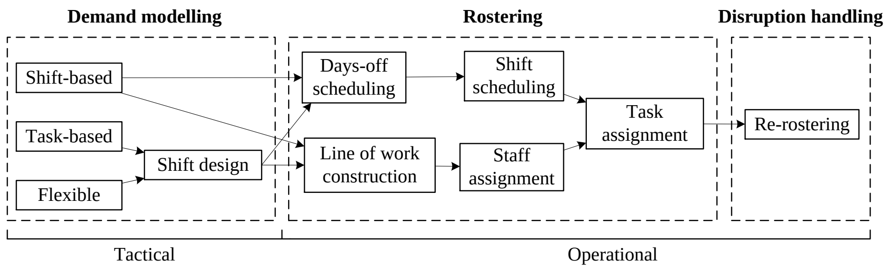


Figure 1.1: Tactical and operational decision making in staff scheduling

classified according to the time horizon of decision making [64]. Decisions regarding manpower planning are typically taken at a strategic level, and are not included in Figure 1.1, even though these decisions have a significant impact on the operational processes [73].

At the tactical level, the number of employees required at different times is determined based on forecasts of the expected workload or tasks to be performed. Three different approaches to demand modelling are distinguished. Shift-based modelling expresses the demand in terms of number of employees needed per shift, per day, which can be directly used as input to the rostering process. In task-based modelling, a set of tasks is defined, typically with additional timing information (e.g. start time, duration, time windows) and qualification requirements. Flexible modelling specifies the number of employees needed at different times, e.g. hourly intervals, on each day. Both task-based modelling and flexible modelling require shift design as a preprocessing step which derives the appropriate input for rostering [42].

The rostering process is the main focus of this dissertation, and is responsible for constructing a roster consisting of shift assignments to employees in a specified scheduling period. There are several steps which can be undertaken to realise this goal. The two main approaches are to either first construct complete lines of work and then to assign these to employees, or to first determine on which days employees are working and then to assign specific shifts on the working days. The final step of the rostering process is to assign tasks to the employees, although this step is often included in the line of work construction.

As staff scheduling is often done in a dynamic environment, unexpected events are common. Typically, such events are related to an unforeseen change in the availability of staff, e.g. acute illness of an employee. Disruptions caused by such events are resolved by the re-rostering process, typically while minimising the impact on the remainder of the current roster [87].

1.1.1 Nurse rostering problems

Nurse rostering problems typically consider the assignment of days-off and shifts simultaneously; they perform the first steps of the rostering process at the same time. The input consists of the outcome of the demand modelling process, and a description of each nurse in terms of qualifications, contracts, availabilities, requests, and any other personal characteristics relevant to the scenario. Typically, additional context-specific inter-personnel restrictions are imposed by the organisation regarding, for example, collaboration, training or supervision.

The goal of the nurse rostering problem is to assign shifts to nurses in order to meet the staffing demands, subject to the aforementioned constraints. The scheduling period in which assignments are made depends on the organisation, but typically ranges from four weeks to three months. In essence, nurse rostering is a multi-objective problem, as each stakeholder considers different aspects of a solution to be important. The hospital management, for example, might be primarily concerned with satisfying health and safety regulations, while a head nurse might want to respect the nurses' preferences as much as possible. Consequently, the different objectives in a nurse rostering problem can be contradicting, making it difficult to model and solve the problem.

1.1.2 Research questions

A recent survey of operational research literature by Van den Bergh et al. [119] showed that 291 articles on the rostering process have been published since 2004. In total, 64 dealt with nurse rostering. The large majority of these articles introduce a particular variant of the problem along with a (problem specific) solution approach. It is clear that several important aspects related to nurse rostering are neglected, regarding both fundamental issues and practical implications.

Due to a lack of theory in nurse rostering, problems are often introduced without any verifiable claims regarding their computational complexity. Researchers assume that the studied problem is hard, but seldom provide actual proof. However, experience tells us that not all nurse rostering problems are hard in practice. Moreover, particular constraints strongly influence an intuitive hardness assessment. Therefore, the present dissertation addresses the following research question.

> *Are there nurse rostering problems that can be solved in polynomial time?*

Another fundamental issue in academic research is the strong abstraction made when evaluating constraints at the boundaries of the scheduling period. Often, the current scheduling period is considered isolated, ignoring assignments from the preceding period. Beyond doubt, such assumptions strongly influence the resolution of problems in subsequent periods, an issue not systematically studied as of yet. This dissertation therefore focuses on this issue in the form of a second research question.

> *What is the impact of inconsistently evaluating constraints at the boundaries of the scheduling period?*

Besides neglecting these fundamental issues in nurse rostering, academic research also tends to overlook some of the practical implications. Kellogg and Walczak [70] conclude that only a few research results find their way to practice. An important cause of this gap between research and practice is the absence of a general model for nurse rostering problems, making it very difficult to apply novel algorithms to problems other than those they were designed for. Furthermore, little attention has been paid to the process of actually implementing a system using automated rostering algorithms in practice. By addressing the following two research questions, the present dissertation attempts to narrow the research-application gap.

> *How can different aspects of practical nurse rostering problems be modelled?*
>
> *Can the transition from manual to automated rostering be supported?*

Ernst et al. [49] structure the practical rostering process in several steps, the last ones dealing with assigning shifts and tasks. In practice, the resolution of rostering problems is intertwined with solving other decision or optimisation problems. The final contribution in this dissertation consider integrated problems that combine characteristics of task scheduling and shift rostering, thereby addressing the following research question.

> *Can the simultaneous assignment of tasks and shifts be optimised?*

These research questions bridge the research-application gap in nurse rostering from two sides. The lack of theory makes it difficult to refute or confirm claims made by practitioners that a particular problem is too complex to be solved efficiently. Clearly, the first research question attempts to directly address this issue by identifying problems solvable in polynomial time. Furthermore, the second research question investigates the impact of inconsistencies in academic models, which have long been ignored, and thereby continue to enforce the research-application gap. The third and fourth research questions aim to support practitioners in applying academic results by addressing two issues often faced by in practice. Finally, the last research question considers the rostering problem in a broader scope, automating additional steps that increase potential efficiency gains in practice.

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.

1.1.3 Background

This dissertation applies many of the established techniques in operational research for solving the presented optimisation problems. 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) integer programming, (a subset of) the decision variables are required to have integer values assigned. A complete theoretical treatment of these topics can be found in the books by Hillier and Lieberman [63], and Wolsey and Nemhauser [123].

Network flow problems are a special type of combinatorial optimisation problems in which a flow must be established in a directed graph with arc capacities. Examples include the shortest path problem, maximum flow problem and minimum cost flow problem. Despite their apparent abstract nature, many practical problems have been reformulated as network flow problems, such as aircraft routing [60] and railroad crew scheduling [117].

Network flow problems have been studied extensively, and, as a result, a wide range of efficient algorithms have been designed for solving different problem variations in polynomial time. For example, in a graph with n nodes and m arcs, Orlin [98] showed that a maximum flow can be found in $O(nm)$ time; a minimum cost flow can be found in $O(n^2m^3 \log n)$ time using the minimum mean cycle-cancelling algorithm [56]. For an extensive introduction to network flow problems and algorithms, the reader is referred to the book by Ahuja et al. [3].

1.2 Structure of the dissertation

This dissertation is structured in four parts: *Theory*, *Practice*, *Integration with other problems* and *Conclusions and future research.* Each of these parts considers particular aspects of nurse rostering, thereby addressing the aforementioned research questions.

Part I focuses on two fundamental aspects of nurse rostering which have received little or no attention in the academic literature.

Chapter 2 particularly studies simplified nurse rostering problems to identify variants which can be solved in polynomial time. The results show that there exist several nurse rostering problems which can be reformulated as minimum cost network flow problems. By linking these original results with previously published proofs, a boundary between easy and hard nurse rostering problems is identified.

Chapter 3 investigates different approaches for consistent constraint evaluation. In academic models, assignments in the preceding period are often simply ignored in constraint evaluation, while in practice, they are crucial to guarantee feasible long term schedules. The effect of incorporating policies accounting for these considerations is investigated.

In Part II, the focus is shifted towards bridging the gap between academic research and practice through two achievements. Both chapters in Part II present contributions that have been integrated in a commercial software package for personnel rostering and management, currently used in four European countries.

Chapter 4 introduces a general object model for rich nurse rostering problems, capable of representing a large variety of constraints. By using modular components, a high degree of flexibility is achieved, allowing a wide range of constraints to be modelled. Furthermore, a new benchmark dataset is introduced, containing 36 instances based on real world data.

Chapter 5 looks beyond the task of constructing rosters in a practical context. Before hospitals can use algorithms for automatically generating schedules, an extensive configuration phase is necessary. While it is a time consuming and an unintuitive task, this step is often ignored in the academic literature. The focus in Chapter 5 is on automating the transition from manual to automated rostering.

Part III broadens the scope of the decision making process involved in constructing work schedules. As is often the case, personnel rostering is not an isolated process, rather it depends on various other decision or optimisation problems. Task scheduling is an important aspect of the rostering process which is intricately linked with assigning shifts, and vice versa. The chapters in Part III focus on problems that combine characteristics of task scheduling and personnel rostering.

Chapter 6 studies the shift minimisation personnel task scheduling problem, where the employees' working times are predetermined and cannot be changed. A set of fixed tasks need to be assigned to skilled employees, while minimising the total number of employees. A decomposition algorithm is presented for this problem which optimally solves all benchmark instances from the literature. New, harder, benchmark instances are generated based on a computational study of which problem characteristics make instances hard.

Chapter 7 studies two problems that generalise the shift minimisation personnel task scheduling problem. Instead of assuming predetermined work times of the employees, assigning shifts is considered part of the decision process. The result is an integrated task and shift scheduling problem. Two variants of this problem are studied: the single day task and shift scheduling problem, and the multi-day task and shift scheduling problem. Both exact and heuristic decomposition approaches are presented and their impact on the obtained solution quality is investigated.

Chapter 8 further focuses on the multi-day task and shift scheduling problem and presents a reformulation of the problem. A column generation algorithm used for solving the linear relaxation of this reformulation is discussed. Furthermore, three heuristics are proposed, using both the primal and dual solutions obtained by column generation. Computational results for all algorithms are presented in detail.

Finally, Part IV summarises the most important conclusions and identifies areas for future research.

Part I

Theory

Introduction to Part I: Theory

Nurse rostering problems are bound to be encountered in every hospital around the world. Despite possible differences between countries, the core problem remains the same: assign a shift or day-off to each nurse on each day of the scheduling period, taking into account a set of personal, organisational and legislative constraints. The academic literature offers many different solution techniques to this problem, ranging from exact methods [55, 85] over metaheuristics [82, 90] to hybrid approaches [40, 118]. Although the problem has received considerable attention in the last decades, several important open issues remain in the academic literature. The chapters in Part I investigate two of these issues: computational complexity and inconsistencies in constraint evaluation. By studying simple variants of the problem, results are obtained which refute some of the common misconceptions in academic research on rostering.

Chapter 2 provides insights into the complexity of nurse rostering. New minimum cost network flow formulations are presented for nurse rostering problems with various types of constraints. Known complexity results are reviewed, and links with the new results are discussed, leading to knowledge about the boundary between easy and hard problems. These insights into the underlying structure support further theoretical studies on models for nurse rostering.

Chapter 3 investigates the evaluation of constraints at the boundary of the scheduling period. Often, academic models consider an isolated scheduling period, without taking into account assignments from the preceding period. In practice, however, such an approach is infeasible in the long term. For example, nurses expect that overtime accumulated in one month is compensated in the next month. Several policies for evaluating constraints in the critical parts of a roster are presented. A systematic study is performed to investigate their impact on roster quality.

Complementary contributions

Apart from the contributions reported in Chapters 2 and 3, additional work, focussing on fairness in nurse rostering, was carried out by the author in the course of the doctoral programme. The results have supported personnel rostering research conducted by Simon Martin of the University of Portsmouth and Komarudin of the Vrije Universiteit Brussel. The aforementioned contributions are briefly reviewed in what follows.

The quality of a satisfactory solution to a nurse rostering problem is typically evaluated by means of a weighted sum objective function, the result of which is proportional to the number of contractual constraint violations [112]. Composite evaluation functions such as the weighted sum are attractive because they are based on crisp mathematical descriptions of the quality measures. However, such approaches do not necessarily compare well with the human way of assessing the quality of a roster. Two solutions with the same objective function value may differ considerably in terms of pairwise comparison of individual rosters. This may lead to the perceived quality being unsatisfactory for nurses, while the solution may actually be optimal according to the weighted sum objective function.

Smet et al. [114] proposed alternatives to the weighted sum objective function which achieve a fair distribution of constraint violations. The proposed alternatives are:

- Improving the worst individual roster
- Minimising the gap between the average and worst individual roster
- Minimising the gap between the best and worst individual rosters

Computational experiments were performed on both artificial benchmark instances from the literature, and on instances based on data from two Belgian hospitals. The results showed that fairness comes at the cost of an increased number of constraint violations. For different objective functions, this trade-off is biased towards either solutions with fewer violations, or solutions in which constraint violations are much more balanced among nurses.

In the work by Simon et al. [90], a cooperative search algorithm was used to construct fair rosters. The analysis of a series of computational results showed that each of the proposed objectives results in a trade-off between solution quality (in terms of number of constraint violations), and fairness.

Komarudin et al. [74] proved a number of properties of the fairness objective functions, and proposed a new lexicographic approach to fair rostering. Computational experiments on real world data showed that the lexicographic objective functions result in a more favourable trade-off between fairness and constraint violations than the other functions.

Chapter 2

On the complexity of nurse rostering problems

Academic advances in nurse rostering mainly focus on solving specific variants of the problem using intricate exact or (meta)heuristic algorithms, while little attention has been devoted to studying the underlying structure or complexity of the problems. The general assumption is that these problems are NP-complete, even in their most simplified form. However, such claims are rarely supported with a proof for the problem under study. The present chapter refutes this assumption by presenting minimum cost network flow formulations for several nurse rostering problems. In addition, these problems are situated among the existing academic literature to obtain insights into what makes nurse rostering hard.

The content of this chapter is based on joint work with Peter Brucker, University of Osnabrück, who sadly passed away on July 24, 2013.

2.1 Introduction

Research on nurse rostering, or personnel rostering in general, has focused mostly on solving some problem at hand. As a result, a large part of the academic literature details algorithms tailored to one specific problem. Typically, general complexity claims are made, thereby referring to NP-complete problems that *resemble* the problem under discussion. However, in many cases there is no certainty that these claims hold for this particular nurse rostering problem.

	Shifts		Demand		Employee availability		
	Single	Multiple	Stable	Varying	Full	Contiguous	Varying
P_{Thm1}	✓	✓	✓		✓		
P_{Thm2}	✓		✓	✓		✓	
This chapter	✓	✓	✓	✓	✓	✓	✓

Table 2.1: Comparison of characteristics of polynomially solvable rostering problems from Brucker et al. [17]

Theoretical studies on models and complexity of nurse rostering have not yet been extensively studied in the present literature.

There are only a few authors who have formally determined the hardness of rostering problems. Osogami and Imai [100] and Brunner et al. [18] prove that rostering problems with constraints on the number of assignments of particular shifts, and with constraints on consecutive days worked and days-off are hard. Lau [80] describes a shift assignment problem closely related to rostering, and proves its NP-completeness. For restricted variants of the problem, Lau [79] and Lau [80] provide polynomial time algorithms, which are discussed in detail in Section 2.5.2 of the present chapter.

To the best of our knowledge, Brucker et al. [17] are the only authors to systematically study personnel scheduling from a theoretical point of view. Based on a general mathematical model, four polynomially solvable cases have been identified, two of which are closely related to rostering. The first problem, P_{Thm1}, considers different shifts requiring a constant number of employees on different days; the employees are assumed to be available on all days. Without any further restrictions on the assignment of shifts to employees, the problem can be solved as a series of transshipment problems. The second problem, P_{Thm2}, assumes one type of shift, and the availability of employees given by one interval, i.e. employee availability is assumed to be contiguous. There are no other restrictions. A reformulation models the problem as a minimum cost network flow problem.

The present chapter identifies new nurse rostering problems that can be solved in polynomial time. Table 2.1 compares the two polynomially solvable rostering problems studied by Brucker et al. [17] with the problems discussed in this chapter. It is worth noting that neither P_{Thm1}, nor P_{Thm2} include any time related constraints, i.e. these problems do not consider the nurses' contractual constraints. Therefore, the present chapter emphasises problems that do include such constraints.

The remainder of this chapter is organised as follows. Section 2.2 outlines the main contributions. Basic definitions of concepts in nurse rostering are introduced in Section 2.3. Section 2.4 investigates problems with restrictions on the number of assignments to each nurse. Several polynomially solvable cases are identified by formulating them as minimum cost network flow problems. Based on these results, an efficient approach to an existing problem from the literature is presented, and the complexity of commonly used benchmark datasets for nurse rostering is discussed. Sections 2.5 and 2.6 consider problems with constraints on consecutive assignments. Again, polynomially solvable cases are presented, and linked with results from the literature. For all results, practical implications are discussed. Finally, Section 2.7 formulates conclusions.

2.2 Contributions

By identifying different nurse rostering problems that can be reformulated as minimum cost network flow problems, a number of problems are identified which can be solved in polynomial time. Based on these new contributions, complexity results from the academic literature are revisited to obtain insights into what makes nurse rostering hard. The contributions of the present chapter provide an update on the current results, and further establish the foundations for theoretical studies on nurse rostering models.

Even though all results are discussed in terms of *shifts* and *days*, the ideas can be directly transferred to the domain of *tasks* and *periods*. This observation underpins the idea that the presented results have a potential impact not only in different rostering application areas (e.g. logistics, manufacturing), but also in personnel scheduling in general. Other problems are also subject to constraints with structures similar to the ones identified in this chapter. High school timetabling, for example, restricts the workload of a resource (e.g. student, teacher or room) by a minimum and maximum value [105].

2.3 Nurse rostering problems

This section introduces common concepts in nurse rostering, which will be used throughout the chapter.

Nurses have to be assigned to shifts in a way that satisfies a variety of constraints. These problems are characterised by a set of n nurses $N = \{1, ..., n\}$, a scheduling period of t days $T = \{1, ..., t\}$ and a set of s shifts $S = \{1, ..., s\}$.

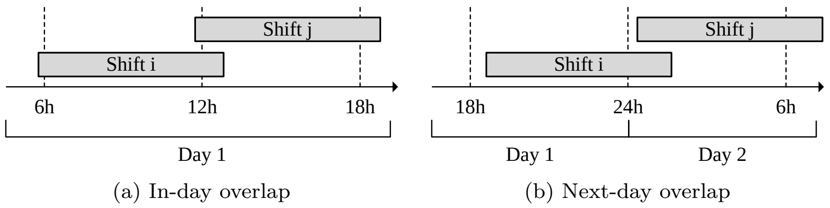


Figure 2.1: Types of overlap between shifts

A *shift* is a fixed time interval which denotes a working period. Each shift is characterised by a unique type which classifies the shifts in various ways, e.g. by time interval (morning, late), by required qualifications (senior, junior), or by a combination of these (morning-senior, late-junior). A shift is considered to occur on the day containing the start time of its time interval.

An *assignment* is the allocation of a nurse to a shift on a day. A *roster* (or schedule) is an $n \times t$ matrix which contains in each cell either an assignment or a day-off. If there is only one shift, the solution is referred to as a *days-off roster* in which the single shift represents a day-on. This work considers non-cyclic rosters, in contrast to cyclic rosters in which all nurses have the same schedule, but lagged in time [108].

Two shifts are *in-day overlapping* if their time intervals overlap when considering the shifts on the same day. An ordered set of two shifts is *next-day overlapping* if a nurse cannot be assigned to these shifts on consecutive days without overlap of their time intervals. Figure 2.1 visualises these concepts. This distinction is important since several models for nurse rostering problems assume that at most one shift can be assigned per day, thereby automatically eliminating in-day overlap, but not necessarily next-day overlap.

Domain constraints define the possible assignments for each nurse on each day. For each nurse i and day j, a set $\bar{S}_{ij}$ is defined which consists of the shifts that can be feasibly assigned to nurse i on day j. In practice, these constraints can be used to model restrictions such as *part-time nurses can only work 4h or 6h shifts* or *a nurse does not want to work late shifts on Wednesday.* This concept can also be used to model nurse skills by only including shifts in $\bar{S}_{ij}$ for which nurse i is qualified.

The *demand* d_{jk} (or coverage requirement) is the required number of nurses on day j, shift k. Demand is *stable* if the same number of nurses is required on each day and shift, i.e. $\forall j \in T, k \in S : d_{jk} = d$ (Figure 2.2a). Furthermore, if on each day and shift only one nurse is required, $\forall j \in T, k \in S : d_{jk} = 1$, there

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Shift 1	5	5	5	5	5	5	5
Shift 2	5	5	5	5	5	5	5

(a) Stable demand

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Shift 1	1	1	1	1	1	1	1
Shift 2	1	1	1	1	1	1	1

(b) Unit demand

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Shift 1	4	2	4	2	4	2	4
Shift 2	2	3	2	3	2	3	2

(c) Varying demand

Figure 2.2: Types of demand

is *unit demand* (Figure 2.2b). In contrast, demand is *varying* if it is not stable (Figure 2.2c).

The demand types shown in Figure 2.2 are examples of *exact* coverage requirements, i.e. the specified value is exactly the number of nurses to be assigned. Alternative definitions are *ranged*, *minimum* or *maximum*. A ranged definition requires that the final value should be within a specified interval. When such an interval has no upper (lower) limit, the constraint is defined as a minimum (maximum).

Apart from the coverage requirements, nurse rostering problems are typically also subject to constraints from the nurses' contracts, which can be categorised as *counters*, *series* or *successions*. Counters restrict the number of times a specific roster item (e.g. weekend assignments or days-off) can occur within a certain period. Series restrict consecutive occurrences of specific roster items [112]. Similar to the coverage requirements, these different types of constraints can be expressed as either ranged, minimum, maximum or exact. Finally, successions denote a special type of series, which restrict occurrences of specific roster items on two consecutive days.

2.4 Results for counter constraints

This section presents results for problems with counter constraints. More specifically, constraints on the number of days worked and on the number of shifts worked of each type are discussed.

Each subtitle describes the problem discussed in that section. The first two elements describe the number of shifts and demand pattern. The last element states the objective, if any. All other elements describe constraints of the problem. The type of definition (exact, range, etc.) for each constraint is mentioned between parentheses.

2.4.1 Single shift, varying demand (minimum), number of days worked (exact), feasibility

The *number of days worked* constraint limits the number of assignments per nurse in the scheduling period. In practice, this constraint is used to model different contract types. For example, a full time nurse will be required to work 20 days in a monthly scheduling period, whereas a part time nurse should only work 15 days.

Consider the set of nurses N to be homogeneous, i.e. each nurse has to work exactly a days. The problem can be formulated as the following integer linear program.

$$x_{ij} = \begin{cases} 1 & \text{if nurse } i \text{ works on day } j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{P}_1 : \sum_{i \in N} x_{ij} \geq d_j \qquad \forall\, j \in T \tag{2.1}$$

$$\sum_{j \in T} x_{ij} = a \qquad \forall\, i \in N \tag{2.2}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i \in N, j \in T \tag{2.3}$$

Constraints (2.1) are the coverage requirements. Constraints (2.2) restrict the number of days worked. Integrality of the decision variables is ensured by Constraints (2.3).

Problem $\mathcal{P}_1$ is a special case of the many-to-many generalised assignment problem [102], in which tasks need to be assigned to agents. The contribution of each task to an agent's workload is always one. Furthermore, all agents are required to work an exact number of tasks, while for each task only a minimum number of required agents is specified.

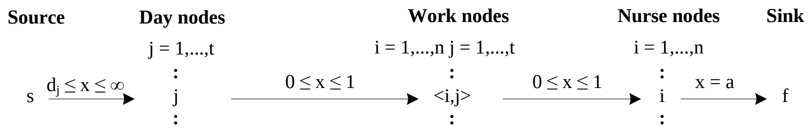


Figure 2.3: Network G_1 corresponding with problem $\mathcal{P}_1$. x denotes the flow through an arc

Minimum number of nurses

Inspired by Burns and Carter [27], a lower bound on the number of nurses required for problem $\mathcal{P}_1$ can be calculated using parameters a and d_j (Equation (2.4)).

$$n = max\left(\left\lceil \frac{\sum_{i=1}^{t} d_i}{a} \right\rceil, d_1, d_2, ..., d_t\right) \tag{2.4}$$

The remainder of this section proves that a solution exists with n nurses, by applying network flow techniques. First, a layered network is constructed in which a feasible flow corresponds to a solution of problem $\mathcal{P}_1$. Let $G_1 = (V, E)$ be a network with V the set of nodes, and E the set of arcs (Figure 2.3). The set V consists of four types of nodes.

Day nodes For each day $j \in T$, one node represents the demand on day j.

Work nodes For each nurse $i \in N$ and each day $j \in T$, one node represents nurse i working on day j.

Nurse nodes For each nurse $i \in N$, one node keeps account of the number of assignments to nurse i.

Other nodes The network contains one source node s and one sink node f.

Each day node has one incoming arc from the source node. Its outgoing arcs are directed towards the work nodes associated with the same day. Each nurse node has incoming arcs only from work nodes associated with the nurse. Finally, each nurse node has one outgoing arc to the sink node.

The supply in all nodes $V \setminus \{\text{source}, \text{sink}\}$ is zero. The supply in the source is na, which can be interpreted as the maximum number of possible assignments, based on the number of nurses and their contracts, i.e. the parameter a. The supply in the sink is $-na$.

Lemma 1. *A feasible flow in network G_1 corresponds to a feasible solution for problem $\mathcal{P}_1$.*

Proof. Due to the configuration of network G_1, a flow respecting the capacity and demand constraints is equivalent to a solution for problem $\mathcal{P}_1$. One unit of flow in an arc between a work node defined for nurse i, day j and a nurse node defined for nurse i, corresponds to a working day for nurse i. By forcing a units of flow in the arc between the nurse node i and the sink node, nurse i will work exactly a days. One unit of flow from the day node associated with day j to the work node associated with day j, nurse i, corresponds to nurse i working on day j. The flow conservation constraints ensure that at least d_j units of flow will be divided among the arcs leaving the associated day node, thereby fulfilling the staffing demands. Since there is an upper bound equal to one on the arcs between the day nodes and work nodes, a nurse can fulfil at most one unit of demand per day. □

Based on Lemma 1, there exists a solution with n nurses for problem $\mathcal{P}_1$ if a feasible flow exists in network G_1. Ahuja et al. [3] show that the latter can be proven by verifying that the circulation feasibility condition (CFC) holds in network G_1.

Theorem 1. *CFC: a circulation problem with non-negative lower bounds on arc flows is feasible if and only if, for every set S of nodes, with $\bar{S} = V - S$ [3]*

$$\sum_{(i,j)\in(\bar{S},S)} l_{ij} \leq \sum_{(i,j)\in(S,\bar{S})} u_{ij} \tag{2.5}$$

Theorem 2. *There exists a solution to problem $\mathcal{P}_1$ with n nurses as calculated by Equation (2.4).*

Proof. First, G_1 is transformed to a network G_1', representing the corresponding circulation problem, by adding a circulation arc from the sink to the source with infinite positive capacity. Next, it is verified that the CFC holds in network G_1' by checking Equation (2.5) for 14 cases which, due to the network's layered structure, cover all possibilities. Table 2.2 shows for each case, which types of nodes are in the sets S and $\bar{S}$, as well as the left and right hand sides of Equation (2.5). As mentioned before, the different types of nodes are the source (s), day nodes (DN), work nodes (WN), nurse nodes (NN), and sink (f). Note that it is assumed that all nodes of a type are in the set for the cases shown in Table 2.2. Changing this assumption does not change the correctness of the proof, but makes the calculation of the CFC terms more complicated.

S	$\bar{S}$	$\sum_{(i,j)\in(\bar{S},S)} l_{ij}$	$\sum_{(i,j)\in(S,\bar{S})} u_{ij}$
$\{s\}$	$\{DN, WN, NN, f\}$	0	$\sum_{j\in T} d_j$
$\{s, DN\}$	$\{WN, NN, f\}$	0	nt
$\{s, DN, WN\}$	$\{NN, f\}$	0	nt
$\{s, DN, WN, NN\}$	$\{f\}$	0	$+\infty$
$\{DN\}$	$\{s, WN, NN, f\}$	$\sum_{j\in T} d_j$	nt
$\{DN, WN\}$	$\{s, NN, f\}$	$\sum_{j\in T} d_j$	nt
$\{DN, WN, NN\}$	$\{s, f\}$	$\sum_{j\in T} d_j$	na
$\{DN, WN, NN, f\}$	$\{s\}$	$\sum_{j\in T} d_j$	$+\infty$
$\{WN\}$	$\{s, DN, NN, f\}$	0	nt
$\{WN, NN\}$	$\{s, DN, f\}$	0	na
$\{WN, NN, f\}$	$\{s, DN\}$	0	$+\infty$
$\{NN\}$	$\{s, DN, WN, f\}$	0	na
$\{NN, f\}$	$\{s, DN, WN\}$	0	$+\infty$
$\{f\}$	$\{s, DN, WN, NN\}$	na	$+\infty$

Table 2.2: Cases used in the proof of Theorem 2

Table 2.2 shows that most of the cases are trivial, except the ones with $DN \in S$ and $s \in \bar{S}$. From Equation (2.4) follows that $na \geq \sum_{j\in T} d_j$. Furthermore, since $a \leq t$ always holds, nt will always be greater or equal then $\sum_{j\in T} d_j$. It will thus always be possible to construct a feasible flow in G'_1. From Lemma 1 follows that there will always be a solution for problem $\mathcal{P}_1$ with n nurses, as calculated by Equation (2.4). $\square$

2.4.2 Multiple shifts, varying demand (range), number of days worked (exact), domain constraints, optimise preferences

An important objective in nurse rostering is to respect the nurses' preferences as much as possible [7, 121]. This is often modelled by minimising an integer cost c_{ijk} defined for assigning nurse i to shift k on day j.

Let a_i be the number of days nurse i is allowed to work in the scheduling period. Note that this constraint definition generalises the *number of days worked* constraint in problem $\mathcal{P}_1$, since now, different nurses can be required to work a different number of days. Let d^l_{jk}, d^u_{jk} be the minimum, maximum number of nurses required to work shift k on day j. The problem can be formulated as an integer linear program.

$$x_{ijk} = \begin{cases} 1 & \text{if nurse } i \text{ works shift } k \text{ on day } j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{P}_2: \ min \sum_{i \in N} \sum_{j \in T} \sum_{k \in S} c_{ijk} x_{ijk} \tag{2.6}$$

$$s.t. \sum_{k \in S} x_{ijk} \leq 1 \qquad \forall\, i \in N, j \in T \tag{2.7}$$

$$d^l_{jk} \leq \sum_{i \in N} x_{ijk} \leq d^u_{jk} \qquad \forall\, j \in T, k \in S \tag{2.8}$$

$$\sum_{j \in T} \sum_{k \in S} x_{ijk} = a_i \qquad \forall\, i \in N \tag{2.9}$$

$$\sum_{k \in S \setminus \bar{S}_{ij}} x_{ijk} = 0 \qquad \forall\, i \in N, j \in T \tag{2.10}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall\, i \in N, j \in T, k \in S \tag{2.11}$$

The objective function (2.6) minimises the assignment costs. Constraints (2.7) limit the number of shifts assigned per nurse and per day to at most one. Constraints (2.8) are the coverage requirements. Constraints (2.9) restrict the number of days worked. Constraints (2.10) are the domain constraints. Constraints (2.11) require the decision variables to be either zero or one.

Problem $\mathcal{P}_2$ can be reformulated as a minimum cost network flow problem in a directed network $G_2 = (V, E)$, with V the set of nodes and E the set of arcs. The set V consists of four types of nodes.

Shift nodes For each day $j \in T$ and each shift $k \in S$, one node represents the demand for shift k on day j.

Work nodes For each nurse $i \in N$ and each day $j \in T$, one node represents nurse i working on day j.

Nurse nodes For each nurse $i \in N$, one node keeps account of the number of assignments to nurse i.

Other nodes The network contains one source node s and one sink node f.

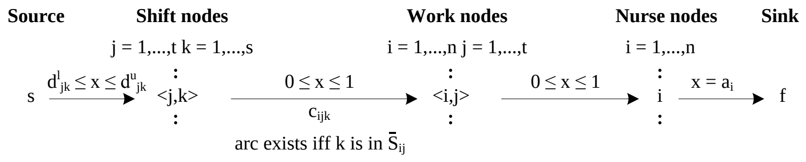


Figure 2.4: Flow network G_2 for problem $\mathcal{P}_2$, x denotes the flow through an arc

Each shift node has one incoming arc from the source node. Its outgoing arcs are directed towards the work nodes associated with the corresponding day, if an assignment of the shift to the nurse on that day does not violate the domain constraints. Each nurse node has incoming arcs only from work nodes associated with the nurse. Finally, each nurse node has one outgoing arc to the sink node. Figure 2.4 shows the structure of network G_2.

Lemma 2. *The size of network G_2 is polynomially bounded by the number of days, nurses and shifts.*

Proof. The number of nodes in G_2 is $t(s + n) + n + 2$.

ts arcs go from the source node to the shift nodes. Each shift node has n arcs to work nodes. ts shift nodes thus induce tsn arcs from shift nodes to work nodes. Each work node has one outgoing arc to a nurse node. Given nt work nodes, nt arcs exist between the work nodes and the nurse nodes. Finally, there are n arcs between the nurse nodes and the sink node. The total number of arcs in G_2 is $t(s + n(s + 1)) + n$, if $\bar{S}_{ij} = S$ for each $i \in N, j \in T$. Smaller sets $\bar{S}_{ij}$ will result in fewer arcs. □

Flow costs are only defined on the arcs between the shift nodes and the work nodes, representing the cost c_{ijk} of assigning a nurse i to shift k on day j. All nodes, except the source and sink nodes, are transshipment nodes. The supply in the source node is $\sum_{i \in N} a_i$, corresponding to the total number of days all nurses can work according to their contracts. The supply in the sink node is equal to $\sum_{i \in N} -a_i$.

Lower and upper bounds on the capacity of the arcs are appropriately defined to correctly represent problem $\mathcal{P}_2$. The arcs between the source node and the shift nodes have a lower (upper) bound equal to the minimum (maximum) coverage requirement. Arcs between the nurse nodes and sink node have a lower and upper bound equal to the required number of days worked. All other arcs require a flow of either zero or one.

Lemma 3. *A minimum cost flow in network G_2 corresponds to an optimal solution for problem $\mathcal{P}_2$.*

Proof. Due to the configuration of network G_2, a minimum cost solution respecting the capacity and demand constraints is equivalent to a solution for problem $\mathcal{P}_2$. A flow in an arc between a work node defined for nurse i, day j and a nurse node defined for nurse i, corresponds with a working day for nurse i. By forcing a_i units of flow in the arc between the nurse node i and the sink node, nurse i will work exactly a_i days. Shift assignments are determined by flows in the arcs between the shift nodes and the work nodes. A flow from the shift node associated with day j, shift k to the work node associated with day j, nurse i, corresponds with nurse i working shift k on day j, thereby incurring cost c_{ijk}. The flow conservation constraints ensure that at least d^l_{jk}, and at most d^u_{jk} units of flow will be divided among the arcs leaving the associated shift node, thereby fulfilling the staffing demands. Since there is an upper bound of one on the arcs between the work nodes and nurse nodes, a nurse cannot be assigned more than one shift per day. □

Theorem 3. *Problem $\mathcal{P}_2$ can be solved in polynomial time.*

Proof. From Lemmas 2 and 3, it follows that problem $\mathcal{P}_2$ can be transformed in polynomial time to a minimum cost network flow problem with integer capacities and arc costs. The network flow problem can be solved in polynomial time [3]. □

In the remainder of this section, two results will be derived from Theorem 3. First, a discussion on the complexity of academic nurse rostering benchmark datasets is presented. Second, a problem from the literature is revisited, and a new network flow-based solution approach is presented.

Complexity of nurse rostering benchmark instances

Benchmark datasets provide interesting indicators for comparing the performance of different algorithms. Table 2.3 shows the hard constraints in three commonly used datasets for nurse rostering: the Nottingham dataset (NOTT) [15], the dataset from the first International Nurse Rostering Competition (INRC) [61], and the KAHO dataset [112]. Based on the hard constraints, INRC can be considered to be a special case of NOTT. Indeed, any algorithm that constructs feasible solutions for NOTT can construct feasible solutions for INRC.

INRC	NOTT	KAHO
Single assignment per day	Single assignment per day	Single assignment per day
Coverage requirements	Coverage requirements	Qualification requirements
	Qualification requirements	Fixed assignments
		Only defined assignments
		No overlapping assignments

Table 2.3: Hard constraints in benchmark instances

The NOTT instances can be straightforwardly transformed to problem $\mathcal{P}_2$ by setting $a_i = t$ for each $i \in N$. The qualification requirements can be modelled by the domain constraints. The coverage requirements in NOTT can be defined as a range, minimum, maximum or exact number. To model these different definitions, the parameters d^l_{jk} and d^u_{jk} should be modified appropriately. An algorithm that generates a feasible solution for problem $\mathcal{P}_2$ can thus produce feasible solutions for NOTT and INRC. The following corollary is an immediate consequence of Theorem 3.

Corollary 1. *A feasible solution for the instances from the NOTT and INRC datasets can be obtained in polynomial time.*

The objective in NOTT and INRC is to minimise the weighted sum of soft constraint violations. These soft constraints are various time related contractual constraints limiting the number of consecutive days worked, weekends worked, etc. The cost of a solution thus depends on the extent to which certain constraints are violated, which is quite different from the assignment cost minimised in problem $\mathcal{P}_2$.

State of the art optimisation algorithms for nurse rostering often use a greedy method to construct an initial solution. Burke et al. [22], for example, use a randomised greedy constructive algorithm to generate initial solutions by assigning uncovered shifts to the nurse who incurs the least gain in penalty. A simple example illustrates how this greedy method can fail to find a feasible solution: consider an instance without any soft constraints and with two nurses A and B. Nurse A has qualifications DH and RN, while nurse B only has qualification RN. The coverage constraints require one RN-shift and one DH-shift to be assigned. If the greedy algorithm of Burke et al. [22] selects nurse A to be assigned to the RN-shift, the DH-shift cannot be assigned, and thus no feasible solution can be constructed without restarting the algorithm. However, by applying the method presented on page 26, Corollary 1 states that a feasible solution can be guaranteed in polynomial time.

Corollary 1 cannot be extended to the KAHO dataset due to the last hard constraint forbidding overlapping assignments. The structure of network G_2

cannot prevent next-day overlap. All other hard constraints can be included in G_2 by making small modifications as follows. Assignments can be fixed by changing the capacity bounds on the arcs between day nodes and work nodes. The second to last constraint states that shift k can only be assigned on day j if $d_{jk} > 0$. This can be modelled in G_2 by only adding shift nodes if at least one nurse is required on that day and shift.

A network based algorithm for a days-off rostering problem with hierarchical substitution

Hung [65] and Billionnet [12] describe a days-off rostering problem with hierarchical substitution. Based on their qualifications, employees are classified into m types, with n_k the number of employees of type k. Coverage requirements are defined in terms of these qualifications: d_{jk} is the number of employees required on day j with qualification k. The qualifications are organised in a hierarchical manner, i.e. a higher qualified employee can substitute for a lower qualified employee, but not the other way around. A cost c_k is associated with each type k employee. When an employee substitutes for a lower qualification, its cost c_k still corresponds to the employee's original, higher qualification. The scheduling period is one week. Each employee should receive o days off, or, equivalently, each employee should work $a = 7 - o$ days. The objective is to find the least cost workforce composition, and to construct a days-off roster.

Hung [65] presents sufficient conditions for a workforce composition to be feasible. A two-phase approach is used for solving the problem: the workforce composition is determined first, the actual roster is constructed afterwards. An exhaustive search determines the workforce composition, suitable for problems with $m \leq 3$. Furthermore, a single pass method is described, which does not guarantee feasible solutions. Billionnet [12] presents an integer programming formulation to determine the number of employees working a particular qualification each day. A feasible roster is constructed with the solution of the integer program.

A network flow reformulation is proposed which enables the direct construction of the roster given the least cost workforce, by solving a minimum cost flow problem in network $G^\star$ (Figure 2.5). This network is near-identical to G_2. However, *skills* are used instead of *shifts*. Arcs between skill nodes and work nodes are only present if the employee is qualified or can substitute for the skill. Lemma 3 holds for $G^\star$; a minimum cost flow solution in network $G^\star$ thus corresponds with a feasible roster.

Note that network $G^\star$ has a pseudo-polynomial number of nodes with respect to the coverage requirements of an instance, since the coverage requirements

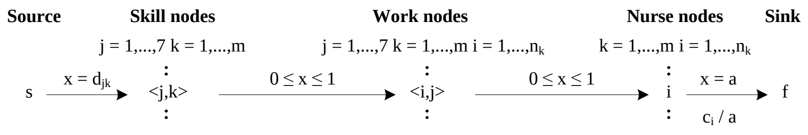


Figure 2.5: Flow network $G^{\star}$ for a days-off rostering problem with hierarchical substitution. x denotes the flow through an arc

directly impact the workforce composition, and thus the number of work nodes and nurse nodes.

By including flow costs on the arcs between the nurse nodes and the sink node, the cost of each employee is modelled. These arcs have $l_{ij} = u_{ij} = a$ and a flow cost of c_i/a, with $c_i = c_k$ if employee i is of type k. If the employee is working, a units of flow result in a cost of $c_i/a \times a = c_i$, which is the employee's cost in the original problem definition.

2.4.3 Multiple shifts, varying demand (range), number of shifts worked of each type (range), domain constraints, feasibility

The constraint on the number of days worked discussed in Section 2.4.1 is a special case of *the number of shifts worked of each type* constraint. The former limits the number of assignments in the scheduling period, whereas the latter restricts the number of assignments of each shift type within the scheduling period. This constraint has various applications in practice, e.g. balancing undesirable shifts among nurses or applying health and safety regulations.

Osogami and Imai [100] discuss a feasibility problem with one counter constraint on the number of shifts worked of each type. This constraint is defined as a range, e.g. nurse i has to be assigned to shift j on at least two days, and at most five days in the scheduling period. Furthermore, there are coverage requirements for each day, shift, also expressed as a range.

