KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Science
Department of Computer Science

Advanced Models and Solution Methods for Automation of Personnel Rostering Optimisation

Burak BİLGİN

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Science

March 2012

Advanced Models and Solution Methods for Automation of Personnel Rostering Optimisation

Burak BİLGİN

Jury:
Prof. Dr. Ir. Maurice Bruynooghe, president
Prof. Dr. Patrick De Causmaecker, promotor
Prof. Dr. Ir. Greet Vanden Berghe
Prof. Dr. Ir. Dirk Cattrysse
Prof. Dr. Daniel De Schreye
Prof. Dr. Graham Kendall (The University of Nottingham)
Prof. Dr. Mario Vanhoucke (Ghent University)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Science

March 2012

D/2012/7515/26
ISBN 978-94-6018-490-1

Preface

Although a PhD appears to be a personal work on the surface, the present dissertation would not be possible without the input of numerous individuals and organisations.

First of all, I would like to thank my promoters Prof. Dr. Patrick De Causmaecker and Prof. Dr. Ir. Greet Vanden Berghe for trusting me with a PhD candidacy, for their continuous efforts to guide me throughout the execution of the project, and for their encouragement throughout the most difficult phases of the project.

I would like to thank the members of my jury, Prof. Dr. Ir. Maurice Bruynooghe, Prof. Dr. Ir. Dirk Cattrysse, Prof. Dr. Daniel De Schreye, Prof. Dr. Graham Kendall and Prof. Dr. Mario Vanhoucke, for their countless remarks on the project and the dissertation. Their feedback has improved the quality of the present dissertation significantly.

The research reported in the present dissertation has been executed within several projects, namely DINGO II (IWT 060376), RAP (IWT / TETRA 060353) and GPS-Plan (IWT 080356). I would like to thank IWT, Institute for the Promotion of Innovation by Science and Technology in Flanders, for funding these projects.

The real world relevance of the present dissertation would be limited without the input provided by the industrial partners of the aforementioned research projects. I would like to thank SAGA NV and GPS NV, Belgian software companies that provide personnel management solutions, for their collaboration. Benoît Rossie (SAGA NV) and Ingo Schelfhout (GPS NV) have provided the challenges their customers were facing, so that these challenges could be incorporated in the problem model and eventually tackled in the solution methods. Moreover, they have incorporated these solution methods in their software and provided them to their customers for invaluable feedback. I have to use this opportunity to thank the health care practitioners that have provided feedback to improve the models and solution methods. I will be glad if the results of the present dissertation will improve the working conditions of these individuals, whose work we take granted most of the time.

The work reported in the present dissertation has also been reported in several academic publications. I would like to thank my co-authors, Prof. Dr. Patrick De Causmaecker, Prof. Dr. Ir. Greet Vanden Berghe, Dr. Peter Demeester, Mustafa Mısır, Wim Vancroonenburg, Pieter Smet and Benoît Rossie for their contributions to these publications. I would also like to thank Dr. Erik Van Achter for his feedback on the use of language in the first draft of this dissertation and for the Dutch translation of the abstract.

I would like to thank Dr. Ender Özcan for introducing me to the research field of heuristic methods when I was a MSc student in Yeditepe University (Istanbul), guiding me through a research project on hyperheuristics, and eventually introducing me to my current promoters as a PhD candidate. Without his efforts and vision, I would not have the chance to start a PhD studentship at KU Leuven.

Besides all the individuals and organisations that have contributed to this PhD professionally, the support of my family members was vital to handle the personal aspects of life as a PhD student. I have to thank Çiğdem and Roel Wondergem for their help and guidance in living and studying in a foreign country. I have to thank my parents, Emin and Meral Bilgin, for their continuous, unconditional support and motivation throughout my life.

Abstract

Personnel rostering is the practice of assigning daily working units, called shifts, to employees with respect to a variety of constraints, such as workforce requirements, labour legislation, union demands and individual employee preferences. The shortage of qualified personnel in the developed countries makes the personnel rostering problem one of the fundamental problems to be addressed in the private and public sector. Personnel rostering is a dynamic problem that evolves in time with the changes in the organisational structures, labour legislations and union demands. The objective of this dissertation is to present a complete solution, ready to be implemented, addressing the real world personnel rostering problem.

The contribution of the present dissertation can be summarised as follows. An extended problem model is introduced to represent the complexity and the diversity of the real world problem instances. An in-depth analysis of the differences between the constraint evaluation methods of the academic literature and the real world practice is reported. The constraint evaluation methods of the real world practice are presented formally and discussed in great detail, referring to examples from the real world. An algorithmic toolbox is developed that consists of the elements from the academic literature as well as new modules that are designed to take into account the extensions made to the problem model. Three quantitative measures are proposed for assessing the properties of a problem instance. Problem instances gathered from real world sources are published as a benchmark data set. An extensive set of experiments and statistical analysis are performed on the benchmark data set and on the problem instances from academic sources. The experimental results are processed to conclude on the most suitable configuration of the algorithmic toolbox on a given problem instance. The resulting body of PhD work benefits both academia and industry, by extending the scope of the personnel rostering theory to cover the complexity and the diversity of the real world practice.

Nederlandse Samenvatting

Personeelsplanning kan worden omschreven als de praktijk van het toewijzen van de dagelijkse arbeidstijd-eenheden (de zogenaamde shifts), waarbij rekening wordt gehouden met een verscheidenheid aan beperkingen zoals werkkrachtbehoeften en de voorkeuren van de individuele werknemer. Het tekort aan gekwalificeerd personeel in de post-industriële samenleving maakt van de personeelsplanning een van de meest fundamentele problemen zowel in de private als in de publieke sector.

Personeelsplanning is een dynamisch gegeven dat mee-evolueert in de tijd, gestuwd door veranderingen in de organisatiestructuren, de arbeidswetgeving en dat rekening houdt met de eisen van het sociale overleg. Het doel van de hiervoor liggende thesis is om meteen een complete oplossing te presenteren, direct implementeerbaar en in staat om de personeelsplanning aan te pakken in de dagelijkse werkelijkheid.

Daartoe wordt in de eerste plaats een uitgebreid model geïntroduceerd om de complexiteit en de diversiteit van het reële probleem weer te geven. Een diepgaande analyse legt de verschillen bloot tussen de evaluatiemethodes uit de wetenschappelijke literatuur en die uit de praktijk.

Vervolgens worden de beperkingen in de evaluatiemethoden van de reële wereld formeel gepresenteerd en in detail besproken met voorbeelden. Een algoritmische *toolbox*, die deels gebaseerd is op elementen uit de wetenschappelijke literatuur, omvat ook nieuwe modules ontworpen om rekening te houden met de uitbreidingen van het model. Drie kwantitatieve maten worden voorgesteld voor de beoordeling van de eigenschappen van een probleeminstantie. Verschillende instanties verzameld in reële planningssituaties worden gepubliceerd als een verzameling benchmarkdata. Een uitgebreide reeks experimenten en een statistische analyse worden nadien uitgevoerd op de benchmarkgegevens en op de probleeminstanties van academische bronnen. Op basis van de experimentele resultaten wordt de meest geschikte onfiguratie van de algoritmische *toolbox* bepaald voor een bepaalde probleeminstantie. De hieruit resulterende doctoraatsverhandeling is dan ook een bijdrage zowel voor de academische wereld als voor de industrie.

Symbols

AR_e	the set of all absence requests of employee $e \in E$
$ar_{e,D',S',K'}$	the absence request of employee $e \in E$ on day set $D' \subset D$, shift type set $S' \subset S$, and skill type set $K' \subset K$
CC	the set of all coverage constraints
$cc_{d,S',K'} \in CC$	a coverage constraint defined on day $d \in D$, for shift type set $S' \subset S$, and skill type set $K' \subset K$
$csv(c)$	counter start value of a counter c
$crv(c)$	counter remainder value of a counter c
D	the set of all days in the current schedule period and in the related parts of the previous and upcoming schedule period
$dc_{e,D',S',K'}$	domain counter defined on day set $D' \subset D$, shift type set $S' \subset S$, and skill type set $K' \subset K$ of employee $e \in E$
$dic_{e,D'}$	days idle counter defined on day set $D' \subset D$ of employee $e \in E$
dis_e	days idle series of employee $e \in E$
$dwc_{e,D'}$	days worked counter defined on day set $D' \subset D$ of employee $e \in E$
dws_e	day worked series of employee $e \in E$
E	the set of all employees
$hwc_{e,D'}$	hours worked counter defined on day set $D' \subset D$ of employee $e \in E$
$JobTime(ar)$	net job time of absence request ar
$JobTime(s)$	net job time of shift type $s \in S$
K	the set of all skill types
$K_e \subset K$	the set of the skill types that employee $e \in E$ has
$l_{E',D',S',K'}$	the collaboration constraint defined on employee set $E' \subset E$, day set $D' \subset D$, shift type set $S' \subset S$, and skill type set $K' \subset K$

$m(x)$	the maximum threshold of constraint x
$n(x)$	the minimum threshold of constraint x
P_{rest}	the total amount of rest time between shift types penalties
P_{skill}	the total amount of employee skill type penalties
$p(x)$	the penalty of constraint x
$p_{e,d}$	auxiliary variable that denotes the presence of employee $e \in E$ on day $d \in D$
$p_{e,d,S'}$	auxiliary variable that denotes the presence of employee $e \in E$ on day $d \in D$ in any of the shift types $s \in S' \subset S$
$p_{e,D',S',K'}$	auxiliary variable that denotes the presence of employee $e \in E$ on any of the days $d \in D' \subset D$, in any of the shift types $s \in S' \subset S$, and with any of the skill types $k \in K' \subset K$
$q_{e,w}$	auxiliary variable that denotes the presence of employee $e \in E$ on weekend $w \in W$
R	the set of shift type pairs (s_i, s_j) such that at least one of s_i and s_j violate the rest time of the other if they are assigned on consecutive days d and $d+1$, respectively
S	the set of all shift types
sst	a variable of type *successive series token*
$sstl$	a list of *successive series tokens*
$sucseries$	a successive series constraint
$swc_{e,D',S'}$	shift types worked counter on day set $D' \subset D$ and shift type set $S' \subset S$ of employee $e \in E$
$sws_{e,S'}$	shift types worked series of employee $e \in E$ defined on shift type set $S' \subset S$
T	the set of shift type pairs (s_i, s_j) such that s_i and s_j overlap if they are assigned on consecutive days d and $d+1$, respectively
$v(x)$	the value of constraint x
W	the set of all weekends in the current schedule period and in the related parts of the previous and upcoming schedule period
w_{rest}	the weight for the rest time between shift types constraint
$wic_{e,W'}$	weekends idle counter of employee $e \in E$ defined on weekends set $W' \subset W$
wis_e	weekends idle series of employe $e \in E$
$wwc_{e,W'}$	weekends worked counter of employee $e \in E$ defined on weekends set $W' \subset W$
wws_e	weekends worked series of employe $e \in E$
$w_{e,k}$	the penalty for making an assignment with skill type $k \in K_e$ to employee $e \in E$
$w(x)$	the weight of constraint x
X	the set of all soft constraints
$x_{e,d,s,k}$	decision variable, whether employee e is assigned on day d for shift type s and skill type k

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The low birth rates and longer life expectancies have deteriorated the age structure in developed countries [59]. The increasing ratio of senior citizens to working people has several direct and indirect impacts. Firstly, the demand for a qualified workforce cannot always be met with the personnel supply at hand. Secondly, the imbalance in the ratio of senior citizens to working people results in higher social security contributions by working people. These facts eventually result in higher employment costs by private and public institutions, which have to compete for qualified personnel and pay the increasing social security contributions for their personnel. As a result, a qualified workforce is one of the scarce and expensive resources that need to be utilised carefully in order to execute economically feasible operations.

Offering higher wages is one method that the institutions use to compete with each other for qualified personnel. An alternative is offering them flexible working hours, an effective recuperation regime and honouring their preferences. Institutions have to balance the preferences of their personnel with the requirements of their own operations. They are bound by legal restrictions to maintain a safe and ergonomic working environment. Consequently, the personnel rostering problem is one of the most challenging, complicated and critical decision problems that is faced by institutions depending on qualified workforce.

Healthcare is one of the sectors that suffer from a shortage of qualified personnel. The workforce demand spans over seven days a week and 24 hours a day and it has to be met by a limited supply of qualified personnel. The working regime of the personnel is governed by an abundance of strict labour legislations, workforce requirements, union demands and personal requests of individual employees. For example, an employee should not work more than six days in a row. He or she is not allowed to work an early shift after a late shift. A series of five late shifts must

be followed by at least two free days. Such rules not only increase the quality of life of the personnel, they also ensure sufficient recuperation and reduce the risk of critical mistakes.

Personnel rostering, a critical and hard task in the healthcare sector, is the responsibility of the head nurses of the respective wards. It has to be carried out among many important tasks, sometimes extending beyond their working hours. Interviews from the first hand have revealed that the preparation of a four week plan by a head nurse exceeds three weeks. This task is perceived as hard, time-consuming and unrewarding. A decision support system that automates personnel rostering is highly appreciated by the real world practitioners.

The personnel rostering problem has been a subject of academic research for more than 40 years. Two comprehensive literature studies have been published recently: *The state of the art of nurse rostering* by Burke et al. [26] and *Staff scheduling and rostering: A review of applications, methods and models* by Ernst et al. [30]. A diverse set of sectors and countries have been the subject of the personnel rostering literature. Transportation, call centres, healthcare, protection, emergency, civic services and utilities, venue management, financial services, hospitality, tourism, retail and manufacturing sectors have been indicated as the main application areas of personnel planning solutions by Ernst et al. [30]. The nurse rostering problems from Belgium [14], the Netherlands [17, 18], the United Kingdom [1, 23], the United States [4], Norway [31] and Italy [5] have been introduced in the scientific literature. The literature studies have scrutinised the problem models, methods and scheduling process types reported in the literature and concluded with the fact that there is a significant gap between the academia and real world practice in nurse rostering. The main reason for this gap has been pointed out to be the academic problem models that are too simple to address the requirements of the real practice. Addressing the needs and requirements of the real world fully has been indicated as a crucial scientific goal by Burke et al. [26].

The objective of this dissertation is to present a complete personnel rostering solution that answers the requirements of the real world practice. In the real world, the problem instances vary over different countries, sectors and institutions. Even different wards in the same institution have distinct personnel rostering problems. Furthermore, the problem definitions evolve over time with changes in the labour legislation, union demands and organisational structure. Therefore, the solution methods have to be able to deal with a diverse set of problem instances. The personnel rostering approach has been developed in close collaboration with two Belgian companies[1]. The resulting solution has been implemented in the personnel management software of these industrial partners and utilised in the field by their customers, which are mostly in the healthcare sector.

The author of the present dissertation has experienced the personnel rostering

[1]SAGA Consulting, GPS NV Belgium

process in the field at first hand and worked with the real world practitioners. If done manually, the personnel rostering process of some wards takes more than three weeks. The utilisation of the solution methods reported in the present dissertation has reduced this process to less than an hour in the automated case. The latter duration includes the analysis of the result by the planner, few manual adaptations to incorporate the planner's personal view and the re-execution of the algorithm if necessary. The net execution time of the algorithm does not exceed ten minutes in most of the cases. The time savings and higher quality of the personnel rosters obtained by the solution methods are recognised by the end users and can be observed in the experimental results presented in Chapter 8, where the quality of the manually and automatically generated rosters are compared.

General purpose approaches are beneficial for both academia and industry, but generic models are required to express a great variety of problem instances. Ernst et al. [30] have pointed to the generalisation of models and methods as a research direction worthy of investigation. Burke et al. [26] have drawn the conclusion that most of the rostering models in the scientific literature are too limited to address the needs and requirements of real world problems. The personnel rostering literature is mostly focused on solution methods instead of on developing generic problem models that are able to express a broad range of problem instances. The development of models that can cover the complexity of the real world problems have been indicated as one of the most relevant research directions in nurse rostering [26].

Throughout the collaboration with industry, personnel rostering problems have been encountered that cannot be expressed with the models presented in the scientific literature. Therefore, a new model with various extensions to the generic model in [14] has been developed (Chapter 3). The new model was inspired by the nurse rostering problem in the Belgian healthcare sector, where a diverse set of problem instances have to be dealt with. It can be categorised as ASBI|RVN|PL according to the notation introduced in [46]. This description reflects the fact that the extended model can express a variety of real world requirements, such as *individual skill types*, *variable number of shift types*, *personnel* and *coverage constraints* as objectives. The resulting problem model is flexible and numerous constraints can be defined and configured with parameters and weights. The model can be further extended with new constraints should it be impossible to express these with the current model. For instance, the problem model is not limited to three distinct *shift types* but allows the user to define as many *shift types* as needed that are configured to address the requirements of the problem at hand.

Accurate evaluation of constraints is essential for the personnel rostering practice. A correct and detailed evaluation function is a first step towards automated decision support of the planners. Such a function has been carefully developed by the author in close collaboration with the industry professionals, planning software developers and end-users. It is not only an essential component of the rostering algorithm, but

it is also helpful to the planners while they are updating the rosters. Ideally, the quality measure used during automated rostering matches the perceived quality measures of the practitioners. Most of the time, this objective cannot be satisfied completely, because the real world practitioners do not use mathematical formulas when evaluating rosters. The real world practitioners use their personal perceptions, which tend to be subjective and vary over time. Consequently, they are inclined to carry out manual adaptations in the generated roster, even if these adaptations result in a worse roster according to the objective function based on the criteria they have expressed at first hand. It has been observed that the representation of the violations in the graphical user interface (GUI) plays an important role in this phenomenon. If a violation with a low importance is represented in a more intense way, then the end users tend to fix it, even if it induces a violation with a higher importance. It is also possible that the vast amount of constraints involved in personnel rostering is beyond the capacity of humans to take into account simultaneously. That means that an end user can fail to recognise a violation induced to a constraint while trying to fix another one. Another reason for the need of manual adaptations may be the impulse to assert a personal control over the situation. However, the assessment of this theory is beyond the scope of the present dissertation. On the other hand, the judgement of the end users are consistent and objective when it comes to the most important constraints. Real world practitioners will not accept any roster that does not satisfy the most important constraints.

The mismatch between academia and real world is one of the fundamental explanations of the fact that the academic solution methods are not commonly utilised in the real world practice. The divergence between the constraint evaluation methods of the academia and the real world practice have been scrutinised in depth on examples from the scientific literature in the present dissertation (Chapter 4). In addition to that, constraint evaluation methods that satisfy the demands of the real world practice have been described formally in the same chapter. The formal descriptions of the constraint evaluation methods incorporate elements that are new to the scientific literature, because they are based on the extended problem model presented in Chapter 3. The utilisation of the constraint evaluation methods presented in Chapter 4 will align the solution methods with the expectations of the real world practitioners. Quantitative measures of problem properties have been introduced in Chapter 5.

Although complexity indicators are not new to the scientific literature [52, 55, 65, 68], the measures introduced in Chapter 5 incorporate elements that address the extensions in the problem model presented in Chapter 3, which makes them unique and relevant to the real world practice.

Burke et al. [26] have advised to evaluate the performance of the solution methods on real world benchmark problems. The idea behind the maintenance of a public benchmark library is to allow researchers to test their solution methods on problems with different properties and compare their results to the state of the art [16].

The test data should ideally stem from the real world practice and exhibit the diversity of the instances encountered in the real world. A nurse rostering data set derived from real world instances is maintained online by the ASAP group at the University of Nottingham (the Nottingham benchmarks for short)[2]. The problem instances in the benchmark data set have been subject of the nurse rostering literature and they have been gathered from different countries. A second real world benchmark set, the KAHO benchmarks, has been gathered from Belgian hospitals throughout collaboration with software developers active in the Belgian human resources planning sector. The KAHO benchmarks have been published online[3] and their properties have been discussed in Chapter 6. The quantitative measures of the Nottingham and KAHO benchmarks, presented in Chapter 6, indicate the complexity and diversity of both benchmark sets.

The scope of the present dissertation has been limited to the real world personnel rostering problems. To increase the probability of finding good enough solutions on a diverse set of real world problem instances, various algorithm configurations have been applied to the problem model presented in Chapter 3. The algorithm configurations reported in Chapter 7 form an algorithmic toolbox. The composition of an integrated, complete algorithmic toolbox is one of the contributions of the present dissertation. The algorithmic toolbox consists of variable neighbourhood search (VNS), adaptive large neighbourhood search (ALNS), hyperheuristics and a set of neighbourhoods and low level heuristics. A subset of the low level neighbourhoods and heuristics of the toolbox have been developed by the author to utilise the new elements of the extended problem model presented in Chapter 3.

An extensive set of experiments has been carried out and a statistical analysis on the experimental results has been reported in Chapter 8. The objective of the experiments has been to determine the most promising configuration of the algorithmic toolbox on a given real world problem instance whose quantitative measure values are within a particular range. The KAHO benchmarks have been used as test data and the Nottingham benchmarks have been used to verify the derived relations. The quantitative measures have been used to divide the KAHO benchmarks into four groups. In total, 54 algorithm configurations have been experimented with and a thorough statistical analysis has been carried out on the experimental results (Chapter 8). The result of the statistical analysis points out the most promising configuration of the algorithmic toolbox on each problem instance group. This relationship has been utilised in the automated personnel rostering software of the industrial partners directly benefiting from the research project. That way, their software has been enabled to autonomously decide on the most promising configuration of the algorithmic toolbox for a given problem instance.

Although the relation derived from the experimental results is of significant value to

[2] http://www.cs.nott.ac.uk/∼tec/NRP/
[3] http://ingenieur.kahosl.be/vakgroep/it/nurse/archive.htm

the real world practitioners, it cannot be generalised to all the personnel rostering problems in its current form. Such a comprehensive study is beyond the scope of the present dissertation and pointed out as future work.