The problem can be formulated as an integer linear program. Let d^l_{jk}, d^u_{jk} be the minimum, maximum number of nurses required to work shift k on day j. Let a^l_{ik}, a^u_{ik} be the minimum, maximum number of days nurse i is allowed to work shift k.

$$x_{ijk} = \begin{cases} 1 & \text{if nurse } i \text{ works shift } k \text{ on day } j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{P}_3 : \sum_{k \in S} x_{ijk} \leq 1 \qquad \forall\, i \in N, j \in T \tag{2.12}$$

$$d^l_{jk} \leq \sum_{i \in N} x_{ijk} \leq d^u_{jk} \qquad \forall\, j \in T, k \in S \tag{2.13}$$

$$a^l_{ik} \leq \sum_{j \in T} x_{ijk} \leq a^u_{ik} \qquad \forall\, i \in N, k \in S \tag{2.14}$$

$$\sum_{k \in S \setminus \bar{S}_{ij}} x_{ijk} = 0 \qquad \forall\, i \in N, j \in T \tag{2.15}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall\, i \in N, j \in T, k \in S \tag{2.16}$$

Constraints (2.12) ensure that at most one shift is assigned per day, per nurse. Constraints (2.13) are the coverage requirements. Constraints (2.14) limit the number of shifts worked of each type. Constraints (2.15) are the domain constraints. Constraints (2.16) require the decision variables to be either zero or one.

Problem $\mathcal{P}_3$ is proven to be NP-complete by reduction from a timetabling problem [100]. However, by relaxing the constraint on the number of shifts worked of each type to a constraint on the number of days worked, problem $\mathcal{P}_3$ can be reduced to a feasible circulation problem in a network obtained by modifying network G_2 in the following way. One additional arc is added to G_2 from the sink node to the source node with infinite positive capacity. Furthermore, the capacities on the arcs between nurse nodes and sink node are modified to correctly model the range on the number of days worked.

The following theorem follows from Lemmas 2 and 3.

Theorem 4. *Problem $\mathcal{P}_3$ without ranged constraints on the number of shifts worked of each type can be solved in polynomial time.*

From Theorem 4, it follows that the granularity of a counter constraint has a significant impact on a problem's complexity. Defining a counter for days worked allows the problem to be solved as a minimum cost network flow problem, whereas restricting the number of shifts worked of each type makes the problem NP-complete. This results in interesting practical considerations when solving nurse rostering problems. It might be sufficient to model regulations at day-level, rather than at shift-level, thereby inevitably making abstraction of some of the administration's guidelines. The result, however, is a computationally tractable problem.

2.5 Results for succession constraints

This section presents results for a problem with succession constraints on shifts. First, a problem is discussed with a constraint that generalises the *number of days worked* constraint, and which can be used to model constraints on day successions. Afterwards, the relationship between this problem and a known academic result is discussed.

As in Section 2.4, the titles of the subsections describe the problems discussed.

2.5.1 Multiple shifts, varying demand (range), number of days worked (exact), domain constraints, incompatible days (range), optimise preferences

Problem $\mathcal{P}_2$ can be extended with a constraint restricting assignments on days from pairwise disjoint sets. For each nurse i, let $\bar{D}_i$ be a set of day sets F, from which at least m^l_{iF}, and at most m^u_{iF} days can be worked. All sets in $\bar{D}_i$ must be pairwise disjoint, i.e. each day in T can occur in at most one set $F \in \bar{D}_i$. There should thus be no overlap between the sets in $\bar{D}_i$. The *incompatible days* constraint restricts the number of days worked by nurse i in set F to values between m^l_{iF} and m^u_{iF}.

This constraint can be interpreted as a *number of days worked* constraint for periods which are subsets of the scheduling period. This first interpretation allows various practical restrictions to be modelled, e.g. balancing the number of assignments per week, or limiting the number of Sundays worked in the scheduling period.

The disjoint sets furthermore offer the opportunity to model succession constraints. Consider the example in which a hospital's administration requires a nurse to have a day-off after working on a bank holiday. For each bank holiday, a set F is added to $\bar{D}_i$ consisting of the bank holiday and the day after. By setting $m^l_{iF} = 0$ and $m^u_{iF} = 1$, nurse i will not be allowed to work on both the bank holiday and the day after. Note that, in this example, it is assumed that there are no consecutive bank holidays, since this would lead to non-disjoint sets in $\bar{D}_i$.

The problem can be formulated as an integer linear program. Let a_i be the number of days nurse i has to work. Let d^l_{jk}, d^u_{jk} be the minimum, maximum number of nurses required to work shift k on day j.

$$x_{ijk} = \begin{cases} 1 & \text{if nurse } i \text{ works shift } k \text{ on day } j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{P}_4 : \; min \sum_{i \in N} \sum_{j \in T} \sum_{k \in S} c_{ijk} x_{ijk} \tag{2.17}$$

$$s.t. \sum_{k \in S} x_{ijk} \leq 1 \qquad \forall\, i \in N, j \in T \tag{2.18}$$

$$d^l_{jk} \leq \sum_{i \in N} x_{ijk} \leq d^u_{jk} \qquad \forall\, j \in T, k \in S \tag{2.19}$$

$$\sum_{j \in T} \sum_{k \in S} x_{ijk} = a_i \qquad \forall\, i \in N \tag{2.20}$$

$$\sum_{k \in S \setminus \bar{S}_{ij}} x_{ijk} = 0 \qquad \forall\, i \in N, j \in T \tag{2.21}$$

$$m^l_{iF} \leq \sum_{j \in F} \sum_{k \in S} x_{ijk} \leq m^u_{iF} \qquad \forall\, i \in N, F \in \bar{D}_i \tag{2.22}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall\, i \in N, j \in T, k \in S \tag{2.23}$$

The objective function (2.17) minimises the assignment costs. Constraints (2.18) ensure that at most one shift is assigned per day, per employee. Constraints (2.19) are the coverage requirements. Constraints (2.20) limit the number of days worked. Constraints (2.21) are the domain constraints. Constraints (2.22) are the *incompatible days* constraints. Constraints (2.23) require the decision variables to be either zero or one.

Problem $\mathcal{P}_4$ can be transformed into finding a minimum cost network flow in a network G_4. This network is obtained by including additional *compatibility* nodes in network G_2. For each $F \in \bar{D}_i$, one compatibility node is added to the network between the work nodes for days in F, and the nurse node of nurse i. Since $\bar{D}_i$ is restricted to pairwise disjoint sets, the maximum size of $\bar{D}_i$ is bounded by t. The flow in the incoming arcs of each compatibility node is bounded between zero and one. The flow in the outgoing arc of each compatibility node is bounded between m^l_{iF} and m^u_{iF}. Figure 2.6 shows the structure of network G_4.

Theorem 5. *Problem $\mathcal{P}_4$ can be solved in polynomial time if $\bar{D}_i$ contains only pairwise disjoint sets of days.*

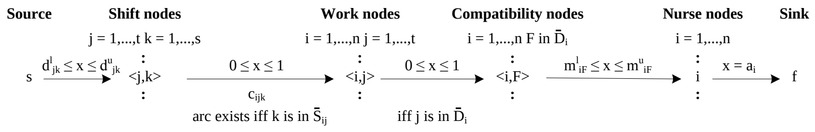


Figure 2.6: Network G_4 corresponding with problem $\mathcal{P}_4$. x denotes the flow through an arc. Arcs from work nodes to nurse nodes for days not in a set F are not drawn

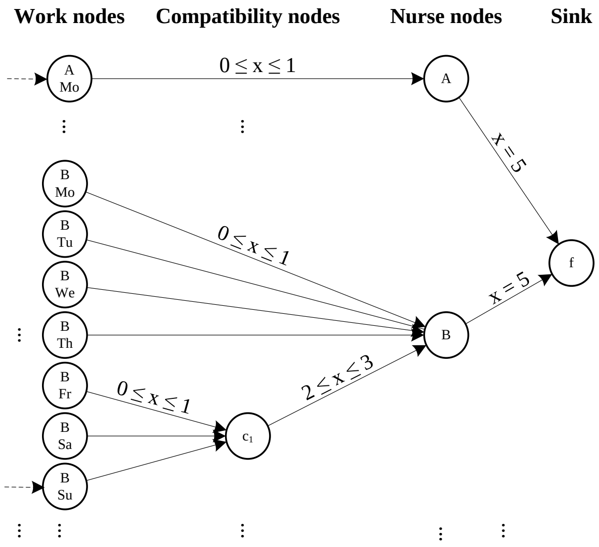


Figure 2.7: Example of network G_4 in which nurse B has to work at least two days in the period from Friday (Fr) to Sunday (Su)

Proof. A large part of network G_4 is identical to G_2; the main ideas from Lemma 3 thus hold for G_4 as well. Since the size of $\bar{D}_i$ is bounded by t, the number of nodes in the network increases maximally with nt. Through the construction of the compatibility nodes, each nurse will be working between m^l_{iF} and m^u_{iF} days from the sets F. □

Figure 2.7 illustrates this formulation with an example. Each nurse in the example has to work exactly five days, and nurse B is required to work at least two days in the period from Friday (Fr) to Sunday (Su). To accommodate for this constraint, the compatibility node c_1 is included in the network.

The requirement for the sets to be pairwise disjoint is a strong modelling restriction. If non-disjoint sets would be allowed, additional constraints could be modelled with the presented networks. Consider the formulation of the *maximum m consecutive days worked* constraint in Equation (2.24).

$$\sum_{g \in \{0,\ldots,m\}} \sum_{k \in S} x_{i(j+g)k} \leq m \qquad \forall\, i \in N,\ j \in \{1,\ldots,t-m\} \tag{2.24}$$

Equation (2.24) forces a nurse to work at most m days in a window of size $m+1$, which slides over the scheduling period. Effectively, this formulation transforms the constraint on consecutive days worked into multiple constraints on non-disjoint incompatible days. However, non-disjoint sets F would result in multiple outgoing arcs from each work node in G_4, which would allow multiple assignments to one nurse on one day, which violates the *single assignment per day* constraint.

2.5.2 Multiple shifts, varying demand (exact), domain constraints, shift succession constraints, feasibility

Health and safety regulations concerning rest time are of major importance in many organisations. Within this class of guidelines, providing sufficient rest time between two consecutive working days is regarded as one of the most important constraints. Forward shift rotation is a common concept in practice, which requires an assignment to not start earlier than the assignment on the previous day. A generalisation of this concept is the *shift succession* constraint, which forbids two particular shifts to be assigned on consecutive days.

Lau [80] discusses a feasibility problem in which the days-off roster has been predetermined. The goal is to assign shifts to working nurses. Let $\bar{T}_i$ be a set of days on which nurse i cannot work, i.e. the days-off in the predetermined roster. Let R be the set of shift pairs (k, k'), which cannot be assigned on two consecutive days. Coverage requirements are defined for each day and shift, expressed as an exact value. The problem can be formulated as the following integer linear program.

$$x_{ijk} = \begin{cases} 1 & \text{if nurse } i \text{ works shift } k \text{ on day } j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{P}_5 : \sum_{k \in S} x_{ijk} \leq 1 \qquad \forall\, i \in N, j \in T \setminus \bar{T}_i \tag{2.25}$$

$$\sum_{i \in N} x_{ijk} = d_{jk} \qquad \forall\, j \in T, k \in S \tag{2.26}$$

$$x_{ijk} + x_{i(j+1)k'} \leq 1 \qquad \forall\, i \in N, j \in T \setminus \{t\}, (k,k') \in R \tag{2.27}$$

$$\sum_{k \in S} x_{ijk} = 0 \qquad \forall\, i \in N, j \in \bar{T}_i \tag{2.28}$$

$$\sum_{k \in S \setminus \bar{S}_{ij}} x_{ijk} = 0 \qquad \forall\, i \in N, j \in T \tag{2.29}$$

$$x_{ijk} \in \{0,1\} \qquad \forall\, i \in N, j \in T, k \in S \tag{2.30}$$

Constraints (2.25) ensure that at most one shift is assigned to a nurse on a working day. Constraints (2.26) are the coverage requirements. Constraints (2.27) are the shift succession constraints. Constraints (2.28) make sure no shifts are assigned on days-off. Constraints (2.29) are the domain constraints. Constraints (2.30) require the decision variables to be either zero or one.

Problem $\mathcal{P}_5$ is proven to be NP-complete by reduction from 3SAT [80]. In addition, Lau [80] presents a greedy algorithm which solves $\mathcal{P}_5$ under two assumptions: the absence of domain constraints, and only allowing monotonic shift changes (i.e. only allowing assignments such that the indices of the assigned shifts are monotonically non-decreasing). The latter is not a strong restriction as it still allows for forward shift rotation to be enforced if the shifts are arranged in order of their start time.

The following theorem is due to Lau [80].

Theorem 6. *Problem $\mathcal{P}_5$ without domain constraints and with monotonic shift succession constraints can be solved in polynomial time [80].*

In a follow-up paper, Lau [79] presents a polynomial time algorithm for problem $\mathcal{P}_5$ when the days-off roster has a particular structure. All nurses' working days are contiguous and the work stretches either start or stop on the same day, i.e. the days-off roster is *tableau shaped*. A solution to this special case can be obtained from a minimal set of node-disjoint paths in a layered network.

The following theorem is due to Lau [79].

Theorem 7. *Problem $\mathcal{P}_5$ without domain constraints and with a tableau shaped days-off roster can be solved in polynomial time [79].*

The network flow model presented in Section 2.4.2 supports the proof of a related result. By omitting the *shift succession* constraint, problem $\mathcal{P}_5$ can be reduced to the problem of finding a feasible flow in network G_2. The fixed days-off roster can be modelled in the network by changing the capacity upper bounds of the correct arcs between work nodes and nurse nodes to zero. Since the number of days worked per nurse is determined by the days-off roster, a_i is set to $t - |\bar{T}_i|$, for each nurse $i \in N$. Finally, the supply in the source node is set to $\sum_{j \in T} \sum_{k \in S} d_{jk}$. The supply in the sink node is equal to $\sum_{j \in T} \sum_{k \in S} -d_{jk}$. Note that neither the structure of the days-off roster, nor the domain constraints are subject to restrictions.

The following corollary is an immediate consequence of Lemmas 2 and 3.

Corollary 2. *Problem $\mathcal{P}_5$ without shift succession constraints can be solved in polynomial time.*

Table 2.4 compares characteristics of the polynomially solvable problems identified by Lau [79, 80], and by Theorem 5 presented in Section 2.5.1.

	Domain constraints	Unconstrained days-off roster	Shift succession	Day succession	Number of days worked
Lau [79]			✓		
Lau [80]		✓	(✓)^1		
Theorem 5	✓	✓		(✓)^2	✓

^{1} Only monotonic shift succession constraints
^{2} Only pairwise disjoint day succession constraints

Table 2.4: Comparison of characteristics of polynomially solvable problems from Lau [79, 80]

Relating the NP-completeness proof from Lau [80] to Theorem 5 allows further examination of the boundary between hard and easy definitions of succession constraints. Problems with the general *shift succession* constraint are NP-complete. However, Theorem 5 proved that a restricted version of the *day succession* constraint can be solved as a minimum cost flow problem. Similar to the findings on counter constraints, the transition from succession constraints on days to succession constraints on shifts transforms a tractable problem into an intractable one. While for counter constraints, in some cases, the abstraction from shifts to days could be justified (e.g. by aggregating constraints on the number of shifts worked of each type to restrict the total number of days worked), it is harder to do so for the succession constraints. Constraints on day successions are not useless, rather they have a different purpose to the shift successions.

2.6 Results for series constraints

This section discusses a problem with series constraints on consecutive days worked and consecutive days-off. Again, the titles of the subsections describe the problems discussed.

2.6.1 Single shift, varying demand (minimum), consecutive days worked (range), consecutive days-off (range), number of days worked (maximum), minimise staff size

The objective of the problem discussed by Brunner et al. [18] is to minimise the size of the workforce. Apart from coverage requirements, expressed as a minimum, there are three other constraints. The first constraint limits the number of days worked to a maximum value. The second and third constraints define a range on the number of consecutive days worked and days-off, respectively.

The problem can be formulated as the following integer linear program. Let $\bar{D}^{\text{work}}$ be the maximum number of days a nurse can work in the scheduling period. Let $\bar{D}^{\text{on}}$, $\bar{D}^{\text{off}}$ be the maximum number of consecutive days worked, days-off. Let $\underline{D}^{\text{on}}$, $\underline{D}^{\text{off}}$ be the minimum number of consecutive days worked, days-off.

$$x_{ij} = \begin{cases} 1 & \text{if nurse } i \text{ works on day } j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{P}_6: \; min \text{ number of nurses} \tag{2.31}$$

$$s.t. \sum_{i \in N} x_{ij} \geq d_j \qquad \forall j \in T \tag{2.32}$$

$$\sum_{j \in T} x_{ij} \leq \bar{D}^{\text{work}} \qquad \forall i \in N \tag{2.33}$$

$$\sum_{n=j}^{\bar{D}^{\text{on}}+j-1} x_{in} \leq \bar{D}^{\text{on}} \qquad \forall i \in N, j \in T \tag{2.34}$$

$$\sum_{n=j}^{\underline{D}^{\text{on}}+j-1} x_{in} \geq \underline{D}^{\text{on}}(x_{ij} - x_{i(j-1)}) \qquad \forall i \in N, j \in T \setminus \{1\} \tag{2.35}$$

$$\sum_{n=j}^{\bar{D}^{\text{off}}+j-1} (1 - x_{in}) \leq \bar{D}^{\text{off}} \qquad \forall i \in N, j \in T \tag{2.36}$$

$$\sum_{n=j}^{\underline{D}^{\text{off}}+j-1} (1 - x_{in}) \geq \underline{D}^{\text{off}}(-x_{ij} + x_{i(j-1)}) \qquad \forall i \in N, j \in T \setminus \{1\} \tag{2.37}$$

$$x_{ij} \in \{0, 1\} \qquad \forall \; i \in N, j \in T \tag{2.38}$$

The objective function (2.31) minimises the number of nurses. Constraints (2.32) are the coverage requirements. Constraints (2.33) limit the number of days worked. Constraints (2.34) and (2.35) are the minimum and maximum number of consecutive days worked, respectively. Constraints (2.36) and (2.37) are the minimum and maximum number of consecutive days-off, respectively. Constraints (2.38) require the decision variables to be either zero or one.

Brunner et al. [18] proved that problem $\mathcal{P}_6$ is NP-complete by showing that it has the *circulant problem* as a special case. Removing all constraints regarding consecutive assignments reduces the problem of finding a feasible solution for $\mathcal{P}_6$ to the problem of finding a feasible flow in network G_1. The lower bound on the flow in the arcs from nurse nodes to the sink node needs to be set to zero, the upper bound should be set to $\bar{D}^{\text{work}}$.

The following corollary is an immediate consequence of Lemmas 1 and 2.

Corollary 3. *A feasible solution for problem $\mathcal{P}_6$ without ranged constraints on the number of consecutive days worked and days-off can be found in polynomial time.*

Limiting the number of consecutive days worked and days-off thus makes the problem difficult. Consequently, almost all problems in practice are difficult.

As was mentioned on page 34, the pairwise disjoint incompatible days constraint strongly resembles the constraint on the maximum number of consecutive working days. To strengthen Corollary 3, the same obstacle holds as was discussed on page 34. Satisfaction of the *maximum number of consecutive working days* constraint cannot be guaranteed without the possibility of modelling a constraint on non-disjoint incompatible days in network G_4.

2.7 Conclusions

The present chapter systematically studied the complexity of nurse rostering problems, thereby further establishing the foundations for theoretical studies on models for rostering. By presenting transformations of different problems to minimum cost network flow problems, new cases were identified which can be solved in polynomial time. Specifically, decision and optimisation problems were reformulated with multiple shifts, varying demand, and constraints on the number of days worked, nurses' domains and incompatible days.

Previously published complexity proofs were discussed in the light of these new results, and, as a result, a boundary between tractable and intractable nurse rostering problems was established. The new results show that for both counter constraints and succession constraints, the difference between easy and hard problems corresponds to defining the constraint on day-level or on shift-level.

These insights allow decision makers to reconsider the formulation of their problem, as it could mean making the problem computationally tractable. In some cases, a revision of the organisational policy could result in alternative constraint definitions, which then allows the reassessed rostering problem to be modelled using the network flow formulations presented in this chapter. As a result, significant savings can be realised by not having to invest in computational resources necessary to obtain optimal rosters.

Chapter 3

Policies for consistent constraint evaluation in nurse rostering

Academic models for nurse rostering typically make significant abstractions of complex real life problems. Often, one isolated scheduling period is considered, contradicting common practice where nurse rostering inherently spans multiple periods. In the present chapter, weaknesses of constraint evaluation processes are revealed through a series of computational experiments on nurse rostering instances. To address the identified problems, policies to consistently evaluate constraints are introduced, when taking into account multiple consecutive scheduling periods. The proposed methods are evaluated on a set of benchmark instances, thereby demonstrating the impact of inconsistent constraint evaluation at the boundaries of the scheduling period.

The content of this chapter is based on joint work with Fabio Salassa, Politecnico di Torino.

3.1 Introduction

In academic studies on automated nurse rostering, the focus is often on solving problems with a single, isolated time horizon rather than on improving the perceived quality of rosters over a long period. Salassa and Vanden Berghe [110] denote the former as a static horizon approach to nurse rostering. An argument for this methodology is that taking into account previous, or future, scheduling periods does not increase the complexity of the problem at hand. However, in practice, real hospital applications are strongly influenced by the inertia of previous periods. Regardless of the performance of the applied algorithm, many approaches based on the restricted academic models fail to produce rosters that meet hospital requirements. Nevertheless, they are common in academic benchmarks [21, 61, 121].

This consideration may, to some extent, explain the gap between academic and applied approaches to nurse rostering [70]. Some practical aspects should be included in a model for automated nurse rostering in order to arrive at a generic automated timetabling procedure. The present chapter discusses issues regarding continuity in constraint evaluation, arising when an isolated scheduling period is considered. Policies for different types of constraints are presented to resolve these issues. In an extensive computational study, the effects of such policies are investigated.

The rest of the chapter is organised as follows. The main contributions are summarised in Section 3.2. Section 3.3 briefly introduces a nurse rostering variant suited for demonstrating the new evaluation methodology. Section 3.4 identifies the main shortcomings with respect to continuity in the current state of the art of nurse rostering. Related work and the discrepancies with the current contributions are also discussed. Remedies for the main identified issues are presented and evaluated in Sections 3.5, 3.6 and 3.7. Finally, Section 3.8 concludes this chapter.

3.2 Contributions

Practitioners are well aware of the pitfalls regarding consistent constraint evaluation and their implications for long term scheduling. To them, it is obvious that assignments at the end of the preceding period influence the assignments at the start of the current scheduling period. This consideration has been largely ignored in the academic literature, even though it has an impact on modelling the problem. This chapter expands upon these issues, and systematically demonstrates potential impact through a series of computational experiments. Furthermore, policies to address the inconsistencies are proposed and evaluated for two types of constraints.

In an effort to bring academic research closer to practice, new benchmark instances have been made publicly available. These instances are available in a widely adopted format, and include realistic contracts and information regarding the preceding scheduling period, thereby allowing the proposed policies to be applied.

3.3 Description of a nurse rostering problem

The problem considered in this study is a common nurse rostering problem, where a working shift or a day-off should be assigned to each multi-skilled nurse on each day in the scheduling period, according to several contractual and operational requirements. There are several types of (possibly overlapping) shifts, to be assigned in a scheduling period, so that coverage requirements are met. This assignment is constrained by two hard constraints: 1) the coverage requirements must be met exactly, and 2) each nurse can be assigned to at most one shift per day. The *quality* of a solution is determined by violations of the soft constraints (Table 3.1). The objective is to maximise the *quality* by minimising the weighted sum of soft constraint violations.

The employees' contractual soft constraints can be categorised as either *counter* constraints or *series* constraints [11]. Counter constraints denote all constraints that can be evaluated by counting the appearance of certain types of assignment in a roster. The series constraints on the other hand, correspond to restrictions on successive assignments.

Counter constraints
Maximum and minimum number of shifts that can be assigned to nurses
Maximum number of working weekends
Day-off/on and shift-off/on requests
Series constraints
Maximum and minimum number of consecutive working days
Maximum and minimum number of consecutive days off
Maximum number of consecutive working weekends
Maximum number of working weekends in four weeks
Avoid isolated days-off
Shift succession (e.g. no Early shift after a Night shift)

Table 3.1: Soft constraints

A more general model for nurse rostering problems is discussed in Chapter 4, while the more concise variant presented here is suitable for demonstrating the concepts in this chapter.

3.4 Inconsistencies in constraint evaluation

Static horizon approaches lead to inconsistent constraint evaluation at the boundaries of the scheduling period. This section supports this claim by identifying consistency issues for both counter and series constraints. Furthermore, related literature is discussed which addresses some of these identified problems.

Section 3.3 defined counters as constraints which limit the occurrence of a particular type of assignment in a period. For example, overtime can be limited by a counter constraint on the number of hours worked. In practice, violations of this constraint are carried over from one scheduling period to the next, such that they can be compensated. This allows overtime to be balanced over a longer time period than just one scheduling period, which is often necessary due to peaks in coverage demand. Furthermore, wards with structural understaffing require nurses to consistently work overtime. In these cases, carrying over information from one period to the next is a vital requirement for a scheduling system. Ignoring overtime from the previous scheduling period, which is what happens when considering a static horizon, results in unbalanced workload and discontented staff.

Series constraints restrict successive assignments. In a static horizon approach, the evaluation of these constraints becomes inconsistent at the boundaries of the scheduling period. Consider the *at least three consecutive days worked* constraint. This series constraint cannot be correctly evaluated using only the data from the current period. Assignments from the preceding schedule, or at least from the last two days from the previous scheduling period, are necessary for a consistent constraint evaluation. Ignoring such considerations can lead to solutions which neglect violations of series constraints at the boundaries of the scheduling period.

To address these issues, Salassa and Vanden Berghe [110] introduced the stepping horizon approach to nurse rostering, in which data from the preceding period imposes restrictions on the current scheduling period. Figure 3.1 illustrates this concept.

The idea of systematically considering information from the previous period has been explored by only a few authors in the context of personnel rostering. Glass and Knight [55] discuss issues regarding continuity in the ORTEC01 benchmark instance. Two ideas are presented to address these problems: 1) adding specialised *continuity constraints* at the start of the scheduling period, and 2) counting additional *implied penalties* to avoid large, avoidable penalties in the next scheduling period. By using these two concepts, a more realistic solution for the ORTEC01 instance is generated.

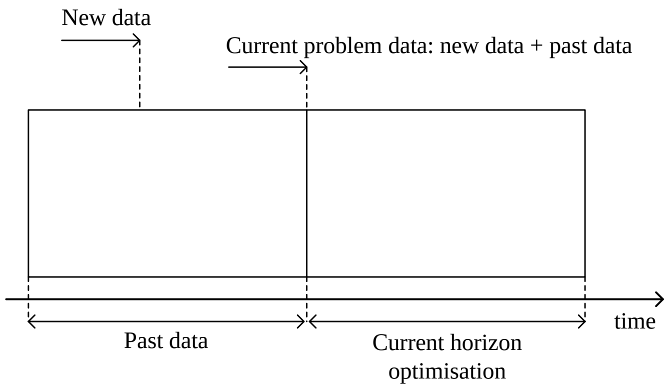
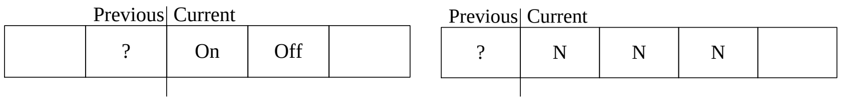

New data
Current problem data: new data + past data
time
Past data
Current horizon optimisation

Figure 3.1: Stepping horizon approach [110]



(a) Example 1: no isolated days-on (b) Example 2: maximum number of consecutive night shifts

Figure 3.2: Examples of series at the boundaries of the scheduling period

Burke et al. [23] describe a commercial nurse rostering system in which assignments from the previous period are used to initialise the constraints that apply to the current scheduling period. Based on relevant previous assignments, the initialisation procedure adjusts the ranges of some constraint parameters.

Smet et al. [112] present a general model for personnel rostering in which information from past (and future) scheduling periods can be included, so that constraints can be correctly initialised. Examples are discussed to illustrate the need for historical information when the evaluation of series constraints is extended over the boundaries of the scheduling period. To correctly evaluate series constraints concerning isolated days-on (Figure 3.2a) or the number of consecutive night shifts (Figure 3.2b), a stepping horizon approach is required such that necessary information from the previous period can be taken into account.

Ikegami and Niwa [66] use previous assignments to explicitly model restrictions on the first few days of the current period, in the form of additional constraints. Warner [122] models the working history such that the quality of the nurses' previous rosters and their preferences for the current period can be balanced.

A systematic study on the impact of taking into account the previous period on the quality of the generated rosters has not yet been undertaken. Neither Burke et al. [23], nor Smet et al. [112] discuss an appropriate methodology for initialisation and its impact on e.g. balancing constraint violations over multiple scheduling periods. Glass and Knight [55] present results for one benchmark instance, but a thorough evaluation of the benefits of their stepping horizon approach is missing. Furthermore, it may become difficult to generate the additional constraints for comprehensive constraints, and, when there are many constraints or nurses, it can increase the problem size significantly. Alternative stepping horizon approaches should thus be 1) sufficiently flexible to be used in various models, and 2) general enough to cope with various types of constraints.

3.5 Policies for consistent constraint evaluation

This section presents policies for consistent constraint evaluation for both counter and series constraints in a stepping horizon approach.

3.5.1 Counter constraints

To alleviate the inconsistencies identified in Section 3.4, the *debit system* is introduced. The idea is to calculate *debits* for each nurse based on assignments in the previous period, which are then used to modify ranges of counters in the current scheduling period. Algorithm 1 shows the pseudocode of the algorithm for calculating the debits. Let C_i be the set of counter constraints associated with contract type i, and N_i the set of nurses with contract type i. Let $v_{n,c}$ be the value of constraint c for nurse n in the previous scheduling period. The range (minimum, maximum) of counter c in the previous and current period are denoted as $l_{n,c}$, $u_{n,c}$ and $l'_{n,c}$, $u'_{n,c}$, respectively.

The mechanism of Algorithm 1 is illustrated through an example concerning a constraint on the number of days worked. Clearly, the debit system can be easily applied to other counter constraints as well.

First, within each set of nurses with identical contracts, the nurse working the most days in the previous scheduling period is identified. Based on the number of days worked of this nurse (τ_c), the amount of debit for all nurses is calculated as the difference between the largest number of days worked and his/her own number of days worked ($d_{n,c}$).

Algorithm 1 Debits calculation algorithm

```
1: for all $i \in$ contract types do
2:     for all $c \in C_i$ do
3:         $\tau_c = \max(v_{1,c}, ..., v_{|N_i|,c})$
4:         for all $n \in N_i$ do
5:             $d_{n,c} = \tau_c - v_{n,c}$
6:             $l'_{n,c} = l_{n,c} + d_{n,c}$
7:             $u'_{n,c} = u_{n,c} + d_{n,c}$
8:         end for
9:     end for
10: end for
```

	Days worked	Debit
Nurse1	20	0
Nurse2	18	2
Nurse3	17	3

Table 3.2: Example debit calculation

A concrete example is shown in Table 3.2. In one period, Nurse1 has worked 20 days, Nurse2 18 days, and Nurse3 17 days. The debit system provides a debit of $20 - 18 = 2$ to Nurse2 while Nurse3 receives a debit of $20 - 17 = 3$ days.

Before solving the problem for the current period, the system changes the range of the allowed number of days worked for each nurse, based on their received debit. Adapting individual conditions based on previous assignments corresponds to what is done in practice when, for example, overtime is carried over to the next period. Continuing the example from Table 3.2, for Nurse1, nothing changes since the debit is zero. For the remaining nurses, a quantity equal to the debit is added to both the minimum and maximum allowed number of days worked. If in the previous period, the number of days worked had to be between 10 (minimum) and 16 (maximum), then the ranges in the current period for Nurse2 are set to 12/18 and for Nurse3 to 13/19.

3.5.2 Series constraints

A policy for consistent evaluation of series constraints in a stepping horizon approach is straightforward. By providing information on the previous period, the stepping horizon approach is no longer limited to the current period for evaluation, but can be extended into the previous period.

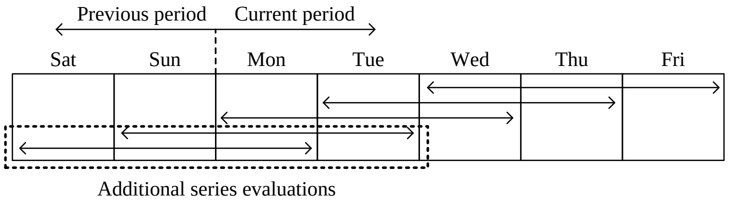


Figure 3.3: Evaluation of series constraints in a stepping horizon approach

Figure 3.3 shows which intervals need to be considered when evaluating the *no isolated day-on* constraint. A static horizon approach only checks the intervals starting in the current scheduling period. The stepping horizon methodology enables evaluation over the boundaries of the current scheduling period, and thus arrives at high quality solutions when data regarding preceding roster periods is available.

3.6 Computational analysis

The identified constraint evaluation inconsistencies influence the quality of employee rosters. In order to analyse the impact, the results of a series of computational experiments are analysed. The benefits of applying the proposed policies in a stepping horizon approach are also evaluated and discussed.

3.6.1 Experimental setup

Instances from the first International Nurse Rostering Competition (INRC) [61] were used in the experiments. This choice is motivated by the fact that 1) these instances have become benchmarks for automated nurse rostering research [19], and 2) they can be solved to optimality, in acceptable computation time, with an integer programming solver. Furthermore, these instances provide realistic scenarios in that some of the contractual constraints can never be satisfied in feasible solutions. For example, to meet the coverage requirements, nurses have to work more than the maximum allowed number of working days. These instances are thus very suitable to illustrate the proposed policies.

All experiments were conducted using an integer programming model implemented in the XPRESS MOSEL modelling language. The XPRESS solver (v. 21.01.06) was used on an Intel Core 2 Duo CPU at 2.13GHz with 4GB RAM.

3.6.2 Counter constraints

The goal of the first set of experiments is to demonstrate that a static horizon approach risks to generate unacceptable results when looking at successive scheduling periods, while the debit system in a stepping horizon approach provides a practical and easily adaptable alternative. The focus is on a constraint on the number of days worked, which allows workload balancing over multiple scheduling periods.

An unbalanced shift roster is quantified by the *maximum imbalance*. This is the largest difference in number of days worked between all nurses with the same contract type. Assume, for example, that in an optimal solution, nurse i works v_i days, and nurse j v_j days. The *imbalance* between these two nurses is $|v_i - v_j|$. The largest difference between all nurses with the same contract is the maximum imbalance. In practice, nurses will want the maximum imbalance to be spread among different staff members in different scheduling periods, such that overtime is fairly distributed over staff.

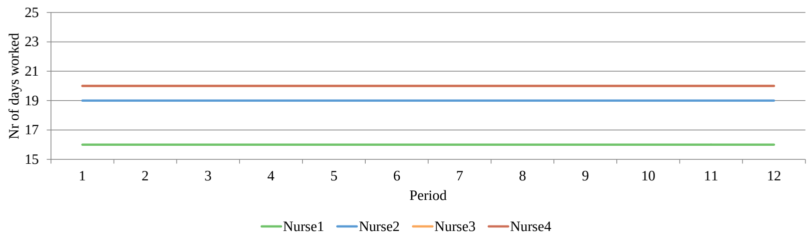
To evaluate the benefit of the debit system over the static horizon approach, the maximum imbalance was calculated for a one-year period. Using the static horizon approach, twelve periods of one month were successively solved without taking into account data from previous periods. In the stepping horizon approach, after solving each period, the debit system was applied before solving the next period.

Table 3.3 shows for ten instances, the maximum imbalance (*MI*), organised per contract type (100%, 75%, 50% and *Night*). The contribution of the debit system is highlighted through the decrease of the maximum imbalance compared to the results obtained with the static horizon approach (Δ). For example, in the *sprint01* instance, the maximum imbalance in one year for the full time nurses is 48 days when using a static horizon approach. This means that among the full time nurses, at least one nurse is working 48 days more than another nurse within one year. This roughly translates to an increased workload of two months of one nurse over another. By applying the debit system, the maximum imbalance is reduced to only 8 days, thereby reducing the imbalance with 40 assignments. In most cases, the maximum imbalance is significantly reduced when using the debit system, while the few increases are relatively small.

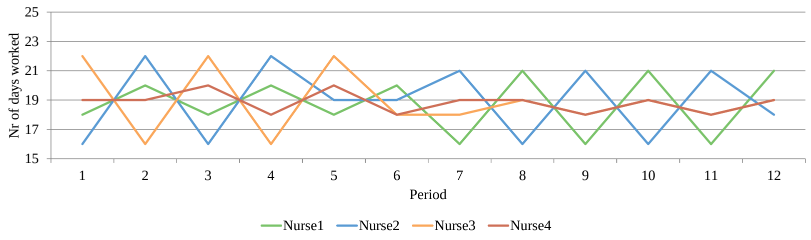
Figure 3.4 illustrates the practical implications of the debit system. In the example, each nurse is allowed to work at most 18 days. However, due to structural understaffing in the ward, working more assignments than allowed is unavoidable. The static horizon results in an unbalanced workload among the nurses, when considering multiple scheduling periods (Figure 3.4a). Each nurse works the same number of days in each period, without any compensation. In

	Static horizon				Stepping horizon (debits)							
	100%	75%	50%	Night	100%		75%		50%		Night	
	MI	MI	MI	MI	MI	Δ	MI	Δ	MI	Δ	MI	Δ
sprint01	48	12	0	0	8	**-40**	2	**-10**	5	5	0	0
sprint02	24	24	24	0	24	0	12	**-12**	6	**-18**	0	0
sprint03	60	0	12	60	2	**-58**	2	2	13	1	2	**-58**
sprint04	12	0	0	24	3	**-9**	29	29	15	15	2	**-22**
sprint05	84	24	24	12	17	**-67**	11	**-13**	4	**-20**	13	1
sprint06	24	12	12	36	3	**-21**	1	**-11**	5	**-7**	9	**-27**
sprint07	36	24	24	12	14	**-22**	1	**-23**	12	**-12**	16	4
sprint08	36	24	0	12	3	**-33**	12	**-12**	8	8	9	**-3**
sprint09	36	36	12	24	6	**-30**	18	**-18**	3	**-9**	1	**-23**
sprint10	48	12	24	12	7	**-41**	2	**-10**	27	3	19	7
Average	41	17	13	19	9	**-32**	9	**-8**	10	**-3**	7	**-12**

Table 3.3: Comparison of the maximum imbalance over a one year period



(a) Static horizon approach (Nurse3 and Nurse4 have identical values)



(b) Stepping horizon approach (debits)

Figure 3.4: Number of days worked per scheduling period

practice, such a solution is unacceptable, since nurses 2, 3 and 4 constantly need to work the same amount of overtime in each period. Using the stepping horizon approach (Figure 3.4b), the number of assigned days varies in time; whenever a nurse works a large surplus in one period, it is compensated in the following period by assigning less days than contractually required.

3.6.3 Series constraints

The second series of experiments compares the impact of the static and stepping horizon approaches on violations of series constraints when several successive scheduling periods are considered.

Table 3.4 shows, for ten instances from the INRC dataset, the total penalty incurred by violations of series constraints over a period of five months (*Penalty*), and the gap to the best possible penalty (Δ). The best possible penalty was obtained by solving the five-month period at once in one pass. Results for the static horizon approach were obtained by solving each month separately, and then calculating the penalty after concatenating the five solutions. Finally, for the stepping horizon approach, information regarding the preceding period was

	Best possible	Static horizon		Stepping horizon	
	Penalty	Penalty	Δ (%)	Penalty	Δ (%)
sprint01	276	332	20.3%	287	4.0%
sprint02	286	306	7.0%	297	3.9%
sprint03	251	287	14.3%	262	4.4%
sprint04	284	315	10.9%	292	2.8%
sprint05	290	310	6.9%	297	2.4%
sprint06	266	314	18.1%	272	2.3%
sprint07	280	312	11.4%	287	2.5%
sprint08	276	296	7.3%	280	1.5%
sprint09	271	307	13.3%	281	3.7%
sprint10	264	304	15.2%	271	2.7%
Average	274	308	12.5%	283	3.0%

Table 3.4: Comparison of static and stepping horizon approaches for series constraints

provided to enable the evaluation procedure described on page 47. Again, the results were obtained by concatenating the five one-month solutions, and then calculating the penalty.

The results in Table 3.4 show that by considering isolated scheduling periods, valuable information is neglected, causing additional violations of series constraints at the boundaries of the scheduling periods. This results in significantly more constraint violations by the static horizon approach than by the stepping horizon approach and the best possible. There still exists a gap of on average 3% between the best possible penalty and the penalty obtained with the stepping horizon approach, mainly due to the fact that the stepping horizon approach still only considers one-month periods, thereby inevitably incurring penalties which can be avoided when considering all five months at once. However, in practice, generating an optimal solution for the aggregated scheduling period is not realistic, since 1) the input often only become known a few months before the scheduling period and 2) the problem size would become too large. The stepping horizon approach thus proves to be a viable alternative.

3.7 Extending the INRC dataset

The instances of the first INRC are used in many academic studies for evaluating the performance of search algorithms. These instances serve as an excellent benchmark although they do not fully represent realistic scenarios. This section introduces new instances based on the INRC data format in an effort to extend the currently available instances with more practical cases. By adhering to the model presented by Haspeslagh et al. [61], these new instances can easily be used for evaluating academic nurse rostering contributions in a more realistic setting.

Two modifications were made to the existing instances. First, additional information regarding the previous scheduling period was added to the new instances so that the evaluation techniques presented in Section 3.5 can be applied. The organisers of the INRC have supported this research by providing the best known solutions for each instance. The dates of the assignments in these solutions were modified by setting them four weeks earlier, so that the best known solutions represent the preceding schedule of the problem to be solved. Second, in some of the instances, new constraints were added to the nurses' contracts to model important organisational and legislative constraints from practice. These additional constraints restrict shift assignments within consecutive working days. For example, in the current instances, there is no penalty for a nurse who works a shift sequence such as 'EEDLDE', with E an early shift, D a day shift, and L a late shift. In practice, however, such assignments will be avoided since nurses prefer to work in blocks of identical shifts or in a forward shift rotation. Constraints penalising such assignments have, therefore, been added to the new instances as forbidden patterns.