The problem model, evaluation method, quantitative measures and solution methods reported in the present dissertation have been implemented in the personnel management software of the industrial partners of the research project. The planner, for example a head nurse in a hospital ward, can use the graphical user interface (GUI) to enter the input data, such as workforce demand and individual constraints, into the system. Several GUI forms and dialogs enable the planner to enter the input data into the system efficiently, save it in a database and then use this data in planning tasks.

When the planner calls the automated personnel rostering functionality, the system collects the necessary data from the database and converts it to the problem model (Chapter 3), selects the most promising configuration of the algorithmic toolbox using the quantitative measures (Chapter 5) and experimental results (Chapter 8) and calls the solution method (Chapter 7), which utilises the real world constraint evaluation methods (Chapter 4). That way several tasks that require a human expert is automated by the personnel management software. These tasks include analysing the problem instance, experimenting with several algorithms and determining the most promising one. The elimination of a human expert in the personnel rostering procedure results in time and cost savings. Another advantage is that a software product can be duplicated and deployed on multiple sites, where a human expert can be involved only in a single site at a time. The economic value produced as a result is significant.

The research reported in the present dissertation has the potential to serve both academia and industry. The results can be directly implemented in personnel management software and deployed in practice. This path is already being explored by two Belgian software companies. At the time of writing, the automated personnel rostering solution documented in the present dissertation was deployed in more than 20 institutions in Belgium, mostly hospitals and rest homes, but also including a retail store and a municipality including its fire department. The conclusions of the present dissertation serve various future research directions for the academic community.

Chapter 2

Academic context

The personnel planning process has been analysed in three phases by Ernst et al. [30]: to determine the size of the staff needed (staffing), to allocate the individual *employees* to *shifts* (rostering) and to assign the tasks to individuals for each *shift*. In practice, the different phases of the personnel planning process are dealt with in different planning horizons (Burke et al. [26]). The staffing phase involves the recruitment of staff and is therefore conceived in longer planning horizons such as a period of one year. Seasonal variations are usually taken into account during the staffing phase. The rostering phase, allocating the individual *employees* to *shifts*, is handled in *schedule periods* of several weeks, typically four. Assigning the tasks to individuals is handled in *schedule periods* of several days [26].

The problem model and solution methods in this dissertation have been developed in order to solve the second phase of the personnel planning problem, i.e., rostering. The objective of personnel rostering is constructing a *schedule* by assigning *shifts* to *employees* [14]. It is a challenging problem, because it is constrained by workload requirements, legal restrictions and contractual agreements, some of which can be conflicting. For example, while workload requirements might induce busier rosters for the personnel, legal restrictions might forbid overloaded rosters. The personnel rostering models and solution methods in this dissertation are primarily based on the nurse rostering problem because of its challenges, such as complex constraints and its relevance to real world practices.

The problem model takes the result of the staffing phase, namely the *coverage constraints*, as input. Several factors are bound to influence the *coverage constraints*. The contractual and preferential constraints of the *employees* have an impact on which types of shifts they work on. Mostly, the full time and half time *employees* work on different shift types. The number of *employees* that can work on a certain shift type needs to be considered while calculating the *coverage constraints*, because

the *coverage constraints* are defined on a date, a set of shift types and a set of skill types. The organisational structure of an institution has an impact on its capacity; for example, on the number of patients that can be served in a hospital. The capacity has a direct impact on the workload, which needs to be taken into account when calculating the *coverage constraints* [26]. The third phase however, assigning the tasks to individuals for each *shift*, is not tackled by the solution method.

Burke et al. [26] have provided a thorough review of the nurse rostering literature, up to 2004. The review paper has presented a terminology set related to nurse rostering problems. The literature has been scrutinised considering problem models, scheduling process types and solution methods. The literature survey has concluded by pointing out nine key areas of the nurse rostering practice for future research. These are *multi-criteria reasoning*, *flexibility and dynamic reasoning*, *robustness*, *ease of use*, *human computer interaction*, *problem decomposition*, *exploitation of problem specific information*, *hybridisation* and *interdisciplinarity*. The academic context of the present dissertation complements Burke et al.'s survey by concentrating on the articles published since that time. Its uniqueness lies in pointing out the challenges that arise from real world applications of personnel rostering approaches.

2.1 Nurse rostering problems

Different working schemes and demands throughout the world lead to a diverse set of problem properties. Even within the same country, each institution and each ward in the institution has its own requirements and organisational structure. As a result, the personnel rostering problem presents a high degree of diversity in addition to its complexity. De Causmaecker and Vanden Berghe [48] have proposed a reference model to categorise the personnel rostering problems according to their properties such as the *personnel environment*, *work characteristics* and *optimisation objective*. The categorisation of the problems will help researchers to study the complexity and hardness of the problem instances and the efficiency of the corresponding algorithms. The categorisation is presented in Table 2.1 with the permission of the authors [48].

Nurse rostering problems are known to be NP-hard [60]. They have been treated mostly as *combinatorial optimisation problems* (COP) in the scientific literature because of their overconstrained nature. The term *solution space* refers to all possible solutions of a COP. Two types of constraints can be modelled in COPs: *hard* and *soft constraints*. *Hard constraints* must be satisfied and *soft constraints* are preferred to be satisfied. The solutions that satisfy all the *hard constraints* of a COP are called the *feasible solutions*. The *soft constraints* are addressed with an *objective function* (Formula 2.1), which is a linear combination of the *weight* (w_c) and the number of violations (n_c) of each soft constraint ($c \in C$). The *weight*

α Personnel environment	Personnel constraints		Skill interactions	
	A	Availability	2,3,...	Fixed number
	S	Sequences	N	Variable number
	B	Balance	I	Individual skill definitions
	C	Chaperoning		
β Work characteristics	Coverage constraints		Shift type	
	R	Range	2,3,...	Fixed number
	T	Time Intervals	N	Variable number
	V	Fluctuating	O	Overlapping
γ Optimisation objective	Objective		Mode	
	P	Personnel constraints	M	Multi objective
	L	Coverage constraints		
	X	Number of personnel		
	R	Robustness		
	G	General		

Table 2.1: Classification of nurse rostering problems by De Causmaecker and Vanden Berghe [48]

of a soft constraint indicates the priority of that soft constraint in a COP. The objective of a COP is to find the *feasible solution(s)* in the *solution space* that has the lowest possible value for the *objective function* (Formula 2.1).

$$\sum_{c \in C} w_c n_c \tag{2.1}$$

Vanhoucke and Maenhout [68] have developed ten *complexity indicators* in four groups for the nurse rostering problem. The indicators are based on problem properties such as problem size, preferences of the nurses, coverage constraints and time related constraints, which restrict the individual schedules of the nurses. The indicators can be used to predict the performance of exact and heuristic methods on a given problem instance. Secondly, they can help to select the most promising algorithm from a set of algorithms to solve a given problem instance. Furthermore, the indicators allow for generating artificial test data under a controlled design. NSPGen is a tool for generating test data using the indicators under controlled design. NSPLib, a data set generated with NSPGen, has been proposed as a benchmark data set to the research community [68]. In order to address the real world challenges faced when developing the solution methods in the present dissertation, the standard academic problem models are extended with several elements. For an accurate assessment of the problem properties, the extensions

to the standard models need to be considered as well. A new set of quantitative measures have been introduced in Chapter 3 to take into account the extensions to the standard models.

Messelis et al. [52] have utilised structural and formal features of the nurse rostering problem to predict algorithm behaviour on a particular problem instance. The studies have been carried out on two data sets, one consisting of small size problem instances and another set consisting of full size problem instances. The structural problem features were size dependent features, coverage constraint structure, workforce structure, contract and request related features. The formal features were structure, balance and Horn proximity based features of the SAT model of the nurse rostering problem under study. The structural and formal features have successfully predicted the quality of the solution in form of an objective value found by a metaheuristic, which was a variable neighbourhood implementation. The resulting approach can be utilised in a system where the solution quality of a problem instance needs to be calculated quickly without finding the actual solution, such as agent negotiation systems between hospital wards [52].

Silvestro and Silvestro [65] have identified four hardness measures: "ward size, demand variability, demand predictability and complexity of skill mix." They have suggested the use of these measures to decide on the operation mode based on a survey they have carried out in hospitals in the United Kingdom. They have recommended self-rostering, team-rostering and departmental rostering for low, mid and high hardness problem instances, respectively. Although operation modes are not the same as algorithms, the proposed measures can be used for determining the most promising algorithms for a given problem instance as well.

Several solution methods have been proposed for tackling the nurse rostering problem in Belgian hospitals, which is also the application field of this dissertation. In Belgian hospitals, the *schedule periods* are flexible, problem elements like *shift types* and *skill types* are user defined, legal restrictions and contractual agreements impose complex constraints and cyclic *assignments* are not the standard practice [14, 25, 47]. Furthermore, the problem has a dynamic nature due to the ever changing labour legislation, contractual agreements and nurse preferences. New aspects and constraint types are introduced in the course of time. The models and solution methods of the automation tools have to comply with the latest state of the problem with the least amount of intervention possible.

2.2 Nurse rostering benchmarks

Benchmarks are a set of problem instances from a particular domain. They are used to test the relative performance of solution methods for that problem domain. Researchers that are new to a problem domain utilise the benchmark problems as

an entry point. Benchmark problems set standards in problem model structure and constraint evaluation methods. The success of the research activities on a particular domain is closely related to how accurately the benchmark problems sample the problem domain. Ideally, the benchmark problems should be an accurate representation of the complexity and diversity of the problem domain. The requirements of the problem domain should be reflected in the benchmark problems. Benchmark problems based on simplified problem models will lead to solution methods that provide unsatisfactory results when applied to the real world problems with higher complexity. Similarly, if benchmark problems do not cover the diversity of a problem domain, the resulting solution methods will be fine-tuned to a limited subset of the problem domain. In this section, the existing nurse rostering benchmarks will be discussed thoroughly.

The Nottingham benchmarks are nurse rostering problem instances collected from Belgium, Canada, France, Greece, Hong Kong, Italy, Japan, the Netherlands, Norway, Saudi Arabia, Spain, the United Kingdom and the United States [20]. The Nottingham benchmarks are maintained in a dedicated website[1]. An XML Scheme Definition (XSD) of the problem, the benchmark instances, their problem descriptions and best known solutions are provided. Furthermore, the source code of the objective function and a visualisation tool are provided publicly so that researchers can verify the solutions they have found. The publication of the problem instances, the source code of the objective function and best known solutions make the research process transparent. This practice allows the verification and reproduction of the results obtained. In addition to that, the visualisation tool and the objective function help the researchers to validate their solutions and see how their approaches compare to the state of the art.

The Nottingham benchmarks share the following hard constraints.

- *Coverage constraints*
- *Single assignment of a certain shift type to a nurse on a given day*
- *Honouring the skill types of assignments by nurses*

The last hard constraint imposes that a nurse needs to have the *skill type* that is required by the assignments allocated to him or her. For example, if an assignment requires the skill type *head nurse*, then the assigned nurse must have the skill type *head nurse* as one of his or her *skill types*. The only hard constraint that is not shared by all of the instances is the *single shift assignment per day per nurse* constraint.

In addition to the hard constraints, the Nottingham benchmark instances have a diverse set of soft constraints. Since the origins of the instances vary over different

[1] http://www.cs.nott.ac.uk/∼tec/NRP/

countries, the sets of soft constraints that apply to each instance vary as well. A discussion about the soft constraint sets of each benchmark instance is beyond the scope of this dissertation. Some real world examples of soft constraints are *no night shifts before a free weekend*, *maximum six consecutive working days* and *maximum two consecutive working weekends*. The details of the soft constraints of each instance can be found in the website. However, the way the soft constraints are evaluated are the same among all the benchmark instances. The set of soft constraints and the weights of each soft constraint is a part of the input data. The aim of the Nottingham benchmarks is to find feasible solutions, thus satisfying all the hard constraints, with the minimum objective function value.

The objective values of the best known solutions and the origins of the instances are given in Tables 2.2 and 2.3. The objective values in bold are the proven optimals with respect to the problem definition of the Nottingham benchmarks. If the solution method of a best known solution is published in the scientific literature, a reference to the publication is also provided.

2.3 Approaches to the Nottingham benchmarks

Burke et al. [20] have stated that some of the smaller instances have been solved to optimality using CPLEX10. The solutions of some of the larger instances have been proven to be optimal via lower bounds using relaxations and decompositions.

Burke et al. [19] have found the best known solution for BCV-6.13.1 using *variable depth search*. *Variable depth search* involves the application of a chain of moves to a candidate solution. The moves in this case were swapping blocks of days between nurses. Applying the swaps concurrently, as chains, helps the algorithm to escape from local optima. Various heuristics have been applied to select the next move to include in the chain. The best results have been achieved with the combination of three heuristics. Two of these are inspired from the manual planners: try to repair a day that has a violation as a result of the last swap and try to improve the schedule of the nurses involved in the current swap. The third heuristic restricts the execution time spent on every chain.

Burke et al. [20] have applied a *scatter search algorithm* to the Nottingham benchmarks. The *scatter search algorithm* maintains a population of diverse candidate solutions throughout the execution. Existing candidates are combined to generate new candidate solutions. New candidate solutions are accepted to the population according to their quality and their contribution to the diversity of the population. Similar to the *memetic algorithms*, newly generated candidate solutions are improved using a *local search algorithm*. The *local search algorithm* in this case was the *variable depth search* introduced in [19]. The resulting solution

Instance	Best known solution	Reference(s) for best solutions	Origin of instance
BCV-1.8.1	252	[20]	Belgium
BCV-1.8.2	853	-	Belgium
BCV-1.8.3	232	-	Belgium
BCV-1.8.4	291	-	Belgium
BCV-2.46.1	**1572**	[20]	Belgium
BCV-3.46.1	3280	-	Belgium
BCV-3.46.2	**894**	[20]	Belgium
BCV-4.13.1	**10**	[20]	Belgium
BCV-4.13.2	**10**	-	Belgium
BCV-5.4.1	**48**	[20, 57]	Belgium
BCV-6.13.1	768	[19]	Belgium
BCV-6.13.2	392	-	Belgium
BCV-7.10.1	**381**	[20]	Belgium
BCV-8.13.1	**148**	[16, 20]	Belgium
BCV-8.13.2	148	-	Belgium
BCV-A.12.1	1294	-	Belgium
BCV-A.12.2	1953	-	Belgium
ORTEC01	**270**	[33]	The Netherlands
ORTEC02	**270**	[33]	The Netherlands
GPost	**5**	[33]	The Netherlands
GPost-B	**3**	[33]	The Netherlands
QMC-1	**13**	-	UK
QMC-2	**29**	-	UK
Ikegami-2Shift-DATA1	**0**	[38]	Japan
Ikegami-3Shift-DATA1	**2**	-	Japan
Ikegami-3Shift-DATA1.1	**3**	-	Japan
Ikegami-3Shift-DATA1.2	**3**	-	Japan
Millar-2Shift-DATA1	**0**	[20, 38, 57]	Canada
Millar-2Shift-DATA1.1	**0**	[20]	Canada
Valouxis-1	**20**	-	Greece
WHPP	**5**	-	France
LLR	**301**	[20]	Hong Kong
Musa	**175**	[57]	USA
Ozkarahan	**0**	[57]	USA
Azaiez	**0**	[2, 57]	Saudi Arabia

Table 2.2: Best known solutions and the origins of the Nottingham benchmarks. Values in bold characters refer to optimal solutions. (part 1)

Instance	Best known solution	Reference(s) for best solutions	Origin of instance
SINTEF	**0**	-	Norway
MER	7081	-	Canada
CHILD	**149**	-	Canada
ERRVH	**2001**	-	Canada
ERMGH	**779**	-	Canada
CHILD-A2	1111	-	Canada
ERMGH-A	795	-	Canada
ERMGH-B	1459	-	Canada
ERRVH-A	2197	-	Canada
ERRVH-B	6859	-	Canada
MER-A	9915	-	Canada
HED01	**136**	-	Spain
BCDT-Sep	**100**	-	Italy

Table 2.3: Best known solutions and the origins of the Nottingham benchmarks. Values in bold characters refer to optimal solutions. (part 2)

method has achieved eight optimal and two best known solutions in the Nottingham benchmarks (see Table 2.2). The execution times were under 15 minutes.

Métivier et al. [57] have solved five instances from the Nottingham benchmarks to optimality (see Table 2.2). The solution method was a *variable neighbourhood search* which involves a *limited discrepancy search* to create new solutions and a *constraint programming approach* to filter the solutions using *soft global constraints*. The *constraint optimisation problem* is transformed into a *constraint satisfaction problem* (CSP) by associating a *violation measure* and a *cost variable* to each *soft global constraint*. In the resulting CSP, all constraints are hard and the sum of all the *cost variables* are to be minimised.

Limited discrepancy search is a backtracking tree search strategy that uses a given heuristic to choose a path at each decision point [37]. If completely traversing the tree does not yield a solution, the backtracking mechanism is executed and the suggestion of the heuristic is ignored at a limited number of decision points, which are called *discrepancies*. After each failing iteration, the number of *discrepancies* is increased. The search is executed until a termination criterion is met.

Ikegami and Niwa [38] have solved the Ikegami-2Shift-DATA1 and Millar-2Shift-DATA1 instances to optimality. The basis of the solution method was to construct overall rosters by fixing the schedules of the individual nurses. Two variations of the solution method have been presented: a *tabu search* and a *branch-and-bound* variant. The solution methods have been developed to tackle problems with two

and three *shift types*.

Brucker et al. [16] have found the optimal solution for BCV-8.13.1 using a two stage constructive method similar to the approach in [38]. In the first stage, the algorithm constructs high quality *shift sequences*. The *shift sequences* from the first stage are used as building blocks in the second stage, where the algorithm generates the overall roster iteratively by constructing schedules for each nurse. The nurse schedules and overall rosters are improved with *greedy local search* during and after the roster construction.

Azaiez and Al Sharif [2] have solved the instance Azaiez to optimality using a *0-1 goal programming approach*. The model of the instance was decomposed into three groups of five, four and four nurses each and solved independently using LINGO, a linear programming optimisation software[2].

In the practical application of their approach in the Riyadh Al-Kharj hospital in Saudi Arabia, Azaiez and Al Sharif [2] came across some continuity problems between two *schedule periods*. The twelve hour shifts in the Azaiez instance result in 24 hour continuous working time, if a schedule ends with a *night shift* and the following schedule starts with a *day shift*. Another similar issue is caused by the *no more than four consecutive working days* constraint. Suppose that three consecutive days are assigned to a nurse at the end of a schedule and three consecutive days are assigned at the start of the following schedule. Both sequences do not pose a constraint violation in their respective *schedule periods*, but they cause violations if the *schedule periods* are considered continuously. In order to overcome this pitfall, Azaiez and Al Sharif [2] have introduced additional constraints based on the schedules of the nurses in the previous *schedule period* in their real world implementation.

Glass and Knight [33] have solved the ORTEC and GPost instances to optimality within half an hour using a mixed integer linear program approach. This was accomplished by deriving rules from presumptions such as the structure of the *schedule period* and the structure of the *coverage constraints* throughout the week, which are satisfied by the problem instances under study. This fact restricts the application of the solution method to a limited group of problem instances that satisfy these presumptions.

Furthermore, Glass and Knight [33] have pointed out that the Nottingham benchmarks are defined for a single isolated *schedule period*. The problem definition implies that the constraints are evaluated in the current *schedule period* and the impact of the schedule elements such as *assignments* and *holidays* from the previous and upcoming *schedule periods* are not taken into account. This approach does not reflect the real world practice, because there are constraints that apply to more than one *schedule period*. Glass and Knight [33] have proposed a methodology for

[2]http://www.lindo.com/

handling such constraints between multiple *schedule periods*.

The KAHO benchmarks, presented in Chapter 6, are a set of nurse rostering problem instances collected from Belgian hospitals. The real world requirements of the nurse rostering problem are reflected in the KAHO benchmarks. Their problem definition incorporates continuous evaluation of the constraints over multiple *schedule periods*. The KAHO benchmarks include actual *assignment* data from the previous *schedule period* to allow the continuous evaluation. The complexity of the real world problems are reflected in the KAHO benchmarks by including composite constraints and constraint parameters. Problem instances with different sizes and difficulty levels ensure the diversity of the KAHO benchmarks. These properties distinguish the KAHO benchmarks from other benchmarks in the nurse rostering literature.

2.4 Problems from various sources

Bard and Purnomo [4] have modelled a nurse rostering problem from a U.S. hospital as an *integer program* and solved it with *Lagrangian relaxation*. Two approaches have been experimented with: the relaxation of the *coverage constraints* and the relaxation of the nurse requests. The first approach has performed better than the second. The test instances could be solved to near-optimality within 20 minutes. The test data consisted of 15 problem instances based on the nurse rostering problem in a 400 bed U.S. hospital. The number of nurses varied between 20 and 100, and the number of contracts between five and 20 among the problem instances. The set of hard and soft constraints, the *schedule period* and the *shift type* structure were the same among all the instances. The *schedule period* was two weeks, because the *counter constraints*, such as the *hours worked counter* and *the number of weekends worked counters* were all defined on periods of two weeks. The continuity between the *schedule periods* were addressed using the assumption that the schedules were cyclic. That means that the schedule repeats itself after the last day of the *schedule period*. Although the test data is not a part of the Nottingham benchmarks, it is published on the website of the Nottingham benchmarks[3].

Burke et al. [18] have developed a three phase iterative solution method. *Variable neighbourhood search* is carried out on the *schedule* in the first phase. The resulting *schedule* is mutated by removing *assignments* from a set of nurses and the *schedule* is repaired using a *heuristic ordering method*. The solution method outperforms the *genetic algorithm* that was deployed in a commercial package[4] for instances with less than 20 nurses. For instances with larger sizes, the *genetic algorithm*

[3] http://www.cs.nott.ac.uk/∼tec/NRP/
[4] Harmony

appeared to be more successful on average. The resulting solution method has been deployed in the commercial package mentioned.

The test data used in [18] has been provided by the industrial partner of the project[5] as a representation of their client's needs. The test data consists of 16 instances. The number of nurses varies between 12 and 30 among the instances. Consequently, the *threshold* values of the *coverage constraints* also vary. There is some variety in the working hours and individual preferences of the nurses. Other properties are the same among all the instances. The number of *shifts types* is fixed to four and the *schedule period* to one month. The same set of hard and soft constraints and the same objective function apply to all of the instances. Fixed *shift type* structure, *schedule period* and the application of the same objective function to all instances might restrict the application of the resulting solution method to the problem instances with the same properties. The *schedule periods* of the KAHO benchmarks vary between four and thirteen weeks (Table 6.1). The numbers of their *shift types* vary between nine and 27 (Table 6.2). The objective function of the KAHO benchmarks is specific to each hospital ward and presented in the respective XML files[6].

Burke et al. [24] have developed a two stage hybrid solution method for the nurse rostering problem. The hard constraints and a subset of the soft constraints are modelled as an *integer programming* model and solved using ILOG CPLEX 10.0 in the first stage. The resulting solution is improved with a *variable neighbourhood search* approach that takes into account all hard and soft constraints in the problem. The test data is derived from the situation in the intensive care unit in a Dutch hospital. The test data consists of twelve instances which are the variations of the same instance over twelve months. It involves four *shift types* and one *skill type*. The uniformity of the test data has resulted in a problem specific *integer programming model*, which needs modification in order to be executable on a different problem. Furthermore, the *integer programming model* does not take into account the assignments in the previous *schedule period*, which might be relevant to the constraints such as *no stand-alone night shift* and *maximum number of consecutive working days*.

The experimental results in [24] have indicated that the *integer programming - variable neighbourhood search hybrid* performs better than the *hybrid genetic algorithm* in [63] and the *hybrid variable neighbourhood search algorithm* in [18] both of which are implemented in a commercial rostering software[7].

Aickelin et al. [1] have developed a *memetic estimation of distribution algorithm* for the nurse rostering problem. The *local search method* used in the *memetic algorithm* was an *ant-miner algorithm*. The solution method has been tested on

[5] ORTEC
[6] http://ingenieur.kahosl.be/vakgroep/it/nurse/archive.htm
[7] Harmony

data that originates from a major UK hospital. The original problem involves three *shift types* and the solution method has been executed on two *shift types*, treating the early and late shifts as day shifts. The *schedule period* used in the hospital is five weeks. However, the problem has been decomposed into instances with *schedule periods* of one week. This raises the question of the relation between the solutions of consecutive weeks. It has been stated in [1] that historic information is taken into account when calculating the penalties for each weekly assignment. However, the penalties of the weekly assignments within a *schedule period* of five weeks might vary according to the assignments in the previous and subsequent weeks. A mechanism that handles the penalty of each week dynamically according to the previous and subsequent weeks might be needed.

Maenhout and Vanhoucke [44] have applied an electromagnetic metaheuristics to NSPLib, a synthetic benchmark data set introduced in [68]. The basic idea of the electromagnetic metaheuristic is similar to *particle swarm optimisation*, which is a solution method based on social intelligence [42]. Both approaches utilise a population where high quality particles attract others and low quality particles repel others on their trajectory throughout the *solution space*. Similar to the practice in the memetic algorithms, the particles have been improved with a local search algorithm at each iteration. *Variable neighbourhood search* (VNS) has been used as the local search algorithm. Three strategies have been used as neighbourhoods in the VNS: optimise the shift-patterns of a given nurse, optimise the shifts assignments on a given day and optimise the global roster.

2.5 Real world implementation

Kellogg and Walczak [40] have scrutinised 50 academic nurse rostering models reported in the scientific literature between 1985 and 2005. They have reported that only 15 out of the 50 models were implemented in real world settings, and only 7 out of the 15 are known to be still in use at the date of the publication. Several reasons have been given for the low ratio of implemented academic models. One of the reasons is the little focus on the real world requirements such as individuality of the nurses, their collaborations with each other and their preferences, which might be too complex for mathematical programming alone [40].

Burke et al. [26] have mentioned three different administrative modes of rostering operation. All the nurses in a hospital are scheduled in one administrative department in the *centralised scheduling* mode. In the *unit scheduling* mode, the head nurses or unit managers are responsible for the scheduling of the nurses in their ward. Some hospitals allow for *self-scheduling* or *interactive scheduling* meaning that staff can schedule themselves.

Variances in the administrative modes of operation [26] continue to exist in the

recent nurse rostering literature. Nurse rostering is a global practice with local applications. Each country, sometimes each hospital, has its own culture of working practices. Although popular with nurses, *self-scheduling* is a difficult operation mode to apply in real world settings [3]. Wang and Wang [69] and Rönnberg and Larsson [64] have proposed automated solutions to overcome the reported difficulties of *self-scheduling* in practice. *Preference scheduling* grants nurses control over their schedules, albeit not as much as in *self-scheduling*. De Grano et al. [50] have proposed a two stage *preference scheduling* approach, where the preferences of the nurses are balanced with the hospital constraints.

Isken [39] has pointed out *tour scheduling*, also known as *cyclic scheduling*, as a solution to the well known challenges of the nurse rostering problems, such as multiple contract types. The solution methods in this dissertation generate *acyclic rosters*. *Cyclic rosters* do not apply to the problems considered in this dissertation, because the problems involve *employees* with individual qualifications, contractual and preferential constraints. The same roster does not satisfy the individual constraints of all the *employees*. The problem model allows variations in *coverage constraints* and various *absence and assignment requests* by *employees*.

2.6 Solution methods

Artificial intelligence approaches, *constraint programming*, *metaheuristics* and *mathematical programming* approaches have been the most common solution methods in the scientific literature on personnel planning [30]. The size and complexity of nurse rostering problems make it difficult to tackle them with exact optimisation methods [26]. *Metaheuristics* are able to find good quality solutions for real world problems in acceptable execution times while they do not need explicit mathematical models. The solution methods presented in this dissertation have been based on *metaheuristics* and *hyperheuristics*. The variants of *metaheuristics* and *hyperheuristics* used in the solution methods will be reviewed in this section.

Metaheuristics are two level search strategies. They deploy local improvement procedures on the low level and they guide the local improvement procedures towards the promising regions of the *solution space* using high level strategies [34].

A *neighbourhood* is the set of all *solutions* that can be derived from a given *solution* using a rule. For example, the set of all *schedules* that can be derived from an *input schedule* by assigning a single shift constitutes the *neighbourhood* of that *schedule* with respect to the *assign shift rule*.

2.6.1 Tabu search

Tabu search is a *metaheuristic* that involves short term memory to guide the search process [35]. The search is carried out on a single *candidate solution* using a given rule to define a *neighbourhood*. An *initial solution* has to be fed into the algorithm as an input. *Tabu search* keeps the executed moves in a *tabu list* for a limited time in order to avoid repeating them. At each iteration, the best *solution* in the *neighbourhood* of the *candidate solution* that is not in the *tabu list* is nominated as the *candidate solution*. The iterative step is repeated until a *termination criterion* is reached. A move that results in an overall *best solution* is accepted even if it is in the *tabu list*. This practice is called the *aspiration criterion*. The length of the *tabu list* is modified throughout the execution. It is shortened in order to intensify the search in the promising areas of the *solution space*, which is indicated by improving moves. It is lengthened in order to diversify the search in the non-promising areas, where no improving moves can be found.

2.6.2 Variable neighbourhood search

Variable neighbourhood search (VNS) is a metaheuristic that involves "systematic change of neighbourhood within a local search" [36]. In contrast to the *tabu search*, VNS involves more than one *neighbourhood*. Several variations, extensions and hybrids of VNS have been presented in [36]. One of the hybrid solution methods is the combination of VNS and *tabu search*. In a variation of this hybrid, the *tabu search algorithm* is executed on multiple *neighbourhoods* which share the same *tabu list*. Token-ring search is a VNS variant, where the neighbourhoods are held in a circular queue that determines their application sequence [32].

2.6.3 Adaptive large neighbourhood search

Similar to VNS, *adaptive large neighbourhood search* (ALNS) utilises a set of neighbourhoods and explores one neighbourhood at each iteration [62]. The *neighbourhood* is selected in a stochastic way using the *roulette wheel method*, where a probability of selection is assigned to each *neighbourhood*. The probabilities of the *neighbourhoods* are increased in three cases: an overall best solution is found, a solution is found that is better than the current solution, or the solution found is feasible and not *tabu*. The probabilities of the *neighbourhoods* are updated regularly by putting more emphasis on the performance in recent iterations [62].

2.6.4 Hyperheuristics

Cowling et al. [28] have defined the term *hyperheuristics* as "heuristics to choose heuristics" to describe a class of combinatorial optimisation methods. *Hyperheuristics* are heuristic search methods that consist of three layers. The *problem model* and the *objective function* are defined on the lowest level. Heuristics that operate directly on the *problem model* and *objective function* are positioned on the middle level. These are also called *low level heuristics*. A heuristic that coordinates the *low level heuristics* operates on the highest level of the architecture. An imaginary *problem domain barrier* is assumed between the *high level heuristic* and the *low level heuristics*. Problem specific information is not allowed to pass this barrier. In other words, the heuristic at the top level only uses problem-independent information such as statistics about the execution time, objective function improvement and the number of calls to a heuristic [22].

The three level architecture allows the developers to apply a *hyperheuristic* implementation to different combinatorial optimisation problems quickly. Once the problem model, objective function and a set of low level heuristics are implemented, they can be deployed in a *hyperheuristic* to tackle the problem under study. This development process is expected to be quicker than the implementation of some tailor-made *metaheuristic* applications. The ease of implementation, however, is expected to come with a slight decrease in the quality of the end results, which is mostly a preferred compromise in the real world practice. This expectation has been expressed as "good enough, soon enough, cheap enough" in the scientific literature.

While collaborating with industrial partners, the author of the present dissertation has witnessed the benefits of the hyperheuristic approach in the real world at first hand. A so-called *agile software development process* is widely expected in the software industry of today. Software companies expect their development teams to quickly implement extra features without compromising the rest of the software. The compartmentalised methodology of the hyperheuristics helps the development teams to fulfill that expectation. For example, an additional feature in the problem model requires little or no modification in the *low or high level heuristics*. A new *low level heuristic* can be implemented and integrated in a hyperheuristic system without any modification in the remainder of the system. The utilisation of the hyperheuristics has allowed the author of the present dissertation to quickly address the requests of the industrial partners. This has been achieved by concentrating on the module addressing the request without having to modify the remainder of the system.

The *hyperheuristic* research has evolved in two main branches. *Hyperheuristics* that select heuristics and *hyperheuristics* that generate heuristics [21]. The solution methods reported in the present dissertation fall into the category of the *selection hyperheuristics*. The *selection hyperheuristics* are composed of two components:

a *selection method* and an *acceptance criterion*. At each iteration, a heuristic is selected by the *selection method* and applied to the *candidate solution*. The modified candidate solution is accepted or rejected according to an *acceptance criterion*. Algorithm 1 presents the pseudocode of the *selection hyperheuristic* [21].

Algorithm 1 Pseudocode of the selection hyperheuristic [21]

```
F(C) := Objective Function
C_0 := Initial Candidate
C ← C_0
BC ← C_0
while Termination criterion not met do
  H ← Select a heuristic
  C* ← H(C)
  if Accept(C*) then
    C ← C*
    if F(C) ≤ F(BC) then
      BC ← C
    end if
  end if
end while
return BC
```

Simple random and *choice function* have been introduced as *selection methods* in [28]. *Simple random* selects a heuristic from the set of low level heuristics in a random fashion with equal probabilities for all heuristics.

A *choice function* maintains a value for each heuristic and selects the heuristic with the highest value at each iteration [28]. The equation of choice function is given in Formula 2.2. The value for heuristic H_i is the weighted sum of three functions. F_1 keeps track of the heuristic performance and it is updated using Formula 2.3 after the execution of heuristic H_i. F_2 keeps track of the performance of a sequence of two heuristics and it is updated after the execution of heuristic sequence H_p, H_i using Formula 2.4. F_3 keeps track of the time passed since heuristic H_i is last called (Formula 2.5). $T(H_i)$ in Formula 2.4 and $T(H_p, H_i)$ in Formula 2.5 refer to the execution times of the heuristic H_i and the heuristic sequence (H_p, H_i), respectively.

The first two variables help to intensify the search by putting more emphasis on the performance of the heuristics. The last variable helps to diversify the search by putting emphasis on heuristics that have not been called recently. α, β and δ vary in the interval of [0.01, 0.99]. They are increased or decreased at each iteration by the magnitude of the change of the variables they are related to. These coefficients serve two purposes. The first purpose is to reduce the effects of events in the earlier

phases of the search process. The second purpose is to help the intensification or diversification of the search process.

$$CF(H_i) = \alpha \cdot F_1(H_i) + \beta \cdot F_2(H_p)(H_i) + \delta \cdot F_3(H_i) \tag{2.2}$$

$$F_1(H_i) = \alpha \cdot F_1(H_i) + \frac{Objective\ Value\ Improvement(H_i)}{Previous\ Objective\ Value \cdot T(H_i)} \tag{2.3}$$

$$F_2(H_p)(H_i) = \beta \cdot F_2(H_p)(H_i) + \frac{Objective\ Value\ Improvement(H_p, H_i)}{Previous\ Objective\ Value \cdot T(H_p, H_i)} \tag{2.4}$$

$$F_3(H_i) = Total\ execution\ time\ since\ H_i\ is\ last\ called \tag{2.5}$$

Simulated annealing and *great deluge* can be used as *acceptance criteria* in *hyperheuristics*. All improving and equal quality moves are accepted by *simulated annealing* [29]. The worsening moves are accepted with a probability that is decreased throughout the execution. The probability of acceptance of worsening moves at iteration $i + 1$ is given in equation 2.6. In this formula, f_i denotes the objective value of the candidate solution at iteration i, t_r the remaining execution time and t_e the total execution time. n is an integer value that determines the cooling scheme. In this study, three cooling schemes are used: *linear* ($n = 1$), *quadratic* ($n = 2$) and *quartic* ($n = 4$).

$$p = exp\left(-\frac{\frac{f_{i+1}-f_i}{f_i}}{\left(\frac{t_r}{t_e}\right)^n}\right) \tag{2.6}$$

Great deluge accepts all improving and equal quality moves [41]. The worsening moves are accepted only if their objective value is smaller than the *deluge value*. The *deluge value* decreases throughout the execution. In equation 2.7, f_0 refers to the objective value of the candidate solution at iteration 0, t_r to the remaining execution time and t_e to the total execution time. Similar to *simulated annealing*, n is an integer value that determines the cooling scheme. The same cooling schemes are applied with *great deluge*. The fundamental difference between *simulated annealing* and *great deluge* is that *simulated annealing* is stochastic and *great deluge*

deterministic. In addition to *simulated annealing* and *great deluge*, *improving and equal moves accepted* is used for performance comparison.

$$d = f_0 * \left(\frac{t_r}{t_e}\right)^n \tag{2.7}$$

2.7 Conclusions

The problem models and solution methods in the scientific literature have mostly been based on a single problem instance in a ward or in a hospital [1, 2, 4, 18, 24, 33, 38]. They have been tested on data that consists of similar instances of the respective problems under study. This might restrict their utilisation to a limited application field. It is economically not feasible for most of the institutions to dedicate resources to a team of researchers to solve their problems to optimality. In addition to that, the personnel rostering problem of each institution, department or ward changes every month because of the changes in the workload, bank holidays, vacation requests and several other factors. Therefore, developing solution methods specific to a single problem instance is not a sustainable approach to the personnel rostering problems in the real world. The alternative is to have a generic problem model that can be used to express a broad range of personnel rostering problems and a generic solution method that is able to tackle the problem instances expressed with the generic model.

The problem model to be presented in Chapter 3 is not based on a single problem instance. It is developed to be as generic as possible, so that the problem instances from different hospitals, sectors and countries can be expressed using the proposed model.

Besides the articles reporting results on the Nottingham benchmarks, the test data used in the experiments have been published in few of the articles published in the nurse rostering literature [4, 44]. Ignoring this practice makes it impossible for other researchers to comprehend the problem under study, verify and reproduce the solution methods, and compare them with other solution methods.

The maintenance of a public benchmark set solves many problems encountered in the nurse rostering research. Researchers can test their solution methods on the problem instances from various sources, so that the resulting solution methods will not be limited to a single problem instance. Researchers can publish their own problem instances in the benchmark set, so that other researchers can understand the problem under study and verify the results. Solution methods from various sources can be compared on a common set of data. The Nottingham benchmarks

[20] fill that gap as a collection of problem instances, best known solutions and references to the corresponding articles[8].

On the other hand, there are significant differences in the problem definitions of the Nottingham benchmarks and the real world nurse rostering practice. In the real world practice, constraints are evaluated over multiple *schedule periods* to ensure the continuity between the *schedule periods*. In Nottingham benchmarks, the constraints are evaluated in isolated *schedule periods*, discarding the actual *assignments* in the previous *schedule period*. Such a practice is likely to result in serious constraint violations that will make the real world practitioners reject the proposed solution altogether.

The continuity between *schedule periods* is a crucial aspect of the real world personnel rostering practice. Any solution that ignores this aspect will be rejected by the real world practitioners. Few of the researchers in the field have recognised this fact and addressed it in their solution methods [2, 4, 33]. Glass and Knight [33] have emphasised the importance of the continuity between *schedule periods* and have proposed a formal method to address it. The critical conclusions of Glass and Knight [33] on the benchmarks have pointed to a new research direction in the nurse rostering research.

Discarding the previous *schedule period* and focusing on a single isolated *schedule period* entails assumptions about the previous *schedule period*. In the real world, the assumptions about the previous *schedule period* will not match the real *assignments* all the time. However, making assumptions about the previous *schedule period* can be acceptable in an academic context given that the assumptions are consistent.

Some instances in the Nottingham benchmarks contain inconsistent assumptions about the previous *schedule period*. Some inconsistencies in the problem instances Valouxis 1, SINTEF, BCV-3.46.2, Millar-2Shift-DATA1, MUSA, Ikegami-2Shift-DATA1, GPost, ORTEC01, Azaiez and WHPP from the Nottingham benchmarks are discovered. They will be discussed in detail in Chapter 4. Furthermore, the evaluation of the constraints that expand over two *schedule periods* will be discussed in greater detail and a set of consistent rules to evaluate these constraints will be introduced in Chapter 4. In this way, we try to offer the research community realistic evaluation methods.

The KAHO benchmarks will be introduced in Chapter 6. The problem structure and constraint evaluation methods of the KAHO benchmarks differ significantly from the Nottingham benchmarks. The differences arise from the objective of the KAHO benchmarks to reflect the real world requirements such as the continuity between *schedule periods*. The main objective of the KAHO benchmarks is to facilitate the research community to develop accurate problem models, constraint evaluation methods and solution methods that are able to cope with those problem

[8] http://www.cs.nott.ac.uk/∼tec/NRP/

models. That way the developed personnel rostering approaches will be able to address real world problems. The Nottingham benchmarks, on the other hand, represent another important research direction, namely to facilitate the research, development and comparison of algorithms in academia.

According to the *no free lunch theorem* [71], a single algorithm configuration cannot be expected to outperform other algorithm configurations on all kinds of optimisation problems. The focus of the present dissertation is limited to the real world personnel rostering problems. However, the real world personnel rostering problems are still a diverse subset of all the personnel rostering problems. Hence, the proposed solution methods must be tested on a diverse set of data. Collecting problem instances from various sources does not guarantee the diversity of the test data. The diversity of test data can be assessed by evaluating the quantitative measures of each instance. Furthermore, a solution method is introduced as an algorithmic toolbox, where each algorithmic configuration is empirically shown to perform well on problem instances that have quantitative measures within a given range. Vanhoucke and Maenhout [68] have proposed a set of *complexity indicators* based on a variation of the nurse rostering problem to address these challenges. Additional elements such as the *schedule locks* and *individual skill types* that arise from the real world requirements induce significant differences between the requirements that are dealt with in the present dissertation (Chapter 3) and the problem variation in [68]. Consequently, a new set of quantitative measures specific to the problem model in Chapter 3 will be introduced in Chapter 5 to address these challenges.

The exact methods applied to the nurse rostering problem have been developed and fine tuned with a single problem instance in mind [2, 4, 33]. In some cases [33], the structure of the problem instance at hand has been hard coded into the *integer programming model*. Although the state-of-the-art results obtained by these methods provide insight from an academic point of view, the utilisation of the resulting solution methods is limited to the problem instances under study. The solution methods presented in Chapter 7 are aimed to cope with a broad range of real world personnel rostering problems that can be expressed using the problem model introduced in Chapter 3. The objective is not to prove the optimality but to offer high quality solutions within limited execution times, which is aligned with the real world personnel rostering practice. Therefore, *metaheuristics* and *hyperheuristics* are chosen as the basis of the solution methods.

Following the suggestions of Ernst et al. [30] and Burke et al. [26], the solution methods in this dissertation involve neighbourhoods and low level heuristics that take advantage of the constraint information next to more general neighbourhoods and low level heuristics (Section 7.6). The neighbourhoods and low level heuristics presented in this dissertation are utilised in *metaheuristics* and *hyperheuristics* (Sections 7.2 and 7.4). The proposed solution methods have the computation time as a termination criterion, which is practical for the real world practitioners [26].

The performance of solution methods in the scientific literature have been assessed through experiments on test data. Some of these solution methods, such as *genetic algorithms*, involve stochastic procedures. A statistical analysis is needed to draw conclusions from the results of experiments carried out on such solution methods. A sound statistical analysis requires a multiple number of executions to satisfy a minimum sample size. Several articles in the scientific literature have based their conclusions on the results of experiments that involve single runs of stochastic methods. In the present dissertation, sound conclusions are ensured by producing experimental results that involve multiple runs and statistical analysis (Chapter 8).