The new instances and the extended data format for the input files have been made available on a dedicated web page[1]. Furthermore, an evaluator was developed to validate solutions for these new instances. The validator is similar to the INRC's validator, but it also incorporates the debit system for counters and the stepping horizon methodology for evaluating series constraints.

3.8 Conclusions

This chapter highlighted issues in academic models for nurse rostering which arise due to the absence of provisions for taking into account continuity. Several examples illustrated that ignoring assignments from previous periods can result in solutions that are unacceptable in practice. Policies were proposed to manage the evaluation of the two main types of constraints in nurse rostering. Both policies require a stepping horizon to be used such that information from outside the current scheduling period can be used during constraint evaluation.

By analysing a series of computational experiments, the effectiveness of the proposed policies in a stepping horizon approach was demonstrated, and compared with a static horizon approach, which is the standard in academic

[1]`http://gent.cs.kuleuven.be/nurserostering.html`

nurse rostering. The proposed policy for counter constraints reduced the unbalanced workload over a one-year period with up to 50%. The stepping horizon evaluation policy reduced the number of violations for the series constraints with 8%.

Part II

Practice

Introduction to Part II: Practice

Despite many advances in the field of automated nurse rostering, a significant gap continues to exist between academic research on nurse rostering and the implementation of results in practice. Kellog and Walczak [70] discuss several reasons for this gap, of which the chapters in Part II address two: the narrow scope of academic models, and academic research neglecting vital practical issues, specifically regarding the implementation of an automated rostering system in practice.

Chapter 4 introduces a general model for rich nurse rostering problems. After several decades of academic research, there exists a large body of literature detailing various nurse rostering problems. Typically, each study considers a small set of constraints, only relevant to their particular problem. Few state of the art models are capable of capturing the complexity of nurse rostering problems from practice. In Chapter 4, such a feature-rich general model is proposed for rostering problems, which has been implemented in commercial software for automated personnel rostering. Additionally, new benchmark instances are published based on real world data and compliant with the presented model.

Another practical issue concerns the implementation of software for automated nurse rostering. Before being able to use such software, typically, practitioners need to go through an extensive configuration phase. First, various legislative and departmental rules and regulations need to be transferred to constraints which can be interpreted by the algorithm. Second, a priority ordering of these constraints needs to be established, indicating which constraints should be less likely to be violated than others. Chapter 5 introduces a technique for partially automating the configuration phase through automatic weight extraction. The practical applicability of the new approach is evaluated in two case studies.

Chapter 4

A general object model for rich nurse rostering problems

In practice, nurse rostering problems are often too complex to be expressed with the available academic models. These models are not rich enough to represent the variegated nature of real world scenarios, and, therefore, rarely find their way to practical applications. This chapter introduces a general model for practical nurse rostering problems, aiming to facilitate re-use of academic nurse rostering approaches in a real world environment. Several complex problem characteristics are discussed, resulting in an object model capable of handling a diverse set of problems. Furthermore, a novel benchmark dataset based on this model is presented.

This chapter is an adaptation of Smet, P., Bilgin, B., De Causmaecker, P. and Vanden Berghe, G. (2014). Modelling and evaluation issues in nurse rostering. *Annals of Operations Research*, 218 (1), 303-326.

Part of the presented model was also included in the doctoral dissertation of Burak Bilgin [10], as he was responsible for defining many of the core concepts in the model.

4.1 Introduction

Nurse rostering belongs to the most varied personnel rostering problems within the academic literature on timetabling and scheduling [119]. With the majority of papers addressing real problems from hospitals all over the world, the differences in hospital management are quite noticeable.

Nevertheless, currently available academic models often do not cover many of the instances encountered in real rostering environments. This may partially explain why a vast academic effort to cope with such problems has not yet led to widespread application in hospitals [70]. Several extensions need to be added to the standard models in order to address real world instances as accurately as possible. As a result, the current nurse rostering benchmarks used in academic research inadequately represent practical scenarios [11, 15, 61, 121].

Bilgin et al. [11] introduced a model for standardising a set of common elements belonging to a large variety of nurse rostering problems. This chapter builds upon this work, and presents extensions to their model, mainly regarding skills and generic constraint domains. The most important components are described in detail. This chapter does not include mathematical formulations of the components, but rather provides a description along with several examples to illustrate their application. An integer programming formulation of the presented model can be found in Smet et al. [113].

A new benchmark dataset complying with this general model is presented, containing rich instances based on data obtained from two Belgian hospitals. This new dataset enables researchers to verify the performance of their algorithms in highly realistic scenarios.

The rest of this chapter is organised as follows. The main contributions are discussed in Section 4.2. Section 4.3 describes the proposed general model for nurse rostering. A new dataset based on this model is introduced in Section 4.4. Finally, Section 4.5 concludes the chapter.

4.2 Contributions

As many of the academic models make strong abstractions of reality, their impact in practice is often not significant. This chapter introduces a general model which takes into account a large variety of important practical considerations. Algorithms designed to cope with this model would thus be able to solve problems which are much closer to the problems faced in practice.

The presented model has been implemented in a commercial software package for (automatically) constructing and managing personnel rosters[2], and is currently

[2]Plan@SAGA, Tobania

used in hospitals and other organisations in Belgium, France, the Netherlands and Luxembourg.

Furthermore, since the standardised model is capable of representing a large variety of nurse rostering problems, it can support academic research in investigating the general applicability of new techniques. As an initial effort, a new benchmark dataset based on real world data has been made publicly available on a dedicated web page[3], which also allows researchers to validate new solutions.

4.3 Object model

This section details the different elements of the general model. De Causmaecker and Vanden Berghe [38] introduced a notation based on the $\alpha|\beta|\gamma$ notation for scheduling problems, to describe and classify nurse rostering problems based on the presence or absence of specific problem characteristics. Where possible, the concepts discussed in this chapter are situated in this classification scheme. It should be noted that some components contain an amount of detail beyond the $\alpha|\beta|\gamma$ scheme and are not classified accordingly.

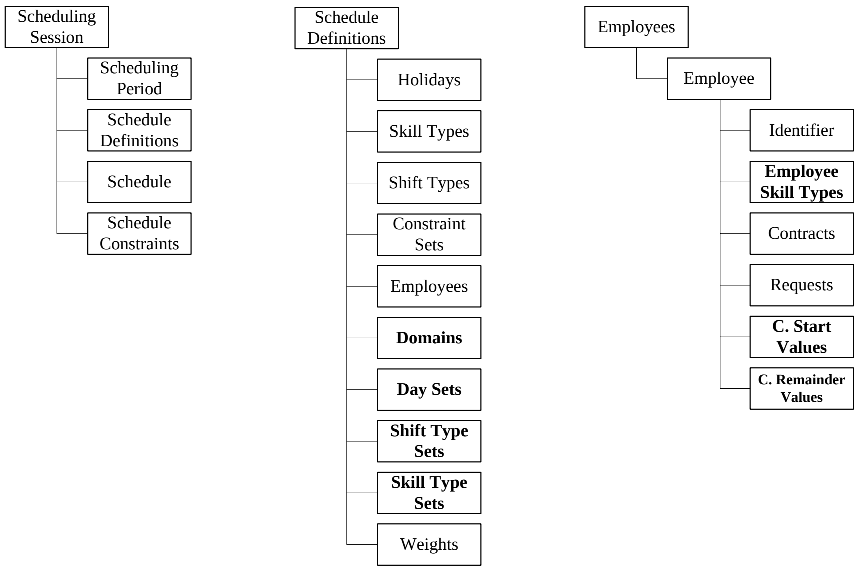
Figure 4.1a shows the rostering model as a tree structure with *scheduling session* as the root element. A scheduling session consists of four elements: *scheduling period*, *schedule*, *schedule constraints* and *schedule definitions*.

4.3.1 Scheduling period

The scheduling period in personnel rostering problems varies among different sectors, countries and times of the year. In Belgian hospitals, scheduling periods are typically four weeks, or one month. Some periods, such as Christmas and the summer holidays, require more attention than the rest of the year due to fewer available personnel. Consequently, in some hospital wards, the scheduling period is limited to two weeks during these seasons.

In order to address these variations, the definition of the scheduling period in a personnel rostering model must be flexible. Therefore, the scheduling period element is defined with a *start date* and an *end date*.

[3]`http://gent.cs.kuleuven.be/nurserostering.html`



(a) Scheduling session (b) Schedule definitions (c) Employees

Figure 4.1: Main elements of the model

4.3.2 Schedule definitions

The schedule definitions element defines the core concepts of any personnel rostering problem instance (Figure 4.1b). Elements in bold refer to additions with respect to the model of Bilgin et al. [11]. The following paragraphs detail these concepts.

Skill types

Skill types define the levels of qualification of employees, e.g. in health care, common skill types include head nurse, regular nurse and caregiver. In addition, skills that require specific training, such as ophthalmology or ambulance driver, also require modelling. In the general model, the *skill types* element is a collection of unique identifiers referring to these skills.

Shift types

Shift types refer to the daily assignment units. In the presented model, neither the number, nor structure of shift types is fixed.

Each shift is uniquely defined under the *shift types* element. Each *shift type* is defined by a specific *start time* and *end time*. Furthermore, a required *rest period before* and *after* is defined for each shift individually. Finally, due to breaks, the actual job time does not always equal the duration of the shift type. An additional element allows to specify the *net job time*, which is an important consideration for the evaluation of constraints related to work time.

In the $\alpha|\beta|\gamma$ notation, the general model covers problems with a variable number of shifts (N) and with overlapping shifts (O).

Constraint sets

A *constraint set* is composed of a unique identifier and a set of constraints corresponding to an employee's contract. These constraints apply restrictions to the schedule of an employee. They are called time related constraints in Burke et al. [23] and correspond to availabilities (A) and sequences (S) in the $\alpha|\beta|\gamma$ notation.

Constraints are modelled around three general types: *counters*, *series* and *successive series* [11]. In real world problem instances, they are usually considered soft constraints by defining a weight to express the relative importance of each constraint. This allows the model to represent rostering problems with personnel regulations constraints (P) as optimisation objective in the $\alpha|\beta|\gamma$ notation.

The three constraint types use a variety of specific subjects and parameters to cover a diverse range of employment constraints encountered in real world problems.

- Counters

  This constraint type limits the number of specific roster items in a certain period, defined by a start date and a number of days. This *counter period* does not need to match the scheduling period. Seven subjects can be restricted by counters: *hours worked*, *shift types worked*, *days worked*, *days idle*, *weekends worked*, *weekends idle* and *domain*.

  Some of these constraints are further parametrised through *domains* (Section 4.3.4). For example, *shift types worked counters* have a *shift type* *set* as a parameter to indicate which shift assignments are constrained. Besides *domain*, *weekends worked* and *weekends idle*, all counters have the *day types* parameter to specify on which days the counter is active, e.g. all days, only holidays, a combination of week and weekend days, etc.

- Series

  This constraint type restricts consecutive occurrences of specific roster items. They are defined for five subjects: *shift types worked*, *days worked*, *days idle*, *weekends worked* and *weekends idle*. Similar to *counters*, *shift types worked series* are also further parametrised through a *shift type set*.

- Successive series

  This constraint type restricts two consecutive series: if the first series appears in a roster, it must be immediately succeeded by the second series. Five successions of series are considered: *days worked* $\rightarrow$ *days idle*, *days idle* $\rightarrow$ *days worked*, *shift types worked* $\rightarrow$ *days idle*, *days idle* $\rightarrow$ *shift types worked* and *shift types worked* $\rightarrow$ *shift types worked*.

Employees

Nurse rostering problems encountered in hospitals are typically very diverse in terms of qualifications, contracts and requests of each employee. The presented model allows for this diversity by defining each employee with specific *employee skill types*, a number of employment *contracts* and *requests* (Figure 4.1c).

- Employee skill types

  The attributes related to the skill types of an employee are expressed under the employee skill type elements. This element varies according to the type of problem. Section 4.3.3 discusses four cases.

- Contracts

  A contract element has a *start date*, *end date* and an identifier referring to a constraint set. Each employee has one, or several non-overlapping contracts. The latter occurs when a new contract starts and an old contract ends within the same scheduling period. Contracts can be defined for each employee separately, thereby allowing a high degree of individuality.

- Requests

  Employees can express *assignment* and *absence requests*, both treated as soft constraints in the presented model. Assignment requests are expressed using domain counters. Absence requests are expressed as separate elements under *requests*. The reason for this is an additional element in absence requests, the *job time*, whose value is added to the *hours worked counter* values of the employee. This can be used to model the paid holidays of the employee.

To correctly evaluate constraints at the boundaries of the scheduling period, additional information is necessary regarding previous and future assignments (see also Chapter 3). This information is defined under the *counter start values* and *counter remainder values* elements.

- Counter start values

  Counter periods may exceed the scheduling period [29]. For example, a counter can be defined for one year, but the scheduling period is defined as four weeks. In this case, the value of the counter up until the current scheduling period is specified as the counter start value.

- Counter remainder values

  If the counter period exceeds the scheduling period, the number of the corresponding roster elements in the remainder of the counter period is called the counter remainder value. For example, if the period of a *weekends worked counter* is one year and the scheduling period is four weeks, the number of remaining weekends after the current scheduling period is given as the counter remainder value.

4.3.3 Variations in skill type definitions

Skill type structures can vary significantly between organisations, thereby also impacting the modelling requirements regarding employee skill types. This section considers four of these structures, and shows how the model copes with them through the flexible definition of employee skill types. Following the $\alpha|\beta|\gamma$ notation, both a variable number of skills (N) as well as individual skill definitions (I) can be modelled. Figure 4.2 shows an overview of how different cases are represented in the model. The elements in bold refer to new elements to the model of Bilgin et al. [11].

One skill type in the problem

In this type of problem, all employees have the same skill type. As a result, the skill type and employee skill type elements are not needed in the model. Note that this type of problem almost never occurs in practice.

Single skill type per employee

In this category, all employees have a single skill type, but there are multiple types overall. The required skill type must be specified in the coverage requirements, but not in the assignments since the skill type information can be retrieved unambiguously (Figure 4.2a). By grouping employees with the same skill types, these problems can be decomposed into instances of the previous category.

Multiple skill types per employee

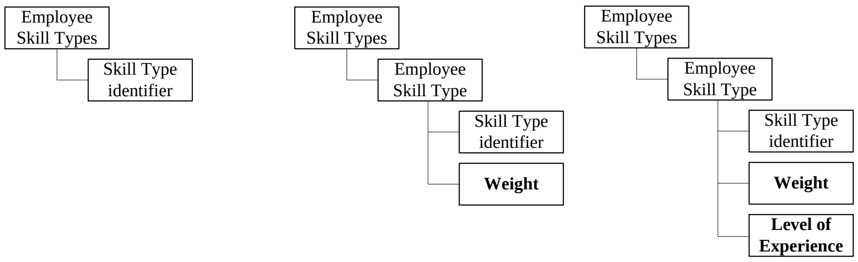
Employees typically have more than one skill type. For example, in some nurse rostering models, skill types are organised hierarchically such that nurses higher in the hierarchy are allowed to substitute nurses at a lower level. Substitutions of a higher ranked nurse by a lower ranked nurse are often not allowed.

In general, the relevance of individual skill types for an employee with multiple skills varies. In cases where a nurse has multiple skills and one is more expensive for the organisation, it is less preferred to assign this nurse to the more menial, but equally paid, role. The opposite can also occur, representing the financial promotion of a lower level nurse to a higher role. Therefore, simply listing the skill types of a nurse in the employment definition does not express the problem sufficiently accurately.

An element consisting of the skill type identifier and a weight representing the relevance of the skill is required (Figure 4.2b). The relevance is accounted for by multiplying the number of assignments with this skill type by the weight and adding the total to the total penalty of the solution.

Different levels of experience for each skill type of the employees

In some working environments, employees have different levels of experience for each skill type. Consider, for example, a senior regular nurse who is in training for the role of head nurse. In this case, the titles ‘regular’ and ‘head nurse’ refer to the skill types, and the titles ‘senior’ and ‘trainee’ refer to the level of experience. In the presented model, this level of experience can be defined using integer numbers, thereby not limiting the number of different levels. For each employee skill type, the level of experience is defined along with the weight (Figure 4.2c).



(a) Case 1 (b) Case 2 (c) Case 3

Figure 4.2: Employee skill type elements for different categories regarding the skill type properties

4.3.4 Domains

In addition to the basic modelling concepts, *domains* are introduced as modelling elements that considerably increase the expressiveness of constraints (Figure 4.3). Note that domains have a different meaning here than the domain constraints discussed in Chapter 2.

Basic domains

- Day sets

  In the problem model, some constraints can be defined for a subset of days. A *day set* consists of a unique identifier and either a *date set* or combination of *day types* (Figure 4.3a). A day type can be defined in various ways, e.g. all holidays or particular weekdays (Figure 4.3b). A date set contains specific dates in the scheduling period, and is given a *handling mode*, which provides essential evaluation information (Figure 4.3c). The handling mode can be either *individual* or as a *block*, depending on how violations should be calculated during evaluation.

- Shift type sets

  In practice, certain constraints involve a set of shift types rather than a single shift type. By allowing more than one shift type as a constraint parameter this problem is addressed. Such *shift type sets* are defined under schedule definitions, and are referenced to by their unique identifiers (Figure 4.3d). For example, a constraint can restrict the number of night shifts assigned to a nurse. However, some wards might have more than

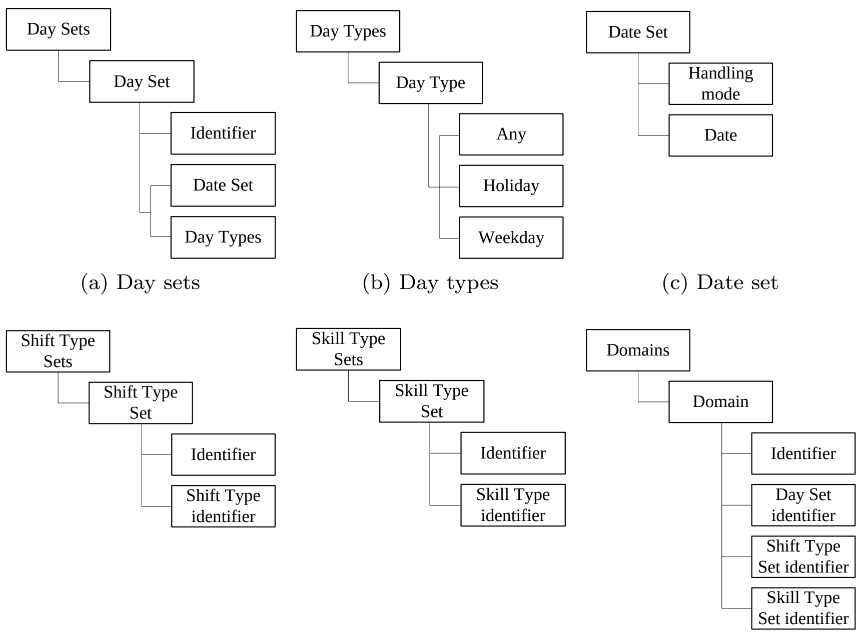


Figure 4.3: Elements used for domain definitions

one type of night shift, e.g. distinguished by duration. In this case, the different night shifts can be combined in a shift type set.

- Skill type sets

  Similar to shift type sets, some constraints are defined for a *skill type set*, rather than for a single skill type (Figure 4.3e).

Domain elements

Apart from the aforementioned basic domains, more complex definitions of domains may be achieved by using *domain* elements. A domain element combines a day set, shift type set and skill type set to specifically identify which part of the schedule a constraint applies to (Figure 4.3f). The domain elements are used by the *domain counter*, *absence request*, *collaboration* and *training* constraints. A domain can thus be considered as an abstract element which is only relevant when used as a parameter in a constraint.

In the following, five examples of employee requests are presented to demonstrate the application of domains as parameters in constraints. Additionally, an example is included of how to use domains to define individual holidays.

- *Absence request on individual holidays.* Suppose an employee applies for a five day absence for home redecoration. This request does not need to be granted in its entirety, but the closer to five days the better for the employee. Here, the date set constitutes the five days being requested, the shift and skill types are all those defined in the problem instance. Note that the days in the date set are handled individually. The job time is the job duration for each granted day which is added to the total working time of the requesting employee.
- *Absence request as a block.* Consider the previous example, but with the reason for absence now being a holiday. The request is fully granted only if all five days are granted. Here the penalty is a fixed value (the weight of the constraint) which is added to the total roster penalty upon the rejection of a request. The date set is considered the block of five days and the shift and skill type sets are defined as before.
- *Absence request on Wednesday afternoons.* Since most schools in Belgium have free Wednesday afternoons, many employees prefer to be at home at this time for their children. This absence request emphasises the domain element's shift type set, which consists of shift types that overlap with the afternoon. The date set consists of Wednesdays and the dates are handled individually. The skill type set is defined as before.
- *Assignment request for a specific skill type.* Employees can increase their level of experience in certain skill types by utilising that particular skill type often. Imagine a senior caregiver who is also a trainee regular nurse. To reach junior level she must work a number of assignments as a regular nurse. In this case the skill type set consists of the 'regular nurse' skill, the date set represents all the dates in the scheduling period which are handled individually as in the first example, and the shift type set again consists of all shift types defined in the problem.
- *Individual holidays.* Employment legislation decrees that each employee is entitled to a specific number of official holidays per year, making it difficult to satisfy coverage requirements on the holidays. However, certain religious holidays are not relevant to all employees, thus it is beneficial for the institution to define an individual set of holidays for each religion. The traditional way to handle the holidays worked constraint is to define an individual counter for each employee which refers to globally defined official holidays. Alternatively, domains can be defined such that each

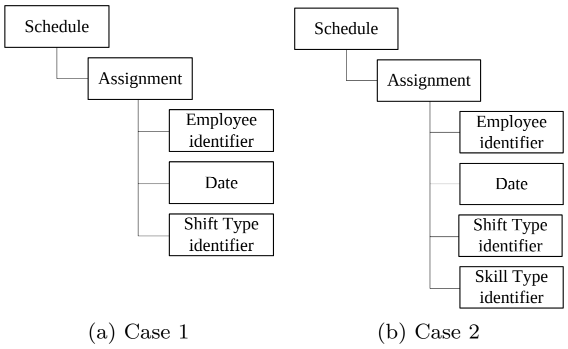


Figure 4.4: Schedule element for different cases regarding the skill type properties

employee has an individual domain representative of their relevant holidays. A domain counter can then limit the number of holidays worked.

4.3.5 Schedule

The goal of the personnel rostering problem is to generate a schedule, i.e. a set of assignments. In the model, an assignment consists of four elements: an employee, a date, a shift type and a skill type.

Contrary to many academic resources on nurse rostering [84, 104], the presented model considers the skill type as part of the assignment. If a nurse has more than one skill type, the skill type that he/she uses for the assignment would be ambiguous unless specified [24]. Figure 4.4 shows the *schedule* element for two cases: one in which each employee only has one skill (Figure 4.4a), and one where employees have multiple skills (Figure 4.4b). The latter requires the used skill to be explicitly part of the assignment.

Some of the time related constraints overlap two scheduling periods, e.g. series constraints. The assignments in the overlapping part of the previous scheduling period must be given as input to ensure an accurate constraint evaluation. Chapter 3 further elaborates on the importance of providing this additional data to ensure consistent constraint evaluation at the boundaries of the scheduling period.

4.3.6 Schedule constraints

The *schedule constraints* element contains four main concepts that further constrain the solution: *coverage constraints*, *schedule locks*, *collaboration constraints* and *training constraints*.

Coverage constraints

The number of employees needed daily for each shift and skill type is expressed using coverage constraints (Figure 4.5). Often, the prime objective of a rostering system is to fulfil these coverage constraints. In the $\alpha|\beta|\gamma$ notation, the proposed model allows for modelling problems that set coverage constraints as determined (D), range (R) and fluctuating (V). The model allows for each coverage constraint to be assigned a weight, enabling modelling problems with a load and coverage constraint objective (L) as part of their optimisation objective.

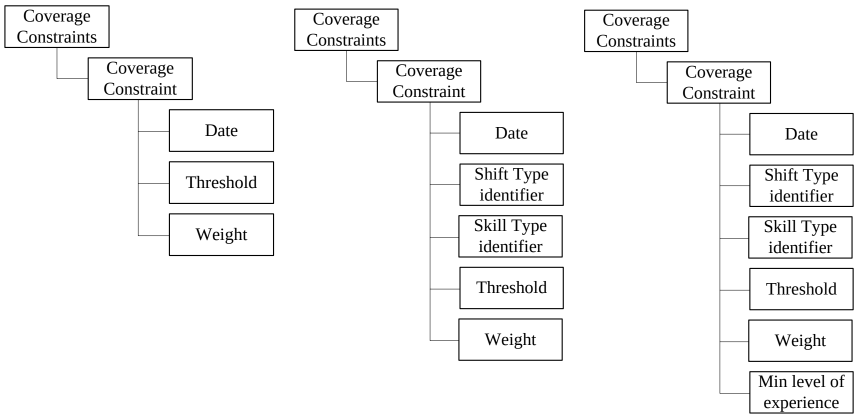
Every type of problem requires a *date* and *threshold* for each defined coverage constraint (Figure 4.5a). The latter is used to specify the minimum and/or maximum required number of nurses. Depending on the skill and shift type properties of a problem instance, additional information may be required. If the problem instance has multiple skill or shift types, then the information must also be provided in the definition (Figure 4.5b). A *minimum level of experience* must be specified in cases where experience levels are associated with skill types (Figure 4.5c). Any employee satisfying the requirements will count towards satisfying the coverage constraint.

Schedule locks

Planners sometimes manually construct a partial schedule. Typically, changes in these parts of the schedule by an algorithm are not allowed. In the presented model, parts that are not allowed to be modified are identified with schedule locks. A schedule lock is defined by an employee and a date.

Collaboration

Situations arise in working environments where specific groups of employees are required to work together due to complementary skills. However, the opposite is also true, e.g. when family members request to work different shifts so that at least one of them can take care of their children. Collaboration constraints are used to model such scenarios (Figure 4.6a). In the $\alpha|\beta|\gamma$ notation, this



(a) Case 1 (b) Case 2 (c) Case 3

Figure 4.5: Coverage constraints

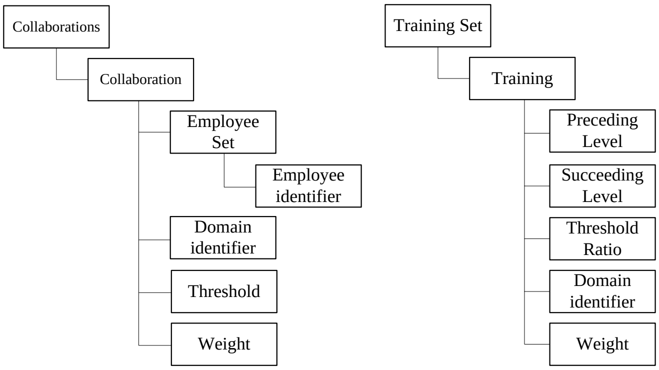
constraint is referred to as chaperoning (C). The collaborating employees are expressed as an *employee set* containing the identifiers of the relevant employees. Note that the number of employees to collaborate is not limited to two.

The scope of collaboration constraints can be restricted using domains. Collaboration constraints can be required only on specific days, for specific shifts and skill types. Finally, the *threshold* element defines the type of collaboration. If the employees are to not work together, then the maximum threshold must be set to zero, and if n employees should work together, then the minimum threshold must be set to n.

Training

In an environment with multiple levels of experience, it is desirable that experienced employees work together with those less experienced to train them. A training constraint restricts the ratio between the numbers of assigned employees with different levels of experience (Figure 4.6b). In the $\alpha|\beta|\gamma$ notation, this constraint also corresponds to chaperoning (C).

Consider the following administrative guideline as an example: *at least one senior caregiver should be assigned for every five junior caregivers.* In this case, a training constraint is defined where the *preceding level* is junior and the *succeeding level* is senior. The *threshold ratio* is 0.2 and the *domain identifier*



(a) Collaboration constraints (b) Training constraints

Figure 4.6: Collaboration and training constraints

refers to a domain consisting of all days and shift types but restricted to caregivers.

This way, the training constraint is defined as a one-way relationship. If a two-way relationship is required, the complementary constraint must be defined as well. The following rule is complementary to the example mentioned above: *at least five junior caregivers should be assigned for each senior caregiver assigned.* In this case, the *preceding level* is senior and the *succeeding level* is junior. The *threshold ratio* is five, while the *domain identifier* remains the same.

4.4 New benchmark dataset

Based on the proposed model, a new benchmark dataset is introduced allowing researchers to evaluate the performance of their algorithms on nurse rostering problems taken from practice. These benchmark problems are based on data collected from six wards in two Belgian hospitals. Table 4.1 shows an overview of some general characteristics of the instances.

For each ward, three scenarios are considered. The *normal* scenario represents regular working conditions with average coverage requirements as well as the standard availability of nurses. The *overload* scenario simulates a situation in which higher coverage requirements are to be met by the regular staff. This can happen in situations where the hospital is required to treat a larger number of patients than normal, e.g. in case of an epidemic. Finally, in the *absence* scenario, a nurse is absent for some time during the scheduling period.

Instance	Employees	Shift types	Skill types	Days
Emergency	27	27	4	28
Psychiatry	19	14	3	31
Reception	19	19	4	42
Meal Preparation	32	9	2	31
Geriatrics	21	9	2	28
Palliative Care	27	23	4	91

Table 4.1: General characteristics of the problem instances

Instance	Skill 1	Skill 2	Skill 3	Skill 4
Emergency	1	15	4	26
Psychiatry	1	17	1	-
Reception	1	1	3	15
Meal Preparation	1	31	-	-
Geriatrics	4	20	-	-
Palliative Care	1	21	4	1

Table 4.2: Skill characteristics of the problem instances

As was discussed on page 65, the presented model includes multi-skilled nurses. Table 4.2 shows how many nurses are qualified for each skill in each instance.

The instances, as well as an XSD of the model, have been made publicly available on a dedicated web page[4]. Current best known results are obtained by the metaheuristic algorithms presented by Smet et al. [112].

4.5 Conclusions

Academic models often fail to consider vital real world aspects of rostering, and therefore, their implementation in practice becomes difficult. The present chapter introduced a general model for feature-rich nurse rostering problems, conceived so that many complex staff-related aspects can be covered, specifically regarding contracts and qualifications. The flexibility of the model is important as it allows representing rostering problems originating from organisations with few regulations, as well as problems with a high degree of variability among the employees and complex time related constraints.

[4]http://gent.cs.kuleuven.be/nurserostering.html

The model has been implemented in a commercial software package for automated personnel rostering and management, and is currently used in hospitals and other organisations in Belgium, the Netherlands, France and Luxembourg.

Complying with the model, a new benchmark dataset was generated, derived from real hospital data. The new instances are publicly available to researchers wishing to evaluate the performance of their algorithms on instances closely resembling real world scenarios.

Chapter 5

Facilitating the transition from manual to automated nurse rostering

After several decades of academic research in the field of nurse rostering, few results find their way into practice. One reason for this is that often, the configuration of a software system for automated rostering is considered too time-consuming and difficult. This chapter introduces an approach for automating part of the costly and unintuitive configuration process by automatically determining the relative importance of soft constraints based on historical data. The approach is evaluated in two case studies and is validated with the help of health care practitioners. The results show that, given relevant historical data, the presented approach simplifies the transition from manual to automated rostering, thus bringing academic research on nurse rostering closer to its practical application.

The content of this chapter is based on joint work with Mihail Mihaylov, Vrije Universiteit Brussel and Wim Van Den Noortgate, KU Leuven.

5.1 Introduction

Despite the advancements of automated nurse rostering techniques [25], scheduling by hand is still standard practice in many hospitals. An important cause of the research-application gap, as identified by Kellogg and Walczak [70], concerns the transition from manual to automated rostering. Many hospitals consider the implementation of an automated rostering approach to be too costly or too difficult. While the nurse rostering literature focuses mainly on automatically constructing rosters, it rarely addresses the implications and difficulties of actually implementing an automated rostering technique in a hospital environment. Expert knowledge is needed for defining all departmental rules, hospital regulations and staff preferences, as well as their relative importance, in an automated rostering system. Configuring the software is time-consuming, as it involves translating skills and expertise in numerical form, which is not intuitive, even for experienced planners. Many health care practitioners see this time investment as a deterrent to the use of automated nurse rostering software and prefer to continue rostering manually.

In modern hospitals, much of the domain knowledge needed to configure a software system for rostering is already available in the form of historical data. This chapter explores the possibility of using this data for partially automating the configuration of a rostering system in order to facilitate the transition from manual to automated nurse rostering. A methodology is proposed for extracting constraint importance, in the form of weights, from past rosters. The approach has been named Automated Weight Extraction (AWE). This method has been empirically evaluated on real world data and with the help of health care practitioners in two case studies concerning Belgian hospitals. The results show that, given an appropriate set of historical data, the proposed AWE method is capable of automatically extracting weights suitable for automatically producing rosters that are usable in practice.

The remainder of this chapter is structured as follows. The main contributions are summarised in Section 5.2. Section 5.3 outlines the difference between the focus of the current academic approach to rostering and the actual approach in health care. The rostering task is classified according to the level of automation in its two main phases. Section 5.4 presents the proposed AWE approach in detail. In Sections 5.5 and 5.6, data from two case studies from practice are studied and AWE is empirically evaluated on hospital data and with the help of health care practitioners. Finally, Sections 5.7 and 5.8 conclude the chapter.

5.2 Contributions

This chapter introduces a methodology for automatically determining the relative priority of constraints in automated nurse rostering. This contribution effectively defines a new rostering technique which alleviates practitioners from the unintuitive and error-prone task of manually defining weights. In two case studies, the proposed approach is evaluated on real world data and with the help of head nurses responsible for creating rosters.

Software vendors are commonly faced with the issue of setting appropriate weights when configuring software for automated nurse rostering. This contribution offers a transparent and easy to understand methodology, which strongly facilitates the configuration process. The AWE approach, introduced in this chapter, has been implemented in a commercial software package for personnel rostering[5], currently in use in hospitals and other organisations in Europe.

5.3 Nurse rostering: theory vs. practice

Due to the large variety of guidelines and regulations, constructing rosters in health care presents a particularly challenging task compared to other personnel scheduling domains. Hospitals operate around the clock, introducing specific constraints on rest times and on night and weekend shifts. The problem is further complicated by a large range of shift types and skill requirements. The scheduling period typically spans a few weeks to a few months, with staff requirements varying each day.

The present contribution focuses on departmental rostering, where the head nurse of the ward is responsible for creating the rosters. This type of rostering is also applied in the hospitals considered in the case studies. Team rostering and self rostering are two other rostering techniques where decision making is delegated to groups or to individual nurses, respectively. These techniques increase the perception of autonomy and reduce the rostering effort of the head nurse. They are restricted to small and medium-sized wards [111].

The academic literature on nurse rostering focuses mainly on optimising rosters of health care personnel. However, in practice, practitioners still need to invest time and effort transitioning to these automated solutions from traditional manual planning. In the following subsections, a brief overview of nurse rostering is presented, both from an academic and a practitioners' point of view.

[5]Plan@SAGA, Tobania

5.3.1 Rostering in the academic literature

Nurse rostering is typically formulated as a constrained optimisation problem, attempting to satisfy a number of soft constraints [113]. Feasible solutions need to respect all hard constraints induced by resource restrictions, departmental requirements and contractual obligations. The quality of a feasible solution (or *penalty*) is measured by an objective function and is related to the number of respected soft constraints concerning staff and contractual preferences. The large number of soft constraints, often mutually conflicting, makes it practically impossible to satisfy all preferences. The objective of nurse rostering problems is to minimise the constraint violations. Such optimisation problems are thus multi-objective in nature. Ideally, the planner would have to select from a set of Pareto-optimal solutions. Here, a Pareto-optimal solution is one where no constraint violation can be reduced, without increasing another one.

Multi-objective optimisation is often addressed by employing a weighted sum objective function due to its simplicity and ease of implementation. This method scalarises the vector of objectives into a single objective value by summing all violated soft constraints, weighted by their relative importance.

Many nurse rostering approaches in the literature utilise weighted sum objective functions and define their constraint weights with the help of health care practitioners [1, 4, 20]. Others simply set the weights by trial-and-error, without elaborating on the choice of values or on their effect on the overall quality of generated schedules [4, 5]. A combination of these two settings also exists, where health care practitioners define only a general preference ordering of constraints and then researchers choose the numerical values according to some rule [9, 26]. These manual methods of determining constraint weights are time-consuming. Practitioners need to be consulted, or a large sample of tests needs to be performed. In addition, there are no explicit guidelines for setting constraint weights, whereas setting them poorly induces biased solutions. Research has indicated that setting static weights may result in a solution landscape that is difficult to explore [101]. Therefore, different methods have been proposed for varying constraint weights, such as SAWing [47], Noising [32] and other adaptive weight methods [69]. However, these methods rely on additional parameters that still need to be tuned manually.

Drawbacks of the weighted sum objective function are that constraint weights are subjective, need to be explicitly defined and that the final solution is sensitive to the chosen values. Numerous other approaches to multi-objective optimisation exist, such as the ϵ-Constraint method [31], compromise programming [6] and evolutionary algorithms [34]. Most of these approaches maintain multiple Pareto-optimal solutions and select a single solution based on some high level problem

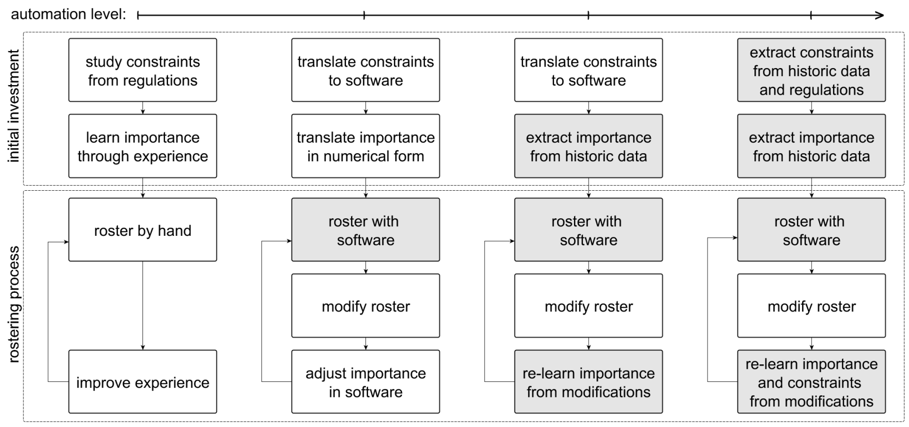

automation level:
initial investment
rostering process
study constraints from regulations
learn importance through experience
roster by hand
improve experience
translate constraints to software
translate importance in numerical form
roster with software
modify roster
adjust importance in software
translate constraints to software
extract importance from historic data
roster with software
modify roster
re-learn importance from modifications
extract constraints from historic data and regulations
extract importance from historic data
roster with software
modify roster
re-learn importance and constraints from modifications

Figure 5.1: Overview of different levels of automation of rostering practices and their two main phases. Non-shaded boxes show manual steps, grey boxes indicate automated ones

information, input by the user. Nevertheless, the weighted sum objective function remains the most preferred multi-objective optimisation method. It is used in the majority of the academic nurse rostering literature [25], as well as in practical applications [1, 50, 93].

5.3.2 Rostering by health care practitioners

In practice, the rostering task encompasses two phases: the initial investment and the rostering process. Academic research mainly focuses on the rostering process, while in practice, the preceding configuration phase is equally important. This first phase is called the initial investment as the hospital needs to invest resources into translating departmental rules and regulations to software constraints and defining their relative importance. Previous academic research has overlooked this stage, under the assumption that it offers little potential for automated improvement. However, it constitutes an important, time-consuming phase in the transition from manual to automated nurse rostering that needs to be appropriately addressed.

Rostering type and transition	Fully manual rostering	Automated rostering, manual transition	Automated rostering, semi-automated transition	Automated rostering, fully automated transition
Performed by	Health care practice	Most academic research and some health care practice	This chapter	Future trends
Benefits	+ considers implicit constraints + answers specific requests	+ saves time in rostering + higher departmental control	+ easier transition to automated + reduced time investment	+ minimal time investment + no human bias
Drawbacks	- labour intensive - requires experience - costs the department	- requires experience - unintuitive to translate experience - biased results	- requires data - difficulty handling transient effects - not suitable after drastic restructuring	- requires data - difficulty handling transient effects - may not capture complex constraints

Table 5.1: Comparison between different levels of automation in nurse rostering and the transition from manual to automated rostering

5.3.3 Automation of rostering practices

Four different rostering practices are identified according to the level of automation of the initial investment and rostering process. These practices are shown in Figure 5.1, where tasks in non-shaded boxes represent work performed by hand, while the grey boxes indicate automated tasks. Table 5.1 shows both benefits and drawbacks of the different approaches. In the following subsections, each of these automation levels is explained, situating the present contribution in the field, and comparing it to academic research, current practice and future trends.

Fully manual rostering

Many hospitals conduct labour-intensive manual rostering as the transition to automated rostering is considered costly and unintuitive. Even though training personnel for manual rostering is equally costly and may sometimes be more expensive, it follows long-established and implicit procedures, which practitioners are reluctant to replace. Manual planners need to study hospital regulations and departmental constraints before they are able to construct rosters. They need to also learn, through experience, the relative importance of these rules, in order to minimise violations of important constraints.

Departmental rostering places the responsibility for designing the rosters with the head nurse, who has the often unrewarding task of balancing departmental regulations and staff preferences. Nevertheless, manual planning allows head nurses to tailor rosters to the particular needs of the department based on their experience and familiarity with the personal qualities of the staff. For example, a head nurse may avoid the assignment of two nurses who do not get along well to the same time period, assign shifts based on personal requests, or assign fewer nurses than required due to a temporary drop in patient admissions. Although handling such implicit constraints may improve staff contentment, it often comes at the expense of the overall organisation, which has little control over the final rosters.

Automated rostering with manual transition

Hospitals have invested in automated rostering software, partially automating the labour-intensive manual rostering while tightening central control [1, 50]. The initial investment enables automated rostering by allowing different automatically generated solutions to be evaluated and compared with the weighted sum objective function. However, as this is only an approximation of the real (perceived) roster quality, staff members often modify the final solution to correct any perceived faults or to incorporate additional implicit constraints and personal requests.

With no explicit guidelines for translating the importance of constraints, practitioners need to rely on their own experience and intuition when configuring the numerical weights. The manual configuration of the rostering software becomes a costly trial-and-error process, where human experts attempt to correct the search by adjusting constraint importance on the basis of the obtained solutions. This iterative adjustment is bound to result in a vicious cycle, as neither the health care practitioners understand the search process, nor can the software interpret the expert's rationale to deliver satisfactory results.

Automated rostering with semi-automated transition

The two aforementioned rostering practices involve manual work by experienced planners and require a costly initial investment phase. Defining constraint weights manually largely depends on the skills of practitioners to translate their expertise into quantitative data, while full manual rostering relies entirely on the experience of the human planner.

Many hospitals aim to increase the rostering process's automation level by relying on computerised solutions. Such solutions are only as good as the domain expertise they contain. In order to incorporate domain expertise into computerised solutions, either humans need to translate their expertise in a language interpretable by a computer, or the machine needs to extract the human expertise. Automated rostering with manual transition addresses the human-computer knowledge gap from the human side first by letting practitioners explicitly translate their knowledge into the language of the software planner.

This chapter proposes a new rostering practice where the knowledge gap is tackled from both sides simultaneously. Human experts only define the variables of the objective function, i.e. the constraints, while the machine automatically extracts their relative importance (the weights) from past data. The transition from manual rostering to automated rostering thus becomes semi-automated. The efforts and costs of the transition are partially performed by the machine. This technique reduces the initial investment by supporting the user in configuring the automated rostering software.

In addition to partially automating the initial investment phase, the proposed technique excludes the need of manual re-adjustment of constraint weights in the rostering phase. The constraint weights are updated in a way similar to the initial phase, namely by examining recent rosters and adjusting the importance of constraints based on the changes introduced by the human planner. Automating the re-adjustment step lowers human bias in the final solution, but has a potential difficulty handling transient effects, such as illness of a staff member or a sudden peak in patient admissions. Another problem is that drastic long-term changes, such as re-structuring or merging of wards, may have a negative influence on the stored domain knowledge and may require re-learning.

Automated rostering with fully automated transition

A natural extension to the proposed rostering practice would be to entirely automate the initial investment phase by extracting not only constraint importance, but also the constraints themselves. The human-computer knowledge gap would then be addressed entirely from the computer side, as the initial investment phase is performed exclusively by the machine. This rostering methodology would allow any manually rostering ward with relevant historical data to effortlessly transition to automated rostering.

Extraction of constraints can be achieved in two distinct ways that can later be combined for improved accuracy. Natural language processing techniques and text mining can be applied to extract information regarding constraints from written hospital regulations and departmental rules. Another method to extract information is to perform pattern detection on historical data using a general knowledge base. This database of constraints can be collected and updated by wards, and can even be shared by hospitals, so that automated methods can detect whether a subset of these constraints has been respected in past (manually constructed) rosters. Once constraints are extracted, their importance can be determined through automated weight extraction. As with the above rostering practice, though, automated knowledge extraction may incorporate transient effects and learn constraints induced by temporary roster modifications.

5.4 Automated constraint weight extraction

The remainder of this chapter focuses on automatically defining the importance of constraints in the initial investment phase. The assumptions here are that sufficient past rosters are available and that their quality is satisfactory to the department and its personnel. In addition, all necessary constraints have been input into the automated rostering software. Although the latter represents a manual task, the process of adding constraints is rather straightforward, as hospital regulations and departmental constraints are often explicitly stated in legal documents. Constraint importance, on the other hand, is implicit and takes experience to learn and translate into numerical weights. The past schedules and constraints need to be available in a structured way. An example is represented by the object model presented in Chapter 4.

The proposed AWE approach works as follows. For each constraint, the number of violations is counted in all rosters, as well as the number of times the constraint is respected. The sum of these two numbers indicates the instances of the constraint, i.e. the number of times the constraint can be identified in all rosters. The new (or automated) weights are defined to be proportional to the ratio of the number of times the corresponding constraint is respected in the past schedules versus the total number of instances of that constraint. Formally, the weight w_c^{new} of each soft constraint c is defined as follows:

$$w_c^{new} = \frac{(n_c - v_c)}{n_c} \tag{5.1}$$

where n_c is the total number of instances of constraint c in all rosters, and v_c is the number of violations of this constraint. The intuition behind this choice is that constraints with a high percentage of violations are less important and should therefore be given low weights. Even though violated constraints might actually be important, they may be difficult (or impossible) to satisfy, e.g. due to conflicts with other constraints. Associating very high weights with such constraints may distort the search process and consume computational resources in vain. Analogously, constraints that are mostly respected in the historical data are either important, or easy to satisfy, and are given a high weight.

It should be stressed that the proposed technique simply extracts information from past data. For this reason, an extracted weight configuration cannot steer an algorithm to generate more preferable schedules than the manual ones, but rather schedules that conform to the historical data. The main advantage of the proposed method is that the obtained weights enable automated configuration of the scheduling software and do not rely on the skills of the practitioner to quantify his/her expertise. Moreover, it allows any department with sufficient historical data to switch to automated rostering by allocating only limited resources during the initial investment phase, as the most time-consuming task is automated. Various machine learning techniques may be applied to extract importance values from manual schedules, or to determine an ordering of the constraints, such as preference learning through regression or classification [54]. However, the proposed method is intuitive, transparent, computationally inexpensive and above all, simple to explain to health care practitioners without a mathematical background.

5.5 A retrospective case study

The weight extraction method was developed when analysing historical data from a Belgian hospital that has already transitioned from fully manual to automated rostering. The initial investment phase in all wards was performed fully manually. Hospital regulations and departmental rules were translated to constraints in the automated planner, and their importance was set based on the intuition of the head nurse of the respective ward. In this section, the following rostering practices are analysed: fully manual rostering, automated rostering with manual transition, and automated rostering with semi-automated transition. Specifically, the weights and constraint violations in different solutions are compared.

5.5.1 Case description

This case study concerns the reception ward of Mariaziekenhuis Noord-Limburg, a modern Belgian hospital with 350 beds. The ward is medium-sized with up to 21 staff members, each working one of three contract types: full time, $\frac{3}{4}$ or another part time. The employees are further categorised according to whether they have telephoning skills or not. Coverage requirements are specified for different types of early, day and late shifts. In addition to the contractually required number of hours worked each month, hospital-wide and ward-specific constraints are specified in the employees' contracts. The former are concerned with providing sufficient rest time — 35 hours rest per week for each employee, and adequate time between two consecutive working days. The ward-specific constraints restrict certain shift types to be assigned to certain employees (e.g. 4h or 6h shifts should not be assigned to full time employees), and ensure a minimum number of particular shifts per employee. Furthermore, the ward-specific constraints state that an employee should never work two consecutive weekends. Finally, a shift may induce a subsequent number of days off. In total, each roster needs to consider 100 soft constraints of varying importance.

5.5.2 Study design

The hospital provided data in the form of three years of past one-month rosters. These rosters had been constructed manually by the head nurse and served as an initial assignment for the ward before corrections were applied due to illness, unavailability of personnel or reduced workload. These corrections were not included in the analysis, as they are incidental and not representative of the ward's operation.

Three sets of one-month rosters were used. The first set consisted of the rosters manually constructed by the head nurse without the support of the automated rostering software. This data set is referred to as the '*Historical data*'. For the same months, two additional sets of rosters were generated using the rostering software system. This system was manually configured by the head nurse with all constraints relevant for the department, as well as the constraint weights. The first of the generated sets, called '*Generated schedules with manual weights*', applied the weights defined by the head nurse. The process of automatically generating rosters was repeated for the last set, but instead of the manual weights, AWE extracted weights from the historical data. This was done by, for each constraint c, calculating n_c and v_c in each monthly roster, summing all these values, and computing w_c^{new} according to Equation (5.1). The extracted weights were then used to generate schedules for each of the monthly periods. This last data set is called '*Generated schedules with extracted weights*'.

To analyse the differences between the three sets of rosters, the following data was collected. First, for each constraint, the weights manually set by the head nurse in the software system were compared with the weights extracted by the AWE approach. Secondly, for each constraint, the total number of violations in all rosters was computed, for all three roster sets. This data was used by the authors to evaluate how the three different types of rosters compare to each other.

The algorithm used to automatically generate rosters is the heuristic search algorithm presented by Smet et al. [112], with a time limit of ten minutes. Mathematical details on the computation of constraint violations are given by Smet et al. [113].

5.5.3 Results

For all 100 constraints, Figure 5.2 compares the values of the manually defined weights with the values of the weights extracted by AWE. Although the values range from 0 to 1500, the manual weights were chosen among only few discrete values, while the AWE approach results in a much more spread out weight distribution. Using a limited number of discrete values for constraint weights may result in multiple plateaus in the search space and therefore numerous solutions with equivalent search score. A wider range of weight values, on the other hand, results in a less monotone search space which may be easier to explore [89].

The graph also shows that the extracted weights deviate significantly from the manual weights. This illustrates that the weights chosen by the head nurse do not correspond to the actual priority of constraints according to the

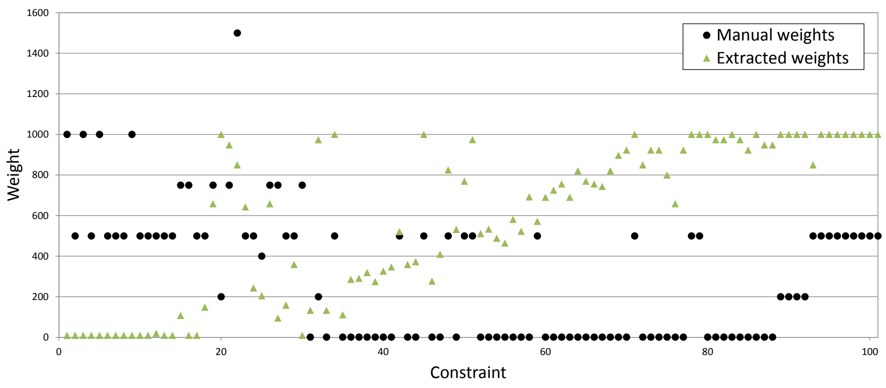


Figure 5.2: Manual and extracted weights for the retrospective case study

accepted rosters. In practice, it is common for head nurses to be satisfied after making changes to the generated rosters, even though the overall penalty after such modifications increases [46]. This also suggests that the manually defined constraint weights differ from the perceived importance of constraints.

In addition to studying the raw values of manual and extracted weights, the constraint violations in the respective rosters are compared. Figure 5.3 shows the total number of violations for each constraint in the three sets of rosters. Highlighted are the number of violations of '*Important constraints*', i.e. constraints weighted 1000 or more by the head nurse (cf. Figure 5.2). Given the assumption that the head nurse was satisfied with the manually produced rosters, the line of the historical data can be interpreted as the preference of the head nurse for trading off constraint violations. Clearly, the constraint violations of the rosters generated with manual weights do not follow the same trend as those of the historical data. This confirms the conjecture that the manually defined weights do not correspond to what the head nurse perceived as important while constructing the rosters. Analogously, this result shows that although the head nurse most certainly has an implicit knowledge of constraint importance, translating experience into numerical form is not trivial and is bound to produce inaccuracies.

The constraint violations of the rosters generated with extracted weights closely approximate the trend in the historical data. This outcome suggests that rosters generated with the weights extracted by AWE resemble the preferences of the human planner much more than the rosters generated with the manual weights do. This illustrates that the proposed AWE approach succeeds better in determining weights corresponding to the real implicit preferences of the head

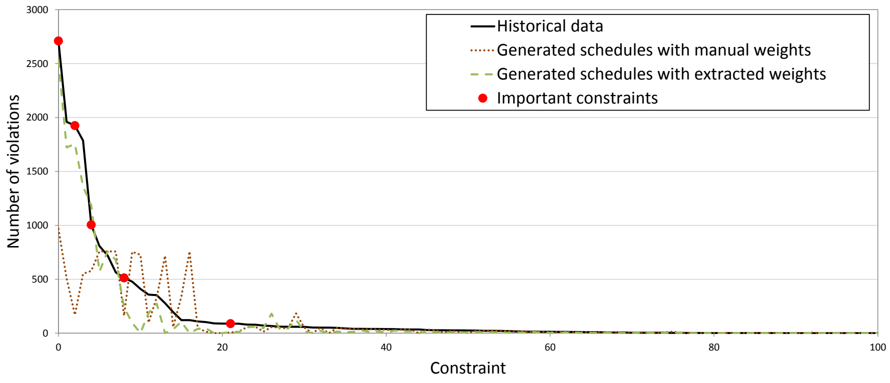


Figure 5.3: Comparison of constraint violations in the different rosters

nurses, than the head nurses are able to manually define.

5.6 A prospective case study

Since the hospital in the first case study had already performed the transition from manual to automated rostering, it was of little interest for the practitioners to let personnel from this hospital evaluate the results of AWE. Therefore, a different Belgian hospital, which has recently begun this transition, was invited to validate the approach in practice. This section first presents an analysis similar to that of the retrospective case study. Additionally, the outcome of an evaluation by the hospital's personnel is discussed.

5.6.1 Case description

This second case study was carried out in AZ Jan Portaels, a medium-sized Belgian hospital with 400 beds. The hospital had purchased the same automated rostering software as the hospital in the first case study, but only a few wards had performed the transition. The head nurses of four of these wards manually incorporated the constraints in the software and associated a weight to each constraint, based on their experience from manual rostering.

On the one hand, the practitioners were interested in comparing the quality of the newly automated rosters with those designed by hand. On the other hand, the hospital was in favour of testing the weight extraction approach in order to

Ward		Employees	Shifts	Skills	Constraints
Maternity	MT	21	7	2	13
General surgery	GS	15	15	2	19
Geriatric medicine	GM	21	17	4	14
Operating theatre	OT	23	12	3	19

Table 5.2: Overview of wards in the prospective case study

facilitate configuration of the software for the remaining wards. Additionally, it presented an interesting opportunity to evaluate AWE by comparing the quality of rosters generated with manual weights versus those generated using the automatically extracted weights. Therefore, an experiment was conducted that aimed to evaluate the same three rostering techniques discussed in the retrospective case study.

The four wards who agreed to participate in this case study were maternity (MT), general surgery (GS), geriatric medicine (GM) and the operating theatre (OT), as displayed in Table 5.2. Similar to the first case study, the wards are medium-sized, but now present a mix of care and non-care units. One difference with the first case study is the number of constraints considered when constructing the rosters. For example, the GS ward specifies no ward-specific constraints. In the MT and GM wards, the number of consecutive working days is limited to a maximum of five, while for night shifts, the number of consecutive days worked should be between four and seven. The OT ward wants to balance the number of undesirable shifts among nurses in order to avoid an uneven distribution of such shifts. All wards, however, are subject to the organisation-wide constraints, ensuring sufficient rest time between shifts, and forbidding some shifts to be assigned to certain nurses (e.g. 4h or 6h shifts to full time nurses).

The number of constraints in the different wards is, furthermore, influenced by the number of coverage constraints. The MT ward, for example, has many of these constraints for the different types of shifts.

Table 5.2 shows that there is a relatively high number of shift types for the GS, GM and OT wards. Typically, there are only four main shift types: early, day, late and night. However, for some of these types, short (4h), medium (6h) and long (8h) variants exist, with different break lengths, which should be assigned to either part time or full time nurses. Table 5.3 shows the different shift types for the GM ward.

	Start	End	Duration	Break
Early shifts				
V01	6:45	15:15	8h30	30 min
V09	6:45	12:45	6h00	0 min
Day shifts				
D44	8:00	14:30	6h30	30 min
D77	8:00	14:30	6h30	22 min
V02	6:45	13:00	6h15	0 min
D10	7:00	15:06	8h06	30 min
D34	8:00	16:30	8h30	30 min
D70	7:00	15:30	8h30	30 min
D83	7:30	13:30	6h00	0 min
D175	7:30	15:36	8h06	30 min
D16	8:00	16:06	8h06	30 min
D82	7:15	15:45	8h30	30 min
Late Shifts				
A01	13:30	21:30	8h00	0 min
A11	15:30	21:30	6h00	0 min
A12	15:00	21:30	6h30	0 min
Night shift				
N03	21:15	7:00	9h45	0 min
Other				
LW8	7:30	11:30	4h00	0 min

Table 5.3: Shifts in the geriatric medicine ward

5.6.2 Study design

For each of the four wards, three years of historical data were supplied in the form of past rosters, constructed by the head nurses. The hospital again provided the manually performed software configuration in the form of constraints and their associated weights. AWE was applied to extract new soft constraint weights for each ward, based on the violations in the manual rosters.

As in the first case study, three sets of rosters were compiled: the '*Historical data*', the '*Generated schedules with manual weights*', and the '*Generated schedules with extracted weights*'. Again, the heuristic from Smet et al. [112] was used to automatically generate the latter two sets of rosters, following the procedure for manually constructing rosters, i.e. per quarter of a year. Each set consists of monthly rosters concerning Q1, Q2 and Q4. July, August and September were not considered, since they present special conditions due to reduced workload.

The evaluation by the head nurses was conducted in the following way. For each month in the considered periods, the rosters from the three sets were printed in a familiar format, bundled, and presented together with an evaluation form on which the head nurses were asked to score, for their ward, each roster on a scale from 1 (very bad) to 10 (very good). To reduce bias, the different rosters were presented anonymously so that nurses were not aware whether they were evaluating a manual or automated roster and whether the latter was generated based on manual or extracted weights.

At the start of the evaluation, an explanation was given, briefly discussing the context of the experiments. The rosters were then (subjectively) evaluated and, relying on the experience of the head nurse, ranked based on their perceived quality. No time limit was imposed for the evaluation of the rosters. The experiment was conducted at a different time for each head nurse and there was no interaction between nurses during the experiment.

The outcome of the evaluation relies on the expertise of the head nurses to evaluate rosters for their ward. It was assumed that each head nurse is sufficiently experienced to correctly perform this task, even though the required effort might be greater for more complex wards such as OT. This assumption is reasonable, since constructing rosters is one of the main responsibilities of the head nurses, implying that they are skilled at the task. Nevertheless, the results should be interpreted with care as it is likely that other head nurses would produce different evaluations for the same rosters.

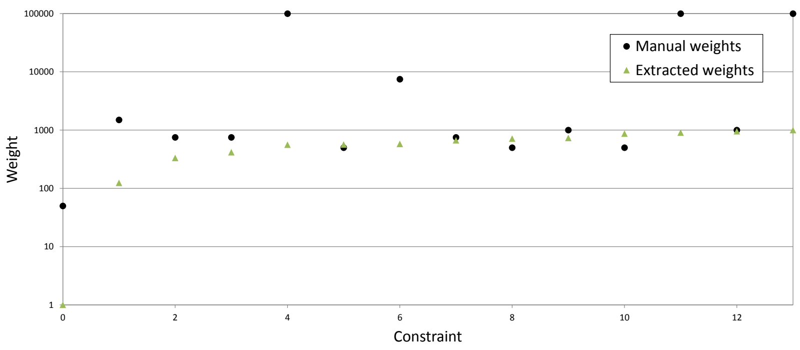
5.6.3 Results

Before presenting the results of the evaluation forms, the data of the second case study is discussed.

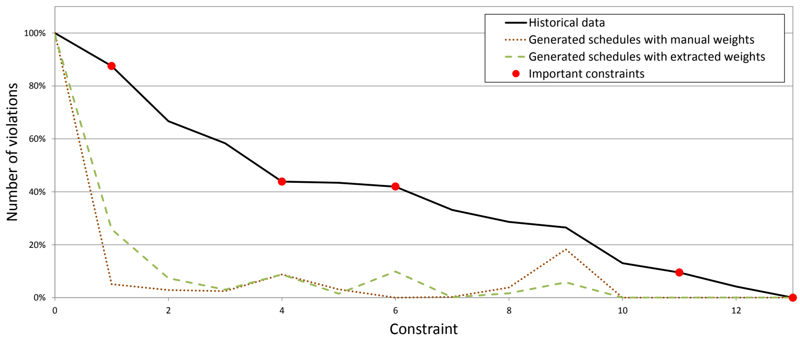
Analysis

The values of the manual and extracted soft constraint weights are compared, as well as the number of constraint violations in the three sets of rosters. Results are only shown for the GM ward. The other wards presented comparable results.

The weights for the 14 constraints of the GM ward are shown in Figure 5.4a. Similar to the first case study, manual weights are selected from a small set of discrete values, while the automatically extracted weights span a wider spectrum of values from 0 to 1000. Figure 5.4b shows the number of soft constraint violations in the three sets of rosters. The violations of the five constraints with the highest manually defined weights are indicated on the graph as '*Important constraints*'.



(a) Weights



(b) Violations

Figure 5.4: Comparison of manually defined and extracted weights, and constraint violations in the different rosters for the GM ward

While the retrospective case study revealed a clear difference between the violations in the two sets of automatically generated rosters, the distinction is less clear in the second case study. Neither the manual, nor the extracted weights contribute to generating rosters that approximate the preferences of the head nurse, i.e. the violations in the historical data. Nevertheless, both sets of weights appear to generate strictly better solutions in terms of the number of soft constraint violations. One should, therefore, expect that the subjective evaluation by the nurses should reveal a comparable score for the two automatically generated rosters and that this score should exceed the one for the manual rosters.

One reason for the different outcomes of the two case studies is the recent

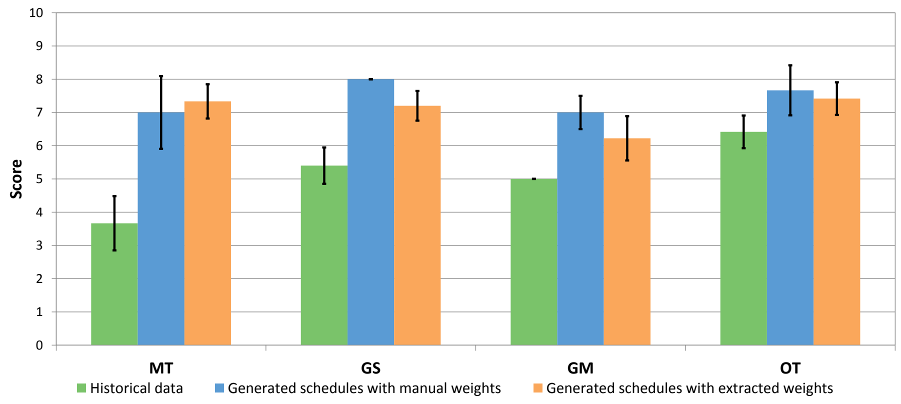


Figure 5.5: Evaluation scores by human planners for different wards. The scores are on a scale of 0 to 10, with higher being better

transition within the second hospital. The manual weights had not significantly been re-adjusted by the nurse from their initially defined values. Due to the low number of constraints in the second case study, the search algorithm succeeds in generating rosters that all Pareto-dominate the manual ones. Another possible explanation for the results is the low number of constraints, making it easier for a head nurse to prioritise and weigh constraints appropriately. Although it may be easier to express their relative importance in numerical form, it is certainly not trivial to actually take this importance into account when manually designing the rosters. Again, it is assumed that significant effort was spent in designing the manual rosters and that the head nurse was satisfied with their final quality.

Evaluation

An important observation made by all head nurses during the evaluation experiments was that for constructing the manual rosters, it was occasionally necessary to consider a different set of constraints than the set defined in the rostering software, e.g. different coverage requirements or staff availability. The head nurses argued that there was no incentive to incorporate these short-term deviations in the software since they were often exceptional. All sets of rosters were evaluated based on the typical constraints defined in the software without considering the exceptional ones. This result highlights an important drawback of automated rostering systems, namely their inability to deal with transient effects, as suggested in Table 5.1.

The averages and standard deviations of the scores given by the head nurses are depicted in Figure 5.5. In all wards, the manually constructed rosters received the lowest scores of the three sets. This result can be read as a strong recommendation to use the automated rostering software, since 1) it appears to construct rosters that are more preferred than the manual ones and 2) it requires significantly less time to do so. Nevertheless, comparison with the manual rosters is difficult (if not inapplicable here), since they had been constructed with an exceptional set of constraints that is not considered by the algorithm. This explains why in this case study, the AWE approach does not produce rosters that conform to the historical data.

On the other hand, the two automatically generated sets of rosters show only a small difference in score. The observations confirm the prediction that the two sets of automated rosters evaluate comparably and have a better quality than the manual ones. In addition, the results do support the applicability of AWE: by automatically extracting weights from past data, the head nurse no longer needs to go through the time-consuming trial-and-error process of appropriately adjusting constraint weights.

As was mentioned in the experimental setup, these conclusions build upon the assumption that head nurses can correctly evaluate rosters for their ward. Even though the assumptions made were reasonable, the evaluations are still subjective, and could be different when other head nurses were consulted.

5.7 Discussion

The results from both case studies deviate in their conclusions. In the retrospective case study, AWE succeeded in producing rosters with a comparable distribution of constraint violations as in those that were manually constructed. This result is not observed in the second case study where neither of the two sets of automated rosters follow the trend of the violation trade-off in the historical data. Based on knowledge obtained *after* discussions with the head nurses, the main cause is that the historical data from which AWE extracted weights considered a different set of constraints than the actual set used for generating the automated rosters. Nevertheless, through validation of the AWE results with health care personnel, it was shown that the results are acceptable for practical execution. This conclusion was also formulated informally by the head nurses responsible for constructing manual rosters, at the end of the experiment when the identity of the three sets of rosters was revealed. The evaluation scores are to be considered carefully as only four head nurses participated in the experiment, and more importantly, the evaluations are subjective.

Both case studies illustrate how AWE succeeds in quantifying the head nurses' preferences regarding constraint priority. This strongly facilitates the initial investment phase, as it supports head nurses in defining the often subtle trade-off between constraint violations [45].

As pointed out, the AWE approach generates weights that conform to the importance of constraints in the historical data. Constraints in the historical data should therefore be identical to those used in the software for automated rostering. This assumption emphasises the need to provide a relevant learning base for harnessing the full potential of automated weight extraction.

Finally, there is no reason to assume that the research results are limited to the hospitals from the case studies. The contributed AWE approach is general enough to be applied in yet other hospitals, countries, or even different types of organisations, provided sufficient relevant data is available.

5.8 Conclusions

Academic work on nurse rostering promises to automate the labour-intensive task of manually constructing rosters. However, in practice, it is hindered by the need to translate the practitioner's experience in numerical form. This chapter introduced automated weight extraction as a new approach capable of narrowing the research-application gap by automatically extracting soft constraint weights from past rosters. The technique facilitates the manual configuration of automated rostering software for hospital wards, allowing for easier adoption of state of the art rostering approaches.

Two case studies confirm the effectiveness of the new weight extraction approach, while also indicating the need for relevant historical data. Initial feedback received from the head nurses who participated in the evaluation of the proposed approach, indicates that the potential time savings are substantial, as the iterative trial-and-error process to determine suitable weights is eliminating. AWE has been integrated in commercial software for (automated) personnel rostering and management, currently used in various hospitals in Belgium, France, the Netherlands and Luxembourg.

Part III

Integration with other problems

Introduction to Part III: Integration with other problems

Real world combinatorial optimisation problems do not usually reduce to neatly delineated theoretical problems. Rather, they combine characteristics of various subproblems, which appear to be strongly intertwined. This is particularly true for nurse rostering, and personnel rostering in general, which is typically driven by other processes. The chapters in Part III focus on such *integrated* problems that combine characteristics of task scheduling and personnel rostering.

In the personnel scheduling literature, assigning shifts to personnel is often the most fine-grained level at which allocation is being discussed [49, 87, 99]. Task assignment to employees is often not incorporated in the roster construction. In some cases, employees know beforehand which tasks to perform during their working hours. In hospitals, for example, nurses know exactly what they must do during a shift. Indeed, some tasks such as meal distribution and hygienic or medical care of patients need to be conducted within set time frames. In many other cases, however, tasks are assigned to employees in an ad hoc manner, often resulting in excess resource utilisation. It is therefore recommendable to incorporate task assignment into roster construction for employees, in order to reduce operational expenses while still maintaining a high quality of service. Maenhout and Vanhoucke [86] point out the importance of such an integrative approach to achieve a more efficient and effective allocation.

Three problems are discussed in the following chapters, of which two are new contributions to the academic literature. The order of the chapters corresponds to the level of generalisation that the studied problems provide. For each of these problems, new solution approaches are presented based on decomposition and the integration of exact and heuristic techniques.

Chapter 6 studies the shift minimisation personnel task scheduling problem, in which a set of predefined tasks is assigned to qualified employees whose working times have been predetermined and cannot be changed. A new two-phase heuristic is presented. First, a constructive matheuristic builds an initial solution by sequentially solving heuristically delineated subproblems. Secondly, the initial solution is refined using a local branching based improvement heuristic. Computational results show that this algorithm succeeds in finding optimal solutions for all available benchmark instances from the literature. In addition, an empirical study is performed to investigate the influence of two problem characteristics on the hardness of instances. Based on these results, ten new benchmark instances are generated with the aim of further challenging the current state of the art algorithms.

In Chapter 7, the scope of decision making is extended to include assigning shifts as well as assigning tasks. This allows for an increased flexibility as now the working times of employees are not predetermined. The resulting problem is denoted as the task and shift scheduling problem and presents a particular challenge as the assignments of tasks and shifts are strongly interdependent. Two problem variants are introduced. Firstly, the problem is considered for a single, isolated day in the single day task and shift scheduling problem. Secondly, the scheduling period is extended to include multiple days in the multi-day task and shift scheduling problem, thereby allowing the definition of time related constraints on the shift allocation to employees. The influence of how the problems are decomposed by algorithms is investigated through a series of computational experiments. The results show that the heuristic algorithms which decompose the problem into horizontal subproblems, are successful in solving large instances.

Finally, Chapter 8 studies a reformulation of the multi-day task and shift scheduling problem. A column generation procedure is proposed for solving its linear programming relaxation. Details are discussed on measures taken to address some of the well-known issues in column generation. To obtain integer solutions, three heuristics are presented which start from the solution obtained by the column generation algorithm. The first two methods are diving heuristics on different sets of decision variables. The third method is a constructive heuristic executed after solving the pricing problem of the column generation, and which uses both primal and dual solutions to generate feasible schedules.

Complementary contributions

While Chapters 6, 7 and 8 focus on integrating personnel rostering and task scheduling problems, additional contributions have been made by the author in the area of integrated vehicle routing and personnel rostering problems. Mısır et al. [95] introduce three related problems dealing with scheduling and routing of home care, security and maintenance personnel.

The author's main contribution was related to formulating the model. Each of these integrated problems is modelled as a vehicle routing problem with a large variety of side constraints corresponding to time related constraints on e.g. hours worked or rest time between two consecutive days worked.

A general set of low-level heuristics was defined, applicable to all three problems. In a computational study, the hyper-heuristic of Mısır [94] was used as a tool to analyse the behaviour of the heuristics. Experimental results revealed that different low-level heuristics perform better on different problems, and that their performance varies during a search. Based on these results, the most relevant set of heuristics was determined for each problem domain. Additional computational experiments showed that using only the relevant low-level heuristics resulted in better solutions than applying the full set of heuristics.

Chapter 6

The shift minimisation personnel task scheduling problem: a new hybrid approach and computational insights

Assigning scheduled tasks to a multi-skilled workforce is a known NP-complete problem with applications in health care, manufacturing, logistics, etc. Optimising the use and composition of costly and scarce resources such as staff has major implications on any organisation's health. This chapter introduces a new, versatile two-phase matheuristic approach to the shift minimisation personnel task scheduling problem (SMPTSP), which considers assigning tasks to a set of multi-skilled employees, whose working times have been determined beforehand. Computational results show that the new hybrid method is capable of finding, for the first time, optimal solutions for all benchmark instances from the literature, in very limited computation time.

The influence of a set of problem features on the performance of different algorithms is investigated in order to discover what makes particular problem instances harder than others. These insights are useful when deciding on organisational policies to better manage various operational aspects related to workforce. The empirical hardness results enable to generate hard problem instances, which have been made publicly available.

This chapter is a minor adaptation of Smet, P., Wauters, T., Mihaylov, M., Vanden Berghe, G. (2014). The shift minimisation personnel task scheduling problem: a new hybrid approach and computational insights. *Omega*, 46, 64-73.

6.1 Introduction

This chapter studies the problem of assigning tasks to multi-skilled employees, while minimising the number of employees. This problem was introduced by Krishnamoorthy and Ernst [76] as the shift minimisation personnel task scheduling problem. Krishnamoorthy et al. [77] propose a Lagrangian relaxation based approach that combines two problem specific heuristics. In doing so, they were able to find 135 feasible solutions and 67 optimal solutions out of 137 benchmark instances. Furthermore, several properties of the problem were discussed, and efficient algorithms for solving special cases of the SMPTSP were introduced. Smet and Vanden Berghe [115] applied a hybrid local search algorithm based on a *fix and optimise* strategy to the dataset and found 68 new best solutions and 81 optimal solutions. They also found new lower bounds for 43 instances.

The SMPTSP is similar to the interval scheduling problem [72], in which a set of jobs with fixed start and end times are given, as well as a set of machines on which the jobs should be processed. The goal is to decide which jobs to process and on which machines, while e.g. maximising profit associated with each job. The SMPTSP differs from the basic interval scheduling problem in that it requires all jobs (tasks) to be assigned while not all machines (employees) can process each job.

This chapter introduces a two-phase approach that can be classified as a matheuristic, as it combines the strengths of both heuristic and exact approaches [88]. This family of hybrid approaches recently gained significant attention because of their ability to solve problems for which traditional (meta)heuristics or exact approaches fail. Della Croce and Salassa [40] describe a variable neighbourhood search-based matheuristic for a real world nurse rostering problem. Neighbourhoods are defined by adding temporary constraints which fix a subset of heuristically selected assignments. The neighbourhoods are searched with an exact branch-and-bound algorithm. Computational experiments show that this matheuristic significantly outperforms a general purpose integer programming solver. Matheuristic approaches have been applied to many other hard combinatorial optimisation problems including vehicle routing [43], permutation flow shop scheduling [39] and the multidimensional knapsack problem [59].

The remainder of this chapter is organised as follows. First, the contributions and their practical relevance are outlined in Section 6.2. The problem definition is provided in Section 6.3. Section 6.4 presents different constructive heuristics for the SMPTSP, as well as a heuristic improvement procedure based on local branching. A discussion on the algorithm design is included in Section 6.5. Furthermore, the performance of the presented algorithm and recent approaches from the literature are compared. The influence of instance specific characteristics on algorithm performance is discussed in Section 6.6. Based on these empirical hardness results, new hard instances are introduced in Section 6.7. Finally, Section 6.8 concludes the chapter.

6.2 Contributions

The first major contribution of this chapter is the introduction of a hybrid heuristic approach, which, at present, represents the state of the art of algorithms for the SMPTSP. A study comparing the performance against recently published solution techniques from the literature, and against a commercial solver (Gurobi 5.1.0) demonstrates the efficiency and effectiveness of the new hybrid heuristic. Furthermore, it is shown that the algorithm was able to find, for the first time, optimal solutions for all instances from the dataset of Krishnamoorthy et al. [77]. The second contribution is an investigation of SMPTSP properties that affect algorithm performance. Finally, as a third contribution, based on the empirical hardness study, a new benchmark dataset is generated containing more challenging problem instances.

The efficient allocation of scarce resources is an ever-present issue for management, particularly when these resources cause high expenses for the organisation. This is especially true for the SMPTSP since inefficient assignment of the available workforce can lead to significant costs, for example when hiring additional temporary workers becomes inevitable. Manual planners often simplify the assignment by making abstraction of intricate problem properties such as start and end times of tasks or qualification requirements. However, many organisations require this complexity to be incorporated in the decision making process. Ignoring it would render any decision support approach inapplicable. Algorithms for problems entailing the full complexity enable better decision making on both strategic and operational level. The former is achieved by determining the optimal composition of an organisation's workforce, the latter by efficiently deploying these costly resources and thereby reducing operational expenses.

6.3 Problem definition

Let $T = \{1, ..., n\}$ be the set of tasks to be assigned and $E = \{1, ..., m\}$ the set of employees. Each task $j \in T$ has a duration u_j, a start time s_j and a finish time $f_j = s_j + u_j$. Each employee e has a set of tasks $T_e \subseteq T$ that he/she can perform. Similarly, for each task j, a set $E_j \subseteq E$ exists, which contains all employees that can perform task j. Both T_e and E_j are defined based on qualifications, time windows of tasks and availabilities of employees.

An interval graph $G = (V, A)$ can be defined with one node for each task, and A the set of arcs. Two nodes i and j are connected if their respective time intervals, $[s_i,f_i]$ and $[s_j,f_j]$, overlap. Let C be the set of maximal cliques in G. This set can be found in polynomial time by first sorting the nodes based on start time and then applying a forward pass algorithm. The set $C = \{K_1, ..., K_t\}$ consists of subsets of T such that any pair of tasks in K overlap in time and K is maximal. No tasks in $T \setminus K$ overlap with any of the tasks in K. In terms of the SMPTSP, it is clear that overlapping tasks, represented by nodes in K, should be assigned to different employees. For each employee e, the set of maximal cliques $C_e = \{K_1, ..., K_t\}$ is constructed in the same way as C, while only considering tasks for which the employee is qualified. An employee e can only be assigned to one of the tasks from each set $K \in C_e$. This prevents overlapping assignments in a solution.

In a feasible solution, all tasks in T are assigned to qualified employees from E in a non-preemptive manner. The objective is to minimise the total number of employees.

Decision variables

$$x_{je} = \begin{cases} 1 & \text{if task } j \text{ is assigned to employee } e \\ 0 & \text{otherwise} \end{cases}$$

$$y_e = \begin{cases} 1 & \text{if employee } e \text{ is assigned to at least one task} \\ 0 & \text{otherwise} \end{cases}$$

Model [77]

$$min \sum_{e \in E} y_e \tag{6.1}$$

$$s.t. \sum_{e \in E_j} x_{je} = 1 \qquad \forall\, j \in T \tag{6.2}$$

$$\sum_{j \in K} x_{je} \leq y_e \qquad \forall\, e \in E,\ K \in C_e \tag{6.3}$$

$$0 \leq y_e \leq 1 \qquad \forall\, e \in E \tag{6.4}$$

$$x_{je} \in \{0, 1\} \qquad \forall\, j \in T,\ e \in E \tag{6.5}$$

The objective function (6.1) minimises the number of employees. Constraints (6.2) ensure that each task is performed by exactly one employee, and that no infeasible assignments in terms of qualifications are made. Constraints (6.3) make sure that tasks are only assigned to active employees and that tasks assigned to an employee do not overlap. Finally, Constraints (6.4) and (6.5) set bounds for the decision variables.

The SMPTSP can be seen as an application of list colouring on interval graphs, which is NP-complete [13]. Colours correspond to employees and nodes correspond to tasks. Two nodes are connected whenever the corresponding tasks overlap in time. The qualifications of the employees are represented by the list of feasible colours on each node. Other applications of list colouring on interval graphs include classroom allocation [30] and register assignment [124].

6.4 Solution procedure

A two-phase hybrid heuristic algorithm is presented which integrates exact optimisation techniques and heuristic search. A constructive heuristic first generates an initial solution, which is improved in the second phase. Section 6.4.1 describes several constructive heuristics for providing the initial solution. The improvement heuristic is presented in Section 6.4.2.

6.4.1 Constructive heuristics

Three different constructive approaches are presented: *first fit*, *best fit* and a *constructive matheuristic* algorithm.

First fit and best fit heuristics

Krishnamoorthy et al. [77] state that when the qualification constraints of the SMPTSP are relaxed, i.e. when all employees are qualified to perform all tasks, the resulting problem can be solved in polynomial time with a forward pass maximal clique algorithm on an interval graph [57]. This algorithm assigns all tasks in order of increasing start time, considering first, if possible, an employee who already has tasks assigned. This property is incorporated in the *first fit* and *best fit* heuristics presented in this section. Both heuristics first order the tasks by start time in ascending order. Ties are broken by taking into account the qualifications of employees, i.e. the tasks are sorted again by the number of qualified personnel able to perform them, also in ascending order. This results in an ordering in which tasks with the smallest number of feasible personnel appear before others. These highly constrained tasks, which are the most difficult to assign, are thus first to be assigned.

An additional mechanism is introduced to ensure that the first fit and best fit heuristics find feasible solutions in cases where tasks can only be assigned to a limited number of employees. Whenever a task j cannot be assigned to a feasible employee due to other overlapping assignments, a qualified employee is randomly selected and his/her assigned tasks overlapping with j are removed. Task j is then assigned to this employee and the removed tasks are reassigned to other employees later in the procedure.

The aforementioned steps outline a general framework for both the first fit and best fit constructive heuristics. The first fit heuristic assigns tasks to the first feasible employee (Algorithm 2). The best fit heuristic is designed by changing line 4 in Algorithm 2 such that it assigns the tasks to the *best* feasible employee instead of to the *first* feasible employee. The best employee is identified by the largest sum of assigned task durations $\sum_{j \in R_e} u_j$, with R_e the set of tasks currently assigned to employee e. This way, employees' schedules will include as many tasks as possible, thereby minimising the number of employees.

It is worth noting that both the first fit and best fit constructive heuristics are not guaranteed to terminate, i.e. it is possible that they enter an infinite loop in which two sets of tasks are repeatedly assigned to and deassigned from the same set of employees. In order to ensure finite behaviour of these heuristics, a termination criterion is included, which stops the algorithm when it enters an

Algorithm 2 First fit constructive heuristic

```
Input:
    $T$ := Tasks to be assigned
    $E_j$ := Employees qualified for task $j \in T$
    $s_j$ := Start time of task $j \in T$
    $R_e := \emptyset$                                  ▷ tasks assigned to employee $e$
Output: (Partial) solution
 1: Order all $j \in J$ by $(s_j + |P_j|)$ in ascending order
 2: while $J \neq \emptyset$ do
 3:     Remove task $j$ from the first position in $T$
 4:     Assign $j$ to the *first* feasible employee
 5:     if Cannot feasibly assign $j$ to any qualified employee then
 6:         Select random employee $e \in E_j$
 7:         $O_j$ := Tasks overlapping with task $j$
 8:         Remove all tasks in $O_j$ from $R_e$
 9:         Assign $j$ to employee $e$
10:         Add tasks in $O_j$ to the list of tasks to be assigned $T$
11:     end if
12: end while
```

infinite loop. As a result, tasks may remain unassigned after the constructive heuristics have finished.