The ratio of nurse rostering approaches implemented in the real world to the approaches reported in the nurse rostering literature has remained relatively low [40]. Mostly, the nurse rostering literature has been focused on algorithms and solution methods. The study of problem models has been given less attention. However, solution methods cannot provide a satisfactory solution to the practitioners in the field, if the underlying problem models do not represent the real world problem accurately. Again solution methods cannot find satisfactory solutions for a problem, if the quality of solutions is not evaluated accurately. Therefore, in addition to algorithms and solution methods, significant emphasis is placed on the problem model and constraint evaluation methods in the present dissertation.

Chapter 3

The personnel rostering model

Literature surveys on personnel rostering have revealed that only a small fraction of the academic solution methods have been implemented in the real world (Chapter 2). The academic research on personnel rostering have mostly been focused on the performance of the solution methods. Many real world requirements concerning the problem definitions have not been addressed in their entirety and that is exactly what the present dissertation is focusing on. A significant number of solution methods reported in the scientific literature have been developed with a single problem instance in mind.

The performance of a solution method is only one of the factors that determines its success in the real world. Another factor is the problem model, which needs to be an accurate and complete representation of the problem at hand. Furthermore, the model needs to be generic enough to address a broad range of problem instances from various sources. The problem instances vary among countries, sectors and institutions, sometimes even within the same institution, for example among the departments of a hospital. In addition to that, the problem of the same unit might exhibit seasonal variations and evolve with changes in the legal and contractual requirements over time. As a result, developing a problem model and a solution method for a single problem instance is not a feasible approach in the real world.

The problem model introduced in this chapter has been developed to be as generic as possible and to represent the real world requirements accurately and completely. Several extensions have been made to the standard academic models to accomplish that objective. The extended personnel rostering model is the main contribution presented in this chapter. The purpose of the generic model is to be utilised in the real world personnel rostering applications. That purpose has been accomplished by deploying it as the core of the KAHO personnel rostering kernel, which is utilised as the rostering engine in the commercial personnel planning software of

our industrial partners[1]. Researchers and software developers who have to address rostering problems with real world complexity can represent their problems using the present model with few or no adjustments. The present chapter contains the information, examples and illustrations that will guide this task.

The problem model is a structure to express the problem data. A complete problem description also includes the constraint evaluation methods in addition to the problem model. The constraint evaluation methods reported in the scientific literature have not always matched the real world practice [33]. The slightest change in the evaluation of the constraints can result in completely different rosters being generated by an algorithm. Therefore, a complete chapter, Chapter 4, has been dedicated to the evaluation of constraints. Chapter 4 will help the researchers and software developers to align their constraint evaluation methods with the practices in the industry.

In practice, it is mostly impossible to satisfy all the constraints of personnel rostering problems. They have been mostly treated as *combinatorial optimisation problems* (COP). The constraints in a COP are handled in two categories, hard and soft constraints. A solution needs to satisfy all hard constraints in order to be considered feasible. The soft constraints are preferred to be satisfied as much as possible. This is accomplished by minimising an objective function (Formula 3.1), which is a linear combination of the *weight* (w_c) and the number of violations (n_c) of each soft constraint ($c \in C$), where each violation is multiplied by the *weight* (w_c) corresponding to the constraint. The quality of a roster is inversely proportional to its objective function value: the lower the objective function value, the higher the quality of the roster.

$$\sum_{c \in C} w_c n_c \tag{3.1}$$

According to the categorisation proposed by De Causmaecker and Vanden Berghe [48], rostering problems with *personnel* (P) and *coverage* (C) constraints as *optimisation objective* (γ) can be modelled using the present data model. *Threshold values* are foreseen for quantitative constraints like *counters*, *series*, *successive series* and *coverage constraints*. A *threshold* value can be either a *minimum*, a *maximum*, or an *interval* defined by a *minimum* and a *maximum* value.

The *constraint sets* and the *weights* of the constraints show differences among sectors, countries and organisations, and even among different *employees* in the same unit. Therefore, the model does not provide a predefined *constraint set* with fixed *weights* that apply to all of the problem instances, but it provides a

[1]SAGA Consulting, GPS NV Belgium

manner for the planner to construct constraints and *constraint sets* that can be specific to each *employee* and problem instance. Consequently, the model does not foresee a predefined *objective function*, but an *objective function* that is constructed according to the *constraint sets* that are defined by the planner using the model.

The problem model is defined with an XML Schema Description (XSD), which has been developed entirely by the author of the present dissertation and provided online[2]. The XSD file enables defining the data model precisely and prevents ambiguities. It also simplifies the input output operation and prevents errors given that the XML files adhere to the XSD file.

Object oriented programming languages like C# and Java provide tools to convert XSD files to source code automatically. The resulting source code, C# or Java classes, are used as the data structures of the problem model. C# and Java provide built-in libraries to serialise the current state of the data structures in an XML file. The input data is loaded from an XML file to C# or Java objects using the deserialisation methods of the corresponding programming languages.

3.1 The differences between the standard academic models and the extended generic model

Most of the rostering problems in the scientific literature involve a limited number of *shift types*, for example early, late and night shifts. In the real world however, a greater number of *shift types* are involved in rostering problems. In Belgian hospitals and rest homes for example, variations in *shift type* structures based on the working hours and *skill types* are common. For example, there can be different early shift type definitions for full time and half time employees as well as for nurses and caregivers. Therefore, a constraint defined on a *shift type* is mostly defined for a *set of shift types*. For example, a counter on the late shift types can be defined on the set of all the late *shift types*.

Although not as common as on *shift types*, some constraints are defined on a *set of skill types*. For example, some constraints that apply to the nurses and caregivers do not apply to the head nurses. Those constraints involve a *set of skill types* that consists of the nurses and caregivers.

Another common parameter of constraints in the real world are *dates* and *day types*. Bank holidays and weekends are important *dates* and *day types* in the rostering problems. Furthermore, it is common in the real world that a specific weekday is more important than others. For example, Wednesdays are important for *employees* that have school children in Belgium, because the schools are closed on Wednesday

[2]http://ingenieur.kahosl.be/vakgroep/it/nurse/archive.htm

afternoons. A set of specific *dates* become the parameter of a constraint, if the *employee* requests a vacation on those days.

A *set of skill types*, a *set of shift types*, or a *set of dates or day types* can be the parameter of a constraint. These cases have to be addressed in a problem model that is claimed to be generic. Therefore, the standard academic models have been extended with three elements that are abstractions of these parameters, namely the *skill type set*, the *shift type set* and the *day set*. Furthermore, a constraint can involve a combination of these three parameters. For example, a constraint can be defined on late and night shift types, for nurses and caregivers, on Wednesdays. To address such cases, the standard academic models have been further extended with a fourth element, called *domain*, which is the combination of a *skill type set*, *shift type set* and a *day set*.

The standard academic models have been extended to address two issues with respect to the *skill types* of the *employees*. It is not uncommon in the real world that some *employees* have more than one *skill type*. This has been modelled to some extent in some academic articles, where the *skill types* have been modelled in a hierarchy and the nurses of a higher *skill type* can substitute the nurses of a lower *skill type*. For example, a head nurse can substitute a caregiver. That is rarely acceptable in the real world because of two reasons. First, *employees* are not willing to carry out tasks that are below their main *skill type*. Second, *employees* with higher *skill types* are paid higher wages. Allocating them for jobs below their main *skill types* is not economical for the employing institution. Despite these arguments, an *employee* has to work in a *skill type* below his or her main *skill type* in extreme cases. In the extended model, a *weight* has been foreseen for each *skill type* of the *employees* to address this fact. The *weight* is inversely proportional to the relevance of a *skill type* to an *employee*. The *weight* for the main *skill type* of an *employee* will be zero and the *weights* for the *skill types* that are less relevant to him or her will be proportionally higher. In some cases, the *employee* will be assigned to the less relevant *skill type* with a penalty in the objective function proportional to that *weight*.

In the scientific literature, the members of a certain *skill type* have been treated equally. In the real world however, people having a certain *skill type* can have different *levels of experience* for that *skill type*. Although not binding for all the rostering problems, common *levels of experience* are senior, junior and trainee. For example, a healthcare practitioner can have three *skill types*: head nurse, nurse and caregiver. He or she can have a different *level of experience* for each of his or her *skill types*. He or she can be a senior caregiver, junior nurse and a trainee as a head nurse. *Level of experience* is considered to be hierarchical. A senior is higher in the hierarchy than a junior, which in turn is higher than a trainee.

Several constraints take into account the *level of experience*. For example, *coverage constraints*, the number of *employees* required on a given *date* for a given *shift* and

skill type, are defined with a minimum *level of experience*. Most of the time, the minimum *level of experience* is junior and the trainees are not counted towards the *coverage constraints*. For example, a *coverage constraint* can be stated as the following: *two caregivers with at least junior as the level of experience are needed for the early shift type on Mondays*. In this case, only junior and senior caregivers will satisfy this *coverage constraint*. The *assignments* made to the trainee caregivers will not be counted towards this *coverage constraint*.

The relevance of a *skill type* to an *employee* and the *level of experience* have been modelled in a new element, called *employee skill type*, in the extended model. Each *skill type* of an *employee* has been modelled as an *employee skill type* that consists of the actual *skill type*, a *weight* and a *level of experience*.

The *coverage constraints* are not the only constraints that make use of the *level of experience*. The planners prefer to maintain a ratio between the staff of different *levels of experience* at certain moments. For example, a senior *employee* is preferred to work with five trainees to supervise and train them. On the other hand, a senior truck driver cannot supervise and train more than one trainee at a time, due to the limitations of the duty they carry out. The standard models have been extended with the *training* constraint to address such requirements.

The *chaperoning* constraint, expressed as (C) in the *personnel environment* α in [49], has various variations in the real world. A set of *employees* can be preferred to work together. For example, *employees* who share the same car to commute to work would request to work in the same working hours. In some cases, a set of *employees* would prefer to work at different times. For example, parents of small children working in the same hospital might prefer to work in distinct working hours, so that at least one them is able to take care of their children. In other cases, a certain number of members from a set of *employees* are required to work together. The latter case is unique in the nurse rostering literature. An extension to the standard models, called *collaboration*, can be used to model all of these and similar cases.

The structures, definitions and examples of the extensions and their relations to the standard models will be presented in greater detail in Section 3.2. Some of the extensions have also been subject of two journal papers [14, 66], publications at conferences [8, 9, 10, 11, 12, 13] and a technical report [15].

3.2 The problem model

The problem model is expressed as a tree structure. The figures between the brackets refer to the multiplicities of the elements. (0..1) refers to an optional element, (1..*) to at least one element. If no multiplicity information is given,

then the multiplicity is exactly one. The elements in boldface characters refer to the extensions to the standard academic rostering models and constitute the contribution of the present chapter. The bullet ⋄ refers to a choice among multiple items.

The root element of the tree is the *scheduling session*, which has four children: *schedule period*, *schedule*, *schedule constraints* and *schedule definitions*.

- Scheduling Session
  - Schedule Period
  - Schedule Definitions
  - Schedule Constraints
  - Schedule (0..1)

3.2.1 Schedule period

The *schedule periods* of personnel rostering problems vary among different institutions, sectors, countries and time of the year. In Belgian hospitals and rest homes, the most common *schedule periods* are four weeks and one month. Some periods, such as the Christmas and summer holidays, require more attention than the rest of the year due to lower numbers of available personnel. Consequently, the *schedule periods* are set to two weeks in some hospital wards during the holiday periods.

The *schedule period* element of a personnel rostering model needs to be flexible in order to address the variations in the *schedule periods* of different problem instances. Therefore, the *schedule period* element is defined with a *start* and an *end date* in the data model.

- Schedule Period
  - Start Date
  - End Date

3.2.2 Schedule definitions

The essential components of the personnel rostering problem are collected under the *schedule definitions* element. The members of the *schedule definitions* element, such as the *skill types*, *shift types* and *employees*, are the basic building blocks of the personnel rostering problem. They are referred by each other and by the rest of the components of the personnel rostering problem.

- Schedule Definitions
  - Holidays (0..1)
  - Skill Types
  - Shift Types
  - **Day Sets**
  - **Shift Type Sets**
  - **Skill Type Sets**
  - **Domains**
  - Constraint Sets
  - Employees
  - Weights

Skill types

The *skill types* that are referred to in a problem instance are defined under this element with their ids. Different job descriptions, qualifications and responsibilities are modelled using the *skill types*. For example, head nurse, regular nurse and caregiver refer to different *skill types* in the healthcare sector. Different classes of driver's licences constitute different *skill types* in the transportation sector. Moreover, the knowledge of particular routes and geographical areas is also categorised as *skill types* in the transportation sector. The ability to utilise a certain machinery or equipment are examples of *skill types*. For example, the ability to ride a horse or command a dog are considered as distinct *skill types* in the security sector. These are just a few examples of the usage of *skill types* in the personnel rostering problem. In the real world practice, each industry has its own standards and definitions of *skill types*.

- Skill Types
  - Skill Type (1..*)
    - Id

Shift types

Shift types refer to daily *assignment* units. The number and structure of the *shift types* are not predefined in order to allow the end user to define any kind of *shift type* he or she has to work with. Each *shift type* is defined by a specific *start* and *end time*. Some working environments such as hospitals require *rest periods before*

and *after* each *shift type*. Healthcare institutions typically foresee eleven hours of recuperation before and after each assignment. That means the rest times between two assignments is not allowed to be less than eleven hours. The *net job time* is not always equal to the duration of the *shift type* due to the breaks. Most of the time, the breaks when the *employee* is allowed to leave the work place for a period of longer than half an hour, for example lunch breaks, are not included in the *net job times*. The *net job time* is a part of the *shift type* element and it is used by the *hours worked counters*.

Common examples of *shift types* in the real world practice are early, late, day and night. Typically, several variations of the *shift types* are defined to be assigned to the *employees* with different *contract* and *skill types*. For example, the starting times of the early and late *shift types* may deviate for the members of the regular nurses and caregivers. The caregivers may require to start earlier in order to prepare the patients for treatment by the nurses. Similarly, *shift types* with different *net job times* will be assigned to the full time and half time *employees*. Consequently, the total number of the *shift types* in most of the personnel rostering problems exceeds three, which is the common number of *shift types* in the scientific literature.

The generic model covers problems that correspond to the *shift type* categories *variable* (N) and *overlapping* (O) in the work characteristics (β) of the categorisation in [48].

- Shift Types
  - Shift Type (1..*)
    - Id
    - Start Time
    - End Time
    - Rest Period Before
    - Rest Period After
    - Net Job Time

Day sets

In the problem model, some constraints are defined on a set of days. A *day set* can be expressed as either a *date set* or a combination of *day types*. A *day type* element can be either a week or weekend day (*Monday, Tuesday, ..., Sunday*), or a *holiday*, or all days in the *schedule period* (*any*). For example, *Monday, Wednesday and Friday* form a *day set* as a combination of three *day types*. If a *day set* is expressed as a *day type* with the value *any*, then the *day set* is considered to be composed of all the dates in the *schedule period*. Similarly, if a *day set* is expressed as a *day type*

with the value *holiday*, then the *day set* consists of all the holidays listed under the *Holidays* element of the *Schedule Definitions* element.

A *date set* is a collection of calendar dates and how they are handled. The calendar dates in a *date set* do not need to be adjacent. For example, a *date set* can consist of the calendar dates 2011 January 2, 3 and 4 as well as the calendar dates 2011 January 1, 2011 May 1 and 2011 November 11. The *handling* mode can be either *individual* or a *block*. For example, an *employee* can request a vacation on the dates 2011 January 2, 3 and 4 as a *block* or as *individual* days. In case of *block handling*, the request is considered to be granted, only if all the requested vacation dates are granted. This is usually the case when the *employee* plans a vacation in a distant location. Alternatively, the *employee* can request a vacation as *individual* days. In that case, each individual day granted will be counted towards the request of the *employee*. This is common when the *employee* plans to stay at home to pursue some kind of private objective, such as redecorating his or her home. Elaborate examples of the *handling* mode will be introduced in the section on employee requests, where they will be discussed in context.

- Day Sets
  - Day Set (1..*)
    - · Id
    - ◇ Date Set
    - ◇ Day Types

- Date Set
  - Handling
  - Date (1..*)

- Day Types
  - Day Type (1..*)
    - ◇ Any
    - ◇ Holiday
    - ◇ Weekday (1..7)

Shift type sets

In the real world practice, different versions of a *shift type*, for example the early shift, are defined to be assigned to the *employees* with different *contract types*, such as full time and half time. Similarly, the members of particular *skill types*, such as

nurses and caregivers, are assigned different versions of a *shift type*, for example the late shift. This practice has been addressed in the extended model by providing a generic structure to define a *shift type* and allowing the end-user to define as much *shift types* as he or she needs (Section 3.2.2). Same practice also results in constraint definitions that involve a set of *shift types* instead of a single *shift type* [14].

Suppose that there are three variances of the early shift with different net job times defined for nurses with different *contract* types and five nurses are needed in the early shift on a particular day. In this case, an assignment with any of these three early shift types should be counted towards this constraint. This is not an uncommon scenario in the real world practice and requires referencing more than a single *shift type* in a constraint. This requirement is addressed in the model by considering a set of *shift types* as a constraint parameter.

Shift types sets are defined under the *schedule definitions* element and addressed by the corresponding constraints with their ids. For example, a constraint can restrict the number of *night shifts* assigned to a nurse, e.g., maximum five night shifts during the *schedule period*. Some wards have more than one type of *night shifts*, which can vary according to their duration. In that case, the *night shifts* with different durations can be combined in a *shift type set*. If a nurse should not work more than five night shifts during the *schedule period*, this means that the sum of all the night shifts of any sort should not exceed five.

- Shift Type Sets
  - Shift Type Set (1..*)
    - Id
    - Shift Type Id (1..*)

Skill type sets

Similar to *shift type sets*, some constraints involve *skill type sets*. For example, a regular nurse or caregiver may be suitable for a particular task, but a head nurse might be overqualified for the same task. A constraint to address that case needs to reference the *skill types*, regular nurse and caregiver, at the same time. This can be accomplished by defining a *skill type set* with the elements, regular nurse and caregiver, and referencing this set in the corresponding constraint.

- Skill Type Sets
  - Skill Type Set (1..*)
    - Id
    - Skill Type Id (1..*)

Domains

As it has been explained in the previous sections, *day sets*, *shift type sets* and *skill type sets* can be parameters of various constraints. Furthermore, a constraint can also reference a certain combination of a *day set*, *shift type set* and a *skill type set*. As a matter of fact, it is common in the real world practice that the same combination is referenced in several constraints. A *domain* element is introduced to address such cases. It is the combination of a *day set*, *shift type set* and a *skill type set*.

The *domain* element benefits the end user by allowing him or her to define a *day set*, *shift type set* and a *skill type set* combination once and then refer to it in different constraints. Sharing a common data structure, i.e., the *domain* element, among different constraints also results in a better source code structure and improved software performance.

The *domain* elements are used by the *domain counter*, *absence request*, *collaboration* and *training* constraints. For example, different versions of late shift types, regular nurses and caregivers as skill types and Fridays as day types can be combined in a single *domain* element and then be referenced in several constraints. More examples of *domains* will be discussed in the following sections about employee requests and constraints.

- Domains
  - Domain (1..*)
    - Id
    - Day Set Id
    - Shift Type Set Id
    - Skill Type Set Id

Constraint sets

The constraints that restrict the *schedule* of an *employee* directly are called *time related constraints* in [27]. The *schedule* of an *employee* can be constrained in various ways. The total number or successions of schedule items such as working days, idle days, assignments of particular *shift types*, working weekends or idle weekends can be limited. For example, the total number of days worked by an *employee* can be set to a certain value. Another example is the prohibition of the assignment of more than three late shifts in a row. A *constraint set* is composed of an *id* and a set of *time related constraints*. It contains all the *time related constraints* of a *contract*.

- Constraint Sets
  - Constraint Set (1..*)
    - Id
    - Constraint (1..*)

The *time related constraints* are modelled around three general constraint types: *counters*, *series* and *successive series*. *Counters* restrict the total number of an item in the schedule of an *employee*. For example, an *employee* should work exactly two weekends in a particular period of four weeks. The successive occurrences of an item is constrained by *Series*. For example, an *employee* should work at least three and at most six days in a row. Successions of two series can also be constrained. For example, an *employee* should have at least two idle days after a *series* of five night shifts.

Time related constraints are usually considered as soft constraints in real world problems. Therefore, a *weight* is foreseen for each *time related constraint* defined in the *constraint sets*. The rostering problems with *personnel regulation constraints* as *optimisation objective* (γ) can be modelled using this definition. Each of the three *time related constraint* types allows a variety of specific subjects and parameters to cover a wide range of employment constraints. For example, working hours, days and weekends or idle days and weekends can be restricted. These subjects can be limited in several ways using minimum, maximum or interval *thresholds*. *Time related constraints* can involve different sets of *days*, *shift types* and *skill types*. Only one of the elements, *counter*, *series* or *successive series* can be the *constraint detail* of a given *constraint* element.

- Constraint
  - Id
  - Constraint Detail
    - Counter
    - Series
    - Successive Series
  - Weight

Quantitative constraints restrict the number or sequence length of schedule items using *threshold* values. Consider the following constraint: at least three and at most five caregivers are needed in the early shift on weekdays. In this example, the numbers three and five are examples of *threshold* values. A *threshold* value is defined as one of the following three options: a *minimum*, a *maximum* or an *interval* defined by a *minimum* and a *maximum* value. Any value that does not

satisfy the *threshold* is considered to be a constraint violation. Imagine only two caregivers are assigned in the early shift on a particular Monday, whereas at least three and at most five caregivers are needed for that shift and day. The value two does not satisfy the *threshold interval*, three and five. Therefore, this situation is considered to be a constraint violation. *Threshold* elements are referred from various positions in the problem model.