Constructive matheuristic

In the constructive matheuristic (CMH), a solution is constructed by optimally solving subproblems one by one using an integer programming solver. In each subproblem, a subset of b employees $E' \subseteq E$ is considered. Still tasks have to be feasibly assigned to employees in E', but instead of minimising the number of employees, the objective is to maximise the sum of assigned task durations, thereby implicitly reducing the number of employees (Equation (6.6)).

$$max \sum_{e \in E'} \sum_{j \in T} u_j x_{je} \tag{6.6}$$

Only the last subproblem is reoptimised for a second time with the original objective function (6.1). Algorithm 3 shows the pseudocode of the constructive matheuristic.

Algorithm 3 Constructive matheuristic

```
Input:
    $T$ := set of tasks to be assigned
    $E$ := set of employees
    $b$ := number of employees in one subproblem
Output: (Partial) solution
1: while $E \neq \emptyset$ or not all tasks assigned do
2:     $E'$ := sample and remove $b$ employees from $E$     ▷ delineate the subproblem
3:     Solve the subproblem for $E'$
4:     Remove the tasks assigned to $E'$ from $T$
5:     if $T = \emptyset$ then
6:         Reoptimise $E'$ using the original objective (Equation (6.1))
7:     end if
8: end while
```

Infeasibility issues

Initial experiments showed that, particularly for small problem instances with $m < 100$, the constructive heuristics are not always able to assign all tasks. The result can thus be infeasible. This issue is addressed in the second phase of the presented approach, which is a local branching based improvement heuristic. This general improvement heuristic is particular in that it does not require a feasible initial solution. Infeasible starting solutions are repaired during execution of the improvement heuristic.

6.4.2 Improvement heuristic

After generating the initial solution with one of the aforementioned constructive heuristics, an improvement procedure attempts to further reduce the number of employees, given that the initial solution was not yet optimal. For this purpose, a matheuristic based on the idea of local branching is presented [52].

The algorithm constructs a solution x in which at most k binary variables have flipped values with respect to a given reference solution $\bar{x}$. This is enforced by adding the asymmetric Hamming distance constraint (6.7) to the original integer programming model.

$$\sum_{j \in T} \sum_{e \in E} \bar{x}_{je}(1 - x_{je}) \leq k'(= \frac{k}{2}) \tag{6.7}$$

In the context of the SMPTSP, k' corresponds to the maximum number of tasks that can be (re)assigned, given the reference solution $\bar{x}$. In essence, the improvement heuristic adds Constraint (6.7) to the mathematical model and solves it to optimality using a general purpose integer programming solver. If the new solution x is not better than the given solution $\bar{x}$, or if the solution's objective value is equal to the lower bound, the procedure stops, else x is set as the new reference solution and the previous steps are reiterated.

Algorithm 4 outlines this local branching improvement heuristic (LBIH).

Algorithm 4 Local branching improvement heuristic

```
Input:
    $F(x)$ evaluation function of $x$
    $LB$ := lower bound on the objective value
    $\bar{x}$ := initial reference solution
    $k'$ := maximum number of tasks to reassign
Output: $\bar{x}$                                ▷ improved initial solution
 1: improved := true      ▷ boolean value to control termination of the algorithm
 2: while improved and $F(\bar{x}) \neq LB$ and stop criterion not met do
 3:     $x$ := solve the model with the Hamming distance constraint with $k'$ given $\bar{x}$
 4:     if $x$ is not feasible then
 5:         $k' := k' + 1$
 6:     else if $F(x) < F(\bar{x})$ then
 7:         $\bar{x} := x$
 8:     else
 9:         improved := false
10:     end if
11: end while
```

Algorithm 4 clearly builds upon the matheuristic concept of combining integer programming and heuristic search by embedding an exact approach for solving subproblems in an iterative improvement framework. Due to the absence of problem specific elements, Algorithm 4 also presents a versatile improvement heuristic applicable to a large class of problems. It can thus be seen as a general method for which only an integer programming formulation is required. Moreover, in contrast to many other general improvement algorithms, it does not require a feasible initial solution. On some occasions, it is worthwhile spending effort in designing a constructive heuristic which provides a good initial solution to yield better results. Feasible, or almost feasible, starting solutions will furthermore benefit the algorithm's performance since this will prevent steps 4 and 5 from being executed excessively.

6.5 Computational results

The behaviour of the proposed heuristics is evaluated by analysing the results of a series of experiments. Furthermore, the performance of the new approaches is compared with the best known results from the literature.

6.5.1 Experimental setup

All instances in the benchmark dataset of Krishnamoorthy et al. [77][6] were used for the experiments and evaluation. The dimensions of these instances range from small (23 employees and 40 tasks) to very large (245 employees and 2105 tasks). On average, all employees can perform either 33% or 66% of the tasks.

All experiments were carried out on an Intel Core i7-2600 at 3.4GHz with 8GB RAM operating on Windows 7. The algorithms were coded in Java. Gurobi 5.1.0 was used as integer programming solver. Each run was repeated ten times with computation time limited to 1800 seconds per run.

6.5.2 New lower bounds

In order to facilitate evaluation of algorithm performance, a new lower bound is presented, which improves upon the best reported lower bounds from the literature.

Proposition 1. *The size of the largest clique in the set C is a valid lower bound for the SMPTSP.*

This is a trivial lower bound since it corresponds to the minimum number of employees needed to cover the largest number of overlapping tasks. A polynomial time algorithm to calculate this lower bound is described on page 108. This bound does not provide the minimum number of employees required in a solution for the SMPTSP, since it does not take into account the employees' qualifications. However, the computational experiments show that this lower bound is equal to the optimal solution for all instances in the dataset of Krishnamoorthy et al. [77].

[6] `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/ptaskinfo.html`

	First fit	Best fit	CMH $b = 10$	CMH $b = 15$
Number of optimal solutions	13	22	118	131
Average solution quality	125.35	125.78	123.01	122.61
Average calculation time (seconds)	0.02	0.11	12.26	44.93
Maximum calculation time (seconds)	0.21	1.10	169.30	889.80

Table 6.1: Summary of results for the constructive heuristics

6.5.3 Constructive heuristics

The first set of experiments compares the performance of the first fit, best fit and CMH with two different block sizes ($b = 10$ and $b = 15$). These values were determined after preliminary experiments. Table 6.1 summarises the performance of the different constructive algorithms. Proposition 1 was used to determine whether a solution is optimal. Detailed computational results can be found on a dedicated web page[7].

Best fit generates more optimal solutions than first fit, whereas first fit has a slightly better average solution quality, meaning that for instances that cannot be solved optimally with best fit, first fit obtains better solutions. The calculation times required for both best fit and first fit are negligible.

The constructive matheuristic produces an optimal solution for the majority of the instances. This improved solution quality comes at the cost of an increased calculation time on large instances. The time required by CMH depends on the block size, which also influences the quality of the constructed solution. Figure 6.1 illustrates this trade-off for an instance with 211 employees and 1647 tasks. When the block size increases, the subproblems become larger and thus require more calculation time. However, larger block size means that more employees are considered in each subproblem, which can result in better solutions.

Based on the results from Table 6.1, CMH with a block size of $b = 10$ was used in remaining experiments to generate the initial solution for the improvement heuristic.

In order to determine the influence of the set of employees composing the subproblems, an experiment was set up in which three approaches were compared. First, employees were selected for subproblems based on the number of tasks they can perform, in ascending order, such that the first subproblem contains the most restricted employees in terms of number of tasks they can perform. Second, this order was reversed, so that the most qualified employees were selected first.

[7] `http://gent.cs.kuleuven.be/smptsp.html`

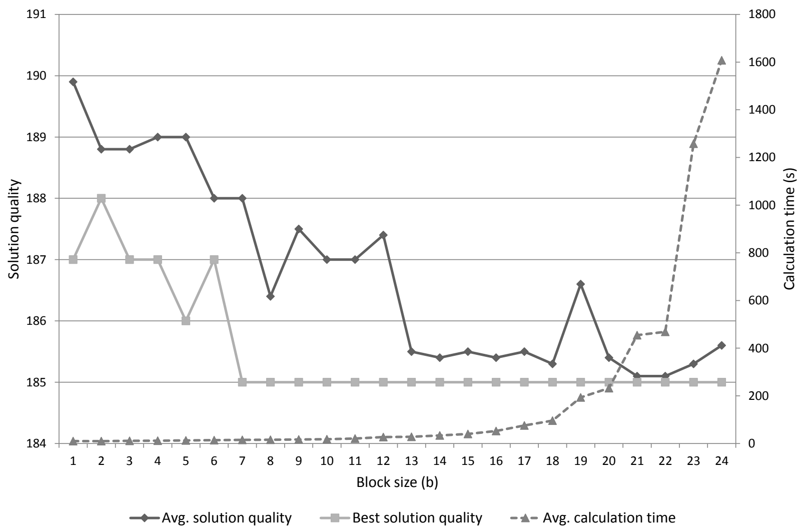


Figure 6.1: Average and best solution quality and average calculation time for the CMH with varying block size $b = [1, 24]$

	Ascending order	Descending order	Random order
Number of feasible solutions	50	50	50
Average solution quality	160.22	160.32	160.31
Average calculation time (seconds)	22.19	23.82	23.79

Table 6.2: Impact of employee selection for the subproblems

Finally, employees were randomly selected for each subproblem. Table 6.2 shows the average performance of these three approaches on 50 instances with number of employees ranging from 88 to 415 and number of tasks ranging from 777 to 2105. The results show that, for the tested instances, the employee selection procedure does not influence the performance of the constructive matheuristic. It is important to note, however, that these results were obtained based on instances in which task qualifications are randomly distributed among employees. The presence of a clear structure in the employee skills (e.g. hierarchical) may lead to different conclusions, and, moreover, may warrant a tailor-made selection mechanism.

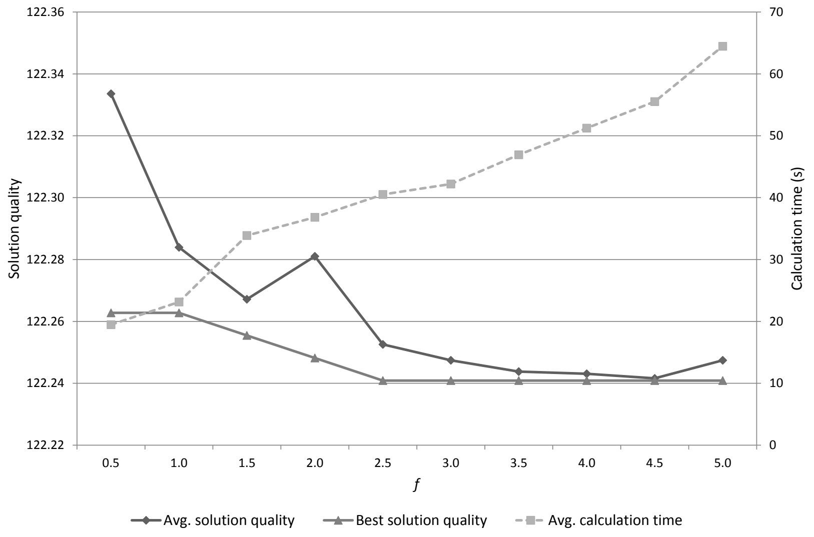


Figure 6.2: Average and best solution quality and average calculation time for the LBIH with varying parameter $f = [0.5, 5]$ on all instances

6.5.4 Improvement heuristic

The parameter k' of the local branching improvement heuristic is set as a function of the problem size and is calculated as $k' = \lceil f\sqrt{n} \rceil$. The rationale behind this is that for instances with many tasks, the number of tasks allowed to be reassigned should not be too large since this would make the calculation time for solving the model with the Hamming distance constraint unacceptable. A linear relation between k' and the number of tasks n would thus not be suitable. Therefore, the square root of n was chosen, multiplied by a factor f. For larger f, more tasks will be allowed to move when solving the model with the Hamming distance constraint. Figure 6.2 shows, for f varying between 0.5 and 5, the best and average solution quality, and average solution time of ten runs, for all instances from the benchmark dataset. The results indicate that when more tasks are allowed to move, better solutions can be obtained, but at the cost of increased calculation time. Based on these experiments, $f = 4.5$ proved to be the most appropriate choice.

The combined matheuristic CMH+LBIH ($b = 10$, $f = 4.5$) was compared with an integer programming solver (MIP) and two methods recently reported in the literature: the Lagrangian relaxation based heuristic from Krishnamoorthy et al. [77] (KEB12) and the fix and optimise heuristic from Smet and Vanden Berghe

	MIP	KEB12	SV12	CMH+LBIH ($b = 10$, $f = 4.5$)
Number of optimal solutions	88	67	81	137
Average solution quality	129.88	127.00	123.04	122.24
Average calculation time (seconds)	-	-	958.91	55.52
Maximum calculation time (seconds)	1800.0	1800.00	1800.00	604.87

Table 6.3: Summary of results for different approaches for the SMPTSP

[115] (SV12). Table 6.3 presents the summarised results. A dedicated web page[8] provides the detailed computational results. The reported calculation times are total times, i.e. the sum of computation time of the constructive heuristic and the time required by the improvement heuristic.

The CMH+LBIH ($b = 10$, $f = 4.5$) approach finds, for the first time, an optimal solution for all the 137 instances, requiring much less than the allowed calculation time. On average, the presented method requires 55 seconds to find an optimal solution while the worst case still only takes little more than ten minutes to produce an optimal solution. Compared to the other approaches, CMH+LBIH ($b = 10$, $f = 4.5$) thus performs significantly better, both in terms of solution quality and the required calculation time.

6.6 Empirical hardness

In order to understand the behaviour of algorithms for the SMPTSP, a series of experiments was conducted to determine what makes particular instances easy or hard for different algorithms. It is generally known that, while the computational complexity of a problem can be established as hard, easy instances may exist [81]. The identification of relevant hardness features enables the development of powerful portfolio techniques [92].

After performing an initial statistical analysis of the performance of different algorithms on the dataset from Krishnamoorthy et al. [77], the skilling level and average task duration were seen to be most influential on the hardness of the problem. This section investigates their influence on the performance of algorithms for the SMPTSP.

[8]`http://gent.cs.kuleuven.be/smptsp.html`

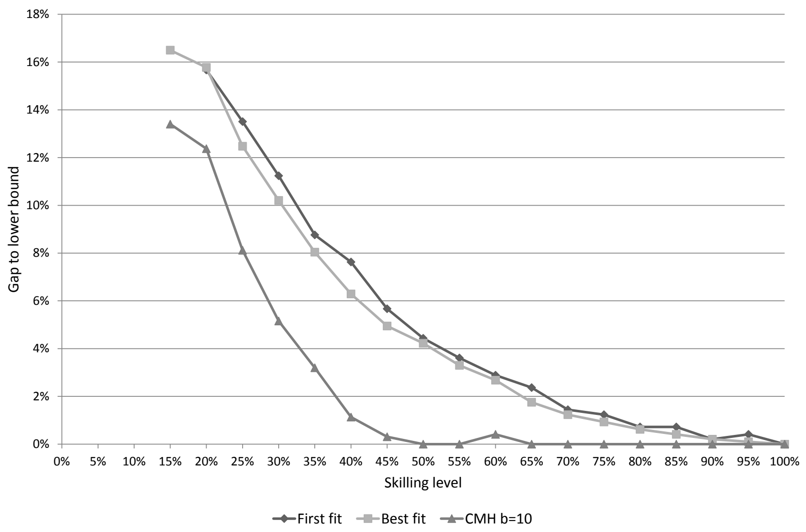


Figure 6.3: Average gap from lower bound for the constructive heuristics with varying skilling level (113 employees, 1112 tasks, 90% tightness)

6.6.1 Skilling level

The skilling level of an instance is the percentage of tasks each employee is qualified for on average. When this level is 100%, each employee can perform all tasks. A series of experiments was set up to determine the influence of this instance feature on the performance of both heuristics and a general purpose integer programming solver. An instance generator, developed according to the description of Krishnamoorthy et al. [77], was used to generate a set of new instances in which the skilling level varied from 5% to 100%. For each level, ten random instances were constructed. The reported results are the average (or median in case of computation time) of one run on each of the ten instances.

Figure 6.3 shows that the gap to the lower bound for the three constructive heuristics, presented in Section 6.4.1, decreases when the skilling level increases. Recall that the constructive heuristics use ideas from the forward pass maximal clique algorithm for interval graphs. When the skilling level is 100%, the SMPTSP reduces to exactly that problem, thus making it easier for the constructive heuristics to find good solutions. For instances with a skilling level lower than 15%, the heuristics were unable to find a feasible solution.

An analogous experiment was conducted in which the integer programming model was solved with a commercial solver (Gurobi 5.1.0), using the computation

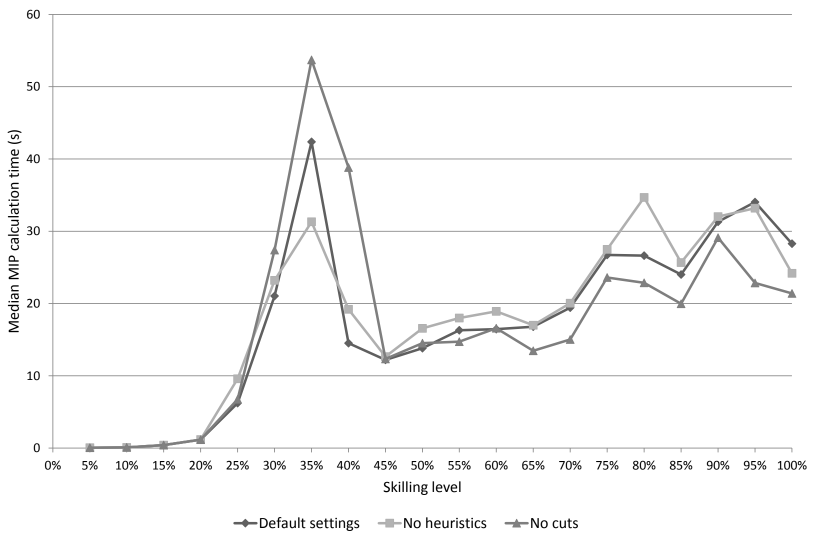


Figure 6.4: Median computation time in seconds for different configurations with varying skilling level (33 employees, 337 tasks, 90% tightness)

time required for finding an optimal solution as a measurement for hardness. Figure 6.4 shows, for different configurations of the solver, that it takes longer to find an optimal solution when the skilling level increases. There is a clear peak in required calculation time at 35%. In the original dataset, instances have a skilling level of either 33% or 66%. Clearly, these are thus instances which are particularly challenging for an integer programming solver, or any procedure which uses a solver as a subroutine.

6.6.2 Average task duration

A similar series of experiments was conducted to investigate the influence of the average task duration. The experimental setup was the same as that described on page 119, but the distribution from which task lengths were sampled was varied, while keeping the number of employees, the number of tasks and the skilling level constant. As in the original dataset, the task lengths were sampled from a triangular distribution $Tri(\alpha, \beta, \gamma)$, with β varying between 100 and 440. The ratios of α and γ to β were kept constant: $\alpha = \beta - 100$ and $\gamma = \beta + 100$.

Figure 6.5 shows that, for all three constructive heuristics, the average gap to the lower bound decreases when tasks become longer. This is particularly clear

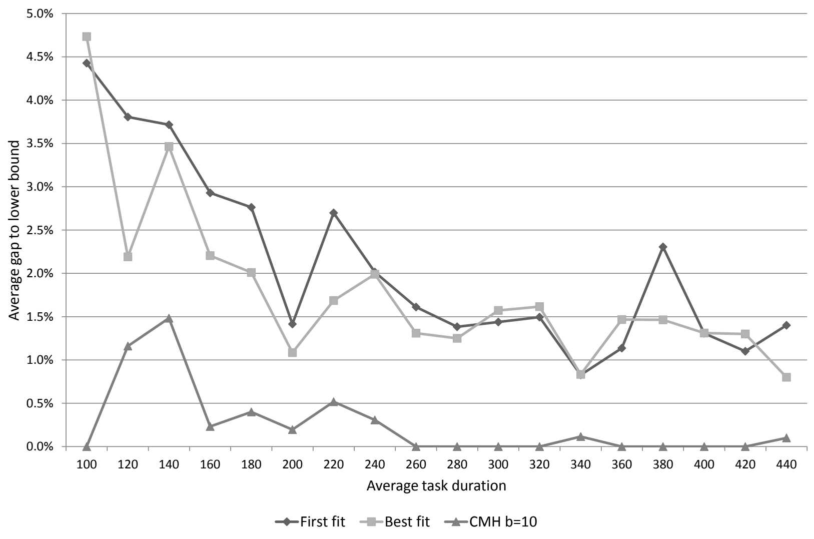


Figure 6.5: Average gap from lower bound for the constructive heuristics with varying average task duration (100 employees, 300 tasks, 60% skilling)

for the first fit and best fit constructive heuristics.

Figure 6.6 shows the same trend for different configurations of a general purpose solver: longer tasks make instances easier to solve. Note that the number of employees and tasks is kept constant and, therefore, the number of variables in the model also remains the same, such that this factor does not alter the solver's performance.

6.7 New benchmark instances

The results from Section 6.6 show that the performance of constructive heuristics suffers in case of a low skilling level or short tasks. The exact solver performs worse when tasks are short, but also when the skilling level is either high, or around 35%. Based on these observations, new benchmark instances were generated with short tasks ($\beta \in \{120, 280\}$) and low skilling level (20%, 30%).

Table 6.4 presents, for each new instance, the clique lower bound (LB), the best solution found by an integer programming solver after 1800 seconds (MIP), the best result found in ten repeated runs by CMH+LBIH ($b = 10$, $f = 4.5$)

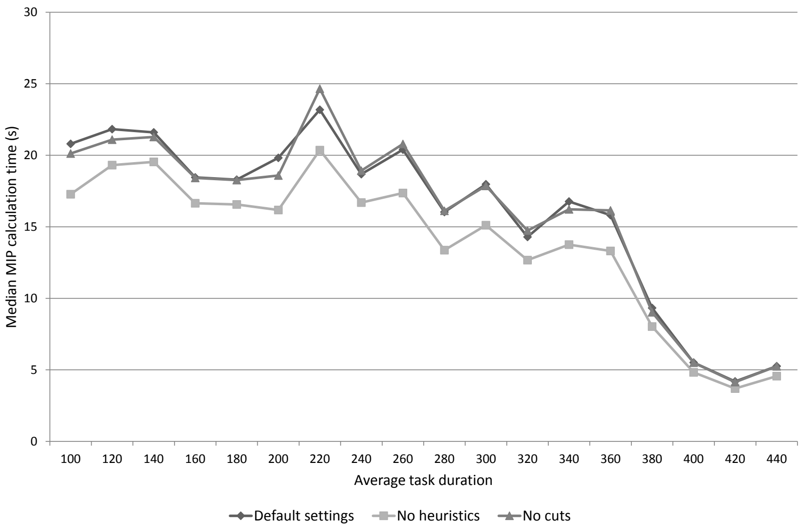


Figure 6.6: Median computation time in seconds for different configurations with varying average task duration (100 employees, 300 tasks, 60% skilling)

Instance	LB	MIP	CMH+LBIH ($b = 10$, $f = 4.5$)		
			F_{min}	F_{avg}	T_{avg}
1_50_258_20	40	40	40	40.80	641.58
2_44_510_20	40	40	41	41.20	683.31
3_102_525_30	77	83	77	77.40	938.62
4_113_647_20	98	99	98	98.00	163.24
5_77_777_30	59	65	59	59.80	1615.46
6_135_777_20	116	119	116	116.90	1699.28
7_70_781_20	59	61	61	61.50	1800.00
8_88_1022_20	79	80	80	80.50	1800.00
9_125_1308_20	98	106	99	101.90	1800.00
10_153_1577_20	116	153	118	123.20	1800.00

Table 6.4: Computational results for the new benchmark dataset instances

(F_{min}), the average result over ten runs (F_{avg}) and the average time required in seconds (T_{avg}).

Table 6.4 shows that a large part of the new instance set remains unsolved in the current experimental setting. A web page[9] keeps track of new results on

[9]`http://sites.google.com/site/ptsplib`

these instances.

6.8 Conclusions

This chapter addressed the shift minimisation personnel task scheduling problem, which deals with the challenge of assigning tasks to employees who are restricted by qualifications and availabilities. The objective is to minimise the number of employees while still assigning all tasks.

A two-phase algorithm design was proposed in which, building upon the matheuristic concept, both phases combine integer programming with heuristic search, resulting in an efficient and versatile general optimisation method. The new hybrid algorithm found, for the first time, optimal solutions for all 137 instances from a benchmark dataset, while consuming little calculation time. Experimental results demonstrate that this novel algorithm holds the state of the art for the SMPTSP.

An empirical hardness study with three different constructive heuristics showed that the problem becomes harder when employees are qualified for relatively few tasks. Furthermore, the constructive heuristics were shown to be sensitive to the average task duration. In problems with long tasks, these algorithms generated better solutions than when the tasks were shorter. The behaviour of an integer programming solver on the standard mathematical model was also studied. Experiments showed that the solver's performance worsens when the average task duration decreases, or when the staff is highly skilled.

A new benchmark dataset was generated consisting of instances satisfying the properties identified as hard. Since their publication, these new benchmark instances have been adopted by researchers to further challenge algorithms for the SMPTSP [51].

Chapter 7

Exact and heuristic decomposition approaches to the single and multi-day task and shift scheduling problem

This chapter investigates the impact of alternative decomposition schemes on a new integrated task and personnel scheduling problem. The problem generalises the SMPTSP discussed in Chapter 6, as now both tasks and shifts have to be assigned to a set of multi-skilled employees. Two variants are considered: one in which the scheduling period is limited to a single, isolated day and one for a period of multiple days.

Different exact and heuristic decomposition schemes are proposed, and their performance is evaluated on a diverse set of instances which have been made publicly available. The research question is whether a composition of strong (heuristic) approaches to the subproblems could outperform exact optimisation approaches applied to the overall problem. Extensive computational experiments show that decomposing the problem into horizontally delineated subproblems is particularly suitable for both the single and multi-day variants, thereby demonstrating the power and versatility of this decomposition scheme.

7.1 Introduction

Scheduling tasks is an important step in the rostering process, which is often solved separately since it is assumed not to be computationally practical to deal with all rostering tasks simultaneously [49]. As a result, the task scheduling problem on its own has been studied extensively in various forms, e.g. by Kolen et al. [72] and in Chapter 6.

This chapter introduces the task and shift scheduling problem (TSS). Tasks, shifts and days-off have to be assigned simultaneously to employees. Here, a task is a time interval, defined by a fixed start and end time, in which a qualified employee is required to perform an activity without interruption. A shift is a time interval in which an employee is available to execute tasks, and is defined by a fixed start and end time. Tasks can only be assigned to qualified employees who are working during the tasks' time interval. Two variants of the TSS are discussed: the single day case and the multi-day case. The single day TSS (SDTSS) denotes the problem for a single, isolated day. The multi-day TSS (MDTSS) extends the scheduling period to multiple days, thereby introducing additional complexity due to restrictions on the assignment of shifts and days-off.

Decomposition of the problem is a natural approach for exploiting the relationship between the personnel rostering and task scheduling components of the problem. In the academic literature, both exact and heuristic cut generation algorithms have been proposed for problems similar to the TSS [41, 58]. These algorithms decompose the problem by solving the task scheduling problem as a subproblem which iteratively adds constraints to the master problem, i.e. the personnel rostering problem.

Part of the present contribution builds upon the general decomposition scheme proposed in Chapter 6 for the SMPTSP. Heuristically delineated subproblems are sequentially solved to optimality. In a second phase, an improvement procedure repairs possible infeasibilities and further improves the solution by iteratively (re)assigning tasks and shifts. This chapter shows that this type of decomposition is very effective for both the SDTSS and the MDTSS. Furthermore, two exact decomposition algorithms are presented for the SDTSS.

All approaches are evaluated on large, varied benchmark datasets generated by a publicly available instance generator. The computational results are analysed to provide insight into the behaviour of different approaches to the SDTSS and MDTSS.

The remainder of this chapter is organised as follows. The main contributions are discussed in Section 7.2. Section 7.3 presents an overview of related literature and situates the TSS among similar problems. Mathematical models, dedicated solution approaches and computational experiments for the SDTSS and MDTSS are presented in Section 7.4 and Section 7.5, respectively. In Section 7.6, conclusions are formulated.

7.2 Contributions

By addressing two levels of decision making simultaneously, larger gains can be realised. The present chapter introduces the integrated task and shift scheduling problem as a new challenging combinatorial optimisation problem. Integer programming formulations are presented for two variants: problems which consider a single isolated day, and problems with a scheduling period consisting of multiple days. Instance generators for both problems are made publicly available.

To solve these new problems, exact and heuristic algorithms are proposed, based on two principles which have proven to be successful in the last decades: decomposition and matheuristics. By decomposing the problem into multiple subproblems, efficient algorithms can be used to solve these subproblems [71], for example, by combining exact and heuristic techniques in a matheuristic framework.

For single day problems, two exact approaches are presented, as well as a heuristic which builds upon the CMH introduced in Chapter 6. For problems with longer scheduling periods, two constructive heuristics are proposed, as well as an improvement heuristic. Extensive computational experiments illustrate the success of the two aforementioned algorithmic paradigms.

7.3 Related literature

Few publications focus on the assignment of non-preemptive tasks and shifts. Dowling et al. [44] assign tasks fixed in time, and flexible shifts to employees. They address the problem in two separate, sequential phases, in which first the shift roster is constructed and afterwards tasks are assigned within the shifts. The objective is to minimise over- and understaffing. Lapègue et al. [78] present a problem in which tasks are also fixed in time, but shifts are not explicitly defined. Rather, a set of guidelines is given to which a shift roster should adhere. Meisels and Schaerf [91] discuss a class of general employee timetabling problems in which both tasks and shifts are fixed in time. The main difference with the TSS is that only one task can be assigned to each shift.

Elahipanah et al. [48] discuss a task scheduling problem with both pre-emptive and non-preemptive tasks. The shift roster is considered to be part of the input. However, there is a high degree of flexibility by allowing additional shifts to be assigned or existing shifts to be modified. Various costs caused by understaffing, overtime, additional shifts and task transitions are minimised.

The largest body of related research concerns pre-emptive tasks. Robinson et al. [107] address a problem in which the pre-emptive tasks are defined by a release date and a deadline, such that their time of execution must also be determined. A tabu search algorithm is used to construct a days-off roster, after which, for each day, a network flow model generates task and shift assignments that minimise personnel costs. Brucker and Qu [16] extend this model with qualification requirements. Brucker et al. [17] study the complexity of various personnel scheduling models, including a project centred planning model that integrates pre-emptive task scheduling and work pattern assignment. The objective is to minimise the completion time of the project. The problem presented by Detienne et al. [41] requires predefined work patterns and activities to be assigned to employees. The costs associated with the assigned work patterns should be minimised. Guyon et al. [58] extend this work by introducing time windows for the tasks.

Côté et al. [36] and Musliu et al. [96] represent pre-emptive tasks as varying staffing requirements in intervals. Côté et al. [36] use implicit models with context-free grammars to model complex rules regarding shift design. Musliu et al. [96] study the minimum shift design problem in which the goal is to decide on an efficient shift structure and a minimal workforce that can carry out all the work without, however, explicitly assigning tasks within the shifts.

Table 7.1 presents an overview of related work and situates the TSS. The literature review on related problems indicates that both the SDTSS and the MDTSS present new academic challenges, while in practice both problems present a complex, time consuming task. Particularly, the combination of fixed shifts and non-preemptive tasks has not been discussed when considering task and shift scheduling simultaneously.

7.4 The single day TSS problem

The single day TSS considers the task and shift scheduling problem for an isolated day, such that there are no restrictions on the assignment of shifts to employees. First, an integer programming formulation of the SDTSS is presented. Second, exact and heuristic decomposition approaches are introduced. Finally, their performance is evaluated.

	Pre-emptive tasks	Fixed tasks	Fixed shifts	Qualifi-cations	Objective
Brucker and Qu [16]	yes	no	no	yes	scheduling costs
Brucker et al. [17]	yes	no	yes	yes	project completion time
Côté et al. [36]	yes	yes	no	yes	scheduling costs
Detienne et al. [41]	yes	yes	yes	yes	scheduling costs
Dowling et al. [44]	no	yes	no	yes	deviation from coverage requirements
Elahipanah et al. [48]	both	no	no	yes	scheduling costs
Guyon et al. [58]	yes	no	yes	yes	scheduling costs
Krishnamoorthy et al. [77]	no	yes	yes	yes	number of employees
Lapègue et al. [78]	no	yes	no	yes	deviation from targeted workload
Meisels and Schaerf [91]	no	yes	yes	yes	scheduling costs
Musliu et al. [96]	yes	yes	no	no	number of different shifts
Robinson et al. [107]	yes	no	no	no	scheduling costs
This chapter	no	yes	yes	yes	scheduling costs and number of assigned shifts

Table 7.1: Overview of related work

7.4.1 Integer programming formulation

In the integer programming formulation of the SDTSS, the maximal cliques in an interval graph are used to efficiently represent overlapping tasks. This concept was also used by Krishnamoorthy et al. [77] and in Chapter 6 to model the SMPTSP. An interval graph $G = (V, A)$ is constructed with one node for each task. Two nodes are connected if the corresponding tasks overlap in time. A maximal clique in G represents a set of pairwise overlapping tasks. It is clear that these tasks should all be assigned to different employees.

Parameters

T	set of tasks
S	set of working shifts
E	set of employees
E_t	set of employees qualified for task t
S_t	set of shifts in which task t can be performed
C_e	set of all maximal cliques in the interval graph for employee e

Decision variables

$$x_{te} = \begin{cases} 1 & \text{if task } t \text{ is assigned to employee } e \\ 0 & \text{otherwise} \end{cases}$$

$$y_{es} = \begin{cases} 1 & \text{if employee } e \text{ is assigned to shift } s \\ 0 & \text{otherwise} \end{cases}$$

Model

$$min \sum_{e \in E} \sum_{s \in S} y_{es} \tag{7.1}$$

$$s.t. \sum_{e \in E_t} x_{te} = 1 \qquad \forall\, t \in T \tag{7.2}$$

$$\sum_{t \in K} x_{te} \leq 1 \qquad \forall\, e \in E,\ K \in C_e \tag{7.3}$$

$$\sum_{s \in S} y_{es} \leq 1 \qquad \forall\, e \in E \tag{7.4}$$

$$\sum_{s \in S_t} y_{es} \geq x_{te} \qquad \forall\, t \in T,\ e \in E \tag{7.5}$$

$$y_{es} \in \{0, 1\} \qquad \forall\, e \in E,\ s \in S \tag{7.6}$$

$$x_{te} \in \{0, 1\} \qquad \forall\, t \in T,\ e \in E \tag{7.7}$$

The objective function (7.1) minimises the number of shifts assigned to employees. Constraints (7.2) make sure that each task is assigned to a qualified employee. Overlapping task assignments are forbidden by Constraints (7.3). Constraints (7.4) require that an employee cannot work more than one shift. Note that, since the set of shifts S does not contain a dummy shift representing a day-off, the left hand side of Constraints (7.4) is not required to be equal to one. When $\sum_{s \in S} y_{es} = 0$, employee e is not working. Constraints (7.5) link the x and y decision variables by stating that tasks can only be assigned to employees who are assigned to a shift in which it can be performed. Finally, Constraints (7.6) and (7.7) impose bounds on the decision variables.

Note that shift breaks are not explicitly included in the model. However, if the breaks are fixed in time, they can be modelled by removing the shifts of which the break overlaps with task t from the set S_t.

7.4.2 Infeasible task pair decomposition

The infeasible task pair decomposition first assigns tasks to employees and then allocates shifts. Algorithm 5 outlines the approach.

Algorithm 5 Infeasible task pair decomposition

```
Input: SDTSS problem instance
Output: Optimal solution
1: Determine infeasible task combinations
2: Solve the task assignment problem taking into account the infeasible task
   combinations
3: Solve the shift assignment problem given an optimal task assignment
```

The main idea is to decompose the problem into an optimisation master problem (task assignment) and a feasibility subproblem (shift assignment). After an optimal solution for the master problem is found, a feasible solution for the subproblem is constructed. As the master problem is solved to optimality, Algorithm 5 returns an optimal solution for the SDTSS.

This approach is inspired by combinatorial Benders decomposition [33] which, in general, generates an exponential number of cuts in the subproblem. A preprocessing step is introduced that adds a polynomial number of combinatorial Benders cuts to the master problem. These cuts dominate all other cuts. As a consequence, the master and subproblems do not need to be solved repeatedly.

Preprocessing

The first step of Algorithm 5 identifies all infeasible task combinations, which are then used as constraints in the second step. A set of tasks is infeasible if there exists no shift that covers the intervals of all tasks in the set. The start and end times of task t and shift s are denoted by a_t and b_t, and v_s and w_s, respectively. Formally, a set of tasks R is infeasible if:

$$\nexists\ s \in S:\ v_s \leq \min\{a_t,\ \forall\ t \in R\} \text{ and } w_s \geq \max\{b_t,\ \forall\ t \in R\}$$

The number of combinations of n tasks is exponential in n, and cannot be checked efficiently for large instances. However, the number of pairs (i.e. combinations of size two) is limited, and can thus all be computed in limited time. More precisely, the total number of task pairs p that need to be checked is:

$$p = \frac{|T|(|T|-1)}{2}$$

By verifying only pairs of tasks, infeasible task combinations of any size are eliminated. This can be easily understood since a set of tasks is infeasible due to the earliest start time and the latest end time being too far apart to be covered by one shift. All tasks between the first and last assigned task can be ignored when determining whether a combination is infeasible or not. Thus, by only checking pairs of tasks, instead of combinations of all sizes, all possibly infeasible combinations are covered.

The task assignment problem

After the infeasible task pairs are identified, a task assignment problem is solved. The model is identical to that of the SMPTSP presented in Chapter 6 (model (6.1) - (6.5)), with additional constraints to ensure that the infeasible task pairs identified in the preprocessing step are not assigned to the same employee. For each employee e, let I_e be the set of all infeasible task pairs identified during preprocessing.

Decision variables

$$x_{te} = \begin{cases} 1 & \text{if task } t \text{ is assigned to employee } e \\ 0 & \text{otherwise} \end{cases}$$

$$y_e = \begin{cases} 1 & \text{if employee } e \text{ is assigned to at least one task} \\ 0 & \text{otherwise} \end{cases}$$

Model

$$min \sum_{e \in E} y_e \tag{7.8}$$

$$s.t. \sum_{e \in E_t} x_{te} = 1 \qquad \forall\, t \in T \tag{7.9}$$

$$\sum_{t \in K} x_{te} \leq y_e \qquad \forall\, e \in E,\ K \in C_e \tag{7.10}$$

$$\sum_{t \in R} x_{te} \leq 1 \qquad \forall\, e \in E,\ R \in I_e \tag{7.11}$$

$$x_{te} \in \{0,1\} \qquad \forall\, t \in T,\ e \in E \tag{7.12}$$

$$y_e \in \{0,1\} \qquad \forall\, e \in E \tag{7.13}$$

The objective function (7.8) states that the number of active employees should be minimised. Constraints (7.9) ensure that each task is performed by exactly one employee, and exclude infeasible assignments in terms of qualifications. Constraints (7.10) make sure that overlapping tasks are not assigned to the same employee. Constraints (7.11) prevent infeasible combinations of tasks to be assigned to the same employee. Note that I_e only contains sets of size two, meaning that these constraints allow at most one of two tasks from each set $R \in I_e$ to be assigned to the same employee. Finally, Constraints (7.12) and (7.13) set bounds for the decision variables.

The shift assignment problem

Based on a solution of the task assignment problem, a feasible shift assignment is constructed as follows. For each active employee, determine the first shift that covers the interval defined by the earliest start time and the latest end time of the assigned tasks. By taking into account the infeasible task pairs when solving the task assignment problem (Constraints (7.11)), a feasible shift will always be found.

7.4.3 Augmented interval graph decomposition

The second step of the infeasible task pair decomposition uses Constraints (7.10) and (7.11) to ensure that conflicting tasks are not assigned to the same employee. These two causes of conflict are considered separately, allowing polynomial time algorithms to be used for constructing the sets C_e and I_e. However, it is possible to model both causes of conflict in one graph by augmenting the interval graph G with additional arcs. Maximal cliques in this augmented interval graph can then result in stronger clique constraints of the form of Constraints (7.10). Algorithm 6 outlines this approach. Note that this approach also returns optimal solutions.

Algorithm 6 Augmented interval graph decomposition

```
Input: SDTSS problem instance
Output: Optimal solution
1: Generate the augmented interval graph $G'$
2: Find all maximal cliques in the augmented interval graph
3: Solve the task assignment problem
4: Solve the shift assignment problem given an optimal task assignment
```

The augmented interval graph $G' = (V, A)$ is constructed with one node for each task. Arcs are added between two nodes in two cases: 1) if the corresponding tasks overlap in time, and 2) if the corresponding tasks cannot be performed within one shift by the same employee. A maximal clique in G' is then a set of conflicting tasks that should be assigned to different employees. All maximal cliques in G' can be found with the Bron-Kerbosch algorithm [14].

The task assignment problem solved at line 3 of Algorithm 6 is identical to solving model (7.8) - (7.13), without Constraints (7.11). Finally, the procedure in the last step is the same as the last step in the infeasible task pair decomposition.

This approach resembles a graph colouring reformulation of the problem. A list colouring problem on G' corresponds to solving the task assignment problem at line 3 of Algorithm 6. The qualifications of the employees define the colour lists of each node.

7.4.4 Horizontal decomposition

When the number of employees and/or tasks increases, exact approaches fail to produce optimal or even feasible solutions within acceptable calculation time. Therefore, an alternative decomposition approach is presented, based on the CMH algorithm introduced in Chapter 6.

Algorithm 7 shows the pseudocode for the horizontal decomposition approach. It is a constructive heuristic which builds a solution by sequentially solving subproblems. In each subproblem, tasks and shifts are assigned to a subset of b randomly selected employees. Note that, for cases in which there is a clear structure in the distribution of skills (e.g. hierarchical), a different, tailor-made selection approach could be more appropriate.

Algorithm 7 Horizontal decomposition approach

```
Input:
    $T$ := set of tasks to be assigned
    $E$ := set of employees
    $b$ := number of employees in one subproblem
Output: $x$                                        ▷ partial or complete solution
 1: $x$ := empty solution
 2: while $E \neq \emptyset$ or not all tasks assigned do
 3:     $E'$ := sample and remove $b$ employees from $E$     ▷ delineate the subproblem
 4:     Solve the subproblem for $E'$ and append the solution to $x$
 5:     Remove the tasks assigned to $E'$ from $T$
 6: end while
```

The size of the subproblems is determined by the parameter b. In general, a higher value of b results in better solutions, but it also increases the calculation

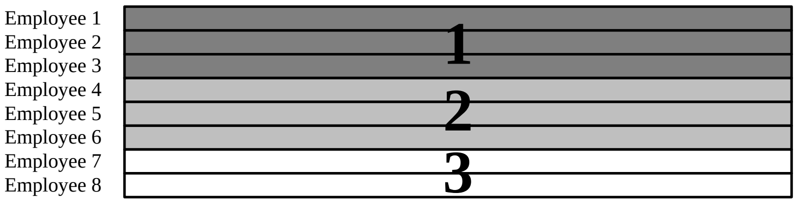


Figure 7.1: Horizontal decomposition of an SDTSS instance with eight employees and block size $b = 3$

time [116]. The decision variables of each subproblem are the same as in the model on page 130. Let u_t be the duration of task t, E' the employees considered in the subproblem and T' the tasks that have not yet been assigned. The model solved at line 4 of Algorithm 7 can then be formulated as:

$$max \sum_{e \in E'} \sum_{t \in T'} u_t x_{te}$$

$$s.t. \sum_{e \in E'_t} x_{te} \leq 1 \qquad \forall\, t \in T'$$

$$\text{Constraints (7.3) - (7.7)} \qquad \text{with } E = E' \text{ and } T = T'$$

Figure 7.1 illustrates the decomposition into subproblems with $b = 3$, for an instance with eight employees. Three subproblems are delineated. The first subproblem is defined over employees $\{1, 2, 3\}$, the second subproblem considers employees $\{4, 5, 6\}$ and the third subproblem only deals with the remaining two employees $\{7, 8\}$.

As is the case for CMH in Chapter 6, the result of applying Algorithm 7 might be a partial solution in which not all tasks are assigned. To fix such infeasible schedules, a second algorithm is applied which builds upon the LBIH algorithm presented in Chapter 6. The repair heuristic (re)assigns a limited number of tasks until a feasible solution is found, or until a time limit is reached. In this process, the impact on the quality of the solution obtained by Algorithm 7 is minimised. Algorithm 8 shows the repair procedure.

The Hamming distance model solved at line 3 of Algorithm 8 is the mathematical model presented on page 130, with the addition of the asymmetric Hamming

Algorithm 8 Repair procedure

```
Input:
    $\bar{x}$ := an infeasible solution
    $k'$ := maximum number of tasks to reassign
Output: $x$                                          ▷ (feasible) solution
1: infeasible := true              ▷ boolean indicating if the solution is feasible
2: while infeasible and time limit not reached do
3:     $x$ := solve the Hamming distance model with $k'$ given $\bar{x}$
4:     if $x$ is feasible then
5:         infeasible := false
6:     else
7:         $k' := k' + 1$
8:     end if
9: end while
```

distance constraint (7.14) [52].

$$\sum_{t \in T} \sum_{e \in E} \bar{x}_{te}(1 - x_{te}) \leq k' \tag{7.14}$$

This constraint limits the number of variables that can change value with respect to a given reference solution $\bar{x}$. In the context of the SDTSS, this constraint ensures that an integer programming solver will (re)assign at most k' tasks. Due to the restriction of the search space imposed by this constraint, the model yields a feasible solution much faster than the original model. Note that k' should be at least as large as the number of unassigned tasks, and possibly larger, to repair a partial solution. Adapting k' to this condition is done automatically at line 7 of Algorithm 8.

Note that the Hamming distance constraint applies to the x variables. Another possibility would be to restrict the number of y variables that can change value. However, intuitively, the assignment of shifts follows from the task assignment. The latter assignment is the most constrained and the most difficult to satisfy. Therefore, focusing on these variables is the most efficient way to a feasible solution.

7.4.5 Computational evaluation

The performance of the different approaches was analysed during extensive computational experiments on a large, varied dataset. First, details on the

Characteristic		Settings
Number of employees	n	20, 40, 60, 80, 100
Skilling	s	0.3, 0.6, 1.0
Tightness	t	0.6, 0.9

Table 7.2: Instance characteristics for the SDTSS dataset

experimental setup are presented. Afterwards the computational results are presented and analysed.

Data and experimental setup

Table 7.2 shows the different values of the characteristics of the instances in the benchmark dataset[10].

Skilling is the percentage of tasks for which, on average, an employee is qualified. For example, $s = 0.6$ means that each employee can, on average, perform 60% of all tasks. $s = 1$ denotes that there are no qualification requirements and all employees can perform all tasks. During instance generation, task qualifications are distributed randomly among employees, resulting in a random skill structure.

The *tightness* of an instance is a parameter of the instance generation procedure to control the number of tasks. It is a measure for how many tasks will be performed in each assigned shift. When the number of employees is kept constant, a higher tightness leads to more tasks in an instance.

The shift structure is the same in all instances, consisting of three partially overlapping shifts of equal duration. Overlapping shifts are particularly challenging since, otherwise, the task assignment problem could be solved per shift, instead of for the entire day. It is conjectured that algorithms performing well for such a shift structure will also be capable of addressing instances with less or no shift overlap.

A full factorial experimental design resulted in $5 \times 3 \times 2 = 30$ instance classes. For each class, ten random instances were generated, resulting in a dataset of 300 instances. The subproblems in the horizontal decomposition approach were constructed with $b = 5$. The repair procedure was initiated with $k' = 40$, based on preliminary experiments and computational results from Chapter 6. The time limit was set to 1800 seconds. For each instance, ten repeated runs were executed. All experiments were carried out on a Dell Poweredge T620, 2x Intel

[10]The instance generator is publicly available at `http://gent.cs.kuleuven.be/tss.html`.

Xeon E5-2670, 128GB RAM. Gurobi 5.6.2 was used as an integer programming solver, configured to use one thread and default settings.

Computational results

Table 7.3 shows computational results for the different algorithms as averages of all instances per class. Four approaches are compared: the integer programming solver on the model from Section 7.4.1 (*MIP*), the infeasible task pair decomposition (*Task pair decomp.*), the augmented interval graph decomposition (*Interval graph decomp.*), and the heuristic horizontal decomposition approach (*Horizontal decomp*). For each of these approaches, the following data is shown: the objective value of the obtained solution (*Obj.*), the percentage of instances in the class for which a feasible solution was found (*Feas.*), and the computation time as elapsed wall time in seconds (*Time*). If an approach failed to find a feasible solution for at least one instance in the class, the objective value is annotated with an asterisk. The best results for each instance class are highlighted in bold.

Furthermore, a lower bound (*LB*) is given, calculated as shown in Equation (7.15), with LB_{MIP} the lower bound of the integer programming solver after 1800 seconds, $LB_{task\ pair}$ the integer programming solver's lower bound of the task assignment problem in the infeasible task pair decomposition after 1800 seconds, and $LB_{interval}$ the integer programming solver's lower bound after 1800 seconds when solving the task assignment problem in the augmented interval graph decomposition.

$$LB = \max(LB_{MIP}, LB_{task\ pair}, LB_{interval}) \tag{7.15}$$

Table 7.4 summarises the computational results. The infeasible task pair decomposition finds feasible solutions for 97% of the instances as well as the highest number of best solutions among the four approaches. The horizontal decomposition, on the other hand, is always capable of finding a feasible solution within the time limit, often very close to the (optimal) lower bound. The difference between the task pair decomposition and augmented interval graph decomposition is small. However, the former does find a higher number of feasible and best solutions in, on average, less computation time.

For instances with a high number of employees and a high tightness, the horizontal decomposition performs better than the other approaches. The other approaches fail to consistently find feasible solutions for these instances. Particularly, when $n > 40$ and $t = 0.9$, the full integer programming model does not always produce a feasible solution.

Instance				MIP			Task pair decomp.			Interval graph decomp.			Horizontal decomp.		
n	s	t	LB	Obj.	Feas.	Time (s)	Obj.	Feas.	Time (s)	Obj.	Feas.	Time (s)	Obj.	Feas.	Time (s)
20	0.3	0.6	15.5	**15.5**	100%	0.2	**15.5**	100%	0.1	**15.5**	100%	0.1	17.9	100%	0.1
		0.9	18.0	**18.0**	100%	0.4	**18.0**	100%	0.3	**18.0**	100%	0.3	20.5	100%	1.2
	0.6	0.6	15.2	**15.2**	100%	0.3	**15.2**	100%	0.3	**15.2**	100%	0.4	15.8	100%	0.9
		0.9	18.0	**18.0**	100%	9.1	**18.0**	100%	1.9	**18.0**	100%	2.6	19.6	100%	2.3
	1	0.6	15.3	**15.3**	100%	8.4	**15.3**	100%	0.6	**15.3**	100%	0.6	15.4	100%	3.6
		0.9	18.0	**18.0**	100%	129.7	**18.0**	100%	2.3	**18.0**	100%	5.5	18.2	100%	9.9
40	0.3	0.6	30.9	**30.9**	100%	118.0	**30.9**	100%	3.1	**30.9**	100%	2.6	35.4	100%	1.2
		0.9	38.9	**39.0**	100%	802.3	39.1	100%	399.2	39.2	100%	412.1	44.0	100%	12.5
	0.6	0.6	30.9	**30.9**	100%	685.8	**30.9**	100%	6.7	**30.9**	100%	9.3	31.4	100%	5.0
		0.9	38.9	**38.9**	100%	1800.0	**38.9**	100%	94.5	**38.9**	100%	135.4	40.4	100%	13.3
	1	0.6	30.5	**30.5**	100%	1800.0	**30.5**	100%	13.1	**30.5**	100%	32.1	**30.5**	100%	20.1
		0.9	38.9	**38.9**	100%	1800.0	**38.9**	100%	163.6	**38.9**	100%	396.4	39.0	100%	67.9
60	0.3	0.6	46.9	**46.9**	100%	936.3	**46.9**	100%	23.4	**46.9**	100%	17.7	51.3	100%	3.0
		0.9	59.4	*61.8	90%	1800.0	60.1	100%	1201.2	**59.9**	100%	1161.4	65.0	100%	10.6
	0.6	0.6	45.8	**45.8**	100%	1800.0	**45.8**	100%	108.0	**45.8**	100%	102.5	46.5	100%	13.7
		0.9	59.6	*69.6	50%	1800.0	**59.6**	100%	1216.7	*71.4	50%	1800.0	60.8	100%	39.0
	1	0.6	46.1	**46.1**	100%	1800.0	**46.1**	100%	186.3	**46.1**	100%	624.6	46.5	100%	64.2
		0.9	59.5	*73.8	60%	1800.0	*72.9	70%	1800.0	*73.6	70%	1800.0	**59.6**	100%	158.3
80	0.3	0.6	58.7	**58.7**	100%	1800.0	**58.7**	100%	346.7	**58.7**	100%	114.3	63.7	100%	5.1
		0.9	76.7	*90.5	20%	1800.0	79.0	100%	1757.0	**77.6**	100%	1678.4	83.7	100%	17.5
	0.6	0.6	57.8	60.9	100%	1800.0	**57.8**	100%	553.5	**57.8**	100%	680.1	58.3	100%	30.5
		0.9	68.0	*93.5	40%	1800.0	82.7	100%	1800.0	*97.8	60%	1800.0	**77.7**	100%	77.2
	1	0.6	52.9	60.9	100%	1800.0	84.4	100%	1695.3	79.2	100%	1304.2	**59.1**	100%	112.9
		0.9	56.5	-	0%	1800.0	87.8	100%	1800.0	89.7	100%	1800.0	**77.5**	100%	389.3
100	0.3	0.6	74.3	79.9	100%	1800.0	**74.3**	100%	436.5	**74.3**	100%	439.8	78.3	100%	26.4
		0.9	89.4	*123.3	30%	1800.0	*111.9	70%	1800.0	*123.3	40%	1800.0	**104.1**	100%	31.5
	0.6	0.6	72.6	80.8	100%	1800.0	79.0	100%	1447.1	110.1	100%	1687.5	**73.7**	100%	56.3
		0.9	65.1	*126.2	50%	1800.0	*122.3	80%	1800.0	*123.1	80%	1800.0	**97.9**	100%	172.8
	1	0.6	66.8	93.3	100%	1800.0	113.7	100%	1800.0	104.0	100%	1730.0	**74.6**	100%	235.0
		0.9	63.8	*120.0	10%	1800.0	112.7	100%	1800.0	117.4	100%	1800.0	**97.3**	100%	761.9

Table 7.3: Comparison of different approaches to the SDTSS, as averages of all instances per class

	MIP	Task pair decomp.	Interval graph decomp.	Horizontal decomp.
Number of feasible solutions	245	292	280	300
Number of best solutions	172	218	196	127

Table 7.4: Summary of computational results for the SDTSS, out of 300 instances

Comparing the required computation time, once $n \geq 40$, the general purpose solver cannot complete its search before the time limit is reached. For both the task pair decomposition and the augmented interval graph decomposition, this only occurs for instances with 80 employees or more. Finally, the heuristic decomposition clearly terminates its search more quickly than the other approaches.

Figure 7.2 allows for a more detailed analysis of the performance difference between two decomposition approaches. The chart shows the relative gap in objective value obtained by the horizontal decomposition and the task pair decomposition, as calculated by Equation (7.16). Negative values indicate that the horizontal decomposition found better solutions than the task pair decomposition. Each data point corresponds to one of the 300 instances for which both approaches found a feasible solution. The vertical lines categorise the instances based on the number of employees. Within each of these categories, the instances are lexicographically ordered by ascending skilling level and tightness.

$$\delta = \frac{\text{Horizontal decomp.} - \text{Task pair decomp.}}{\text{Task pair decomp.}} \times 100 \tag{7.16}$$

This data visualisation reveals the following three insights: 1) the horizontal decomposition scales better than the task pair decomposition with an increasing number of employees, 2) the tightness has the same influence on both approaches, and 3) the horizontal decomposition performs better than the task pair decomposition when employees are qualified for a higher number of tasks. The latter can be explained by the fact that it becomes easier for the horizontal decomposition to construct solutions when more employees are qualified for each task, as was observed in Chapter 6. Furthermore, symmetry in the mathematical model could influence the time required to solve the mathematical model. The model, and thus the consequences of symmetry, in the task pair decomposition are typically much larger than the subproblems to be solved in the horizontal decomposition method.

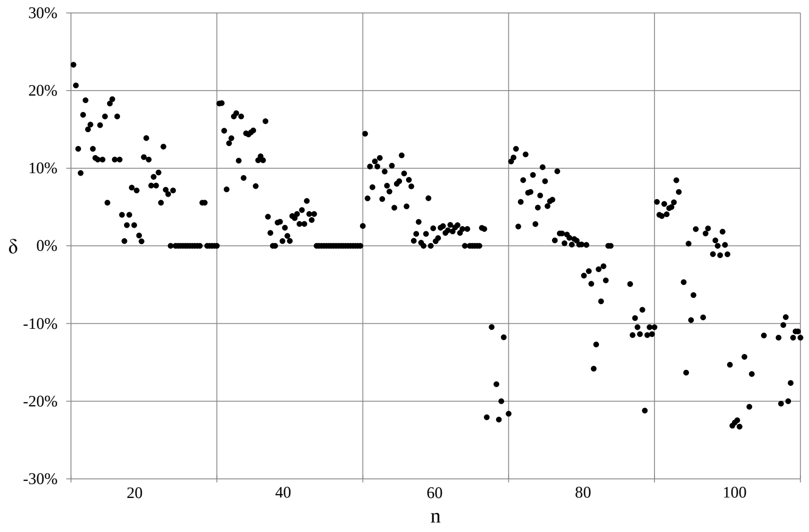


Figure 7.2: Relative gap δ (Equation (7.16)) between the results obtained by the task pair decomposition and the horizontal decomposition, for varying number of employees n

7.5 The multi-day TSS problem

The problem definition of the SDTSS can be extended to multiple days. With the addition of the dimension *day*, shift assignments are now restricted by time related constraints, such as forbidden shift successions and maximum number of days worked in the scheduling period. First, an integer programming formulation for the MDTSS is presented. Afterwards, two constructive heuristics and one improvement heuristic are proposed. The effect of decomposing the problem into either horizontal or vertical subproblems on the solution quality is investigated.

7.5.1 Integer programming formulation

Let D be the set of days in the scheduling period, and let $D' \subset D$ be the set containing all Saturdays in D. The model assumes that the last day in D is never a Saturday, i.e. the scheduling period always includes full weekends (Saturday and Sunday). The day on which task t is to be performed is denoted by d_t. For modelling purposes, the set of shifts S now also includes a dummy shift s_0 which represents a day-off. The expression $S \setminus \{s_0\}$ refers to the set of working shifts in which tasks can be performed.

Decision variables

$$x_{te} = \begin{cases} 1 & \text{if task } t \text{ is assigned to employee } e \\ 0 & \text{otherwise} \end{cases}$$

$$y_{esd} = \begin{cases} 1 & \text{if employee } e \text{ is assigned to shift } s \text{ on day } d \\ 0 & \text{otherwise} \end{cases}$$

Model

$$min \text{ days worked} + \sum \text{soft constraint penalty} \tag{7.17}$$

$$s.t. \sum_{e \in E_t} x_{te} = 1 \qquad \forall\, t \in T \tag{7.18}$$

$$\sum_{t \in K} x_{te} \leq 1 \qquad \forall\, e \in E,\; K \in C_e \tag{7.19}$$

$$\sum_{s \in S} y_{esd} = 1 \qquad \forall\, e \in E, d \in D \tag{7.20}$$

$$\sum_{s \in S_t} y_{esd_t} \geq x_{te} \qquad \forall\, t \in T,\; e \in E \tag{7.21}$$

$$y_{esd} \in \{0,1\} \qquad \forall\, e \in E,\; s \in S, d \in D \tag{7.22}$$

$$x_{te} \in \{0,1\} \qquad \forall\, t \in T,\; e \in E \tag{7.23}$$

Constraints (7.18) ensure that each task is assigned to a qualified employee. Constraints (7.19) make sure that there are no overlapping task assignments. Constraints (7.20) require an employee to be assigned to one shift (working or dummy) per day. Constraints (7.21) enforce tasks to only be assigned to employees who are working a suitable shift on the tasks' day of execution. Constraints (7.22) and (7.23) require all decision variables to be binary.

The objective function (7.17) minimises the cost of the schedule, consisting of two parts: 1) the number of days necessary to perform all tasks, and 2) a weighted sum of soft constraint violations. The latter includes different time related constraints, restricting the assignment of shifts (Table 7.5). Typically, in practice, it is impossible to respect all time related constraints since they are often imposed by authorities with conflicting priorities. The general methodology to deal with such constraints is by considering them as soft constraints, and penalising violations in the objective function. Additional variables p^c count the degree of violation of each constraint c. These variables are added to the objective function with a weight corresponding to their relative importance.

7.5.2 Horizontal decomposition

Due to the intricate dependencies between the task and shift assignments, even constructing a feasible solution for the MDTSS presents a challenge. This is further complicated by the tasks' qualification requirements. Two heuristic decomposition approaches have been developed to construct solutions, as well as a heuristic to improve these initial solutions. This section details a heuristic which decomposes the problem horizontally (employee-based). Section 7.5.3 presents an algorithm which delineates subproblems vertically (day-based). Due to the aforementioned feasibility issues, both approaches build upon the idea of simultaneously assigning tasks and shifts to efficiently find feasible solutions. The problem is solved partly with an integer programming solver, which strongly facilitates constructing a feasible solution for this problem.

The horizontal decomposition of the MDTSS is similar to the CMH algorithm presented on page 111 in Chapter 6. Subproblems consisting of b employees are sequentially solved to optimality. The employees in each subproblem are selected randomly without taking into account any instance specific information.

The decision variables in the subproblems are the same as in the model on page 143. Furthermore, let u_t be the duration of task t, l_s the duration of shift s, E' the employees considered in the subproblem, T' the tasks that are still unassigned, and $T'_{ed} \subseteq T'$ the unassigned tasks on day d for which employee e is qualified. The model of the subproblem can then be formulated as:

Description	Formulation	
Max η^1 working days	$\sum_{d \in D} \sum_{s \in S \setminus \{s_0\}} y_{esd} - p_e^1 \leq \eta^1$	$\forall\ e \in E$
Min η^2 working days	$\sum_{d \in D} \sum_{s \in S \setminus \{s_0\}} y_{esd} + p_e^2 \geq \eta^2$	$\forall\ e \in E$
Max η_s^3 assignments of shift s	$\sum_{d \in D} y_{esd} - p_{es}^3 \leq \eta_s^3$	$\forall\ e \in E$
Min η_s^4 assignments of shift s	$\sum_{d \in D} y_{esd} + p_{es}^4 \geq \eta_s^4$	$\forall\ e \in E$
Max η^5 consecutive working days	$\sum_{s \in S \setminus \{s_0\}} \sum_{t=0}^{\eta^5} y_{es(d+t)} - p_{ed}^5 \leq \eta^5$	$\forall\ e \in E,\ d \in [0, \|D\| - \eta^5[$
Max η^6 consecutive days-off	$\sum_{t=0}^{\eta^6} y_{es_0(d+t)} - p_{ed}^6 \leq \eta^6$	$\forall\ e \in E,\ d \in [0, \|D\| - \eta^6[$
Complete weekends	$\sum_{s \in S \setminus \{s_0\}} (y_{esd} - y_{es(d+1)}) - p_{ed}^7 + p_{ed}^{7'} = 0$	$\forall\ e \in E,\ d \in D'$
No isolated day-off	$y_{es_0(d-1)} - y_{es_0 d} + y_{es_0(d+1)} + p_{ed}^8 \geq 0$	$\forall\ e \in E,\ d \in [1, \|D\| - 1[$
Shift succession (s, s')	$y_{esd} + y_{es'(d+1)} - p_{ed}^9 \leq 1$	$\forall\ e \in E,\ d \in [0, \|D\| - 1[$

Table 7.5: Time related constraints in model (7.17) - (7.23)

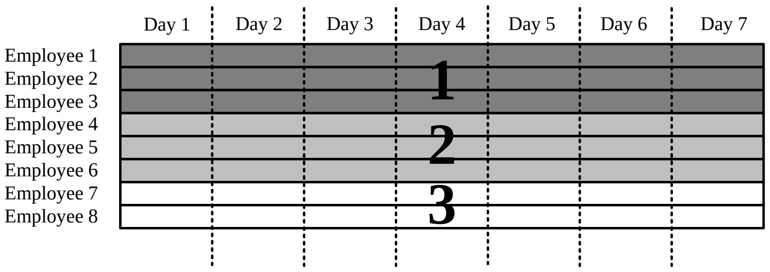


Figure 7.3: Horizontal decomposition of an MDTSS instance with eight employees and a block size $b = 3$

$$min \sum_{e \in E'} \sum_{d \in D} (\sum_{s \in S} l_s y_{esd} - \sum_{t \in T'_{ed}} u_t x_{te}) +$$

$$\sum \text{soft constraint penalty}$$

$$s.t. \sum_{e \in E'_t} x_{te} \leq 1 \qquad \forall\, t \in T'$$

$$\text{Constraints (7.19) - (7.23)} \qquad \text{with } E = E' \text{ and } T = T'$$

Figure 7.3 illustrates how these subproblems are delineated for an instance with eight employees and seven days. As mentioned on page 136, it is possible that this method returns an infeasible partial solution, i.e. a solution in which some tasks remain unassigned. These infeasible assignments are afterwards repaired by an improvement heuristic (Section 7.5.4).

7.5.3 Vertical decomposition

Constantino et al. [35] decompose a nurse rostering problem into one assignment problem for each day of the scheduling period. The vertical decomposition approach presented here is based on this idea and decomposes the problem into one subproblem per day, taking into account a large variety of soft constraints. Based on shift assignments from preceding days, a cost is calculated for assigning a particular shift to an employee such that expected violations of the soft constraints are minimised.

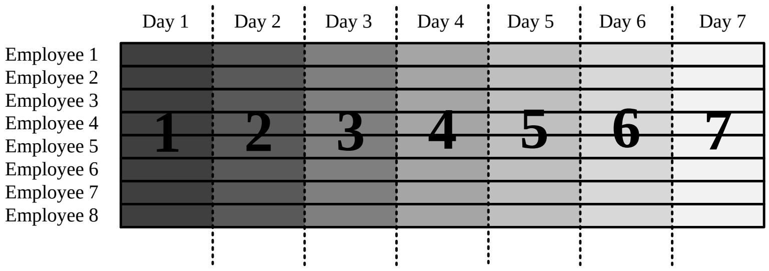


Figure 7.4: Vertical decomposition of an MDTSS instance with eight employees and seven days

Algorithm 9 outlines the vertical decomposition approach. An example application of this vertical decomposition is shown in Figure 7.4 for an instance with eight employees and seven days.

Algorithm 9 Vertical decomposition for the MDTSS

```
Input:
    $D$ := set of days in the scheduling period
    $E$ := set of employees
    $S$ := set of shifts
Output: $x$                                    ▷ partial or complete solution
 1: $x$ := empty solution
 2: for all $d \in D$ do
 3:     for all $e \in E$ do
 4:         for all $s \in S$ do
 5:             Calculate cost for assigning shift $s$ on day $d$ to employee $e$ (Algorithm
    10)
 6:         end for
 7:     end for
 8:     $x_d$ := solve the subproblem for day $d$
 9:     if $x_d = null$ then
10:         Assign $s_0$ to each employee on day $d$     ▷ assign days-off to all employees
11:     end if
12:     Append $x_d$ to $x$
13: end for
```

The vertical subproblems are similar to the SDTSS, apart from the addition of a dummy shift representing an idle day, and the objective function which now includes a cost c_{esd} for assigning shift s to employee e on day d. Let T_d be the tasks to be performed on day d. The subproblem for day d, which is solved at line 8 of Algorithm 9, can be formulated as:

$$
\begin{aligned}
& min \sum_{e \in E} \sum_{s \in S \setminus \{s_0\}} c_{esd} y_{es} \\
& s.t. \sum_{s \in S} y_{esd} = 1 && \forall\ e \in E \\
& \qquad \sum_{s \in S_t} y_{esd} \geq x_{te} && \forall\ t \in T_d,\ e \in E \\
& \qquad y_{esd} \in \{0, 1\} && \forall\ e \in E,\ s \in S \\
& \qquad \text{Constraints (7.2), (7.3), (7.7)}
\end{aligned}
$$

The shift assignment costs c_{esd} for day d are calculated before the subproblem corresponding to this day is solved. Algorithm 10 shows the pseudocode for this procedure, which is executed at line 5 of Algorithm 9. The costs are calculated based on the sequence of shifts assigned to employee e on the days preceding d. For each constraint, the algorithm determines whether assigning shift s on day d results in a violation or not, and appropriately adjusts c_{esd} with either the weight of the constraint, or with zero. Clearly, this approach can be generalised to include many other personnel rostering constraints which are not considered in the current study.

Figure 7.5 illustrates this approach with an example for the *maximum* η_s^3 *assignments of shift s* constraint from Table 7.5. The parameters of this constraint state that an employee can work at most three early (E) shifts. Suppose that an employee is already assigned to three early shifts. This means that the employee should avoid working another early shift, or else a penalty is incurred. This is expressed by assigning a high cost to the early shift, and no costs to any of the other shifts. By minimising the shift assignment costs when solving the subproblem, the assignment of another early shift is avoided.

7.5.4 Improvement heuristic

Initial solutions constructed by the decomposition heuristics are improved with the LBIH algorithm presented in Chapter 6. This improvement heuristic tries to iteratively improve (or repair, if necessary) the solution by reassigning a limited number of tasks. The initial solution is generated by one of the decomposition methods. Iteratively, the method solves the Hamming distance model using an integer programming solver. When there is no improvement in two consecutive iterations, or when the time limit is reached, the algorithm stops.

Algorithm 10 Shift assignment cost calculation

```
Input:
    $e$ := employee for which the shift assignment cost is calculated
    $\bar{s}$ := shift for which the assignment cost is calculated
    $d$ := day up to which shifts have been assigned
Output: $c_{e\bar{s}d}$                                    ▷ shift assignment cost
 1: $c_{e\bar{s}d} := 0$
    ▷ Min/max working days
 2: $C_t$ := calculate number of working days
 3: if $C_t \geq$ max allowed and $\bar{s}$ is not an idle shift then
 4:     $c_{e\bar{s}d}$ += $(C_t -$max allowed$+1) \times$weight ‘max working days’
 5: end if
 6: if $C_t <$ min allowed and $\bar{s}$ is an idle shift then
 7:     $c_{e\bar{s}d}$ += (min allowed$-C_t) \times$weight ‘min working days’
 8: end if
    ▷ Min/max assignments per shift type
 9: for all $s \in S$ do
10:     $C_s$ := calculate number of assignments per shift type $s$
11:     if $C_s \geq$ max allowed and $s = \bar{s}$ then
12:         $c_{e\bar{s}d}$ += $(C_s -$max allowed$+1) \times$weight ‘max assignments of shift $s$’
13:     end if
14:     if $C_s <$ min allowed and $s \neq \bar{s}$ then
15:         $c_{e\bar{s}d}$ += (min allowed$-C_s) \times$weight ‘min assignments of shift $s$’
16:     end if
17: end for
    ▷ Max consecutive working days/days-off
18: if $(d-1)$ is a working day then
19:     $L_w$ := calculate length of ongoing stretch of working days
20:     if $L_w \geq$ max allowed consecutive working days and $\bar{s}$ is not an idle shift then
21:         $c_{e\bar{s}d}$ += $(L_w -$max allowed$+1) \times$weight ‘max consecutive working days’
22:     end if
23: else
24:     $L_o$ := calculate length of ongoing stretch of days-off
25:     if $L_o \geq$ max allowed consecutive days-off and $\bar{s}$ is an idle shift then
26:         $c_{e\bar{s}d}$ += $(L_o -$max allowed$+1) \times$weight ‘max consecutive days-off’
27:     end if
28: end if
    ▷ Isolated days
29: if $(d-2)$ is a working day and $(d-1)$ is a day-off and $\bar{s}$ is not an idle shift then
30:     $c_{e\bar{s}d}$ += weight of constraint ‘no isolated day-off’
31: end if
    ▷ Shift succession
32: for all forbidden shift successions do
33:     if the succession is matched by assigning $\bar{s}$ on $d$ then
34:         $c_{e\bar{s}d}$ += weight of constraint ‘shift succession’
35:     end if
36: end for
    ▷ Complete weekends
37: if $d$ is a Sunday and the assignment on $(d-1) \neq \bar{s}$ then
38:     $c_{e\bar{s}d}$ += weight of constraint ‘complete weekends’
39: end if
```

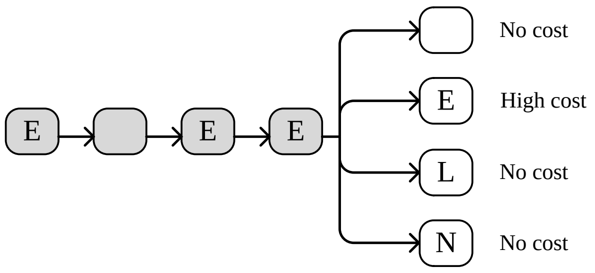


Figure 7.5: Example of determining c_{esd} in the vertical decomposition for the *maximum three assignments of shift E* constraint

Characteristic		Settings
Number of employees	n	10, 20, 40
Number of days	d	7, 28
Skilling	s	0.3, 0.6, 1
Tightness	t	0.6, 0.9

Table 7.6: Instance characteristics for the MDTSS dataset

7.5.5 Computational evaluation

A series of experiments was performed to compare the two decomposition methods for the MDTSS, and to illustrate how an integer programming solver is able to deal with this problem.

Data and experimental setup

Table 7.6 shows the parameter settings used in the instance generation procedure of the used instances[11].

A full factorial design resulted in $3 \times 2 \times 3 \times 2 = 36$ instance classes. For each class, ten random instances were generated, resulting in a dataset of 360 instances.

The set of employees is considered homogeneous in terms of contract, i.e. each employee's shift allocation is constrained by the same time related constraints. Table 7.7 lists the contract's time related constraints for the instances with $d = 7$ and $d = 28$. All constraints have a weight of 100.

[11]The instance generator is publicly available at `http://gent.cs.kuleuven.be/tss.html`.

Constraint	$d = 7$	$d = 28$
Minimum number of working days	2	8
Maximum number of working days	6	24
Minimum number of assignments per shift	[2, 1, 0, 1]	[8, 4, 0, 4]
Maximum number of assignments per shift	[4, 6, 7, 6]	[16, 24, 28, 24]
Maximum number of consecutive working days	4	4
Maximum number of consecutive days-off	3	3
Complete weekends	yes	yes
Allow isolated days-off	no	no
Shift succession	$(3, 1)$	$(3, 1)$

Table 7.7: Definition of time related constraints

All algorithms were tested on the 360 instances. The time limit for the decomposition approaches was set to 3600 seconds. In the improvement procedure, the maximum number of tasks to reassign was set to $k' = \max(40, \mu)$, with μ the largest number of unassigned tasks over all days in the scheduling period. The subproblems in the horizontal decomposition were constructed with $b = 5$. A minimum of 50% of the time was allocated to the improvement procedure in both decomposition approaches. This is achieved by imposing a time limit of $\frac{1800}{\lceil |E|/b \rceil}$ seconds on each subproblem in the horizontal decomposition. In the vertical decomposition a time limit of $\frac{1800}{d}$ seconds was imposed on each subproblem. All experiments were carried out in the same computing environment as in Section 7.4.5.

Computational results

Tables 7.8 and 7.9 show computational results as averages of all instances per class. Four different approaches are compared: an integer programming solver on the model from Section 7.5.1 with a time limit of one hour (*MIP-1h*) and ten hours (*MIP-10h*), the horizontal decomposition followed by the improvement heuristic (*Horizontal decomp.*) and the vertical decomposition followed by the improvement heuristic (*Vertical decomp*). For each of these approaches, three metrics are shown: the objective value of the obtained solution (*Obj.*), the relative gap to a lower bound (*Gap*), and the percentage of instances in the class for which a feasible solution was found (*Feas.*). The lower bound (*LB*) is the lower bound reported by the integer programming solver after ten hours of calculation time.

If an approach failed to find a feasible solution for at least one instance in the class, the objective value and gap are annotated with an asterisk. If no feasible solutions were found within one instance class, a dash is shown. The best results for each instance class are highlighted in bold.

Table 7.8 shows that for the 7-day instances, feasibility can be obtained by all approaches, except for MIP-1h on the largest instances. For the smaller instances, however, MIP-1h and MIP-10h compare very well to the decomposition approaches as they are able to find the optimal solution within the time limit, which is not always guaranteed by the heuristics. Out of the latter, the horizontal decomposition clearly results in the best solutions. Moreover, for instances with 40 employees, the horizontal decomposition even improves upon MIP-10h. Note that the gap values are to be considered with some care, as the objective value rapidly increases with the number of violations of soft constraints since each constraint weight is set to 100. Due to weight scaling, the differences in objective values can result in very large gaps, while in practice, the difference is not significant.

The results for the 28-day instances (Table 7.9) show that even finding a feasible solution is very challenging. Solving the integer program with a time limit of one hour leads to poor results, and even when allowing ten hours of computation time, feasible solutions are not consistently obtained. The decomposition approaches find significantly more feasible solutions within one hour. The vertical decomposition has a success rate of 100%. Regarding objective value, the horizontal decomposition generally finds better solutions.

Table 7.10 summarises the results from Tables 7.8 and 7.9, and shows that when considering a time limit of one hour, the horizontal decomposition is favourable to the other approaches, whereas it does not always guarantee a feasible solution for the larger instances within that time limit. Unsurprisingly, allowing the integer programming solver ten hours of computation time results in the highest number of best solutions, thereby establishing it as a very powerful but often impractical approach to the problem.

The methods always used all available time, except for the small instances with $n = 10$ for which the MIP approaches converged earlier than the decomposition algorithms.

Instance				MIP-1h			MIP-10h			Horizontal decomp.			Vertical decomp.		
n	s	t	LB	Obj.	Gap	Feas.	Obj.	Gap	Feas.	Obj.	Gap	Feas.	Obj.	Gap	Feas.
10	30	60	1256.6	**1256.6**	0.0%	100%	**1256.6**	0.0%	100%	1260.5	0.5%	100%	1256.7	0.0%	100%
		90	1839.6	**1839.6**	0.0%	100%	**1839.6**	0.0%	100%	1843.6	0.3%	100%	1839.8	0.0%	100%
	60	60	130.6	**130.6**	0.0%	100%	**130.6**	0.0%	100%	132.8	1.4%	100%	131.0	0.8%	100%
		90	894.3	1016.7	23.6%	100%	**926.6**	11.3%	100%	1278.2	40.3%	100%	1328.3	42.2%	100%
	100	60	211.4	**211.4**	0.0%	100%	**211.4**	0.0%	100%	**211.4**	0.0%	100%	**211.4**	0.0%	100%
		90	472.9	**472.9**	0.0%	100%	**472.9**	0.0%	100%	**472.9**	0.0%	100%	**472.9**	0.0%	100%
20	30	60	126.7	391.9	42.0%	100%	**159.8**	15.0%	100%	444.3	60.3%	100%	808.0	69.3%	100%
		90	311.3	2984.8	90.1%	100%	**1598.8**	81.5%	100%	2155.9	87.2%	100%	3615.1	92.1%	100%
	60	60	104.4	106.0	1.8%	100%	106.0	1.8%	100%	**105.9**	1.7%	100%	208.3	17.7%	100%
		90	146.4	2078.6	91.6%	100%	**367.0**	31.9%	100%	1066.7	79.1%	100%	2198.8	92.8%	100%
	100	60	113.3	**115.8**	2.6%	100%	**115.8**	2.6%	100%	**115.8**	2.6%	100%	116.4	2.9%	100%
		90	547.9	1441.5	57.7%	100%	**647.9**	22.7%	100%	803.0	40.3%	100%	2133.2	72.1%	100%
40	30	60	171.5	1812.2	83.3%	100%	**211.6**	9.2%	100%	230.0	18.7%	100%	1463.5	72.4%	100%
		90	200.4	*25086.6	*99.2%	80%	7931.7	97.0%	100%	**4984.5**	95.3%	100%	8914.7	97.7%	100%
	60	60	142.2	1231.7	68.9%	100%	490.6	45.0%	100%	**184.7**	22.1%	100%	976.0	50.8%	100%
		90	146.6	*21206.0	*99.3%	70%	10209.1	98.1%	100%	**5052.2**	95.7%	100%	5980.0	97.4%	100%
	100	60	140.7	527.6	40.7%	100%	285.3	31.7%	100%	**225.8**	25.8%	100%	1518.8	64.7%	100%
		90	149.6	*13936.0	*98.6%	80%	10272.9	98.2%	100%	**2953.5**	88.9%	100%	5686.5	97.1%	100%

Table 7.8: Comparison of solution quality for different approaches to the MDTSS, 7-day instances, as averages of all instances per class

Instance				MIP-1h			MIP-10h			Horizontal decomp.			Vertical decomp.		
n	s	t	LB	Obj.	Gap	Feas.	Obj.	Gap	Feas.	Obj.	Gap	Feas.	Obj.	Gap	Feas.
10	30	60	5883.4	6057.7	3.2%	100%	**6018.6**	2.4%	100%	6660.2	12.2%	100%	6828.1	14.4%	100%
		90	11294.6	**11294.8**	0.0%	100%	**11294.8**	0.0%	100%	11336.2	0.4%	100%	11344.0	0.5%	100%
	60	60	873.8	1249.1	22.0%	100%	**907.9**	3.0%	100%	1966.6	58.0%	100%	4151.7	78.9%	100%
		90	2024.7	*8679.2	*77.9%	90%	**5878.7**	65.3%	100%	7635.7	74.4%	100%	12284.9	83.9%	100%
	100	60	1707.4	1772.3	2.9%	100%	**1752.3**	1.8%	100%	2046.7	18.5%	100%	4281.0	58.5%	100%
		90	1712.2	*4418.3	*69.1%	80%	**2021.2**	21.4%	100%	2600.8	38.9%	100%	5662.4	70.8%	100%
20	30	60	380.3	13302.4	96.6%	100%	**4507.6**	90.8%	100%	6347.1	93.8%	100%	14138.3	97.2%	100%
		90	1965.5	-	-	0%	*32164.6	*93.4%	90%	**23079.0**	91.9%	100%	35571.7	94.5%	100%
	60	60	379.8	9433.6	95.2%	100%	**1019.2**	36.7%	100%	2115.6	78.7%	100%	9020.2	95.7%	100%
		90	351.8	*57922.0	*99.5%	10%	*31813.4	*98.9%	50%	*13569.9	*97.0%	80%	**21358.3**	98.3%	100%
	100	60	307.6	3815.7	88.1%	100%	2078	81.4%	100%	**1533.7**	75.5%	100%	7162.9	95.6%	100%
		90	296.2	*43325.3	*99.3%	30%	33314.2	98.9%	100%	**9436.2**	96.4%	100%	13521.3	97.8%	100%
40	30	60	565.2	*91667.4	*99.1%	90%	39880.7	98.4%	100%	**8155.6**	91.7%	100%	25256.0	97.7%	100%
		90	601.2	-	-	0%	-	-	0%	*178971.0	*99.7%	40%	**56141.6**	98.9%	100%
	60	60	568.8	*123068.9	*97.8%	90%	8920.4	92.1%	100%	**3368.3**	79.3%	100%	24204.0	97.5%	100%
		90	-	-	-	0%	*94051.0	*99.4%	10%	*166671.8	*99.7%	50%	**49668.9**	99.1%	100%
	100	60	567.6	*177024.0	*99.7%	90%	7930.6	85.6%	100%	**2823.1**	76.7%	100%	26458.9	97.8%	100%
		90	-	-	-	0%	-	-	0%	117528.7	99.2%	100%	**46547.2**	99.5%	100%

Table 7.9: Comparison of solution quality for different approaches to the MDTSS, 28-day instances, as averages of all instances per class

		MIP-1h	MIP-10h	Horizontal decomp.	Vertical decomp.
7 days	Number of feasible solutions	173	180	180	180
	Number of best solutions	74	130	105	56
28 days	Number of feasible solutions	118	145	167	180
	Number of best solutions	23	77	45	10
Total	Number of feasible solutions	291	325	347	360
	Number of best solutions	97	207	150	66

Table 7.10: Overview of computational results for the MDTSS, out of 360 instances

7.6 Conclusions

The idea of decomposing a large problem into smaller, tractable subproblems has been explored in the academic literature before [8, 53, 83]. This chapter investigated decomposition schemes for integrated task and shift scheduling problems, which are encountered in various environments such as logistics, manufacturing, etc.