Worked days, *idle days*, *worked weekends*, *idle weekends*, *assignments of certain shift types* and *hours worked* are examples of roster items in this context. The following constraints demonstrate how they can be restricted. If an employee has to work at least 38 hours a week, the value 38 refers to the minimum *threshold* value. If an employee should not be assigned more than two worked weekends in four weeks, the value two is the maximum *threshold* in this constraint. If there must be at least three and at most five caregivers present on the week days in the early shift, then the *threshold* value is defined as an interval, with three as the minimum and five as the maximum value.

- Threshold
  - Minimum
  - Maximum
  - Interval
    - Minimum
    - Maximum

Counters The *counter* constraints restrict the number of specific roster items over the *counter period*. The *counter period* is modelled by a *start* and *end date* and does not need to match the *schedule period*. Imagine a situation where the *schedule period* is the month of April and there is a counter restricting the number of bank holidays worked by an employee throughout the year. In this case, the *counter period* is a calendar year, expanding from January 1 to December 31. However, the *schedule period* is the month April. In this example, the *schedule period* is a subset of the *counter period*. The other way around is also possible. For example, the *counter period* of the number of hours worked in a week by an employee is a single week, which is a subset of any *schedule period* longer than one week. It is also not rare in the real world practice that the *counter period* matches the *schedule period*. Therefore, the *counter period* is treated independently from the *schedule period* to allow the practitioners to represent their counters accurately.

A *counter* has a *counter type*, which is a choice among seven elements: *hours worked*, *shift types worked*, *days worked*, *days idle*, *weekends worked*, *weekends idle* and *domain*. *Hours worked counters* are widely utilised in the real world practice. The contracts of the employees are mostly based on the weekly working hours.

Typically, a full time employee is expected to work 38 hours a week. This is an example of a *hours worked counter*. In working environments that operate on 24 hours a day and seven days a week basis, *shift types worked*, *weekends worked* and *weekends idle* counters become critical. For example, the number of night shift type assignments to an employee can be restricted using a *shift types worked counter*. Similarly, employees are expected to work a certain number of weekends and to have rest on the remaining weekends. *Weekends worked* and *weekends idle* counters are used to enforce this working regime. These *counters* ensure that the employees are not overloaded with assignments that interfere with their private lives. They also balance the rosters of the employees and provide a sense of fairness by assigning similar numbers of unpleasant shifts to each of the employees.

In cases where there are *shift types* with different working hours, *days worked* and *days idle counters* can be used to complement the *hours worked counters*. Suppose that there are *shift types* for full timers and half timers in a working environment. Assigning the *shift types* for half timers to a full timer will result in him or her working on more days than he or she actually should in order to fulfill his or her *hours worked counter*. This can be overcome by using a *days worked counter* to restrict the number of days worked by him or her. It is common in the real world practice to restrict the number of assignments of a certain set of *shift types* with a particular set of *skill types* on a particular set of days. This is represented with a *domain counter*. For example, the number of night shift assignments to nurses and caregivers on Fridays should be less than two in a *schedule period* of four weeks. Here, the night shifts refer to the *shift type set*, nurses and caregivers to the *skill type set* and Fridays to the *days set*. The combination of the *shift type set*, *skill type set* and *day set* constitute a *domain*, which is one of the options for a *counter type*.

Shift types worked counters have a *shift types set* as a parameter. Any *shift type* in the *shift types set* is counted in the *shift types worked counter*. In working environments with various *contract types*, there are distinct *shift types* to be assigned to the employees with different *contract types*. For example, if there are full time, half time and 75% contracts, then there will be variants of *shift types* specific to each of those *contract types*. Moreover, in some cases, the *shift types* also vary among the members of different *skill types*. For example, the night shift types can be different for the employees with full time, half time and 75% contracts. In addition to that, the night shift types can also be different for nurses and caregivers. However, if a *counter* has to be defined on night shifts, it has to count all occurences of different night shifts. Therefore, a *shift types set* parameter is used in the *shift types worked counters* instead of defining these counters on a single *shift type*.

Except *domain*, *weekends worked* and *weekends idle*, all *counters* have the *day types* parameter, which can have the value of either *any*, *holidays* or a set of week and weekend days. For example, the application of a counter can be limited to certain week days. Late and night shift types are usually unwanted on Fridays,

Saturdays and Sundays. To address this preference, a *shift types worked counter* can be defined with a *shift types set* that consists of the late and night shifts. The *day types* parameter of that counter will be Fridays, Saturday and Sundays.

- Counter
  - Period
    - Start Date
    - End Date
  - Counter Type
    - ◇ Hours Worked Counter
      - Day Types
    - ◇ Days Worked Counter
      - Day Types
    - ◇ Days Idle Counter
      - Day Types
    - ◇ Shift Types Worked Counter
      - **Shift Types Set**
      - Day Types
    - ◇ Weekends Worked Counter
    - ◇ Weekends Idle Counter
    - ◇ **Domain Counter**
      - **Domain**
  - Threshold

The following constraints taken from Burke et al. [20] are examples of *counters*:

- "Maximum and minimum number of hours worked during the scheduling period or per week"
  - *hours worked counter*
- "Maximum number of a specific shift type worked"
  - *shift types worked counter*
- "Maximum number of weekends worked"
  - *weekends worked counter*
- "Maximum total number of assignments for all Mondays, Tuesday, Wednesdays, ..."
  - *days worked counter* with Mondays, Tuesdays, Wednesdays, ... as *day types* parameter

Series The *series* constraints restrict consecutive occurrences of specific roster items. For example, an *employee* should not work more than two weekends in a row. They can be defined with five *subjects*: *shift types worked*, *days worked*, *days idle*, *weekends worked* and *weekends idle*. Similar to *counters*, *shift types worked series* are also defined on a *shift types set*.

- Series
  - Series Type
    - ◇ Days Worked Series
    - ◇ Days Idle Series
    - ◇ Shift Types Worked Series
      - · **Shift Types Set**
    - ◇ Weekends Worked Series
    - ◇ Weekends Idle Series
  - Threshold

The following constraints presented by Burke et al. [20] are examples of *series*:

- "Maximum and minimum number of consecutive working days"
  - *days worked series*
- "Maximum and minimum number of consecutive non-working days"
  - *days idle series*
- "Maximum number of consecutive weekends worked"
  - *weekends worked series*
- "Valid numbers of consecutive shift types"
  - *shift types worked series*

Successive series Discussions with practitioners have revealed that the succession of two *series* can be restricted. A common example in the real world is the following: the assignment of at least five night shifts in a row should be followed immediately by at least two idle days. If the first *series* appears in a roster, it needs to be succeeded immediately by the second *series*. The first series in this formulation is the conditional part and the second is the restricting part. The conditional part is not evaluated and its violations are not penalised. In the example above, the assignment of four night shifts in a row does not satisfy the conditional part

and it will not result in any violation of the constraint in question. However, the conditional part triggers the evaluation of the restricting part. If the conditional part is true, then the restricting part is evaluated, and the violations of the restricting part are penalised. That means, only assignments of five and more night shifts in a row will trigger the evaluation of the restricting part. In that case, if the night shift series is not followed immediately by at least two idle days in a row, then this constraint is considered to be violated. Such constraints ensure the sufficient recuperation of the *employees*. That in turn increases the productivity and job satisfaction of the *employees* and minimises the risks of job accidents and other critical mistakes.

- Successive Series
  - Series 1
  - Series 2

The constraint *minimum two idle days after a series of at least five night shifts* can be translated into a successive series as following.

- Successive Series
  - Series 1
    - Shift Types Worked Series
      - Shift Type Set: Night Shifts
    - Threshold
      - Minimum: 5
  - Series 2
    - Days Idle Series
    - Threshold
      - Minimum 2

In this example, the *shift types worked series* is the conditional part, and the *days idle series* the restricting part. Violations of the conditional part, i.e., the *shift types worked series*, are not going to be penalised. If that is desired, then a seperate constraint, a *shift types worked series*, needs to be defined in addition to the *successive series* constraint. In the example above, if the number of night shift assignments in a row is desired to be restricted to exactly five, then an additional *shift types worked series* with night shifts as the *shift type set* and five as the *minimum* and *maximum threshold* needs to be defined.

Valid successions of shift types can also be modelled using the *successive series* constraint. For example, the constraint, *no early shifts after night shifts*, can be modelled as the following.

- Successive Series
  - Series 1
    - Shift Types Worked Series
      - Shift Type Set: Night Shifts
    - Threshold
      - Minimum: 1
  - Series 2
    - Shift Types Worked Series
      - Shift Type Set: Early Shifts
    - Threshold
      - Maximum 0

Five *series* successions have been identified that are relevant in real world problems: *days worked - days idle*, *days idle - days worked*, *shift types worked - days idle*, *days idle - shift types worked* and *shift types worked - shift types worked*. Further examples of successive series that are requested in many situations include:

- Two idle days should be assigned after six days worked in a row. (Days worked - days idle)
- After two idle days, four days should be worked. (Days idle - days worked)
- An early shift type should be assigned after two idle days. (Days idle - shift types worked)
- At least two late shift types should be assigned after two early shift types. (Shift types worked - shift types worked)

Employees

The individuality of *employees* varies among different working environments. The term individuality is used here to refer to the diversity between the *skill type sets*, *contracts*, *constraints* and *requests* of each *employee*. The nurse rostering problem in Belgian hospitals involves a high degree of individuality. The model enables that by allowing each *employee* to be defined with a *skill type set*, a number of employment *contracts* and *requests*.

- Employees
  - Employee (1..*)
    - Id
    - **Employee Skill Types**
    - Contracts
    - Requests (0..1)
    - **Counter Start Values (0..1)**
    - **Counter Remainder Values (0..1)**

Employee skill types Imagine a healthcare practitioner starting her career as a caregiver. Throughout her career, this practitioner gains enough experience as a caregiver and becomes a regular nurse by receiving additional education. The practitioner continues her work as a regular nurse and carries on with the education to become a head nurse. In order to be qualified as a head nurse, she has to apply this education as a head nurse for a certain number of shift assignments. However, these shift assignments are not counted towards the workforce requirements of her ward, because she has to be supervised by a qualified head nurse. This is a common scenario in the healthcare sector.

Several properties of the *skill types* can be recognised in this scenario. First, the actual *skill types*, i.e., caregiver, regular nurse and head nurse, stand out. Second, the *level of experience* for each *skill type* can be differentiated. The hypothetical healthcare practitioner has a high *level of experience* as a caregiver and a normal *level of experience* as a regular nurse. However, her *level of experience* as a head nurse is so low that she has to be supervised by an actual head nurse. In addition to that, her work as a head nurse is not counted towards the workforce requirements of her ward. The different *levels of experience* of the healthcare practitioner can be stated as senior as a caregiver, junior as a regular nurse and trainee as a head nurse.

A third aspect in this scenario will be noticed if the scenario is studied carefully. The common practice in the healthcare sector is to pay the nurses a fixed wage based on their qualifications. In this case, the hypothetical healthcare practitioner will be paid as a regular nurse, even if she works as a caregiver or a head nurse occasionally. Therefore, the employing institution would desire to minimise her shift assignments as a caregiver or a head nurse. Similarly, the hypothetical nurse would prefer not to work as a caregiver, because she would be overqualified for the tasks of a caregiver. Consequently, a third element is needed to represent the suitability of a *skill type* for an *employee*.

The attributes of the *skill types* of an *employee* are expressed under the *employee skill type* element. The *employee skill type* element consists of three items, the *skill type id*, *level of experience* and *weight*. The *skill type id* refers to the id of the *skill*

type as it is defined under the *skill types* element of the *schedule definitions* element. Examples of the *skill type id* values are caregiver, regular nurse and head nurse.

The *level of experience* element states how experienced the *employee* is on the corresponding *skill type*. There are three *levels of experience* in most of the cases. They are represented as integers in the model. As a result, more than three *levels of experience* can be defined if necessary. The *levels of experience* are considered to be hierarchical in the real world practice. Such a hierarchy can easily be represented using integers by considering the greater values to be higher in the hierarchy. For example, the values 3, 2 and 1 can refer to senior, junior and trainee, respectively.

The suitability of a *skill type* to an *employee* is addressed as a *weight* in the extended model. The *weight* of an *employee skill type* is inversely proportional to the suitability of the corresponding *skill type* for the *employee* and considered as a penalty for every assignment made with that *skill type*. In the example above, the *weight* of the *employee skill type* regular nurse should be minimum, i.e., 0. The *weights* of the *employee skill types* head nurse and caregiver should be higher than 0, indicating a penalty in case they are used in the assignments of this practitioner.

As it can be seen from the example, *employees* can have more than one *skill type*. Therefore, the extended model allows for multiple *employee skill types* for an *employee*.

The modelling of the *employee skill types* as presented in this dissertation is novel to the scientific literature. In the scientific literature, the *level of experience* element has not been considered. Also penalising less suitable *skill types* with a *weight* is new to the personnel rostering literature. Another unique feature of the *employee skill type* element presented in this dissertation is that it allows an *employee* to have an arbitrary non-empty subset of the *skill types* in the problem as her *employee skill type* set.

Organising the *skill types* hierarchically is a common practice in the scientific literature. That approach entails that a head nurse has all the *skill types*, for example regular nurse and caregiver in her *employee skill type* set. However, this is rarely true in the real world practice. *Employees* that are higher in the hierarchy are rarely planned for the tasks that have to be carried out by *employees* that are lower in the hierarchy. For example, the common practice in the scientific literature may result in planning a retail store manager as a cashier. This is not acceptable in the real world practice.

- Employee Skill Types
  - Employee Skill Type (1..*)
    - Skill Type Id
    - Level Of Experience
    - Weight

Contracts A *contract* element includes a *start date*, an *end date* and a *constraint set* that applies to the *schedule* of the *employee*. Each *employee* has at least one *contract*. In practice, an *employee* can have more than one *contract* with one institution. This happens when a new *contract* starts and an old *contract* ends within the same *schedule period*. For example, when an *employee* switches from a *half time contract* to a *full time contract* in the same *schedule period*, he or she will have two *contracts* in that *schedule period*. Each *employee* can have an exclusive *contract*, which allows a high degree of individualisation among the *employees*.

- Contracts
  - Contract (1..*)
    - Period
      - Start Date
      - End Date
    - Constraint Set Id

Requests *Employees* can express their *absence* and *assignment requests* prior to the planning process. For example, an *employee* can request to have free afternoons on Fridays or free mornings on Mondays. *Employees* have rights to paid vacations. They can request certain dates to be counted towards their paid vacations. As a result, if these *requests* are granted, then a certain amount of working time will be added to their *hours worked counters* for the granted *absence requests*. As an example, consider a full time employee that has to work 38 hours a week. That *job time* corresponds to seven hours and 36 minutes for every working day. If she requests a paid vacation on a particular date and her *request* is granted, than seven hours and 36 minutes must be added to all of her *hours worked counters* that cover the granted day.

Similarly, *employees* may prefer to work on certain dates, in certain *shift types* and using a certain subset of their *skill types*. For example, a healthcare practitioner might prefer to work as a regular nurse, in late *shift types*, on weekends, so that she is entitled to receive extra compensation from her institution. As it can be seen from the examples, the *requests* of the *employees* can be diverse. A generic structure is needed to represent all types of *requests* that *employees* might come up with.

Two types of *requests* can be differentiated, *absence requests* and *assignment requests*. The *assignment requests* are represented via the *domain counters*, because *domain counters* cover all the requirements of the *assignment requests*. However, *domain counters* do not meet all the requirements of the *absence requests*, because an extra element, the *job time*, is needed in case of *absence requests*. Therefore, the problem model is extended with an *absence request* element that consists of a *domain*, *job time* and *weight*.

The *domain* element indicates on which days, for which *shift* and *skill types*, the *absence request* is submitted. For example, an *employee* may prefer not to be assigned on Wednesdays for the late shift types as a caregiver, because the workload may be especially high for that combination. The *absence request* element does not have a *threshold*, because in *absence requests*, the *threshold* is by default maximum 0, since no assignment is desired at all.

The *job time* value is added to the *hours worked counter* values of the *employee*, if the *absence request* is granted. The *job time* element is used to model the paid vacations of the *employee*. Since the *requests* are considered as *soft constraints*, the *absence requests* have a *weight* element. A multiple of the *weight* is added to the objective value if the *absence request* is not granted.

- Requests
  - Absence Request (1..*)
    - **Domain Id**
    - Job Time
    - Weight

In order to demonstrate the modelling power of *domains*, *absence* and *assignment requests*, five complex *request* examples from the real world practice and their representations in the extended model will be presented in the following paragraphs. These examples are *absence requests on individual holidays*, *absence request as a block*, *absence request on Wednesday afternoons*, *assignment request for a specific skill type* and *individual bank holidays*. These and countless others provide challenges to the planners every time a roster is prepared. The satisfaction of these *requests* plays a critical role in the job satisfaction of the *employees* by balancing work and private life. It also results in higher *employee* loyalty and performance, which are crucial factors in the sectors where the qualified workforce is a scarce resource.

Absence request on individual holidays Suppose that an *employee* submits a five day *absence request* for home redecoration. This *request* does not need to be granted in its entirety, but the more days granted, the better it is for the *employee*. For example, there is no problem for the *employee* if only four days between Monday and Friday are granted. Even if a day in the middle of the period is not granted, the situation is still acceptable. In that case, the *date set* consists of the five days that the *employee* requests, but these dates are handled individually. The penalty of this request is calculated using equation (3.2). Another parameter of the *absence request* constraint is the *job duration* for each granted day. This *job duration* will be added to the total working time of the requesting *employee* for each day the *absence request* is granted. Suppose that the *employee* in this

example has a full time *contract* that requires him or her to work 38 hours per week, which corresponds to seven hours and 36 minutes a day. If this *absence request* is preferred to be a part of the paid vacation of the *employee*, then for each granted day, seven hours and 36 minutes will be added to his or her *hours worked counters* that cover this *absence request*. If the employee is granted four free days for this *absence request*, that will add an extra 30 hours and 24 minutes to the related *hours worked counters*.

$$Penalty = Number\ of\ Days\ Not\ Granted * Weight \tag{3.2}$$

An *assignment* in any *shift type*, with any *skill type*, on the requested dates will result in a violation of this *request*. Therefore, the *skill type set* and the *shift type set* of the *domain* used in this *absence request* consist of all the *skill types* and *shift types* in the problem. Instead of enlisting all the *shift types* and *skill types* in the problem, these *skill* and *shift type sets* are expressed with the value *any*.

- Domain
  - Day Set
    - Date Set: requested dates
    - Handling: individual
  - Shift Type Set: any
  - Skill Type Set: any

Absence request as a block Suppose that an *employee* requests a five day *absence request* for a ski vacation. The *absence request* is considered to be granted, if and only if all the five days in the request are granted. In contrast to the *absence request on individual days*, the penalty of this constraint is not scaled by the number of days not granted. The penalty, the *weight* of the constraint, is a fixed value and it is added to the objective value in case the *request* is not granted. For this *absence request as a block*, the *date set* consists of the five days that the *employee* requests and these dates are handled as a block. Similar to the *absence request on individual days*, the *shift* and *skill type set* consists of all the *shift* and *skill types* defined in the problem instance. Despite being encountered frequently in the real world practice, handling *absence requests* as a block is a completely new concept in the personnel rostering literature.

- Domain
  - Day Set
    - Date Set: requested dates
    - Handling: block
  - Shift Type Set: any
  - Skill Type Set: any

Absence request on Wednesday afternoons Primary and secondary schools in Belgium, as well as in many other countries, have free Wednesday afternoons. Therefore, many *employees* with school children prefer to stay at home on Wednesday afternoons. This *absence request* emphasises the *domain* element's *shift type set*, which, in this case, consists of the *shift types* that overlap with the afternoon. The *date set* consists of Wednesdays and the dates are handled *individually*. The reader is cautioned that any arbitrary set of day types or dates can be represented using the *domain* element and Wednesday is selected as a relevant example in this case to demonstrate how a real world example can be converted to a *domain* element. Since the *skill types* do not play any role in this constraint, the *skill type set* consists of all the *skill types* in the problem definition.

- Domain
  - Day Set
    - Day Types: Wednesdays
  - Shift Type Set: shift types that overlap with the afternoon
  - Skill Type Set: any

Assignment request for a specific skill type In working environments with multiple *levels of experience* for *employee skill types*, *employees* can increase their *level of experience* by working for the corresponding *skill types* as much as possible. Suppose that a senior caregiver is also a trainee as a regular nurse. She needs to work a number of *shifts* as a regular nurse in order to increase his or her *level of experience* from trainee to junior. Therefore, he or she prefers as many *assignments* as possible as a regular nurse. This example emphasises the role of the *skill type set* in the *domain* element. In this case, the *skill type set* consists of *regular nurse* only. The dates and *shift types* do not play any role in this *assignment request*. The *date set* therefore includes all the dates in the *schedule period* and the *shift type set* consists of all the *shift types* defined in the problem instance. Similar to the *absence request* for home redecoration case, the more dates are assigned, the better it is for the nurse. Therefore, the dates are handled individually and the penalty for this *request* is calculated using the formula in equation (3.2).

- Domain
  - Day Set
    - Day Types: any
  - Shift Type Set: any
  - Skill Type Set: regular nurse

Individual bank holidays According to the employment legislation, each *employee* is allowed to take a specific number of *bank holidays* each year. Therefore, the planners have difficulties in satisfying the *coverage constraints* on *bank holidays*. On the other hand, the official *bank holidays* are not equally relevant to all *employees*. For example, the religious holidays of the most common religion in the country are not relevant to the members of other religions. In this case, it is beneficial for the institution as well as for the *employees*, to define an *individual set of bank holidays* related to their own religion or culture.