Two TSS problems were introduced: the first one deals with a single, isolated day, while the second variant extends the scheduling period to multiple days. Exact and heuristic decomposition methods were introduced for both variants. Extensive computational experiments showed that a decomposition scheme which solves horizontally delineated subproblems, found near-optimal solutions for SDTSS problems with up to 100 employees, and MDTSS problems with up to 40 employees and a scheduling period of 28 days. Furthermore, it was shown how various time related personnel rostering constraints could be accounted for by the different algorithms. Finally, for both problem variants, an instance generator was made publicly available to enable researchers further investigating properties and solution approaches to these problems.

The results in this chapter clearly indicate the need to investigate how to decompose a problem, and give way to insightful recommendations regarding decomposition for other problems. The obtained results confirm previous academic results which have demonstrated the effectiveness of horizontal decomposition and the LBIH algorithm on other problems, e.g. the SMPTSP (Chapter 6) and the nurse rostering problem [40].

Chapter 8

Column generation based heuristics for the multi-day task and shift scheduling problem

Column generation is a well-known, exact method for solving large, complex optimisation problems. This chapter presents a reformulation of the multi-day task and shift scheduling problem, introduced in Chapter 7. Column generation is used to solve the linear programming relaxation of the reformulation. Furthermore, several heuristics are introduced which use the primal and dual solutions of this relaxation to generate feasible integer solutions. A series of computational experiments investigate the behaviour of both the column generation and the heuristics. In addition, a comparison is made with the approaches proposed in Chapter 7.

The content of this chapter is based on joint work with Andreas Ernst, CSIRO.

8.1 Introduction

Column generation is a well-known method for solving large scale linear programming problems. Successful applications in various settings have illustrated both the power and versatility of this technique. Also for personnel rostering, column generation has been able to address large problems for which other exact approaches fail.

In general, most applications of column generation to personnel rostering problems follow the same decomposition scheme. The master problem assigns lines of work to employees, while the subproblem is responsible for generating new lines of work [28, 67]. Typically, column generation is used within a branch-and-price algorithm to obtain optimal integer solutions [19, 85, 106].

This chapter proposes a column generation algorithm for solving the linear relaxation of the MDTSS. Tasks and shifts, both fixed in time, need to be assigned a set of multi-skilled employees, considering a scheduling period of multiple days. The objective is to minimise the cost of the schedule, consisting of the total number of days employees are required to work to perform all tasks, and a weighted sum of penalties incurred by violating time related constraints. A detailed description, as well as an integer programming formulation of this problem are presented in Chapter 7.

In addition to column generation, this chapter also introduces three heuristics which use both the primal and dual solutions of the linear relaxation. Computational results are presented, comparing the performance of these heuristics with the decomposition algorithms presented in Chapter 7.

The remainder of this chapter is organised as follows. Section 8.2 discusses the main contributions. A set covering reformulation of the MDTSS is presented in Section 8.3. Details on a column generation algorithm used for solving the linear relaxation of the reformulation are presented in Section 8.4. Section 8.5 presents the three column generation based heuristics. Computational results are presented in Section 8.6. Finally, Section 8.7 concludes this chapter.

8.2 Contributions

This chapter applies, for the first time, column generation to the MDTSS. A dedicated approach is presented for solving the subproblems, using a decomposition scheme in which the task assignment and shift assignment decisions are separated, resulting in subproblems which are computationally feasible. Furthermore, details are discussed on measures taken to address some of the well-known issues in column generation.

The second contribution of this chapter concerns three column generation based heuristics for the MDTSS: two diving heuristics and a new priority adjustment heuristic To generate feasible integer solutions, these methods use both the primal and dual solutions of the linear relaxation.

8.3 Set covering reformulation

Chapter 7 presented an integer programming formulation for the MDTSS (model (7.17) - (7.23)), which will be referred to as the *original formulation*. The problem is now reformulated using Dantzig-Wolfe decomposition, resulting in a set covering problem with exponentially many variables. Here, a column represents a complete line of work (a *pattern*), consisting of both task and shift assignments. The decision variables of the set covering problem assign patterns to employees.

Let P be the set of patterns, with P_e the subset of patterns for which employee e is qualified. A binary value a_{tp} indicates whether task t is covered in pattern p ($a_{tp} = 1$), or not ($a_{tp} = 0$). Finally, there is a cost c_{ep} for assigning pattern p to employee e.

Decision variables

$$x_{ep} = \begin{cases} 1 & \text{if pattern } p \text{ is assigned to employee } e \\ 0 & \text{otherwise} \end{cases}$$

$$u_t = \begin{cases} 1 & \text{if task } t \text{ is not assigned} \\ 0 & \text{otherwise} \end{cases}$$

Model

$$min \sum_{e\in E}\sum_{p\in P_e} c_{ep}x_{ep} + \sum_{t\in T} M u_t \tag{8.1}$$

$$s.t. \sum_{e\in E}\sum_{p\in P_e} a_{tp}x_{ep} + u_t \geq 1 \qquad \forall t \in T \tag{8.2}$$

$$\sum_{p\in P_e} x_{ep} = 1 \qquad \forall\, e \in E \tag{8.3}$$

$$x_{ep} \in \{0,1\} \qquad \forall\, e \in E,\ p \in P_e \tag{8.4}$$

$$u_t \in \{0,1\} \qquad \forall\, t \in T \tag{8.5}$$

This set covering problem is denoted as the *master problem*. Constraints (8.2) ensure all tasks are covered, either by an employee or by the corresponding u

variable. Constraints (8.3) assign exactly one feasible pattern to each employee. Constraints (8.4) and (8.5) enforce integrality of the decision variables. The first term of the objective function (8.1) minimises the cost of the schedule. In the second term, the u variables with sufficiently large weights M are used to force all tasks to be assigned to employees. By using the u variables in the model, any feasibility issues arising when solving the master problem are avoided.

8.4 Column generation

Due to the combinatorial nature of the patterns, the master problem contains too many variables to be solved directly. The integrality constraints (8.4) and (8.5) can be relaxed such that $x_{ep} \geq 0$ and $u_t \geq 0$. A relaxation of the master problem is thus obtained, which can be solved using column generation [83].

Instead of solving the master problem directly, the *restricted master problem* is solved in which only a subset of variables (patterns) is considered. After solving the restricted master problem, the *pricing problem* is solved to determine if there are non-basic variables with negative reduced cost. Let π be the dual prices of Constraints (8.2), and γ the duals of Constraints (8.3). The reduced cost $\bar{c}_{ep}$ of a variable x_{ep} is defined as:

$$\bar{c}_{ep} = c_{ep} - \sum_{t \in T} a_{tp}\pi_t - \gamma_e$$

The pricing problem is solved for each employee separately. Since γ_e can thus be considered constant, the pricing problem is to determine whether a new pattern exists such that $c_{ep} - \sum_{t \in T} a_{tp}\pi_t \leq \gamma_e - \epsilon$, with ϵ a small constant value to adjust for rounding errors.

If there are no new variables to enter the basis, the column generation algorithm has converged and an optimal solution for the master problem has been found. If a negative reduced cost variable is found, it is added to the restricted master problem, which is then solved again.

8.4.1 The pricing problem as an integer program

The pricing problem generates one or more feasible complete lines of work. Let q_{et} be a binary value which is one if employee e is qualified for task t, and zero otherwise. The problem can be formulated as an integer program that is almost identical to the original formulation (7.17) - (7.23), without the employee index.

Decision variables

$$y_t = \begin{cases} 1 & \text{if task } t \text{ is assigned} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{sd} = \begin{cases} 1 & \text{if shift } s \text{ is assigned on day } d \\ 0 & \text{otherwise} \end{cases}$$

Model

$$min\ c - \sum_{t \in T} \pi_t y_t \tag{8.6}$$

$$s.t. \sum_{t \in K} y_t \leq 1 \qquad \forall K \in C_e \tag{8.7}$$

$$y_t \leq q_{et} \qquad \forall\ t \in T \tag{8.8}$$

$$\sum_{s \in S} z_{sd} = 1 \qquad \forall d \in D \tag{8.9}$$

$$\sum_{s \in S_t} z_{sd_t} \geq y_t \qquad \forall\ t \in T \tag{8.10}$$

$$c = \text{days worked} + \sum \text{soft constraint penalty} \tag{8.11}$$

$$y_t \in \{0, 1\} \qquad \forall\ t \in T \tag{8.12}$$

$$z_{sd} \in \{0, 1\} \qquad \forall\ s \in S, d \in D \tag{8.13}$$

The objective function (8.6) minimises the reduced cost. Constraints (8.8) make sure that tasks are only assigned to qualified employees. Constraints (8.7), (8.9), (8.10), (8.12) and (8.13) have their counterparts in the original formulation.

Preliminary computational experiments showed that solving the pricing problem as an integer program requires too much computation time, thereby severely impacting the performance of the column generation algorithm. Therefore, an alternative approach to solving the pricing problem is presented.

8.4.2 The pricing problem as a decomposed shortest path

The construction of a line of work is decomposed into packing tasks in shifts, and constructing the shift roster. First, for each day d and shift s, a problem is solved which selects tasks for shift s on day d, thereby maximising the sum of the dual prices of the selected tasks (the *total profit* σ_{sd}). Afterwards, the shift roster is constructed while minimising the cost, and maximising the sum of total profits.

Since a suboptimal task selection can never lead to a better pattern, this decomposition approach is equivalent to assigning tasks and shifts simultaneously. Whenever a shift assignment is made, it will always be packed with the most 'profitable' tasks, i.e. tasks with the largest dual prices, since the sum of these duals reduces the objective value.

Step 1: packing tasks

The problem of packing tasks in shift s on day d can be formulated as an integer program. Let T_{sd} be the set of tasks to be executed on day d, and whose time intervals fit in shift s. C_{sd} is the set of all maximal cliques in the interval graph constructed for tasks in T_{sd}.

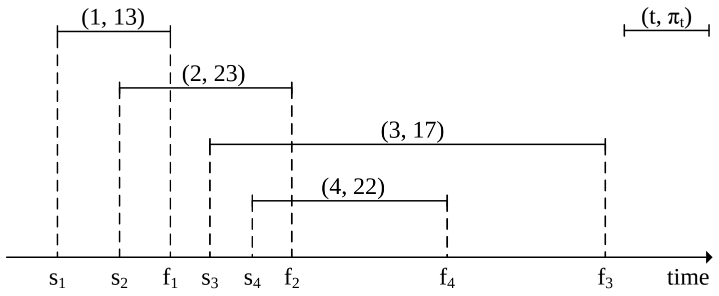
$$max \sum_{t \in T} \pi_t y_t \tag{8.14}$$

$$s.t. \sum_{t \in K} y_t \leq 1 \qquad \forall K \in C_{sd} \tag{8.15}$$

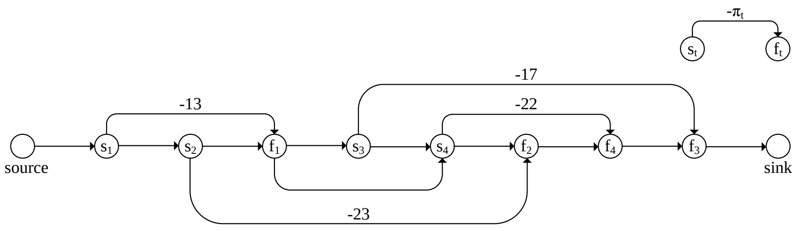
$$y_t \in \{0,1\} \qquad \forall\, t \in T_{sd} \tag{8.16}$$

The objective function (8.14) maximises the total profit σ, while making sure that no overlapping tasks are selected (Constraints (8.15)), and tasks are assigned at most once (Constraints (8.16)).

This problem can be reformulated as a shortest path problem in a graph $G = (V, A)$, with V the set of nodes and A the set of arcs [77]. The set V contains $2|T_{sd}| + 2$ nodes. The first and last nodes in G are the source and sink node, respectively. Other nodes correspond with the start and end times of the tasks in T_{sd}, ordered by increasing time. Each node is connected with its neighbours by a directed arc without cost, in the direction of the time horizon. Furthermore, directed arcs are defined between the start and end time nodes of



(a) Tasks



(b) Shortest path graph

Figure 8.1: Example of packing tasks in shifts [77]

a task. These arcs have an cost of $-\pi_t$. Finally, there are directed arcs with zero cost between each end node and each subsequent start node. A shortest path between the source and sink node represents an optimal packing of tasks according to π.

Consider an example with $T_{sd} = \{1, 2, 3, 4\}$. Figure 8.1a shows the tasks' time intervals, labelled as (t, π_t). The graph G for this example is shown in Figure 8.1b. The value above each arc is the cost. For arcs with no cost, the value is not shown. In this example, the shortest path from source to sink is $\{source, s_1, f_1, s_4, f_4, sink\}$, and has a distance of -35. The corresponding optimal task packing is $\{1, 4\}$, with a total profit $\sigma_{sd} = 35$.

Step 2: assigning shifts

After determining the optimal task selection for each shift and each day, the following integer program is solved to construct the shift roster.

$$min\ c - \sum_{s \in S} \sum_{d \in D} \sigma_{sd} z_{sd} \tag{8.17}$$

$$s.t. \sum_{s \in S} z_{sd} = 1 \qquad \forall d \in D \tag{8.18}$$

$$c = \text{days worked} + \sum \text{soft constraint penalty} \tag{8.19}$$

$$z_{sd} \in \{0,1\} \qquad \forall\ s \in S, d \in D \tag{8.20}$$

The objective function (8.17) minimises the cost of the roster, while maximising the sum of σ_{ds}. Constraints (8.18) assign one shift on each day. Constraints (8.19) and (8.20) calculate the cost of the solution and require integrality of the variables, respectively.

Once the shift roster is constructed, task assignments are made based on the task packing solutions calculated in the first phase. For each assigned shift, the packed tasks are assigned.

8.4.3 Initialisation

The u variables allow for the restricted master problem to be solved with only variables representing *empty patterns*. An empty pattern only has day-off assignments, and no tasks assignments. For each employee, this pattern is added to P_e. In the first solution of the restricted master problem, all u variables will be set to one, and all employees will have their empty pattern assigned.

8.4.4 Lagrangian dual bound

In each iteration of the column generation, a dual bound on the optimal solution of the master problem is calculated as follows. First, Constraints (8.2) are dualised with a vector of multipliers π.

$$
\begin{aligned}
&min \sum_{e\in E}\sum_{p\in P_e} c_{ep}x_{ep} + \sum_{t\in T} Mu_t + \\
&\qquad \sum_{t\in T}\pi_t\left(1-\sum_{e\in E}\sum_{p\in P_e} a_{tp}x_{ep} - u_t\right) \\
&s.t. \sum_{p\in P_e} x_{ep} = 1 && \forall\ e\in E \\
&\quad x_{ep}\geq 0 && \forall\ e\in E,\ p\in P_e \\
&\quad 0\leq u_t\leq 1 && \forall\ t\in T
\end{aligned}
$$

Reordering the terms gives:

$$
\begin{aligned}
&min \sum_{t\in T}(M-\pi_t)\,u_t + \sum_{e\in E}\sum_{p\in P_e}\left(c_{ep}-\sum_{t\in T}\pi_t a_{tp}\right)x_{ep} + \sum_{t\in T}\pi_t \\
&s.t. \sum_{p\in P_e} x_{ep} = 1 && \forall\ e\in E \\
&\quad x_{ep}\geq 0 && \forall\ e\in E,\ p\in P_e \\
&\quad 0\leq u_t\leq 1 && \forall\ t\in T
\end{aligned}
$$

A solution to this problem is a lower bound for the master problem. The term $\sum_{e\in E}\sum_{p\in P_e}\left(c_{ep}-\sum_{t\in T}\pi_t a_{tp}\right)x_{ep}$ corresponds to the objective value from the pricing problem (summed over all employees) and is calculated as such in each iteration. The problem is thus equivalent to:

$$
\begin{aligned}
&min \sum_{t\in T}(M-\pi_t)\,u_t + \sum_{e\in E}\min_p \bar{c}_{ep} + \sum_{t\in T}\pi_t \\
&s.t.\ 0\leq u_t\leq 1 && \forall\ t\in T
\end{aligned}
$$

Since $\pi_t\leq M$, u_t will always be zero in an optimal solution to this problem. The lower bound can thus be calculated as:

$$\sum_{e \in E} \min_p \bar{c}_{ep} + \sum_{t \in T} \pi_t \tag{8.21}$$

This bound can be used to terminate the column generation earlier in order to avoid the tailing-off effect. The algorithm can be stopped if the difference between the objective value and the lower bound is smaller than some small constant ϵ. Moreover, since the original objective value will always be integer, the lower bound can be rounded up to the nearest integer. If the objective value of the master problem is then smaller or equal to this rounded value, the column generation can be safely terminated.

8.4.5 Stabilisation

Column generation often convergences slowly due to degeneracy of the set covering problem. State of the art linear programming solvers usually return an extreme point of the dual polyhedron, which results in dual variables taking either very large or very small values. This causes the pricing problem to often generate columns that will never be used in an optimal solution [109].

To address this issue, different stabilisation approaches have been proposed in the academic literature [120]. In this work, the column generation is stabilised by smoothing the dual prices [103]. The dual prices π^k passed to the pricing problem in iteration k are 'corrected' based on the dual prices from the previous iteration. Specifically, the stabilised duals $\tilde{\pi}^k$ passed to the pricing problem in iteration k are calculated as a convex combination of π^k and $\tilde{\pi}^{k-1}$ (Equation (8.22)).

$$\tilde{\pi}^k = \alpha \tilde{\pi}^{k-1} + (1 - \alpha)\pi^k \tag{8.22}$$

The parameter $\alpha \in [0, 1[$ determines the level of smoothing and is fixed to 0.5.

Solving the pricing problem with the stabilised duals might not yield a negative reduced cost column, even when one exists for π^k. This situation is the result of a *mis-pricing*. In this case, the pricing problem is solved again using $\tilde{\pi}^k = \pi^k$.

8.4.6 Breaking symmetry

If the workforce is homogeneous, i.e. employees have identical qualifications and soft constraints, the master problem exhibits a large degree of symmetry. To break this symmetry *employee classes* are introduced. An employee is added to a class if he/she is identical to the other employees in the class. Clearly, in case of a completely homogeneous workforce, there is only one employee class, of which the size is equal to the number of employees. In contrast, if the workforce is completely heterogeneous, there is one employee class for each employee.

The master problem is modified to decide which patterns to assign to the employees in each class. Let L be the set of employee classes. The decision variables x_{ep} of the master problem are redefined as:

$$x_{lp} = \begin{cases} 1 & \text{if pattern } p \text{ is assigned to an employee in employee class } l \\ 0 & \text{otherwise} \end{cases}$$

Let $size_l$ be the number of workers in employee class l. Constraint (8.3) is changed to:

$$\sum_{p \in P_l} x_{lp} = size_l \qquad \forall\, l \in L \tag{8.23}$$

The Lagrangian dual bound (8.21) becomes:

$$\sum_{l \in L} size_l \min_p \bar{c}_{lp} + \sum_{t \in T} \pi_t \tag{8.24}$$

8.5 Heuristics

Solving the master problem with column generation produces fractional primal and dual solutions, as well as a lower bound on the optimal integer solution. In this section, three heuristic algorithms are presented which use this information to find feasible integer solutions for the problem.

8.5.1 Iterated diving heuristics

A diving heuristic explores one branch of a search tree and is intended to quickly obtain feasible integer solutions in branch-and-bound algorithms [68]. The algorithm starts by solving the root node to optimality. In the resulting fractional solution, variables equal to one are fixed to this value. Out of the other (non-zero) fractional variables, one is selected and also fixed to one. These steps are repeated until a leaf node at the bottom of the search tree is reached.

This basic algorithm is embedded in an iterative framework as follows. Once a leaf node is reached, and while the time limit is not reached, the diving heuristic is restarted. To avoid exploring the same branch, randomness is introduced when selecting a fractional variable to fix. Roulette wheel selection is used such that a variable whose value is close to one has a high probability of being selected. Furthermore, in the spirit of branch-and-bound, if the objective value of the master problem in a node is larger than the best known integer solution found so far, the diving heuristic is immediately restarted.

Algorithm 11 outlines this approach.

Algorithm 11 Iterated diving heuristic

```
Input:
    MDTSS problem instance
    $F(x)$ evaluation function of x
Output: $x_{best}$                                   ▷ best solution found
 1: while time limit not reached do
 2:     while leaf node not reached do
 3:         Solve LP relaxation using column generation
 4:         Fix all variables with values equal to one
 5:         Select one fractional variable with roulette wheel selection
 6:         Fix the selected variable to one
 7:     end while
 8:     if $F$(leaf node solution) $\leq F(x_{best})$ and leaf node solution is feasible
    then
 9:         $x_{best}$ := leaf node solution
10:     end if
11: end while
```

The choice of which variables to fix strongly affects the behaviour of the heuristic. The next sections discuss two diving heuristics, the first branching on variables of the master problem, the second branching on variables of the original formulation.

Branching on master variables

Branching on the variables x_{ep} only impacts the master problem, i.e. the branching decisions do not need to be enforced when solving the pricing problem. In the master problem, branching is done by changing the bounds of the fixed variables. However, this branching scheme is possibly too aggressive to obtain feasible solutions in the leaf nodes.

Branching on original variables

A (fractional) solution for the master problem can be projected onto the task assignment variables v_{et} of the original formulation. For each task t and employee e, the values of the master variables for the patterns containing task t, are summed (Equation (8.25)).

$$v_{et} = \sum_{p \in P_e : a_{tp} = 1} x_{ep} \qquad \forall e \in E,\ t \in T \tag{8.25}$$

Fixing an original variable corresponds to fixing the assignment of one task to an employee. To respect these branching decisions, the algorithm for solving the pricing problem is modified as follows. First, the dual prices used to pack the tasks are modified based on whether the original variable is fixed to zero or one. In case of zero, the dual price of the corresponding task for that employee is set to $-\infty$, if the original variable is one, the corresponding dual price is set to $+\infty$. This modification of the dual prices is only done for packing the tasks, the total profit σ_{sd} is still calculated using the original dual prices. When packing tasks in shifts, for each day d, a set of shifts $\tilde{S}_d$ is maintained out of which at least one needs to be assigned such that the fixed tasks can be assigned. In the second step, the integer program (8.17) - (8.20) for constructing the shift rosters is extended with Constraints (8.26) to ensure that at least one of the required shifts from $\tilde{S}_d$ is assigned.

$$\sum_{s \in \tilde{S}_d} z_{sd} = 1 \qquad \forall d \in D \tag{8.26}$$

After each branching step, all patterns which do not contain the fixed assignments are removed from the model.

8.5.2 Priority adjustment heuristic

The priority adjustment heuristic is an iterative constructive heuristic which attempts to build a feasible integer solution based on the information available in each column generation iteration.

Parameter	Value
Number of employees	10, 20, 40
Number of days	7, 28
Skilling	0.3, 0.6, 1
Tightness	0.6, 0.9

Table 8.1: Parameter settings for the instance generator

The heuristic uses a value θ_{et} to indicate the preference of assigning task t to employee e. High values for θ_{et} correspond to a high preference. At the start of the algorithm, θ_{et} is initialised with dual prices π of the current column generation iteration. Initially, all employees will thus have the same preference for each task. In one iteration, the algorithm builds a solution by generating a complete line of work for each employee sequentially, using the current priority values θ as an indication for the task assignments. Once all employees have a complete line of work, the priorities of unassigned tasks are updated so that they receive a higher priority in the next iteration. The update rule increases the current priority with the product of the highest π value, and the fractional solution of the master problem projected onto the original variables v_{et}, according to Equation (8.25).

If there are no unassigned tasks, a feasible solution is obtained, otherwise the algorithm continues until a maximum number of iterations is reached.

Algorithm 12 outlines the procedure.

8.6 Computational results

8.6.1 Data and experimental setup

To evaluate the different algorithms, a set of instances was generated with the instance generator used in Chapter 7. Table 8.1 shows the parameter settings used during instance generation. In total, 24 instance classes were defined, each containing two instances.

All employees are subject to the same set of time related constraints (Table 8.2). All constraints have a weight of 100. The shift structure is the same in all instances, consisting of three partially overlapping shifts of equal duration.

Algorithm 12 Priority adjustment heuristic

```
Input: $\pi$ := current dual prices of Constraints (8.2)
Output: $\langle shifts, tasks \rangle$                              ▷ (feasible) solution
 1: $\theta_{et} := \pi_t, \forall e \in E$
 2: $feasible := false$
 3: $iterations := 0$
 4: $shifts := \emptyset$                        ▷ Vector of shift assignments of length $|E|$
 5: $tasks := \emptyset$                         ▷ Vector of task assignments of length $|E|$
 6: $unassigned := \emptyset$
 7: $E := shuffle(E)$            ▷ Randomise order in which employees are processed
 8: while $iterations < 50$ and $!feasible$ do
 9:     for all $t \in unassigned$ do               ▷ Increase priority of unassigned tasks
10:         for all $e \in E$ qualified for $t$ do
11:             $\theta_{et} := \theta_{et} + v_{et} max(\pi)$
12:         end for
13:     end for
14:     $shifts := \emptyset$                               ▷ Clear previous shift assignments
15:     $tasks := \emptyset$                                ▷ Clear previous task assignments
16:     $\tilde{\theta}_{et} := \theta_{et}$                    ▷ Copy priorities to be used in this iteration
17:     for all $e \in E$ do
18:         $c$ := generate column for $e$ with $\tilde{\theta}_{et}$              ▷ Solve pricing problem
19:         $shifts_e := shifts(c)$                          ▷ Assign shifts from column $c$
20:         $tasks_e := tasks(c)$                            ▷ Assign tasks from column $c$
21:         for all $t \in tasks_e$ do
22:             for all $e \in E$ qualified for $t$ do
23:                 $\tilde{\theta}_{et} := -\infty$      ▷ Update priority to reflect already assigned tasks
24:             end for
25:         end for
26:     end for
27:     $unassigned := \emptyset$                               ▷ Collect all unassigned tasks
28:     for all $t \in T : t \notin tasks$ do
29:         $unassigned := unassigned \cup t$
30:     end for
31:     if $unassigned = \emptyset$ then                      ▷ Check feasibility of solution
32:         $feasible := true$
33:     end if
34:     $iterations := iterations + 1$
35: end while
```

Constraint	7 days	28 days
Minimum number of working days	2	8
Maximum number of working days	6	24
Minimum number of assignments per shift	[2, 1, 0, 1]	[8, 4, 0, 4]
Maximum number of assignments per shift	[4, 6, 7, 6]	[16, 24, 28, 24]
Maximum number of consecutive working days	4	4
Maximum number of consecutive days-off	3	3
Complete weekends	yes	yes
Allow isolated days-off	no	no
Shift succession	$(3, 1)$	$(3, 1)$

Table 8.2: Definition of time related constraints

All experiments were carried out on an Intel Core i5 CPU at 2.53GHz with 4 cores and 4GB RAM. CPLEX 12.6 was used as a linear and integer programming solver. For solving the linear programming problems, the primal simplex algorithm was used. In the pricing problem, the number of presolves was limited to one. In all experiments ϵ was set to $1e - 6$. The allowed computation time was limited to 3600 seconds wall time.

8.6.2 Column generation

This section investigates two aspects related to the column generation. Firstly, general characteristics of the algorithm for different problem instances are analysed. Secondly, the computed lower bound is compared with another lower bound obtained by solving a relaxation of the original formulation.

Tables 8.3, 8.4 and 8.5 show the main computational results for instances with a scheduling period of one week and 30%, 60% and 100% skilling, respectively. Table 8.6 shows the results for instances with a four week period. $|E|$ is the number of employees, $|T|$ the number of tasks.

Two approaches are compared: solving the linear relaxation with column generation, and solving the root node of the original formulation using CPLEX. The following data are shown: the final objective value, i.e. the lower bound (*Obj*), the total number of iterations (*Iters*), the number of columns generated in the algorithm (*Cols*), the elapsed wall time in seconds (T_{total}), the time spent in solving the master problem (T_{ma}), and the time spent in solving the pricing problems (T_{pr}).

Instance		Root node		Column generation					
$\|E\|$	$\|T\|$	Obj	T_{total}	Obj	T_{total}	T_{ma}	T_{pr}	Iters	Cols
Small									
11	205	542.6	6.4	**594.9**	4.7	0.5	4.1	57	1261
11	200	**1213.2**	4.3	1144.8	4.6	0.6	4.0	61	1397
11	318	**2459.0**	3.0	2325.3	16.7	7.8	8.9	157	2681
11	314	**2058.0**	2.7	1977.7	11.8	4.7	7.1	121	2321
Medium									
23	454	101.5	188.2	**102.0**	13.8	7.2	6.6	43	2821
24	474	99.0	301.2	**99.0**	15.7	9.5	6.2	45	2713
23	657	642.1	961.0	**808.8**	198.8	170.9	27.8	135	7948
23	636	108.9	405.1	**445.8**	164.3	139.4	24.9	134	7115
Large									
49	923	147.0	2117.8	**176.0**	69.8	54.4	15.4	46	5184
44	873	132.0	1372.9	**177.0**	68.9	53.8	15.1	51	5367
47	1326	141.0	2133.9	**210.7**	3637.9	3559.1	78.8	136	19815
45	1322	135.0	1938.6	**199.9**	3608.1	3541.8	66.4	133	17543

Table 8.3: Results for column generation on instances with 30% skilling

Instance		Root node		Column generation					
$\|E\|$	$\|T\|$	Obj	T_{total}	Obj	T_{total}	T_{ma}	T_{pr}	Iters	Cols
Small									
11	214	51.0	13.1	**151.0**	4.2	0.7	3.5	43	1262
11	201	49.0	11.1	**49.0**	3.6	0.6	3.0	39	1181
11	353	829.8	153.2	**958.3**	45.1	31.6	13.5	164	4116
11	326	484.3	175.6	**659.4**	27.5	16.3	11.2	113	3520
Medium									
25	444	75.0	286.2	**90.0**	13.4	6.3	7.1	47	2674
23	456	100.0	632.5	**200.0**	12.9	5.3	7.6	42	2810
23	692	109.0	3480.0	**509.0**	254.0	228.1	25.9	113	7200
23	685	69.0	1226.9	**307.0**	482.0	450.5	31.4	135	9014
Large									
48	899	144.0	3600.1	**177.0**	77.8	49.7	28.0	57	6507
50	908	150.0	2801.7	**176.0**	83.1	54.4	28.6	56	6696
46	1402	140.0	3600.1	**416.0**	3274.0	3162.5	111.5	127	17404
44	1385	135.0	3600.1	**820.2**	3668.5	3553.8	114.7	119	17275

Table 8.4: Results for column generation on instances with 60% skilling

Instance		Root node		Column generation					
$\lvert E\rvert$	$\lvert T\rvert$	Obj	T_{total}	Obj	T_{total}	T_{ma}	T_{pr}	Iters	Cols
Small									
12	221	36.0	27.0	**152.0**	1.9	0.3	1.6	140	346
12	234	51.8	34.2	**154.0**	1.8	0.3	1.5	128	368
11	347	655.0	366.9	**855.0**	93.9	80.3	13.6	882	2823
11	355	1239.3	138.0	**1456.0**	228.0	203.4	24.6	1541	4808
Medium									
24	437	72.0	237.5	**97.0**	3.8	1.0	2.8	209	532
23	503	69.0	739.5	**304.0**	10.8	5.7	5.1	322	1055
22	653	82.0	1205.9	**204.0**	680.2	646.4	33.8	1647	5486
23	670	78.0	1326.1	**1006.0**	1042.4	1009.0	33.4	1646	6693
Large									
47	885	141.0	3600.1	**176.0**	34.6	22.3	12.4	600	1350
46	901	138.0	3600.1	**181.0**	30.4	18.9	11.5	514	1358
47	1333	-	3600.4	**281.1**	3603.5	3534.7	68.8	1757	6737
44	1359	-	3600.4	-	3601.6	3531.6	70.0	1707	5548

Table 8.5: Results for column generation on instances with 100% skilling

Instance		Root node		Column generation					
$\lvert E\rvert$	$\lvert T\rvert$	Obj	T_{total}	Obj	T_{total}	T_{ma}	T_{pr}	Iters	Cols
30% skilling									
12	918	4504.3	108.4	**4801.3**	1388.5	1146.8	241.7	311	11850
12	853	4098.7	92.6	**4173.8**	501.9	362.3	139.5	221	8652
11	1345	**13107.8**	8.9	5935.4	3607.2	3521.2	86.0	466	11482
12	1289	**8484.3**	71.4	6868.9	3605.6	3460.8	144.8	329	14827
60% skilling									
11	855	648.5	473.0	**1597.0**	317.3	215.2	102.1	124	6738
11	893	409.0	499.2	**1707.0**	600.4	430.1	170.3	142	8047
12	1427	**1230.5**	3600.3	-	3641.4	3498.0	143.3	202	14155
11	1295	**1911.8**	3603.2	-	3622.2	3468.9	153.3	244	15794
100% skilling									
11	954	1164.5	1429.1	**2812.0**	490.5	261.2	229.4	778	4639
11	798	187.0	782.3	**685.0**	132.5	80.6	51.9	567	2786
12	1348	**779.0**	3482.0	-	3603.3	3430.3	173.0	1281	8174
12	1343	**613.0**	3600.1	-	3606.2	3429.8	176.4	1251	8415

Table 8.6: Results for column generation on instances with 28 days

Runtime characteristics

Computational results for non-homogeneous instances in Tables 8.3 and 8.4 show that the column generation algorithm requires around 50 iterations for instances with a small number of tasks. When the number of tasks increases, the number of iterations generally increases to around 130 iterations. The total number of columns generated is proportional to the number of iterations and the number of employees in an instance, and thus follows a similar trend. Table 8.5 shows that instances with homogeneous employees require significantly more iterations.

For most instances, the column generation converges within one hour. This is true for instances with both heterogeneous and homogeneous sets of employees, indicating that the overall time required to solve the master and pricing problems is significantly less for the homogeneous instances. In all cases, for the smallest instances, more time is spent in solving the pricing problems. However, with increasing instance size, this ratio shifts such that up to an order of magnitude more time is spent on solving the master problem than on solving the pricing problems.

Most of these conclusions hold for instances with 28 days as well, but the number of iterations, and consequently the total computation time, is generally higher (Table 8.6). Again, most time is spent on solving the master problem.

Lower bound comparison

Tables 8.3 - 8.6 also compare two lower bounds (Obj). For 39 out of the 48 instances, solving the linear relaxation of the set covering formulation results in a better lower bound compared to solving the root node of the original formulation. In the nine other cases, either the column generation did not terminate within the time limit, or the cuts added by CPLEX when solving the root node significantly strengthened the linear relaxation of the original formulation.

Comparing the required computation time of both approaches, column generation generally requires less time to find the optimal solution. For the instances with homogeneous employees, the difference in required time is up to two orders of magnitude in favour of column generation.

Influence of skilling level

As was shown in Section 6.6.1, the skilling level has a significant impact on the performance of algorithms. Figure 6.4 on page 120 clearly illustrated that SMPTSP instances with a skilling level of 35% are particularly hard for integer programming solvers. To investigate whether a similar effect also occurs when solving linear relaxations of the MDTSS, additional experiments were performed.

All instances from Table 8.3, and the 30% skilling instances from Table 8.6 were modified to a skilling level of 35%. For these modified instances, Table 8.7 shows the required calculation time (*Time*), and the relative difference to the time required for solving the same instances with a skilling level of 30% (δ). Instances for which δ is negative require less calculation time when the skilling level is 35%. In contrast, when δ is positive, more time is required to solve the 35% skilling instance; these cases are highlighted in bold. The results are shown for solving the root node of the original formulation using CPLEX (*Root node*), and for solving the linear relaxation with column generation. For the latter, time spent in solving the master and pricing problems is shown separately.

The original formulation generally requires more calculation time of an integer programming solver for instances with 35% skilling level. In some cases, the required time doubles or triples, and in one extreme case, the time required for the modified instance is almost ten times larger. This effect is less outspoken for the column generation. There are fewer increases, and they are generally smaller. Furthermore, there is no notable difference between the results for the master problem and for the pricing problems.

These results confirm that for problems in which the task assignment constraints make up an important part of the model, a skilling level of 35% is harder for integer programming solvers on the original formulation. By reformulating the problem, and in this case solving it with column generation, this effect is much less noticeable.

8.6.3 Heuristics

Tables 8.8, 8.9 and 8.10 show computational results of the three heuristics for instances with a scheduling period of one week, and varying number of employees. Results for instances with a period of four weeks are presented in Table 8.11. In addition to the number of employees and number of tasks in each instance, the best known lower bound is also shown (*LB*). For each of the heuristics, the objective value of the best solution found (*Obj*), and the gap

Instance		Root node		Column generation					
				Total		Master		Pricing	
$\|E\|$	$\|T\|$	Time	δ	Time	δ	Time	δ	Time	δ
Small (7 days)									
11	205	5.8	-11%	**5.1**	8%	**0.7**	28%	**4.4**	6%
11	200	**6.2**	43%	3.3	-29%	0.6	-9%	2.7	-32%
11	318	**12.2**	308%	13.8	-18%	6.9	-12%	6.8	-23%
11	314	**8.5**	216%	10.5	-11%	4.1	-12%	6.4	-10%
Medium (7 days)									
23	454	**207.0**	10%	11.3	-18%	5.6	-23%	5.8	-13%
24	474	**316.1**	5%	12.7	-19%	7.4	-21%	5.2	-16%
23	657	**1970.0**	105%	**226.0**	14%	**197.0**	15%	**29.0**	4%
23	636	**838.3**	107%	**182.6**	11%	**159.5**	14%	23.2	-7%
Large (7 days)									
49	923	1427.8	-33%	**71.3**	2%	**55.1**	1%	**16.1**	4%
44	873	**3278.4**	139%	67.8	-2%	52.4	-3%	**15.4**	2%
47	1326	**3600.1**	69%	2423.4	-33%	2355.3	-34%	68.1	-14%
45	1322	**3600.1**	86%	2515.9	-30%	2453.6	-31%	62.3	-6%
Small (28 days)									
12	918	**297.9**	175%	740.8	-47%	554.4	-52%	186.5	-23%
12	853	**175.7**	90%	**577.9**	15%	**429.3**	18%	**148.6**	7%
11	1345	**92.6**	944%	3607.2	0%	3512.6	0%	**94.6**	10%
12	1289	**322.6**	352%	3610.2	0%	3423.4	-1%	**186.8**	29%

Table 8.7: Calculation times for instances with a skilling level of 35%

calculated as $\frac{Obj-LB}{Obj}$ (Gap) is also shown. The best results for each instance are highlighted in bold.

For the priority adjustment heuristic, Algorithm 12 was executed every ten column generation iterations. In the diving heuristics, only the root node is solved to optimality each time. In the other nodes, at most ten column generation iterations are done.

With increasing problem size, there is a clear trend in performance of the different heuristics. For the small instances (Table 8.8), the diving heuristic on the master variables obtains the highest number of best solutions, with six out of twelve instances solved to optimality. As the number of tasks and employees increases, both diving heuristics fail to consistently find feasible solutions (Tables 8.9 - 8.11). The priority adjustment heuristic succeeds in finding feasible solutions for all instances within the time limit. The reason for this is that the diving heuristics require several nodes to be solved with

Instance			Priority adjustment		Diving - master		Diving - original	
$\|E\|$	$\|T\|$	LB	Obj	Gap	Obj	Gap	Obj	Gap
30% skilling								
11	205	595	855	30.4%	**753**	21.0%	854	30.3%
11	200	1214	1356	10.5%	**1355**	10.4%	1455	16.6%
11	318	2459	2560	3.9%	**2459**	0.0%	2558	3.9%
11	314	2058	2159	4.7%	**2058**	0.0%	2258	8.9%
60% skilling								
11	214	151	252	40.1%	**151**	0.0%	**151**	0.0%
11	201	49	50	2.0%	**49**	0.0%	149	67.1%
11	353	959	**1558**	38.4%	-	-	-	-
11	326	660	**1458**	54.7%	**1458**	54.7%	1960	66.3%
100% skilling								
12	221	152	256	40.6%	**152**	0.0%	**152**	0.0%
12	234	154	357	56.9%	**154**	0.0%	**154**	0.0%
11	347	855	**1457**	41.3%	-	-	-	-
11	355	1456	**2358**	38.3%	-	-	-	-

Table 8.8: Comparison of heuristics for small instances with a scheduling period of one week

Instance			Priority adjustment		Diving - master		Diving - original	
$\|E\|$	$\|T\|$	LB	Obj	Gap	Obj	Gap	Obj	Gap
30% skilling								
23	454	102	1511	93.2%	**1210**	91.6%	-	-
24	474	99	**109**	9.2%	810	87.8%	-	-
23	657	809	**2922**	72.3%	-	-	-	-
23	636	446	**2924**	84.7%	-	-	-	-
60% skilling								
25	444	90	98	8.2%	295	69.5%	**90**	0.0%
23	456	200	203	1.5%	**200**	0.0%	**200**	0.0%
23	692	509	**2321**	78.1%	-	-	-	-
23	685	307	**2823**	89.1%	-	-	-	-
100% skilling								
24	437	97	**97**	0.0%	99	2.0%	-	-
23	503	304	**805**	62.2%	-	-	-	-
22	653	204	**1210**	83.1%	-	-	-	-
23	670	1006	**2216**	54.6%	-	-	-	-

Table 8.9: Comparison of heuristics for medium sized instances with a scheduling period of one week

Instance			Priority adjustment		Diving - master		Diving - original	
$\|E\|$	$\|T\|$	LB	Obj	Gap	Obj	Gap	Obj	Gap
30% skilling								
49	923	176	**207**	15.0%	-	-	-	-
44	873	177	**4424**	96.0%	-	-	-	-
47	1326	211	**24603**	99.1%	-	-	-	-
45	1322	200	**24796**	99.2%	-	-	-	-
60% skilling								
48	899	177	**189**	6.3%	-	-	-	-
50	908	176	**188**	6.4%	-	-	-	-
46	1402	416	**21882**	98.1%	-	-	-	-
44	1385	821	**17073**	95.2%	-	-	-	-
100% skilling								
47	885	176	**176**	0.0%	-	-	-	-
46	901	181	**181**	0.0%	-	-	-	-
47	1333	282	**2227**	87.3%	-	-	-	-
44	1359	0	**3926**	100.0%	-	-	-	-

Table 8.10: Comparison of heuristics for large instances with a scheduling period of one week

Instance			Priority adjustment		Diving - master		Diving - original	
$\|E\|$	$\|T\|$	LB	Obj	Gap	Obj	Gap	Obj	Gap
30% skilling								
12	918	4802	**9143**	47.5%	-	-	-	-
12	853	4174	**8547**	51.2%	-	-	-	-
11	1345	13108	**21360**	38.6%	-	-	-	-
12	1289	8485	**14465**	41.3%	-	-	-	-
60% skilling								
11	855	1597	**3409**	53.2%	-	-	-	-
11	893	1707	**3816**	55.3%	-	-	-	-
12	1427	1231	**11866**	89.6%	-	-	-	-
11	1295	1912	**10941**	82.5%	-	-	-	-
100% skilling								
11	954	2812	**5620**	50.0%	-	-	-	-
11	798	685	**2893**	76.3%	-	-	-	-
12	1348	779	**6036**	87.1%	-	-	-	-
12	1343	613	**6538**	90.6%	-	-	-	-

Table 8.11: Comparison of heuristics for instances with a scheduling period of four weeks

the column generation, while the priority adjustment heuristic can obtain a feasible solution before the column generation has terminated. Clearly, for larger problem instances, the time required to solve one node increases, making it less likely for the diving heuristics to find a feasible solution.