- Domain
  - Day Set
    - Date Set: individual bank holidays
    - Handling: individual
  - Shift Type Set: any
  - Skill Type Set: any

In the traditional way of handling the *holidays worked constraint*, a *counter* is defined for each *employee*. This *counter* refers to the bank holidays that are defined globally. On the other hand, the *holidays worked counter* can be handled using *domains*. In this case, each *employee* has an *individual domain* that consists of the *bank holidays* related to him or her. A *counter* is defined on this *domain* to restrict the number of worked individual *bank holidays*.

Counter start values In some cases, the *counters* cover longer periods than the *schedule period* itself. For example, a *counter* can be defined on one year, while the *schedule period* is defined on four weeks. In this case, the *counter* value up until the current *schedule period* is given as the *counter start value* in the problem input.

Suppose that the number of bank holidays worked by an *employee* in a year should be exactly five and the *schedule period* is the month June. In this case, the *counter period* is one year and it covers the *schedule period*, the month June, completely. The *minimum* and *maximum thresholds* have the value 5, but the bank holidays

worked by the *employee* before the *schedule period*, i.e., before June, should be taken into account as well.

Suppose that the *employee* has already worked three bank holidays before June. In this case, the *counter start value* is 3. Any bank holidays worked in the month of June should be added on top of the *counter start value*, i.e., 3, and evaluated against the *threshold* value, i.e., 5. In rare cases, the *counter value* might have already exceeded the *threshold* before the *schedule period*.

Suppose that the *employee* has already worked 6 bank holidays before June. In that case, a constraint violation will be reported even if no assignment is made to any bank holiday in the current *schedule period*. This is for information purposes only and will not impact the algorithm, because the violations that are carried from the previous schedule periods will remain constant, since the algorithm cannot change the assignments in the previous *schedule periods*. The formal definition of how the *constraint start values* are used in the evaluation of the *counters* will be discussed in 4.3.5.

- Counter Start Values
  - Counter Start Value (1..*)
    - Constraint Id
    - Value

Counter remainder values The *counter remainder value* element is similar to the *counter start value* element. If the *counter period* exceeds the *schedule period*, then the number of the corresponding roster elements in the remainder of the *counter period* is given as the *counter remainder value*.

The concept of the *counter remainder value* can be demonstrated on the same example as the *counter start value*. In that example, the *employee* has to work exactly five bank holidays in a calendar year and the *schedule period* is the month June. Suppose that he or she has not worked on any bank holidays before the month June and there are only three bank holidays after the month of June. The three bank holidays after the month of June is the *counter remainder value* to be considered for this *schedule period*. The *counter remainder value*, i.e., 3, will be subtracted from the *minimum threshold* of the *counter*, i.e., 5. However, the *maximum threshold*, i.e., 5, will remain as it is. The rationale behind this is that the three of the five bank holidays can still be assigned after the *schedule period* and it is sufficient to assign at least two bank holidays in this *schedule period*. The *maximum threshold* remains the same, because *assignments* of up to five bank holidays will not cause any violation throughout the *counter period* of this constraint.

The *counter remainder value* is slightly different from the *counter start value*. The actual *assignments* or *idle days* that are processed by the *counter* are considered in the *counter start value*. However, all the possible *assignments* or *idle days* that can be processed by the *counter* are taken into account in the *counter remainder value*.

For example, the *counter start value* of the bank holidays worked *counter* takes into account the actual assigned bank holidays before the *schedule period*. On the other hand, the *counter remainder value* of the same *counter* takes into account all the bank holidays after the *schedule period* to which future assignments are possible. Suppose that, an *employee* has to work 5 bank holidays in a year. There has been 6 bank holidays before the *schedule period* and the *employee* has worked on 3 of these 6 bank holidays. Suppose that there are 4 bank holidays after the *schedule period*. In this case, the *counter start value* is 3, not 6, and the *counter remainder value* is 4.

Connecting the *schedule period* with the previous and upcoming *schedule periods* using *counter start* and *remainder values* addresses the expectations of the real world practitioners and it is a novel concept in the personnel rostering literature.

- Counter Remainder Values
  - Counter Remainder Value (1..*)
    - Constraint Id
    - Value

Rest time between shift types

Each *shift type* is given with a *rest period before* and *after*. Although eleven hours of *rest time before* and *after* is a common practice in the healthcare sector, the *rest time before* and *after* a *shift type* might be different in exceptional cases. For example, a *shift type* might require fourteen hours of *rest time before* and eleven hours of *rest time after*. *Assignments* of other *shifts* that overlap with these *rest periods* are considered as soft constraint violations. The *weight* of the *rest time between shift type* violations is given under the *weights* element of the *schedule definitions*.

- Weights
  - Rest Time Between Shifts

3.2.3 Schedule constraints

The constraints that restrict the *schedules* of more than one *employee* are expressed in the *Schedule Constraints* element. These are *coverage constraints*, *schedule locks*, *collaborations* and *trainings*.

- Schedule Constraints
  - Coverage Constraints
  - Schedule Locks (0..1)
  - **Collaborations (0..1)**
  - **Training Set (0..1)**

Coverage constraints

The primary objective of a rostering system is to fulfill the required number of *assignments* for each *day*, *shift type* and *skill type*. The definition of *coverage constraints* allows modelling the rostering problems that have *coverage constraints* as *determined* (D), *range* (R) and *fluctuating* (V) in *work characteristics* (β). The model also allows the definition of a *weight* for each *coverage constraint*, so that rostering problems with the *load and coverage constraint objective* (L) as a part of their *optimisation objective* (γ) can be modelled.

Following the common practice in the healthcare sector, the *coverage constraints* in this dissertation are determined per *day*, *skill* and *shift type* in the problem model. The *threshold* value restricts the number of *employees* on each *day*, for each *shift* and *skill type*. If more than one *level of experience* is possible for the *skill types* of the *employees*, then the *minimum level of experience* that satisfies this *coverage constraint* needs to be specified. The *levels of experience* are ordered hierarchically. Any *employee* with a *level of experience* higher than or equal to the minimum satisfies the *coverage constraint* in question.

The following example demonstrates how a complex real world *coverage constraint* can be expressed using the extended problem model. On July 4, 2011, at least two and at most four practitioners should be assigned a long early or long late shifts, either as a caregiver or a regular nurse. The practitioners should be at least a junior in the assigned skill types. In this case, the date is July 4, 2011. The *shift type set* consists of the long early and long late shifts. Long *shift types* are the ones that are assigned mostly to the full time *employees*. The *skill type set* consists of the *skill types* caregiver and regular nurse. The *minimum level of experience* is junior in this case. That means the *employees* who are at least junior in the required *skill types* are qualified for this *coverage constraint*. For example, a senior caregiver is qualified, but a trainee regular nurse is not. Although it might seem

complicated, *coverage constraints* such as this example are common in the real world practice.

- Coverage Constraints
  - Coverage Constraint (1..*)
    - Date
    - **Shift Type Set Id**
    - **Skill Type Set Id**
    - **Minimum Level of Experience**
    - Threshold
    - Weight

Schedule locks

Sometimes, planners construct a partial *schedule* that contains *assignments* and *idle days*. Modification of these parts of the *schedule* by the solution method is not desired. The parts that are not allowed to be modified by the solution method are identified with the *schedule locks* in the model.

- Schedule Locks
  - Schedule Lock (1..*)
    - Employee Id
    - Date

Collaboration

In some working environments, there are situations where specific groups of *employees* are required to work together. There are also cases where the opposite is desired. A specific group of *employees* request not to work together. For example, *employees* with complementary *skills* can be required to work together. Family members can request to work in different *shifts* so that at a given time, at least one of them can take care of their children. This constraint is expressed as *chaperoning* (C) in the *personnel environment* α in [49].

In the present model, the number of *employees* to collaborate is not limited to two. The *employees* to collaborate are expressed as an *employee set*. The application of the *collaboration* constraint can be restricted using the *domain* element. The planner can define *collaboration* constraints that are required only on specific days, for specific *shift* and *skill types*.

The *threshold* element defines the type of the *collaboration*. If the *employees* should not work together, then the maximum *threshold* needs to be set to zero. If at least n *employees* should work together, then the minimum *threshold* needs to be set to n.

- Collaborations
  - Collaboration (1..*)
    - Employee Set
      - Employee Id (2..*)
    - Domain Id
    - Threshold
    - Weight

Training

In a working environment with multiple *levels of experience*, it is desired that the experienced *employees* work together with less experienced *employees* in order to train them. A *training* constraint restricts the ratio between the numbers of assigned *employees* with different *levels of experience*.

The following rule is an example of the *training* constraint: *At least one senior caregiver should be assigned for every five trainee caregivers assigned*. In that case, the *preceding level* is trainee and the *succeeding level* is senior. The *threshold ratio* is 0.2. The *domain* consists of all days and *shift types*, but it is restricted to caregivers for the *skill type*. The *training* constraint is defined as a one way relationship.

If a two way relationship is preferred, the complementary *training* constraint needs to be defined as well. The following rule is complementary to the example mentioned above: *At least five trainee caregivers should be assigned for each senior caregiver assigned*. In this case, the *preceding level* is senior and the *succeeding level*, trainee. The *threshold ratio* is 5. The *domain* is the same as for the complementary example.

- Training Set
  - Training (1..*)
    - Preceding Level
    - Succeeding Level
    - Threshold Ratio
    - Domain Id
    - Weight

3.2.4 Schedule

The solution for the personnel rostering problem is a *schedule*, which is a set of *assignments*. An *assignment* consists of four elements: the *employee* that is assigned, the *date*, *shift type* and the *skill type* that is required.

Contrary to many academic resources on nurse rostering, e.g., [45, 61], the *skill type* is considered as a part of the *assignment*. If a nurse has more than one *skill type*, the *skill type* that he or she uses for the *assignment* would be ambiguous unless specified. This ambiguity is not a problem if the employee can use all of his or her *skill types* in the same *assignment* and if such an *assignment* is counted by the *coverage constraints* of all of these *skill types*. For example, a production supervisor has to be present throughout a shift, and in the meanwhile, he or she can operate a machine or perform other duties.

However, there are also problem instances where an *employee* can use only one of his or her *skill types* in a *shift*. For example, if two *skill types* are associated with different locations, then the *employee* can use only one of the *skill types* in a given *shift*. Suppose that a truck driver has a driver's licence for several types of vehicles, but that does not mean he or she can drive all those vehicles at the same time. If each vehicle type is modelled as a *skill type*, then the *skill type*, i.e., the vehicle type, must be specified in the *assignment* to overcome any ambiguity. Certainly, an *assignment* to this truck driver cannot be counted towards all the different *coverage constraints* that require the vehicle types he or she is qualified to drive. The *assignment* can be counted towards only one of those *coverage constraints*.

The *schedule* is the main output of a solution method that addresses personnel rostering problems. However, the *schedule* can also be a part of the problem input. Some of the personnel rostering constraints, such as the *series* constraints, overlap with two *schedule periods*. The *assignments* in the overlapping part of the previous *schedule period* need to be given as input in order to ensure an accurate constraint evaluation. This subject is studied in greater detail in Chapter 4. In some cases, planners construct a partial *schedule* and give it as input to the solution method in order to receive a complete *schedule*.

- Schedule
  - Assignment (1..*)
    - Employee Id
    - Date
    - Shift Type Id
    - Skill Type Id

3.3 Conclusions

A gap between academia and the real world has been identified in the literature surveys on personnel rostering (Chapter 2). One of the main reasons of this gap is the use of problem models that do not address the complexity and the diversity of the real world problems. The emphasis of the scientific literature has been on the algorithms and mostly simplified problem models have been used to prove the usefulness of the proposed algorithms. As a result, the simplified problem models cannot cope with the complexity of the real world problems. Academic problem models are mostly defined with a single problem instance in mind. Consequently, such problem models cannot deal with the diversity of the real world problems. The standard academic models have been extended by the author of the present dissertation with new elements to address the complexity and the diversity of the real world problems. In the present chapter, the extensions and the motivations behind these extensions have been discussed with their relevance to the standard academic models. The extensions stem from the real world requirements of the personnel rostering problem in the Belgian healthcare sector.

The constraints encountered in the real world practice are mostly complex and diverse. They are defined on a set of shift types, skill types and/or days. Shift type sets, skill type sets and day sets have been introduced to address this requirement. Furthermore, some constraints have been defined on a combination of these elements. Therefore, a *domain* element has been defined that is the combination of a *day set*, a *shift type set* and a *skill type set*.

The skill types of the employees constitute another area where the standard academic models have been extended. In the real world practice, employees can have more than one skill type. Mostly, not all the skill types of an employee have the same relevance to that employee. Therefore, a weight has been associated with each skill type of an employee. In some cases, different employees have different levels of experience for the same skill type. The skill type element of an employee has further been extended with a level of experience element to address such cases. The coverage constraint definition has therefore been extended with a *minimum level of experience* element. Furthermore, the variations in the *level of experience* have resulted in another constraint, namely the *training* constraint. Some employees are expected to work together or not to work together in various settings. A *collaboration* element has been introduced to address various requirements of employees working together or not working together.

The new problem model in this chapter has been presented as a data structure to express the problem instances accurately. The problem model has been presented in an XSD file in order to define it unambiguously. In order to facilitate future research as well as application development, the XSD file has been provided online for researchers and software developers. The extended problem model presented

in this chapter is at the core of the KAHO personnel rostering kernel that has been integrated in the commercial personnel planning software of our industrial partners[3]

The problem model is one part of the problem definition and it does not include formal definitions of constraint evaluations, which is another major part of the problem definition. The evaluation of constraints is one of the areas where the academic and real world practices diverge. Therefore, the evaluation of constraints will be discussed in greater detail in Chapter 4.

[3]SAGA Consulting, GPS NV Belgium

Chapter 4

Evaluation of constraints

The problem model in Chapter 3 provides a structured model for representing the problem data. The present chapter adds to it the formal definitions of the evaluations of constraints expressed in the data model. In real world problems, it is not always possible to satisfy all the constraints. Therefore, the constraints are divided into two groups, hard and soft constraints. Hard constraints must be satisfied in order to accept a solution as feasible. For example, an *employee* cannot be assigned to two *shifts* in different places at the same time. Soft constraints are preferred to be satisfied, but the violation of them does not make the solution infeasible. For example, granting the *absence request* of an *employee* is usually considered as a soft constraint. The violations of the soft constraints are to be minimised.

The relative priorities, often referred to as *weights*, of the soft constraints are usually not the same. Some soft constraints are more important than others. An objective function is constructed to model this fact. The objective function is the weighted sum of all the soft constraint violations. The relative priority of a soft constraint is expressed through its *weight* in the objective function. The higher the weight of a soft constraint, the more important it is in the problem. In the real world practice, it is not always clear if a constraint is soft or hard. Some constraints are treated as hard in some problems and soft in other problems. The constraints that are not hard but critical are mostly treated as soft constraints with very high weights in the objective function so that they are violated only in extreme cases.

As explained in Chapter 3, the problem model does not foresee a predefined set of soft constraints, but a data model to express a set of soft constraints specific to each problem instance and to each employee in the problem instance. Consequently, there is no predefined objective function associated with the problem definition. Rather, the objective function is defined in a generic way as the sum of the penalties

of all the soft constraints in a problem instance.

The evaluation of hard and soft constraints will be presented formally in this chapter. In addition to that, constraint evaluation over multiple schedule periods will be discussed in greater detail.

The continuity between two *schedule periods*, might be perceived as a minor detail from an academic point of view. However, it plays a major role in the rosters of actual nurses. Close collaboration with industrial partners and real world practitioners makes it possible to gather such requirements.

A set of consistent rules dealing with constraint evaluation over different *schedule periods* will be presented in this chapter. The importance of the rules is illustrated with examples. Ignoring these rules makes the comparison of approaches developed by different researchers impractical. It also excludes possible application in many real world environments.

4.1 Definitions and variables

- E: the set of all employees
- D: the set of all days in the current schedule period and in the related parts of the previous and upcoming schedule period
- S: the set of all shift types
- K: the set of all skill types
- W: the set of all weekends in the current schedule period and in the related parts of the previous and upcoming schedule period

$x_{e,d,s,k}$ in equation 4.1 are the decision variables of the problem.

$$\forall e \in E, \forall d \in D, \forall s \in S, \forall k \in K :$$

$$x_{e,d,s,k} = \begin{cases} 1 \text{ if employee } e \text{ is assigned on day } d, \\ \qquad \text{in shift type } s \text{ and skill type } k \\ 0 \text{ otherwise} \end{cases} \tag{4.1}$$

$p_{e,d}$ in equation 4.2 is an auxiliary variable that denotes the presence of employee $e \in E$ on day $d \in D$. $p_{e,d}$ takes the value 1 if employee $e \in E$ is assigned on day $d \in D$. It takes the value 0 otherwise.

$$
\begin{aligned}
&-|S||K|p_{e,d} + \sum_{s \in S} \sum_{k \in K} x_{e,d,s,k} \leq 0 \\
&-p_{e,d} + \sum_{s \in S} \sum_{k \in K} x_{e,d,s,k} \geq 0
\end{aligned}
\tag{4.2}
$$

$p_{e,d,S'}$ in equation 4.3, another auxiliary variable, denotes the presence of employee $e \in E$ on day $d \in D$, in any of the shift types $s \in S' \subset S$. $p_{e,d,S'}$ takes the value 1 if employee $e \in E$ is assigned on day $d \in D$, in any of the shift type $s \in S' \subset S$. It takes the value 0 otherwise.

$$
\begin{aligned}
&-|S'||K|p_{e,d,S'} + \sum_{s \in S'} \sum_{k \in K} x_{e,d,s,k} \leq 0 \\
&-p_{e,d,S'} + \sum_{s \in S'} \sum_{k \in K} x_{e,d,s,k} \geq 0
\end{aligned}
\tag{4.3}
$$

$p_{e,D',S',K'}$ in equation 4.4 denotes the presence of employee $e \in E$ on any of the days $d \in D' \subset D$, in any of the shift types $s \in S' \subset S$ and with any of the skill types $k \in K' \subset K$. $p_{e,D',S',K'}$ takes the value 1 if employee $e \in E$ has at least one assignment on any of the days $d \in D' \subset D$, in any of the shift types $s \in S' \subset S$, with any of the skill types $k \in K' \subset K$. It takes the value 0, otherwise.

$$
\begin{aligned}
&-|D'||S'||K'|p_{e,D',S',K'} + \sum_{d \in D'} \sum_{s \in S'} \sum_{k \in K'} x_{e,d,s,k} \leq 0 \\
&-p_{e,D',S',K'} + \sum_{d \in D'} \sum_{s \in S'} \sum_{k \in K'} x_{e,d,s,k} \geq 0
\end{aligned}
\tag{4.4}
$$

Similarly, $q_{e,w}$ denotes the presence of employee e in weekend w (equation 4.5). This variable becomes 1 if employee e works at least one shift in weekend w, and 0 otherwise. l refers to the length of the weekend. $d_{w,i}$ denotes day i of weekend w.

$$\forall e \in E, \forall w \in W :$$

$$\begin{aligned} -l\ q_{e,w} + \sum_{i=1}^{l} p_{e,d_{w,i}} &\leq 0 \\ -q_{e,w} + \sum_{i=1}^{l} p_{e,d_{w,i}} &\geq 0 \end{aligned} \tag{4.5}$$

4.2 Hard constraints

4.2.1 Single assignment start per day per employee

Single assignment start per day per employee is a hard constraint (Equation 4.6).

$$\forall e \in E, \forall d \in D : \sum_{s \in S} \sum_{k \in K} x_{e,d,s,k} \leq 1 \tag{4.6}$$

4.2.2 Schedule locks

In some cases, manual planners can assign a preset value v to an employee $e \in E$, on a day $d \in D$, for a shift type $s \in S$ and a skill type $k \in K$. These preset assignments are hard constraints and they are not allowed to be changed. These preset values can be 1 or 0.

$$x_{e,d,s,k} = v \tag{4.7}$$

4.2.3 Honour skill types

Let the skill types of an employee $e \in E$ be $K_e \subset K$. In that case, Equation 4.8 forbids any assignment to e with a skill type k that e does not have. *Honouring*

skill types is a hard constraint.

$$\sum_{d \in D} \sum_{s \in S} \sum_{k \in K/K_e} x_{e,d,s,k} = 0 \tag{4.8}$$

4.2.4 Defined assignments only

If an *assignment* on day $d \in D$, with *shift type* $s \in S$ and *skill type* $k \in K$ is not defined in any of the *coverage constraints* $cc_{d,S',K'} \in CC$, then such an *assignment* cannot be made to any *employee* $e \in E$. *Assignments* that are provided in the input *schedule* cannot be modified, deleted or reassigned to another *employee* if no *coverage constraint* is defined for that *day*, *skill* and *shift type*.

The motivation behind this constraint is the fact that the *coverage constraints* are considered as soft constraints. For example, an *assignment* can be made even if the *maximum threshold* of the corresponding *coverage constraint* is equal to zero. This is necessary in practice when the working hours of an *employee* does not satisfy his or her *minimum number of required working hours* specified in his or her *contract*. However, there are situations when the *assignment* of a particular *shift type* to an *employee* with a particular *skill type* cannot be considered at all. For example, in some wards, a head nurse is never assigned in the *night shift* and therefore such a *coverage constraint* is never defined. Hence, the *defined assignments only* constraint is treated as a hard constraint.

$$\begin{aligned} &\forall d \in D, \forall s \in S, \forall k \in K : \\ &\neg\exists \ (cc_{d,S',K'} \in CC | s \in S', k \in K') \Rightarrow \sum_{e \in E} x_{e,d,s,k} = 0 \end{aligned} \tag{4.9}$$

4.2.5 Overlapping shift types

Let T be the set of shift type pairs (s_i, s_j) such that s_i and s_j overlap if they are assigned on consecutive days d and $d+1$, respectively. Equation 4.10 ensures that no overlapping shift type assignments are made to any employee on any day.

$$\forall d \in D, \forall e \in E, \forall (s_i, s_j) \in T : \sum_{k \in K} x_{e,d,s_i,k} + \sum_{k \in K} x_{e,(d+1),s_j,k} \leq 1 \tag{4.10}$$

4.3 Soft constraints

Let $v(x)$, $w(x)$, $m(x)$, $n(x)$, $p(x)$ refer to the value, weight, maximum threshold, minimum threshold and the penalty of a constraint x.