Tables 8.12 and 8.13 compare the best results obtained by the column generation based heuristics (*CG based*) with the decomposition heuristics from Chapter 7 (*Horizontal decomp* and *Vertical decomp*), for instances with a scheduling period of one week and four weeks, respectively. Additionally, the results of an integer programming solver on the original formulation with a time limit of one hour are also shown (*MIP-1h*).

For instances considering a one-week scheduling period, the column generation based heuristics find the best solution for 15 out of 36 instances, while the horizontal decomposition finds the best solution for 28 out of 36 instances. When a scheduling period of four weeks is considered, the horizontal decomposition approach always outperforms the best column generation based heuristic. These results strongly advocate the type of heuristic decomposition which was the subject of Chapter 7. However, the column generation based heuristics should not be dismissed completely, as they outperform the other approaches on several instances. For example, on large instances with completely homogeneous workforce, the column generation based heuristics obtain better solutions than both heuristics from Chapter 7 and the integer programming solver, thereby presenting a complementary approach.

8.7 Conclusions

This chapter proposed a reformulation of the MDTSS, and a column generation algorithm to solve its linear programming relaxation. Several of the well-known issues of column generation were addressed by including dual smoothing, Lagrangian relaxation and symmetry breaking structures. The algorithm for solving the pricing problem uses an effective decomposition scheme that significantly reduces the time required to check for negative reduced cost columns. Furthermore, it could be adapted to a heuristic algorithm, resulting in an even smaller calculation time.

The objective value obtained by the column generation was compared with solving the root node of the original formulation. Computational experiments showed that the former presents a stronger lower bound, which was, generally, obtained in less computation time. These results indicate that solving the reformulation in a branch-and-bound framework could lead to optimal integer solutions much quicker than solving the original formulation.

Instance		CG based	Horizontal decomp.	Vertical decomp.	MIP-1h
$\|E\|$	$\|T\|$				
30% skilling					
11	205	**753**	**753**	754	**753**
11	200	**1355**	**1355**	**1355**	**1355**
11	318	**2459**	**2459**	**2459**	**2459**
11	314	**2058**	2078	**2058**	**2058**
23	454	1210	467	909	**205**
24	474	109	228	1715	**101**
23	657	2922	2861	3622	**2620**
23	636	2924	**2380**	3020	**2320**
49	923	207	**185**	3317	2522
44	873	4424	**248**	403	694
47	1326	24603	**5042**	9742	23006
45	1322	24796	**4991**	7435	27908
60% skilling					
11	214	**151**	**151**	**151**	**151**
11	201	**49**	**49**	**49**	**49**
11	353	1558	1638	1558	**1257**
11	326	1458	1157	1356	**1156**
25	444	**90**	**90**	94	91
23	456	**200**	**200**	1004	**200**
23	692	2321	**1816**	2416	2618
23	685	2823	**1856**	3320	3924
48	899	189	**177**	190	802
50	908	188	**176**	1112	203
46	1402	21882	**4737**	6537	22299
44	1385	17073	**7297**	7629	23993
100% skilling					
12	221	**152**	**152**	**152**	**152**
12	234	**154**	**154**	**154**	**154**
11	347	1457	**855**	**855**	**855**
11	355	2358	**1456**	**1456**	**1456**
24	437	**97**	**97**	**97**	**97**
23	503	805	**304**	308	**304**
22	653	1210	**363**	2207	1104
23	670	2216	**1048**	3013	1811
47	885	**176**	**176**	182	182
46	901	**181**	**181**	183	186
47	1333	**2227**	2479	6031	11859
44	1359	**3926**	4548	5825	14061

Table 8.12: Comparison with heuristic decomposition algorithms from Chapter 7 for instances with a scheduling period of one week

Instance		CG based	Horizontal decomp.	Vertical decomp.	MIP-1h
$\|E\|$	$\|T\|$				
30% skilling					
12	918	9143	6353	7035	**5832**
12	853	8547	5970	6330	**5129**
11	1345	21360	**13130**	**13130**	**13130**
12	1289	14465	9502	9739	**9345**
60% skilling					
11	855	3409	2805	5113	**1798**
11	893	3816	**2770**	5620	3408
12	1427	11866	**9132**	14160	-
11	1295	10941	**8148**	12736	8526
100% skilling					
11	954	5620	3093	7718	**2812**
11	798	2893	983	1294	**685**
12	1348	6036	**1875**	4324	3618
12	1343	6538	**2035**	5128	3621

Table 8.13: Comparison with heuristic decomposition algorithms from Chapter 7 for instances with a scheduling period of four weeks

Three heuristic algorithms were introduced, all using the primal and dual solutions found by the column generation algorithm. The presented priority adjustment heuristic was able to find feasible solutions for all instances, whereas the diving heuristics only succeeded in this for small instances with around ten employees. For instances with a large number of employees and tasks, the new column generation based heuristics improved over the results obtained by the heuristics presented in Chapter 7. This illustrates the potential of these algorithms for other problem formulations that have an exponential number of variables.

Part IV

Conclusions

Chapter 9

Conclusions and future research

9.1 Conclusions

This dissertation investigated theory and practice of nurse rostering, thereby making significant scientific, social and industrial contributions. The majority of state of the art studies in nurse rostering focuses on introducing new solution approaches for particular problem variations. As a result, important aspects of nurse rostering are neglected, regarding both fundamental issues and practical implications. The new theoretical insights in this dissertation enable a deeper understanding of nurse rostering problems, and have direct potential implications in the way people will address such problems in the future. Decision support systems, for example, could incorporate the network flow models to address problem relaxations, embedded in a framework to efficiently find optimal solutions for problems previously considered too large or too complex. The practical contributions directly impact the industry by facilitating practitioners towards applying academic results. Evidently, this would benefit the staff too, as the literature has repeatedly illustrated that algorithms generate better solutions than human planners are able to construct.

Furthermore, the dissertation studied problems that combine personnel rostering with other hard combinatorial optimisation problems. The definition of such integrated problems addresses challenges deemed too complex to solve by hand. The contributions in this dissertation, however, show that this type of problem can be solved, allowing for a better resource utilisation, and consequently, higher gains.

Contributions to theory: Part I investigated two issues which are generally neglected in academic research on nurse rostering. Specifically, Chapter 2 identified several nurse rostering problems that can be reformulated as minimum cost network flow problems. The identified problems included constraints on day successions and on the number of days worked in both the complete scheduling period and in subsets thereof. In the light of these new results, previously published complexity proofs were revisited to obtain a deeper understanding of what makes nurse rostering hard. For two types of constraints, it was shown that changing the constraint definition from day-level to shift-level makes the problem hard.

Chapter 3 addressed inconsistencies which occur when rostering nurses in multiple, consecutive scheduling periods. Typically, academic models build upon a static horizon approach to the problem, thereby considering an isolated scheduling period which ignores assignments from preceding, or future, periods. This is a significant abstraction of reality, since in practice, assignments in the current period are strongly influenced by the inertia of the previous scheduling period. To address this issue, stepping horizon policies for two types of constraints were introduced and evaluated through a series of computational experiments. The results showed the impact of incorrectly evaluating constraints at the boundary of the scheduling period, and illustrated the importance of including information of the preceding scheduling period. Specifically, the proposed policies reduced the workload unbalance in a one-year period by up to 50%, while for series constraints, the stepping horizon evaluation policy reduced the number of violations caused by considering a static horizon by 8%.

Contributions to practice: Part II detailed two academic contributions which have found their way to practice. The proposed models and algorithms have been implemented in a commercial software package for personnel rostering and management, and are currently used in hospitals and other organisations in Belgium, the Netherlands, France and Luxembourg.

Chapter 4 presented an alternative to the many academic models that fail to consider vital real world aspects of rostering. A general object model was introduced, suitable for representing the complex problem characteristics that originate from practical nurse rostering problems. Particular attention was devoted to modelling the large variety of organisational and legislative regulations, and personal requests. A structured description of the model in the form of an XSD, as well as a large collection of new benchmark instances based on real world data, have been made publicly available to accommodate those researchers wishing to work on realistic data.

Chapter 5 presented a solution approach to a problem often faced by software vendors and practitioners when implementing a system for automated nurse rostering. To allow algorithms to generate solutions automatically, several parameters need to be configured beforehand, defining the organisation's guidelines and regulations, and their relative importance (the weights). Many organisations consider this configuration phase too difficult and time-consuming, as it requires experienced planners to quantify their implicit knowledge; a task which is also prone to errors. To facilitate this task, a new approach was developed for automatically extracting weights of constraints from historical data. The rationale behind the proposed technique is that constraints with many violations in past rosters are less important and are consequently assigned a low weight. This approach was evaluated on two case studies concerning Belgian hospitals, and included a (subjective) evaluation by the head nurses responsible for creating rosters. The results indicated that the proposed technique succeeds in extracting suitable weights, which, when used in an algorithm for nurse rostering, resulted in workable and at the same time high quality rosters.

Contributions to the integration with other problems: Part III investigated problems which consider the interaction of personnel rostering and task scheduling. In many organisations, these problems are strongly intertwined and cannot be considered separately. The three chapters in this part introduced and discussed problems in which both tasks and shifts need to be assigned simultaneously.

Chapter 6 introduced new hybrid algorithms for the shift minimisation personnel task scheduling problem (SMPTSP). Staff availability is predetermined and cannot be changed, while tasks have to be assigned to the multi-skilled employees. In addition to the traditional first fit and best fit techniques, a new constructive heuristic was presented, which combines integer programming and heuristic search. To further improve the constructed solutions, an algorithm was presented which integrates local branching in an iterative improvement framework. The resulting algorithms succeeded in finding the optimal solutions for all benchmark instances from the literature, for the first time. A series of computational experiments was conducted to investigate the influence of two problem characteristics on the performance of the heuristics and of an integer programming solver. Based on these results, ten new instances were generated, which have since then been adopted by the research community as *hard* instances.

Chapter 7 generalised the SMPTSP to include the decision of *when* employees have to work; decisions have to be made regarding both task and shift assignments. Two variants of this problem were introduced: one considering a single, isolated day, and one considering a longer scheduling period. For both variants, integer programming formulations as well as several exact and heuristic decomposition approaches were presented. Extensive computational experiments investigated the different decomposition schemes using a diverse set of instances. The results showed that the heuristic decomposition of the problem into horizontal subproblems was particularly successful in solving large instances.

Finally, Chapter 8 presented a set covering reformulation of the multiple day task and shift scheduling problem. Due to the large number of variables in the reformulation, column generation was used for solving the linear programming relaxation. The algorithm's performance was improved by stabilising the duals, reducing the tailing-off effect and breaking symmetry in the set covering problem. Computational results showed that, for all tested instances, the lower bound obtained by column generation is stronger than the bound obtained for the integer programming formulation presented in Chapter 7 by an integer programming solver. Finally, three heuristics were presented to generate feasible integer solutions, based on the primal and dual solutions obtained by column generation. The new heuristics succeeded in finding new best solutions to instances for which the approaches from Chapter 7 failed.

Overall, the results in this dissertation contribute towards bridging the gap between theory and practice in nurse rostering from two sides. The theoretical contributions reinforce the academic foundations of the nurse rostering problem, allowing researchers to better understand rostering problems, and the implications of various assumptions. Clearly, such insights are useful for practitioners as well, as they provide a framework in which the consequences of redefining a rostering problem can be evaluated. The impact of the practice-oriented contributions on the research-application gap manifests itself in two ways that facilitate the application of state of the art academic results. Firstly, by providing a means to represent feature-rich rostering problems. Secondly, by automating part of the configuration process of rostering software. The final contributions in this dissertation illustrate that, when integrating rostering with other problems, it remains computationally feasible to obtain (near-)optimal solutions, while enabling decision makers to model problems much closer to reality.

9.2 Future research

The present dissertation has made several new contributions to the state of the art of 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, a list of possible further extensions is given regarding both models and algorithms.

Models: Chapter 2 showed that the common assumption that all nurse rostering problems are hard is not valid. The scope in Chapter 2 was limited to problems with two types of constraints, naturally leading to the question of whether or not there are other nurse rostering problems that can be solved in polynomial time. Problems with constraints on consecutive assignments are of particular interest. Limited appearance of such constraints give rise to dramatic complexity shifts. Investigating properties of the coefficient matrix of integer programming formulations (e.g. total unimodularity) can identify efficiently solvable (sub)problems.

Chapter 3 focused on the impact of the preceding scheduling period. However, future assignments also influence the current scheduling period. Anticipating the future can be realised in several ways, e.g. by considering uncertainty of demand at the start of the next scheduling period through stochastic programming, or by taking into account already fixed assignments similarly to the techniques presented in Chapter 3. While it is expected that the impact of ignoring the next scheduling period(s) is less significant than that of ignoring the preceding period, it does warrant study as it leads to completely consistent constraint evaluation policies for rostering problems.

The technique presented in Chapter 5 calculated new weights based on two solution characteristics: the number of instances of a constraint and the number of violations of a constraint. While this has the advantage of providing a transparent methodology and clear understanding of how each weight was determined, taking into account other characteristics might result in a more accurate approximation of the *real* constraint weight. Apart from the number of violations, the calculation could incorporate the degree of violation, as well as whether the violation was a result of exceeding a maximum or a minimum. Furthermore, the weight extraction problem could be reformulated as a machine learning problem, allowing established techniques to be adapted and applied.

The models for the integrated task and shift scheduling problems introduced in Chapter 7 are the first formalisation of the problems. However, to facilitate their practical application, these models should be enriched, much like was done in Chapter 4 for the rostering problem. For example, in many organisations, only a subset of tasks will be fixed in time. Other tasks have time windows in which they should be executed. Such an extension to the models presented in Chapters 6 and 7 would be interesting both from a practical and academic point of view.

Algorithms: The algorithms introduced in Chapters 6, 7 and 8 illustrate the effectiveness of two algorithmic paradigms: hybridisation and decomposition. Future research should focus on further exploiting a problem's structure with algorithms based on these two principles.

The results in Chapter 2 show that restricted variants of the nurse rostering problem can be solved efficiently. The presented network flow models allow fixing assignments, thereby making these reformulations particularly suitable for enforcing branching decisions. Since only few constraints are captured in the networks, dualising other constraints would allow Lagrangian relaxation to calculate lower bounds on the optimal objective value. These two properties of the reformulations indicate that they are particularly suitable for integration in a branch-and-bound algorithm for rostering problems with complex constraints.

Smet et al. [112] have applied a suite of metaheuristics to the benchmark instances introduced in Chapter 4. However, it remains difficult to evaluate the performance of such results without any indication of what the optimal objective value is. Therefore, an investigation into a lower bounding procedure or an exact algorithm for these complex instances presents a relevant, albeit challenging research objective. Due to the size of the instances and the complex constraint definitions, traditional techniques such as integer programming cannot be straightforwardly applied. Decomposition approaches such as branch-and-price might be more suitable for these instances.

The integrated task and shift scheduling problems discussed in Part III could be optimally solved using decomposition techniques such as branch-and-price or (combinatorial) Benders decomposition. The latter could be conceived in the following way. The master problem would assign shifts to employees, while the subproblem decides whether a given shift roster is feasible, i.e. whether all tasks can be assigned to qualified employees. If not, the subproblem generates cuts which prevent (parts of) solutions to be generated again in the next iterations.

Finally, all instances considered for the problems discussed in Part III have a random skill structure. It was argued that such a structure presents a relevant challenge, as in practice, often, the skill structure might vary significantly among organisations, or even among departments within the same organisation. Nevertheless, hierarchical skill structures are common [37]. Dedicated approaches for problems with such skill structure could possibly outperform the approaches presented in this dissertation, as they currently do not exploit this information in any way.

Bibliography

[1] S. Abdennadher and H. Schlenker. INTERDIP – an interactive constraint based nurse scheduler. In *Proceedings of the First International Conference and Exhibition on the Practical Application of Constraint Technologies and Logic Programming*, 1999.

[2] Ablecare. Verpleegkundigen in vlaamse woonzorgcentra, vraag en aanbod, 2010.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

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

[5] E. Ásgeirsson. Bridging the gap between self schedules and feasible schedules in staff scheduling. *Annals of Operations Research*, 218(1):51–69, 2014.

[6] E. Ballestero. Compromise programming: A utility-based linear-quadratic composite metric from the trade-off between achievement and balanced (non-corner) solutions. *European journal of operational research*, 182(3):1369–1382, 2007.

[7] J. F. Bard and H. W. Purnomo. Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164(2):510–534, 2005.

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

[9] I. Berrada, J. A. Ferland, and P. Michelon. A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences*, 30(3):183–193, 1996.

[10] B. Bilgin. *Advanced Models and Solution Methods for Automation of Personnel Rostering Optimisation.* PhD thesis, KU Leuven, 2012.

[11] B. Bilgin, P. De Causmaecker, B. Rossie, and G. Vanden Berghe. Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research*, 194(1):33–57, 2012.

[12] A. Billionnet. Integer programming to schedule a hierarchical workforce with variable demands. *European Journal of Operational Research*, 114(1):105–114, 1999.

[13] F. Bonomo, G. Duran, and J. Marenco. Exploring the complexity boundary between coloring and list-coloring. *Annals of Operations Research*, 169(1):3–16, 2009.

[14] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[15] 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.

[16] P. Brucker and R. Qu. Network flow models for intraday personnel scheduling problems. *Annals of Operations Research*, 218(1):107–114, 2014.

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

[18] J. O. Brunner, J. F. Bard, and J. M. Köhler. Bounded flexibility in days-on and days-off scheduling. *Naval Research Logistics*, 60(8):678–701, 2013.

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

[20] 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.

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

[22] 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.

[23] E. K. Burke, P. De Causmaecker, S. Petrovic, and G. Vanden Berghe. Fitness evaluation for nurse scheduling problems. In *Proceedings of the Congress on Evolutionary Computation*, pages 1139–1146, 2001.

[24] E. K. Burke, P. De Causmaecker, S. Petrovic, and G. Vanden Berghe. Metaheuristics for handling time interval coverage constraints in nurse scheduling. *Applied Artificial Intelligence*, 20(9):743–766, 2006.

[25] 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):441–499, 2004.

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

[27] R. N. Burns and M. W. Carter. Work force size and single shift schedules with variable demands. *Management Science*, 31(5):599–607, 1985.

[28] A. Caprara, M. Monaci, and P. Toth. Models and algorithms for a staff scheduling problem. *Mathematical programming*, 98(1-3):445–476, 2003.

[29] R. C. Carrasco. Long-term staff scheduling with regular temporal distribution. *Computer methods and programs in biomedicine*, 100(2):191–199, 2010.

[30] M. W. Carter and C. A. Tovey. When is the classroom assignment problem hard? *Operations Research*, 40(S1):28–39, 1992.

[31] V. Chankong and Y. Y. Haimes. *Multiobjective decision making: theory and methodology.* North Holland, 1983.

[32] I. Charon and O. Hudry. The noising methods: A generalization of some metaheuristics. *European Journal of Operational Research*, 135(1):86–101, 2001.

[33] G. Codato and M. Fischetti. Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

[34] C. Coello, G. Lamont, and D. Van Veldhuisen. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Springer Science & Business Media, 2007.

[35] A. A. Constantino, D. Landa-Silva, E. L. de Melo, C. F. X. Mendonça, D. B. Rizzato, and W. Romão. A heuristic algorithm based on multi-assignment procedures for nurse scheduling. *Annals of Operations Research*, 218(1):165–183, 2014.

[36] M.-C. Côté, B. Gendron, and L.-M. Rousseau. Grammar-based integer programming models for multiactivity shift scheduling. *Management Science*, 57(1):151–163, 2011.

[37] P. De Bruecker, J. Van den Bergh, J. Beliën, and E. Demeulemeester. Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243(1):1–16, 2015.

[38] P. De Causmaecker and G. Vanden Berghe. A categorisation of nurse rostering problems. *Journal of Scheduling*, 14(1):3–16, 2011.

[39] F. Della Croce, A. Grosso, and F. Salassa. A matheuristic approach for the total completion time two-machines permutation flow shop problem. In P. Merz and J.-K. Hao, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6622 of *Lecture Notes in Computer Science*, pages 38–47. Springer Berlin Heidelberg, 2011.

[40] 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.

[41] B. Detienne, L. Péridy, E. Pinson, and D. Rivreau. Cut generation for an employee timetabling problem. *European Journal of Operational Research*, 197(3):1178–1184, 2009.

[42] L. Di Gaspero, J. Gärtner, G. Kortsarz, N. Musliu, A. Schaerf, and W. Slany. The minimum shift design problem. *Annals of Operations Research*, 155(1):79–105, 2007.

[43] K. Doerner and V. Schmid. Survey: matheuristics for rich vehicle routing problems. In M. Blesa, C. Blum, G. Raidl, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 6373 of *Lecture Notes in Computer Science*, pages 206–221. Springer Berlin Heidelberg, 2010.

[44] D. Dowling, M. Krishnamoorthy, H. Mackenzie, and D. Sier. Staff rostering at a large international airport. *Annals of Operations Research*, 72:125–147, 1997.

[45] R. G. Drake. The nurse rostering problem: from operational research to organizational reality? *Journal of Advanced Nursing*, 70(4):800–810, 2014.

[46] R. G. Drake. The ‘robust’ roster: exploring the nurse rostering process. *Journal of Advanced Nursing*, 70(9):2095–2106, 2014.

[47] A. Eiben and J. Van Hemert. SAW-ing EAs: Adapting the fitness function for solving constrained problems. In D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, editors, *New Ideas in Optimization*, pages 389–402. McGraw-Hill, 1999.

[48] M. Elahipanah, G. Desaulniers, and E. Lacasse-Guay. A two-phase mathematical-programming heuristic for flexible assignment of activities and tasks to work shifts. *Journal of Scheduling*, 16(5):443–460, 2013.

[49] 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.

[50] P. Eveborn and M. Rönnqvist. Scheduler–a system for staff planning. *Annals of Operations Research*, 128(1-4):21–45, 2004.

[51] J.-G. Fages and T. Lapègue. Filtering AtMostNValue with difference constraints: Application to the shift minimisation personnel task scheduling problem. *Artificial Intelligence*, 212:116–133, 2014.

[52] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming Series B*, 98:23–47, 2003.

[53] M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[54] J. Fürnkranz and E. Hüllermeier. *Preference learning.* Springer, 2010.

[55] 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.

[56] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36(4):873–886, 1989.

[57] U. Gupta, D. Lee, and J.-T. Leung. An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers*, C-28(11):807–810, 1979.

[58] O. Guyon, P. Lemaire, E. Pinson, and D. Rivreau. Cut generation for an integrated employee timetabling and production scheduling problem. *European Journal of Operational Research*, 201(2):557–567, 2010.

[59] S. Hanafi, J. Lazic, N. Mladenovic, C. Wilbaut, and I. Crévits. New hybrid matheuristics for solving the multidimensional knapsack problem. In M. Blesa, C. Blum, G. Raidl, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 6373 of *Lecture Notes in Computer Science*, pages 118–132. Springer Berlin Heidelberg, 2010.

[60] M. Haouari, N. Aissaoui, and F. Z. Mansour. Network flow-based approaches for integrated aircraft fleeting and routing. *European Journal of Operational Research*, 193(2):591–599, 2009.

[61] 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.

[62] B. Hayes, A. Bonner, and J. Pryor. Factors contributing to nurse job satisfaction in the acute hospital setting: a review of recent literature. *Journal of Nursing Management*, 18(7):804–814, 2010.

[63] F. S. Hillier and G. J. Lieberman. *Introduction To Operations Research*. McGraw-Hill Education, 2014.

[64] P. Hulshof, N. Kortbeek, R. Boucherie, E. Hans, and P. Bakker. Taxonomic classification of planning decisions in health care: a structured review of the state of the art in OR/MS. *Health Systems*, 1(2):129–175, 2012.

[65] R. Hung. Single-shift off-day scheduling of a hierarchical workforce with variable demands. *European Journal of Operational Research*, 78(1):49–57, 1994.

[66] A. Ikegami and A. Niwa. A subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3):517–541, 2003.

[67] B. Jaumard, F. Semet, and T. Vovor. A generalized linear programming model for nurse scheduling. *European journal of operational research*, 107(1):1–18, 1998.

[68] C. Joncour, S. Michel, R. Sadykov, D. Sverdlov, and F. Vanderbeck. Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics*, 36:695–702, 2010.

[69] Ö. Kelemci and S. Uyar. Application of a genetic algorithm to a real world nurse rostering problem instance. In *ICEIS (2)*, pages 474–477, 2007.

[70] D. L. Kellogg and S. Walczak. Nurse scheduling: From academia to implementation or not? *Interfaces*, 37(4):355–369, 2007.

[71] J. Kinable. *Decomposition approaches for optimization problems.* PhD thesis, KU Leuven, 2014.

[72] A. Kolen, J. Lenstra, C. Papadimitriou, and F. Spieksma. Interval scheduling : a survey. *Naval Research Logistics*, 54(5):530–543, 2007.

[73] Komarudin, M.-A. Guerry, T. De Feyter, and G. Vanden Berghe. The roster quality staffing problem - a methodology for improving the roster quality by modifying the personnel structure. *European Journal of Operational Research*, 230(3):551–562, 2013.

[74] Komarudin, M.-A. Guerry, P. Smet, T. De Feyter, G. Vanden Berghe, et al. A two-phase heuristic and a lexicographic rule for improving fairness in personnel rostering. In *Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling*, pages 292–308, 2014.

[75] C. T. Kovner, S. Fairchild, S. Poornima, H. Kim, and M. Djukic. Newly licensed RNs' characteristics, work attitudes, and intentions to work. *The American Journal of Nursing*, 107(9):58–70, 2007.

[76] M. Krishnamoorthy and A. Ernst. The personnel task scheduling problem. In X. Yang, K. Teo, and L. Caccetta, editors, *Optimisation methods and application*, pages 434–368. Kluwer, 2001.

[77] M. Krishnamoorthy, A. Ernst, and D. Baatar. Algorithms for large scale shift minimisation personnel task scheduling problems. *European Journal of Operational Research*, 219(1):34–48, 2012.

[78] T. Lapègue, D. Prot, and O. Bellenguez-Morineau. A constraint-based approach for the shift design personnel task scheduling problem with equity. *Computers & Operations Research*, 40(10):2450–2465, 2013.

[79] H. C. Lau. Combinatorial approaches for hard problems in manpower scheduling. *Journal of the Operations Research Society of Japan*, 39(1):88–98, 1996.

[80] H. C. Lau. On the complexity of manpower shift scheduling. *Computers & Operations Research*, 23(1):93–102, 1996.

[81] K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 556–572, 2002.

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

[83] M. E. Lübbecke. Column generation. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, 2010.

[84] B. Maenhout and M. Vanhoucke. Comparison and hybridization of crossover operators for the nurse scheduling problem. *Annals of Operations Research*, 159(1):333–353, 2008.

[85] B. Maenhout and M. Vanhoucke. Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling*, 13(1):77–93, 2010.

[86] B. Maenhout and M. Vanhoucke. An integrated nurse staffing and scheduling analysis for longer-term nursing staff allocation problems. *Omega*, 41(2):485–499, 2013.

[87] B. Maenhout and M. Vanhoucke. Reconstructing nurse schedules: Computational insights in the problem size parameters. *Omega*, 41(5):903–918, 2013.

[88] V. Maniezzo, T. Stutzle, and S. Voss, editors. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, 2009.

[89] M.-E. Marmion, C. Dhaenens, L. Jourdan, A. Liefooghe, and S. Verel. Nils: A neutrality-based iterated local search and its application to flowshop scheduling. In P. Merz and J.-K. Hao, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6622 of *Lecture Notes in Computer Science*, pages 191–202. Springer Berlin Heidelberg, 2011.

[90] S. Martin, D. Ouelhadj, P. Smet, G. Vanden Berghe, and E. Özcan. Cooperative search for fair nurse rosters. *Expert Systems with Applications*, 40(16):6674–6683, 2013.

[91] A. Meisels and A. Schaerf. Modelling and solving employee timetabling problems. *Annals of Mathematics and Artificial Intelligence*, 39(1-2):41–59, 2003.

[92] T. Messelis and P. De Causmaecker. An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 233(3):511–528, 2014.

[93] S. Mirrazavi and H. Beringer. A web-based workforce management system for Sainsburys Supermarkets Ltd. *Annals of Operations Research*, 155(1):437–457, 2007.

[94] M. Mısır. *Intelligent Hyper-heuristics: A Tool for Solving Generic Optimisation Problems.* PhD thesis, KU Leuven, 2012.

[95] M. Mısır, P. Smet, and G. Vanden Berghe. An analysis of generalised heuristics for vehicle routing and personnel rostering problems. *Journal of the Operational Research Society*, 66:858–870, 2015.

[96] N. Musliu, A. Schaerf, and W. Slany. Local search for shift design. *European Journal of Operational Research*, 153(1):51–64, 2004.

[97] OECD. *Health at a Glance 2013: OECD Indicators.* OECD Publishing, 2013.

[98] J. B. Orlin. Max flows in O(nm) time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774, 2013.

[99] E. L. Ormeci, F. S. Salman, and E. Yücel. Staff rostering in call centers providing employee transportation. *Omega*, 43:41–53, 2014.

[100] T. Osogami and H. Imai. Classification of various neighborhood operations for the nurse scheduling problem. In G. Goos, J. Hartmanis, J. Leeuwen, D. Lee, and S.-H. Teng, editors, *Algorithms and Computation*, volume 1969 of *Lecture Notes in Computer Science*, pages 72–83. Springer Berlin Heidelberg, 2000.

[101] D. Parr and J. Thompson. Solving the multi-objective nurse scheduling problem with a weighted cost function. *Annals of Operations Research*, 155(1):279–288, 2007.

[102] D. W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793, 2007.

[103] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. In-out separation and column generation stabilization by dual price smoothing. In V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, editors, *Experimental Algorithms*, pages 354–365. Springer, 2013.

[104] S. Petrovic, G. Beddoe, and G. Vanden Berghe. Storing and adapting repair experiences in personnel rostering. *Lecture Notes in Computer Science*, 2740:148–165, 2003.

[105] G. Post, S. Ahmadi, S. Daskalaki, J. Kingston, J. Kyngas, C. Nurmi, and D. Ranson. An XML format for benchmarks in high school timetabling. *Annals of Operations Research*, 194(1):385–397, 2012.

[106] H. W. Purnomo and J. F. Bard. Cyclic preference scheduling for nurses using branch and price. *Naval Research Logistics (NRL)*, 54(2):200–220, 2007.

[107] R. Robinson, R. Sorli, and Y. Zinder. Personnel scheduling with time windows and preemptive tasks. In *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, pages 561–566, 2005.

[108] M. Rocha, J. F. Oliveira, and M. A. Carravilla. Cyclic staff scheduling: optimization models for some real-life problems. *Journal of Scheduling*, 16(2):231–242, 2013.

[109] L.-M. Rousseau, M. Gendreau, and D. Feillet. Interior point stabilization for column generation. *Operations Research Letters*, 35(5):660–668, 2007.

[110] F. Salassa and G. Vanden Berghe. A stepping horizon view on nurse rostering. In *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling*, pages 161–174, 2012.

[111] R. Silvestro and C. Silvestro. An evaluation of nurse rostering practices in the national health service. *Journal of Advanced Nursing*, 32(3):525–535, 2000.

[112] P. Smet, B. Bilgin, P. De Causmaecker, and G. Vanden Berghe. Modelling and evaluation issues in nurse rostering. *Annals of Operations Research*, 218(1):303–326, 2014.

[113] P. Smet, P. De Causmaecker, B. Bilgin, and G. Vanden Berghe. Nurse rostering: A complex example of personnel scheduling with perspectives. In A. S. Uyar, E. Özcan, and N. Urquhart, editors, *Automated Scheduling and Planning*, volume 505 of *Studies in Computational Intelligence*, pages 129–153. Springer Berlin Heidelberg, 2013.

[114] P. Smet, S. Martin, D. Ouelhadj, E. Özcan, and G. Vanden Berghe. Fairness in nurse rostering. Technical report, KU Leuven, 2013.

[115] P. Smet and G. Vanden Berghe. A matheuristic approach to the shift minimisation personnel task scheduling problem. In *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling*, pages 145–160, 2012.

[116] P. Smet, T. Wauters, M. Mihaylov, and G. Vanden Berghe. The shift minimisation personnel task scheduling problem: A new hybrid approach and computational insights. *Omega*, 46:64–73, 2014.

[117] B. Vaidyanathan, K. C. Jha, and R. K. Ahuja. Multicommodity network flow approach to the railroad crew-scheduling problem. *IBM Journal of Research and Development*, 51(3):325–344, 2007.

[118] 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.

[119] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

[120] F. Vanderbeck. Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 331–358. Springer, 2005.

[121] M. Vanhoucke and B. Maenhout. NSPLib – a nurse scheduling problem library: a tool to evaluate (meta-)heuristic procedures. In *Proceedings of the 31st Annual Meeting of the working group on Operations Research Applied to Health Services*, pages 151–165, 2007.

[122] M. Warner. Nurse staffing, scheduling, and reallocation in the hospital. *Hospital & Health Services Administration*, 21(3):77–90, 1976.

[123] L. A. Wolsey and G. L. Nemhauser. *Integer and combinatorial optimization.* John Wiley & Sons, 2014.

[124] T. Zeitlhofer and B. Wess. List-coloring of interval graphs with application to register assignment for heterogeneous register-set architectures. *Signal Processing*, 83(7):1411–1425, 2003.

Publications

Articles in internationally reviewed academic journals

T. Wauters, J. Kinable, **P. Smet**, W. Vancroonenburg, G. Vanden Berghe, and J. Verstichel. The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling*, 2015.

M. Mısır, **P. Smet**, and G. Vanden Berghe. An analysis of generalised heuristics for vehicle routing and personnel rostering problems. *Journal of the Operational Research Society*, 66:858-870, 2015.

P. Smet, T. Wauters, M. Mihaylov, and G. Vanden Berghe. The shift minimisation personnel task scheduling problem: a new hybrid approach and computational insights. *Omega*, 46:64-73, 2014.

P. Smet, B. Bilgin, P. De Causmaecker, and G. Vanden Berghe. Modelling and evaluation issues in nurse rostering. *Annals of Operations Research*, 218(1):303-326, 2014.

S. Martin, D. Ouelhadj, **P. Smet**, G. Vanden Berghe, and E. Özcan. Cooperative search for fair nurse rosters. *Expert Systems with Applications*, 40(16):6674-6683, 2013.

Article in academic book, internationally recognised scientific publisher

P. Smet, P. De Causmaecker, B. Bilgin, and G. Vanden Berghe. Nurse rostering: a complex example of personnel scheduling with perspectives. In A. S. Uyar, E. Özcan, and N. Urquhart, editors, *Automated Scheduling and Planning*, volume

505 of *Studies in Computational Intelligence*, pages 129-153. Springer Berlin Heidelberg, 2013.

Papers at international scientific conferences and symposia, published in full in proceedings

P. Smet, P. Brucker, P. De Causmaecker, and G. Vanden Berghe. Polynomially solvable formulations for a class of nurse rostering problems. *Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2014).* York, United Kingdom, 26-29 August 2014, pages 408-419, 2014.

Komarudin, M.-A. Guerry, **P. Smet**, T. De Feyter, and G. Vanden Berghe. A two-phase heuristic and a lexicographic rule for improving fairness in personnel rostering. *Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2014).* York, United Kingdom, 26-29 August 2014, pages 292-308, 2014.

P. Smet, G. Vanden Berghe. A matheuristic approach to the shift minimisation personnel task scheduling problem. *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012).* Son, Norway, 28-31 August 2012, pages 145-160, 2012.

M. Mısır, **P. Smet**, K. Verbeeck, and G. Vanden Berghe. Security personnel routing and rostering: a hyper-heuristic approach. *Proceedings of the 3rd International Conference on Applied Operational Research (ICAOR 2011).* Istanbul, Turkey, 24-26 August 2011, pages 193-205, 2011.

Meeting abstracts, presented at international scientific conferences and symposia, published or not published in proceedings or journals

W. Vancroonenburg, **P. Smet**, and G. Vanden Berghe. A two phase heuristic approach to multi-day surgical case scheduling considering generalized resource constraints and desiderata. *Operational Research Applied to Health Services (ORAHS 2014).* Lisbon, Portugal, 20-25 July 2014.

P. Smet, P. Brucker, P. De Causmaecker, and G. Vanden Berghe. A network flow formulation and computational experiments for a class of nurse scheduling

problems. *New Challenges in Scheduling Theory.* Aussois, France, 31 March - 4 April 2014.

P. Smet and G. Vanden Berghe. A heuristic approach to an integrated personnel rostering and task assignment problem. *Proceedings of the 6th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2013).* Ghent, Belgium, 27-30 August 2013.

P. Smet and G. Vanden Berghe. A hybrid constructive algorithm for the integrated task and shift scheduling problem. *The 10th Metaheuristics International Conference (MIC 2013).* Singapore, 5-8 August 2013.

P. Smet and G. Vanden Berghe. A decomposition approach for the integrated task and shift scheduling problem. *Proceedings of the 26th European Conference on Operational Research (EURO 2013).* Rome, Italy, 1-4 July 2013.

M. Garraffa, **P. Smet**, and G. Vanden Berghe. A hybrid heuristic for a real world task assignment problem. *Proceedings of the 26th European Conference on Operational Research (EURO 2013).* Rome, Italy, 1-4 July 2013.

D. Ouelhadj, S. Martin, **P. Smet**, E. Özcan, and G. Vanden Berghe. Fairness and cooperation in nurse rostering. *Proceedings of the 26th European Conference on Operational Research (EURO 2013).* Rome, Italy, 1-4 July 2013.

P. Smet, S. Martin, D. Ouelhadj, E. Özcan, and G. Vanden Berghe. Investigation of fairness measures for nurse rostering. *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012).* Son, Norway, 28-31 August 2012.

P. Smet and G. Vanden Berghe. A comparison of fairness objectives for nurse rostering. *Operational Research Applied to Health Services (ORAHS 2012).* Enschede, Netherlands, 15-20 July 2012.

S. Martin, **P. Smet**, D. Ouelhadj, E. Özcan, and G. Vanden Berghe. Agent-based cooperative meta-heuristic search for fairness in nurse rostering. *The 25th European Conference on Operational Research (EURO 2012).* Vilnius, Lithuania, 8-11 July 2012.

P. Smet, W. Vancroonenburg, and G. Vanden Berghe. Automated scheduling at a joinery site. *Proceedings of the 5th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2011).* Phoenix, USA, 9-11 August 2011.

Meeting abstracts, presented at other scientific conferences and symposia, published or not published in proceedings or journals

P. Smet and A. Ernst. Lower bounds for an integrated task and personnel scheduling problem. *The 29th Annual Conference of the Belgian Operations Research Society (ORBEL 29).* Antwerp, Belgium, 5-6 February 2015.

P. Smet and G. Vanden Berghe. Integrated task scheduling and personnel rostering problems. *The 28th Annual Conference of the Belgian Operations Research Society (ORBEL 28).* Mons, Belgium, 30-31 January 2014.

P. Smet, T. Wauters, and G. Vanden Berghe. Hardness analysis and a new approach for the shift minimisation personnel task scheduling problem. *The 27th Annual Conference of the Belgian Operations Research Society (ORBEL 27).* Kortrijk, Belgium, 7-8 February 2013.

M. Mihaylov, **P. Smet**, and G. Vanden Berghe. Automatic Constraint Weight Extraction for Nurse Rostering: A Case Study. *The 27th Annual Conference of the Belgian Operations Research Society (ORBEL 27).* Kortrijk, Belgium, 7-8 February 2013.

P. Smet and G. Vanden Berghe. A study on fairness objectives for nurse rostering. *The 26th Annual Conference of the Belgian Operations Research Society (ORBEL 26).* Brussels, Belgium, 2-3 February 2012.

P. Smet, M. Mısır, and G. Vanden Berghe. An introduction to new application domains for the home care scheduling problem. *The 25th Annual Conference of the Belgian Operations Research Society (ORBEL 25).* Ghent, Belgium, 10-11 February 2011.

FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE
COMBINATORIAL OPTIMISATION AND DECISION SUPPORT (CODES)
Celestijnenlaan 200A box 2402
B-3001 Heverlee