The objective value of a candidate solution is the sum of the penalties of all soft constraints X in the problem instance (Equation 4.11).

$$\sum_{x \in X} p(x) \tag{4.11}$$

4.3.1 Rest times between shift types

Let R be the set of shift type pairs (s_i, s_j) such that at least one of s_i and s_j violates the rest time of the other if they are assigned on consecutive days d and $d+1$, respectively. Let w_{rest} be the weight for the *rest time between shift types* constraint. P_{rest} in Equation 4.12 refers to the total *rest time between shift types* penalty, which is the number of violations of the *rest time between shift types* constraint multiplied by the corresponding weight.

$$P_{rest} = \sum_{e \in E} \sum_{d \in D} \sum_{(s_i, s_j) \in R} \sum_{k \in K} w_{rest} \cdot x_{e,d,s_i,k} \cdot x_{e,(d+1),s_j,k} \tag{4.12}$$

4.3.2 Employee skill type penalties

Let the skill types of an employee $e \in E$ be $K_e \subset K$. Each skill type $k \in K_e$ has a penalty $w_{e,k}$ that is added to the objective value, in case an assignment is made with that skill type. P_{skill} in Equation 4.13 refers to the total amount of employee skill type penalties of a schedule.

$$P_{skill} = \sum_{e \in E} \sum_{d \in D} \sum_{s \in S} \sum_{k \in K} w_{e,k} \cdot x_{e,d,s,k} \tag{4.13}$$

4.3.3 Coverage constraints

Coverage constraints are considered to be soft constraints. Let $S' \subset S$ and $K' \subset K$. The coverage constraint $cc_{d,S',K'}$ restricts the number of employees assigned on

day $d \in D$, for shift type set $S' \subset S$ and for skill type set $K' \subset K$. As it can be understood from its formulation, any assignment with any shift type s in the shift type set $S' \subset S$ and any skill type k in the skill type set $K' \subset K$ on day $d \in D$ is counted by the coverage constraint $cc_{d,S',K'}$. $v(cc_{d,S',K'})$ refers to the total number of assignments on day d, with any shift type $s \in S' \subset S$ and any skill type $k \in K' \subset K$. The total penalty $p(cc_{d,S',K'})$ of coverage constraint $cc_{d,S',K'}$ is calculated using Equation 4.15.

$$v(cc_{d,S',K'}) = \sum_{e \in E} \sum_{s \in S'} \sum_{k \in K'} x_{e,d,s,k} \tag{4.14}$$

$$\begin{aligned} & p(cc_{d,S',K'}) = \\ & w(cc_{d,S',K'}) \cdot \max\{0, v(cc_{d,S',K'}) - m(cc_{d,S',K'})\} + \\ & w(cc_{d,S',K'}) \cdot \max\{0, n(cc_{d,S',K'}) - v(cc_{d,S',K'})\} \end{aligned} \tag{4.15}$$

4.3.4 Collaboration

Let $E' \subset E$ be the employee set, $D' \subset D$ be the day set, $S' \subset S$ be the shift type set, $K' \subset K$ be the skill type set and the triple $< D', S', K' >$ be the domain of the collaboration constraint $l_{E',D',S',K'}$. $v(l_{E',d,S',K'})$ refers to the total number of assignments made to the members of the employee set E' on day $d \in D' \subset D$, for shift type set $S' \subset S$ and skill type set $K' \subset K$. The total penalty $p(l_{E',D',S',K'})$ of collaboration constraint $l_{E',D',S',K'}$ is calculated using Equation 4.17.

$$\forall d \in D' : v(l_{E',d,S',K'}) = \sum_{e \in E'} \sum_{s \in S'} \sum_{k \in K'} x_{e,d,s,k} \tag{4.16}$$

$$\begin{aligned} & p(l_{E',D',S',K'}) = \\ & \sum_{d \in D'} w(l_{E',D',S',K'}) \cdot \max\{0, v(l_{E',d,S',K'}) - m(l_{E',D',S',K'})\} + \\ & \sum_{d \in D'} w(l_{E',D',S',K'}) \cdot \max\{0, n(l_{E',D',S',K'}) - v(l_{E',d,S',K'})\} \end{aligned} \tag{4.17}$$

4.3.5 Counters

Let $csv(c)$ be the counter start value and $crv(c)$ the counter remainder value of counter c. The penalty $p(c)$ of counter c is calculated using Equation 4.18.

$$\begin{aligned} p(c) = & \\ & w(c) \cdot \max\{0, v(c) - m(c)\} + \\ & w(c) \cdot \max\{0, n(c) - v(c) - crv(c)\} \end{aligned} \tag{4.18}$$

Days worked counters

Let $D' \subset D$ be the day set of the days worked counter $dwc_{e,D'}$ of employee $e \in E$. The counter value $v(dwc_{e,D'})$ of days worked counter $dwc_{e,D'}$ is calculated using Equation 4.19.

$$v(dwc_{e,D'}) = csv(dwc_{e,D'}) + \sum_{d \in D'} p_{e,d} \tag{4.19}$$

Days idle counters

Let $D' \subset D$ be the day set of the days idle counter $dic_{e,D'}$ of employee $e \in E$. The counter value $v(dic_{e,D'})$ of days idle counter $dic_{e,D'}$ is calculated using Equation 4.20.

$$v(dic_{e,D'}) = csv(dic_{e,D'}) + \sum_{d \in D'} 1 - p_{e,d} \tag{4.20}$$

Shift types worked counters

Let $D' \subset D$ be the day set and $S' \subset S$ be the shift types set of the shift types worked counter $swc_{e,D',S'}$ of employee $e \in E$. The counter value $v(swc_{e,D',S'})$ of

shift types worked counter $swc_{e,D',S'}$ is calculated using Equation 4.21.

$$v(swc_{e,D',S'}) = csv(swc_{e,D',S'}) + \sum_{d \in D'} \sum_{s \in S'} \sum_{k \in K} x_{e,d,s,k} \tag{4.21}$$

Domain counters

Let $D' \subset D$ be the day set, $S' \subset S$ be the shift types set, and $K' \subset K$ be the skill type set of the domain counter $dc_{e,D',S',K'}$ of employee $e \in E$. The counter value $v(dc_{e,D',S',K'})$ of domain counter $dc_{e,D',S',K'}$ is calculated using Equation 4.22.

$$v(dc_{e,D',S',K'}) = csv(dc_{e,D',S',K'}) + \sum_{d \in D'} \sum_{s \in S'} \sum_{k \in K'} x_{e,d,s,k} \tag{4.22}$$

Weekends worked counters

Let $W' \subset W$ be the weekends in the counter period of the weekends worked counter $wwc_{e,W'}$ of employee $e \in E$. The counter value $v(wwc_{e,W'})$ of weekends worked counter $wwc_{e,W'}$ is calculated using Equation 4.23.

$$v(wwc_{e,W'}) = csv(wwc_{e,W'}) + \sum_{w \in W'} q_{e,w} \tag{4.23}$$

Weekends idle counters

Let $W' \subset W$ be the weekends in the counter period of the weekends idle counter $wic_{e,W'}$ of employee $e \in E$. The counter value $v(wic_{e,W'})$ of the weekends idle counter $wic_{e,W'}$ is calculated using Equation 4.24.

$$v(wic_{e,W'}) = csv(wic_{e,W'}) + \sum_{w \in W'} (1 - q_{e,w}) \tag{4.24}$$

Hours worked counters

Let $D' \subset D$ be the day set of the hours worked counter $hwc_{e,D'}$ of employee $e \in E$. Let $D_{ar} \subset D$, $S_{ar} \subset S$, and $K_{ar} \subset K$ be the day set, shift type set and skill type

set of absence request ar, respectively. Let $D'_{ar} = D' \cap D_{ar}$. Let AR_e be the set of all absence requests of an employee $e \in E$. Let $JobTime(s)$ be the net job time of a shift type $s \in S$ and $JobTime(ar)$ the net job time of an absence request ar.

$AR^c_e = \{ar \in AR_e | D'_{ar} \neq \emptyset \wedge \ handling\ of\ D_{ar}\ is\ complete\}$

$AR^i_e = \{ar \in AR_e | D'_{ar} \neq \emptyset \wedge \ handling\ of\ D_{ar}\ is\ individual\}$

$$
\begin{aligned}
&v(hwc_{e,D'}) = \\
&csv(hwc_{e,D'}) + \sum_{d \in D'} \sum_{s \in S} \sum_{k \in K} x_{e,d,s,k} \cdot JobTime(s) + \\
&\sum_{ar \in AR^c_e} (1 - p_{e,D_{ar},S_{ar},K_{ar}}) \cdot JobTime(ar) + \\
&\sum_{ar \in AR^i_e} \sum_{d \in D'_{ar}} \sum_{s \in S_{ar}} \sum_{k \in K_{ar}} (1 - x_{e,d,s,k}) \cdot JobTime(ar)
\end{aligned}
\tag{4.25}
$$

4.3.6 Series

Days worked series

The calculation of the total penalty $p(dws_e)$ of days worked series dws_e with a maximum threshold $m(dws_e)$ of an employee $e \in E$ is given in Equation 4.26.

$$
p(dws_e) = w(dws_e) \sum_{d \in D} max \left\{ \left(\sum_{i=0}^{m(dws_e)} p_{e,d+i} \right) - m(dws_e), 0 \right\} \tag{4.26}
$$

The calculation of the total penalty $p(dws_e)$ of days worked series dws_e with a minimum threshold $n(dws_e)$ of an employee $e \in E$ is given in Equation 4.27.

$$
\begin{aligned}
&p(dws_e) = \\
&w(dws_e) \sum_{d \in D} (1 - p_{e,d}) \cdot p_{e,d+1} \cdot max_{i=1}^{n(dws_e)} \{(n(dws_e) - i)\,(1 - p_{e,d+i+1})\}
\end{aligned}
\tag{4.27}
$$

Days idle series

The calculation of the total penalty $p(dis_e)$ of days idle series dis_e with a maximum threshold $m(dis_e)$ of an employee $e \in E$ is given in Equation 4.28.

$$p(dis_e) = w(dis_e) \sum_{d \in D} max \left\{ \left(\sum_{i=0}^{m(dis_e)} 1 - p_{e,d+i} \right) - m(dis_e), 0 \right\} \tag{4.28}$$

The calculation of the total penalty $p(dis_e)$ of days idle series dis_e with a minimum threshold $n(dis_e)$ of an employee $e \in E$ is given in Equation 4.29.

$$p(dis_e) = w(dis_e) \sum_{d \in D} p_{e,d} \cdot (1 - p_{e,d+1}) \cdot max_{i=1}^{n(dis_e)} \{(n(dis_e) - i)\,(p_{e,d+i+1})\} \tag{4.29}$$

Weekends worked series

The calculation of the total penalty $p(wws_e)$ of weekends worked series wws_e with a maximum threshold $m(wws_e)$ of an employee $e \in E$ is given in Equation 4.30.

$$p(wws_e) = w(wws_e) \sum_{w \in W} max \left\{ \left(\sum_{i=0}^{m(wws_e)} q_{e,w+i} \right) - m(wws_e), 0 \right\} \tag{4.30}$$

The calculation of the total penalty $p(wws_e)$ of weekends worked series wws_e with a minimum threshold $n(wws_e)$ of an employee $e \in E$ is given in Equation 4.31.

$$p(wws_e) = w(wws_e) \sum_{w \in W} (1 - q_{e,w}) \cdot q_{e,w+1} \cdot max_{i=1}^{n(wws_e)} \{(n(wws_e) - i)\,(1 - q_{e,w+i+1})\} \tag{4.31}$$

Weekends idle series

The calculation of the total penalty $p(wis_e)$ of weekends idle series wis_e with a maximum threshold $m(wis_e)$ of an employee $e \in E$ is given in Equation 4.32.

$$p(wis_e) = w(wis_e) \sum_{w \in W} max \left\{ \left(\sum_{i=0}^{m(wis_e)} 1 - q_{e,w+i} \right) - m(wis_e), 0 \right\} \quad (4.32)$$

The calculation of the total penalty $p(wis_e)$ of weekends idle series wis_e with a minimum threshold $n(wis_e)$ of an employee $e \in E$ is given in Equation 4.33.

$$p(wis_e) = w(wis_e) \sum_{w \in W} q_{e,w} \cdot (1 - q_{e,w+1}) \cdot max_{i=1}^{n(wis_e)} \{(n(wis_e) - i)\,(q_{e,w+i+1})\} \quad (4.33)$$

Shift types worked series

The calculation of the total penalty $p(sws_{e,S'})$ of shift types worked series $sws_{e,S'}$ defined on $S' \subset S$ with a maximum threshold $m(sws_{e,S'})$ of an employee $e \in E$ is given in Equation 4.34.

$$p(sws_{e,S'}) = w(sws_{e,S'}) \sum_{d \in D} max \left\{ \left(\sum_{i=0}^{m(sws_{e,S'})} p_{e,d+i,S'} \right) - m(sws_{e,S'}), 0 \right\} \quad (4.34)$$

The calculation of the total penalty $p(sws_{e,S'})$ of shift types worked series $sws_{e,S'}$ defined on $S' \subset S$ with a minimum threshold $n(sws_{e,S'})$ of an employee $e \in E$ is given in Equation 4.35.

$$p(sws_{e,S'}) = w(sws_{e,S'})$$

$$\sum_{d \in D} (1 - p_{e,d,S'}) \cdot p_{e,d+1,S'} \cdot max_{i=1}^{n(sws_{e,S'})} \{(n(sws_{e,S'}) - i)\,(1 - p_{e,d+i+1,S'})\} \quad (4.35)$$

4.3.7 Successive series

As the name suggests, the *successive series* constraint restricts the succession of series that occur in the schedule of an employee. The successive series are evaluated in two steps. In the first step, the schedule of the employee is divided into series according to the definition of the successive series. A data structure, *successive series token*, is used for this purpose. A *successive series token* item consists of two elements: *type* and *length*. The *type* element can take one of three values: *series 1*, *series 2* or *irrelevant*. The *length* element refers to the length of the series that the token represents.

- Successive series token
  - Type
  - Length

The first step of the successive series evaluation is creating a list of successive series tokens corresponding to the schedule of the employee. The algorithm of the *CreateToken(SeriesType)* procedure is given in Algorithm 2. This procedure is called from numerous places in the algorithms of the first step.

Algorithm 2 CreateToken(SeriesType) procedure

```
INPUT : SeriesType
sst := new SuccessiveSeriesToken()
sst.Type := SeriesType
sst.Length := 1
return sst
```

Algorithm 3 refers to the first step of the evaluation of the *days worked - days idle successive series*. Since a day in the schedule of an employee is either *worked* or *idle*, only two series types will be used for this successive series. *Series 1* refers to the series of *worked* days and *series 2* to the series of *idle* days. Algorithm 3 returns a list of successive series tokens of types *series 1* and *series 2*.

Algorithm 3 Days worked - days idle successive series

```
INPUT : p_{e,d}
sstl = new SuccessiveSeriesTokenList()
for d = 1 → |D| do
  if p_{e,d} == 0 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series2))
    else
      if sstl.LastElement.Type == Series1 then
        sstl.Add(CreateToken(Series2))
      else if sstl.LastElement.Type == Series2 then
        sstl.LastElement.Length++
      end if
    end if
  else if p_{e,d} == 1 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series1))
    else
      if sstl.LastElement.Type == Series1 then
        sstl.LastElement.Length++
      else if sstl.LastElement.Type == Series2 then
        sstl.Add(CreateToken(Series1))
      end if
    end if
  end if
end for
return sstl
```

The first step of the evaluation of the *days idle - days worked successive series*, i.e., Algorithm 4, is similar to the *days worked - days idle successive series*. The only difference is that *series 1* refers to the *idle* days and *series 2* refers to the *worked* days in this case.

Algorithm 4 Days idle - days worked successive series

```
INPUT : p_{e,d}
sstl = new SuccessiveSeriesTokenList()
for d = 1 → |D| do
  if p_{e,d} == 0 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series1))
    else
      if sstl.LastElement.Type == Series1 then
        sstl.LastElement.Length++
      else if sstl.LastElement.Type == Series2 then
        sstl.Add(CreateToken(Series1))
      end if
    end if
  else if p_{e,d} == 1 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series2))
    else
      if sstl.LastElement.Type == Series1 then
        sstl.Add(CreateToken(Series2))
      else if sstl.LastElement.Type == Series2 then
        sstl.LastElement.Length++
      end if
    end if
  end if
end for
return sstl
```

The remainder of the successive series involve a third series type, i.e., the *irrelevant* series type. The *irrelevant* series consists of the shift type assignments that do not belong to a shift type set of any of the series in the successive series. In case of the *shift types worked - shift types worked* successive series, *idle days* are also considered to be *irrelevant*. Algorithms 5, 6 and 7 refer to the algorithms of the first step of the evaluations of the *shift types worked - days idle*, *days idle - shift types worked* and *shift types worked - shift types worked* successive series, respectively.

Algorithm 5 Shift types worked - days idle successive series

```
INPUT : p_{e,d}
INPUT : p_{e,d,S'}
sstl = new SuccessiveSeriesTokenList()
for d = 1 → |D| do
  if p_{e,d} == 0 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series2))
    else
      if sstl.LastElement.Type == Series2 then
        sstl.LastElement.Length++
      else
        sstl.Add(CreateToken(Series2))
      end if
    end if
  else if p_{e,d,S'} == 1 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series1))
    else
      if sstl.LastElement.Type == Series1 then
        sstl.LastElement.Length++
      else
        sstl.Add(CreateToken(Series1))
      end if
    end if
  else
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Irrelevant))
    else
      if sstl.LastElement.Type == Irrelevant then
        sstl.LastElement.Length++
      else
        sstl.Add(CreateToken(Irrelevant))
      end if
    end if
  end if
end for
```

Algorithm 6 Days idle - shift types worked successive series

```
INPUT : p_{e,d}
INPUT : p_{e,d,S'}
sstl = new SuccessiveSeriesTokenList()
for d = 1 → |D| do
  if p_{e,d} == 0 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series1))
    else
      if sstl.LastElement.Type == Series1 then
        sstl.LastElement.Length++
      else
        sstl.Add(CreateToken(Series1))
      end if
    end if
  else if p_{e,d,S'} == 1 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series2))
    else
      if sstl.LastElement.Type == Series2 then
        sstl.LastElement.Length++
      else
        sstl.Add(CreateToken(Series2))
      end if
    end if
  else
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Irrelevant))
    else
      if sstl.LastElement.Type == Irrelevant then
        sstl.LastElement.Length++
      else
        sstl.Add(CreateToken(Irrelevant))
      end if
    end if
  end if
end for
return sstl
```

Algorithm 7 Shift types worked - shift types worked successive series

```
INPUT : p_{e,d,S'_1}
INPUT : p_{e,d,S'_2}
sstl = new SuccessiveSeriesTokenList()
for d = 1 → |D| do
  if p_{e,d,S'_1} == 1 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series1))
    else
      if sstl.LastElement.Type == Series1 then
        sstl.LastElement.Length++
      else
        sstl.Add(CreateToken(Series1))
      end if
    end if
  else if p_{e,d,S'_2} == 1 then
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Series2))
    else
      if sstl.LastElement.Type == Series2 then
        sstl.LastElement.Length++
      else
        sstl.Add(CreateToken(Series2))
      end if
    end if
  else
    if sstl.Length == 0 then
      sstl.Add(CreateToken(Irrelevant))
    else
      if sstl.LastElement.Type == Irrelevant then
        sstl.LastElement.Length++
      else
        sstl.Add(CreateToken(Irrelevant))
      end if
    end if
  end if
end for
return sstl
```

The first step of the evaluation is specific to each successive series type. The second step, however, is common among all successive series types. In the second step, which is presented as Algorithm 8, the penalty is calculated using the list of tokens constructed in the first step.

Algorithm 8 Second step of the evaluation of successive series

$INPUT : sstl := SuccessiveSeriesTokenList$
$INPUT : Min1 := minimum\ threshold\ of\ series\ 1$
$INPUT : Max1 := maximum\ threshold\ of\ series\ 1$
$INPUT : Min2 := minimum\ threshold\ of\ series\ 2$
$INPUT : Max1 := maximum\ threshold\ of\ series\ 2$
$p(sucseries) := 0$
for $i = 1 \rightarrow sstl.Length - 1$ **do**
  if $\{(sstl[i+1].Type == Series2) \wedge$
  $(Max2 < sstl[i+1].Length) \wedge$
  $(sstl[i].Type == Series1) \wedge$
  $(Min1 \leq sstl[i+1].Length) \wedge$
  $(sstl[i+1].Length \leq Max1)\}$ **then**
    $p(sucseries) += w(sucseries) \cdot (sstl[i+1].Length - Max2)$
end if
if $\{(sstl[i+1].Type == Series2) \wedge$
  $(sstl[i+1].Length < Min2) \wedge$
  $(sstl[i].Type == Series1) \wedge$
  $(Min1 \leq sstl[i+1].Length) \wedge$
  $(sstl[i+1].Length \leq Max1)\}$ **then**
    $p(sucseries) += w(sucseries) \cdot (Min2 - sstl[i+1].Length)$
end if
  end for

4.3.8 Absence request

Let $D' \subset D$ be the day set, $S' \subset S$ the shift type set and $K' \subset K$ be the skill type set of the domain $< D', S', K' >$ of the absence request $ar_{e,D',S',K'}$ of employee $e \in E$. The handling of the days set in the domain of the absence request determines how the absence request is evaluated.

If the handling is individual, then the total penalty $p(ar_{e,D',S',K'})$ of the absence request is calculated using Equation 4.36.

$$p(ar_{e,D',S',K'}) = w(ar_{e,D',S',K'}) \sum_{d \in D'} \sum_{s \in S'} \sum_{k \in K'} x_{e,d,s,k} \tag{4.36}$$

If the handling is complete, then the total penalty $p(ar_{e,D',S',K'})$ of the absence request is calculated using Equation 4.37.

$$p(ar_{e,D',S',K'}) = w(ar_{e,D',S',K'}) \cdot p_{e,D',S',K'} \tag{4.37}$$

4.4 Constraint evaluation over multiple schedule periods

The common practice in academia has been to deal with problem instances considering an isolated *schedule period* only. This is fine provided that the evaluation of all the constraints is sorted out, which is often not the case. This simplification does not match the real world practice in hospitals, when constraints apply to consecutive days. Moreover, it makes the comparison of approaches impossible, because methods that have been developed for real world purposes include more complex aspects.

Burke et al. [27] have presented a constraint evaluation method for nurse rostering problems. The initialisation algorithm of the evaluation method takes into account the previous *schedule period*. The motivation behind this practice is to evaluate the consecutiveness constraints that overlap with the previous *schedule period* accurately. In [33], Glass and Knight have studied the nurse rostering practice and pointed at the importance of the continuity between two *schedule periods*. The ideas that have been asserted in that paper are in alignment with the case of this dissertation.

In real world practice, a solution is evaluated taking into consideration the previous and to a lesser extent also the upcoming *schedule periods*. The *assignments* at the end of the previous *schedule period* need to be taken into account to evaluate the constraints that apply to assignments overlapping with the previous and current *schedule periods*. Similarly, the information in the upcoming *schedule periods* often needs to be considered when evaluating the constraints that overlap with the current and upcoming *schedule periods*.

4.4.1 Example 1

Consider the *min three consecutive idle days* constraint. The evaluation of this constraint requires taking into account the previous *scheduling period* as well. The *idle days block* at the end of the previous *schedule period* in Table 4.1 is not penalised as a constraint violation in the previous *schedule period*, because the constraint violation can be overcome in the current *schedule period*. However, the same block needs to be taken into account when evaluating the constraint in the current *scheduling period*. A *working day assignment* on the first day of the current *schedule period* will result in a constraint violation. In order to avoid a constraint violation caused by the *idle days block* at the end of the previous *schedule period*, an *idle day* needs to be assigned on the first day of the current *schedule period*.

		Previous	Current		
On	Off	Off	?		

Table 4.1: Continuity Example 1

4.4.2 Example 2

Consider the *no isolated assignment* constraint and the situation in Table 4.2. This constraint cannot be evaluated based on the information in the current *schedule period* only. Whether there is a constraint violation in the current *schedule period* or not, depends on the *assignment* on the last day of the previous *schedule period* (the day marked with '?' in Table 4.2).

		Previous	Current		
		?	On	Off	

Table 4.2: Continuity Example 2

4.4.3 Example 3

It is common practice not to penalise a constraint violation at the end of a *schedule period* if it could be overcome by *assignments* or *idle days* in the upcoming *schedule period*. Consider the *at least three consecutive days worked* constraint. The two *assignments* at the end of the current *schedule period* in Table 4.3 do not contribute to a constraint violation because the problem could be overcome by an *assignment* at the start of the upcoming *schedule period*.

This kind of practice is relevant in the real world only if such violations are handled in the upcoming *schedule period*. Table 4.4 pictures the same situation as it is observed in the upcoming *schedule period*. The violation of the *at least three*

		Current	Upcoming		
Off	On	On	?		

Table 4.3: Continuity Example 3a

		Previous	Current		
Off	On	On	?		

Table 4.4: Continuity Example 3b

consecutive days worked constraint at the end of the previous *schedule period* is not penalised in the previous *schedule period* because this violation could be overcome in the current *schedule period*. This constraint overlaps with the previous *schedule period*. In its turn, this means that the constraint violations in the previous *schedule period* that can be overcome in the current *schedule period* need to be handled in the current *schedule period*. In this case, either an *assignment* needs to be made to the first day of the current *schedule period*, or the corresponding constraint violation needs to be penalised in the objective function.

4.4.4 Example 4

The following situation is an example of how the information in the upcoming *schedule period* is used in the current *schedule period*. The *schedule* in Table 4.5 is the same as the *schedule* in Table 4.3 with one exception, namely the nurse is on vacation on the first day of the upcoming *schedule period*. That day corresponds to an *idle day*. This implies that the violation of the *at least three consecutive days* constraint cannot be overcome in the upcoming *schedule period*. Therefore, this violation needs to be penalised in the current *schedule period*, because this is the only opportunity to address this violation.

		Current	Upcoming		
Off	On	On	V		

Table 4.5: Continuity Example 4. V refers to vacation

4.4.5 Example 5

The *holidays worked counter* is considered in Example 5. Sometimes, *holidays worked counters* are defined over a period of one year in real world practice. During a given *schedule period* in a year, the *holidays worked* in the previous *schedule periods* need to be known. Information on the number of remaining holidays in the rest of the reference period, which in the present case is one year, is also required.

Previous				Current				Upcoming			
	HW	HW			H	H	H		H		

Table 4.6: Continuity Example 5. HW refers to a holiday that is worked. H refers to a bank holiday.

Table 4.6 shows a nurse who has to work on at least five bank holidays in a year. There are three bank holidays in the current *schedule period*. He or she has already worked on two holidays in the previous *schedule periods* and there is only one holiday left after the current *schedule period* in the current year.

If the information from the previous *schedule periods* is taken into account, he or she still has to work on three bank holidays in the remainder of the year, because he or she has already worked two holidays. If the information from the upcoming *schedule periods* is taken into account, he or she has to work at least two holidays in the current *schedule period*, because there is one more holiday in the remainder of the year. Any number less than two holidays worked in the current *schedule period* will result in a violation of this constraint at the end of the year.

4.4.6 Example 6 (Valouxis 1)

Although the continuity between *schedule periods* seems to be a minor boundary issue, it can result in big differences in practice. The *schedule* in Fig. 4.1 has been presented as the optimal solution for the problem instance Valouxis 1 in the Nottingham benchmarks. In Fig. 4.1, each row represents the schedule of a nurse and each column a day in the *schedule period*. The filled squares represent the days with *assignments* and the empty ones the *idle days*. The penalty for this solution has been reported to be 20. However, if the roster in Fig. 4.1 is examined carefully, three specific *assignments* can be seen at the beginning of the *schedule period*. Depending on the *assignments* in the previous *schedule period*, these *assignments* may be *isolated assignments*. Since the *no isolated assignment* constraint has a penalty of 1000, it is possible that the penalty of this solution is greater than 3000. This number exceeds the reported value of 20 by a margin that can make this solution unacceptable in practice. This anomaly makes academic comparison of algorithm performance impossible, because other algorithms have been developed to work with the real world evaluation functions that consider continuity rules.

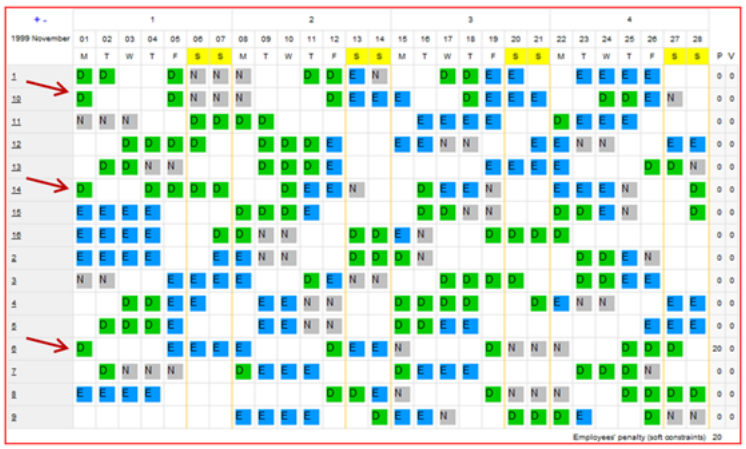

Figure 4.1: Optimal Solution for Valouxis 1

4.4.7 Example 7 (SINTEF)

The problem instance SINTEF from the Nottingham benchmarks contains pattern definitions of the *min two consecutive day shifts* and *min two consecutive early shifts constraints*. These pattern definitions have exceptions for the beginning of the current *schedule period*. A {day-on, day-off} sequence at the start of the *schedule period* is considered to be a constraint violation. In practice, an *assignment* of a relevant *shift type* at the end of the previous *schedule period* would make a {day-on, day-off} sequence at the start of the *schedule period* perfectly legal.

4.4.8 Example 8 - Maximum consecutive free days (BCV-3.46.2, Millar-2Shift-DATA1, MUSA, Ikegami-2Shift-DATA1)

The pattern definition of the *max seven consecutive free days* constraint in the problem instance BCV-3.46.2 from the Nottingham benchmarks does not take into account the idle days at the end of the previous *schedule period*. Even if there are seven consecutive idle days at the end of the previous *schedule period*, up to seven idle days can be assigned at the beginning of the current *schedule period* without a penalty according to the pattern definition. This results in 14 consecutive idle days that are not penalised. Similar continuity problems occur in the Millar-2Shift-Data1, MUSA and Ikegami-2Shift-DATA1 instances, where the *max X consecutive free days* constraints have been defined in exactly the same way.

The *max seven consecutive free days* constraint in BCV-3.46.2 has generated the most severe differences between the benchmark evaluation and the evaluation

function presented in this dissertation. The previous *schedule period* is assumed to be empty, because no *assignments* are defined in that period. As a result, the algorithms developed by the author (Chapter 7) corrects the violations at the end of the previous *schedule period* by making *assignments* at the start of the current *schedule period*.

4.4.9 Example 9 - Maximum consecutive working days (BCV-3.46.2)

The pattern definition of the *max seven consecutive working days* constraint in contract *Waverpleeg* has been defined as the following unwanted pattern: {8-days-on, 1-day-off}. This unwanted pattern definition evaluates any *assignment* sequence of eight or more *assignments* as one constraint violation. There is a fixed amount of penalty for any sequence of *assignments*, be it eight or 27. In the real world however, the amount of penalty is proportional to the size of the violation. Similar pattern definitions are present for *maximum consecutive night shift* and *maximum consecutive working days* constraints in *Waverpleeg* and other contracts.

4.4.10 Example 10 (GPost, ORTEC01)

The pattern definition of the *min 2 consecutive free shifts* constraint penalises the {day-off, day-on} sequence at the beginning of the current *schedule period*. This pattern definition assumes that the last day of the previous *schedule period* is a working day. This contradicts with most of the consecutiveness constraints in the same instances, which assume the last day of the previous *schedule period* to be an idle day. For example, the *no isolated assignment* constraint penalises the {day-on, day-off} sequence at the beginning of the current *schedule period*, which is based on the assumption that the last day of the previous *schedule period* is an idle day. This contradiction exists in both instances, GPost and ORTEC01.

Furthermore, the assumption that the last day of the previous *schedule period* to be an idle day is not completely accurate. Such an assumption implies that no assignment is made in the last day of the previous *schedule period*. This implication contradicts with the coverage constraints, which are hard constraints in the Nottingham benchmarks.

4.4.11 Example 11 (Azaiez, WHPP)

The unwanted pattern of the *no isolated assignment* constraint in Contract RKH in the problem instance Azaiez from the Nottingham benchmarks has been defined as {day-off, day-on, day-off} sequence. Similar to Valouxis 1 (Example 6), it is not

clear how an isolated *assignment* at the beginning of the current *schedule period* is going to be evaluated. The same pattern definition occurs in the problem instance WHPP from the Nottingham benchmarks.

4.5 Conclusions

The problem model of the nurse rostering problem has been introduced in Chapter 3 as a collection of data structures. Complementary to Chapter 3, the formal definitions of the evaluations of the constraints have been presented in this chapter. The nurse rostering problem has been treated as a constraint optimisation problem. First the decision variables and the auxiliary variables based on the decision variables have been defined. The evaluations of the hard and soft constraints have formally been defined using the decision and auxiliary variables. This information is essential to the developer, in order to provide consistent evaluation methods that comply with the user's quality perception.

A candidate solution, a schedule in the present case, is feasible if and only if it satisfies all the hard constraints. The objective value of the candidate solution is the linear combination of the soft constraint violations. The calculation of the values and penalties of all the soft constraint types have been presented in this chapter.

Furthermore, constraint evaluation over multiple *schedule periods* has been discussed in greater detail with hypothetic examples as well as with examples from the literature. Although continuity issues in constraint evaluation seem to be boundary conditions, they can result in significant deviances from the solutions that are acceptable and satisfactory in real world practice. The deviances in the objective values between the real world and academic constraint evaluations impede the intended purpose of algorithm comparison. The algorithms presented in this dissertation deploy the constraint evaluation methods developed for the real world practice. Therefore, the comparison of the objective values achieved by the algorithms presented in this dissertation and by the academic algorithms do not provide an accurate performance comparison. Such issues can be overcome by incorporating the schedule information from the previous and upcoming *schedule periods* in the constraint evaluation of the academic algorithms as well. In any case, we propose modifications to the academic benchmark constraint evaluation functions so that they are compliant with real world practice.

Chapter 5

Quantitative measures of problem properties

One of the conclusions of the academic context in Chapter 2 is that, despite considerable academic efforts, real world applications of the solution methods to the personnel rostering problem have been very limited. The academia's approaches have not been able to address a number of real world requirements for the personnel rostering problem. One of these requirements is to be able to address a variety of problem instances. A single algorithm configuration cannot be expected to perform well on all personnel rostering problems. Wolpert and Macread [71] have formalised this fact in their work on *no free lunch* theorems. They have stated that the performance of algorithms vary over different problems. If algorithm a_1 outperforms algorithm a_2 on problem p_1, then there must be a problem p_2 on which the algorithm a_2 outperforms algorithm a_1. Consequently, if a solution method has to address a broad range of problem instances, it cannot rely solely on a single algorithm configuration.

In case of personnel rostering, a single algorithm configuration cannot be expected to perform well on all problem instances from various sources, such as different countries, sectors, institutions and on problem instances created synthetically in laboratory conditions. A single algorithm configuration is expected to perform well on some instances and unsatisfactorily on others. Therefore, first, a range of problem instances have to be identified as target instances. In the present dissertation, real world personnel rostering problems have been identified as target instances. Second, a solution method consisting of several algorithms with a variety of configurations is needed, in order to increase the probability of finding good enough solutions on the targeted range of problems.

The solution method needs to decide somehow which algorithm configuration to choose to tackle a given problem instance. This practice prevents the interference of an expert by circumventing his task of analysing the problem instance, experimenting with several algorithms and determining the most promising one. A fully automated system eliminating the involvement of an expert leads to time and cost savings. In addition, such a system provides a vast economic value, because as a software product, it can be duplicated easily and deployed on multiple sites simultaneously, compared to the deployment of an expert on a single site.

The relations between the algorithm performance and the problem properties are required for an automated decision of which algorithm configuration to execute on a given problem instance. Such a decision process involves measuring the problem properties and selecting the most promising algorithm configuration for those properties. This process adds an overhead to the execution time of the algorithm. Therefore, it should not take longer than an acceptable fraction of the execution time of the algorithm, in order to satisfy the end users who expect a good enough solution in a limited amount of execution time.

Relationships between algorithm performance and problem properties have been investigated for the personnel rostering problem in the scientific literature [52, 55, 65, 68]. These studies give a profound insight in the contribution of the problem properties to the hardness of a problem instance. Properties such as the size of a problem and the ratio between the supply and demand of the workforce are the main factors of the hardness of a personnel rostering problem instance.

The problem model introduced in Chapter 3 involves numerous real world requirements. *Schedule locks*, one of the frequently used requirements in the real world practice, have not been considered in the hardness measures reported in the scientific literature. If a part of the *schedule* is locked by the end user, then the algorithm is not allowed to make any changes on that part. *Schedule locks* reduce the size of the problem instance for the algorithm. Depending on the constraints that apply to the locked parts, *schedule locks* can increase or decrease the hardness of a problem instance.

In the scientific literature, the size of a problem instance has been measured either as the number of *employees*, *days* or *shift types*. All of these measures are factors of the overall size of a problem instance, but none of them represent the overall size by itself. Therefore, quantitative measures that take into account the *schedule locks* and the overall size of the problem instance are needed. Three measures are introduced in this chapter to address these requirements. Two of the quantitative measures are based on the problem size: *minimum number of required assignments* and *minimum number of required and free assignments*. The third quantitative measure, the *tightness ratio*, takes into account to which extent the *coverage constraints* can be addressed by the employees. In other words, the *tightness ratio* indicates the ratio of workforce demand to workforce supply. This value is ideally

below 1 indicating that there is sufficient workforce supply to cover the demand. Although rare, the value of the *tightness ratio* exceeds 1 in the real world practice. That points to a workforce supply shortage in the corresponding ward.

The quantitative measures of several nurse rostering benchmarks will be presented in Chapter 6. The analysed benchmarks will be presented in four groups according to their measures in the same chapter. The solution methods presented in Chapter 7 are applied to the nurse rostering benchmarks and the results of these experiments will be reported in Chapter 8. As a result, a relation between the quantitative measures and the performance of the algorithm configurations will be inferred (Chapter 8). The most promising algorithm variant and parametrisation for each group will be pointed out in Section 8.2. The inferred relation can then be used in personnel rostering software that utilise the problem model introduced in Chapter 3 and the solution methods introduced in Chapter 7.

5.1 Minimum number of required assignments

The number of *assignments* an algorithm has to deal with is a more accurate estimation of the problem size than the number of *days*, *employees* or *shift types* alone. This measure can be derived from the number of assignments required by the *coverage constraints*. However, the *coverage constraints* do not always express an exact number of required *assignments*. Sometimes they are defined with an *interval*: a *minimum* and a *maximum* number of *employees* needed for a date, *shift* and *skill type*. For this measure, only the *minimum number of assignments* that are required by the *coverage constraints* are used.

There are two cases, in which the value of the *minimum number of required assignments* measure can exceed the actual number of assignments that the algorithm has to deal with. In some rare cases such as personnel shortage, the *minimum number of required assignments* can exceed the workforce supply in a problem instance. In such a case, the actual number of the *assignments* an algorithm has to deal with is lower than the *minimum number of required assignments*. The second case is when a considerable number of *schedule locks* prevent the algorithm from modifying a subset of the *assignments* required by the *coverage constraints*. In both cases, the number of assignments that an algorithm has to deal with is actually smaller than the value of the *minimum number of required assignments*. To address both cases a second measure of problem size, *minimum number of required and free assignments*, has been introduced in the following section.

5.2 Minimum number of required and free assignments

In practice, planners construct some parts of the rosters manually and then lock these parts before executing the automated rostering tool on the problem instance (Section 3.2.3). The *assignments* and *idle days* in the locked parts of the roster are not allowed to be modified by the algorithm. The locked parts of the input *schedule* can reduce the number of the *required assignments* that can be processed by the rostering tool. On the other hand, in some rare cases, the number of minimum required *assignments* can exceed the number of the employees whose schedules are not locked on the *day* of the *required assignment*. Such cases also reduce the number of the *required assignments* that the rostering algorithm has to deal with. A second problem size measure, the *minimum number of required and free assignments* (MNRFA), has been introduced to take into account both cases.

Let D refer to the set of all days in the *schedule period* and K to the set of all skill types in the problem instance. Let $CC_{d,k} \subset CC$ be the set of *coverage constraints* defined on day d that contain skill type k in their skill type set. Let $n(cc_{d,S',k})$ be the *minimum threshold* of *coverage constraint* $cc_{d,S',k}$. Let $S' \subset S$ be the shift type set of *coverage constraint* $cc_{d,S',k}$. Let $E'_d \subset E$ be the set of employees whose schedules are locked on day d. Let $E''_{d,k} \subset E$ be the set of employees who have skill type k and whose schedules are not locked on day d. The *minimum number of required and free assignments* measure is calculated using the formula in Equation 5.1.

$$\sum_{d \in D} \sum_{k \in K} \sum_{cc_{d,S',k} \in CC_{d,k}} min\left(|E''_{d,k}|, \left(n(cc_{d,S',k}) - \sum_{s \in S'} \sum_{e \in E'_d} x_{e,d,s,k}\right)\right) \quad (5.1)$$

5.3 Tightness ratio

The *tightness ratio* measures how well the *coverage constraints* can be matched by the available workforce for a given problem instance. The *tightness ratio* of a problem instance is calculated by taking the average of the *tightness ratios* for each date and *skill type*. The *tightness ratio* for a date and *skill type* is the ratio of the *required number of working hours* to the *available number of working hours by employees*. The *required number of working hours* are calculated using *coverage constraints*. The number of *required shift type assignments* is multiplied by the *net working duration* (working hours or job time) of the *shift type*. The resulting value is the *required number of working hours*. Usually, *employees* have a maximum

number of working hours per time reference in their contracts. This information is used as the *available number of working hours by employees*. Similar to MNRFA, only the working hours data of the *assignments* that are not locked and the available *employees* are used. The only exception to this calculation is when a problem instance does not have *hours worked counters* for the employees. That is the case for example in the *Valouxis-1* instance from the Nottingham benchmarks. In such cases, the calculations are based on the *number of available working days of the employees* instead of the *available working hours by employees*.

Let D refer to the set of all days in the *schedule period* and K to the set of all skill types in the problem instance. The tightness ratio of the problem instance is calculated using equation 5.2.

$$\frac{\sum_{d \in D} \sum_{k \in K} \frac{requiredJobTime_{d,k}}{availableJobTime_{d,k}}}{|D||K|} \tag{5.2}$$

Let $CC_{d,k} \subset CC$ be the set of *coverage constraints* defined on day d that contain skill type k in their skill type set. Let $n(cc_{d,S',k})$ be the *minimum threshold* of *coverage constraint* $cc_{d,S',k}$. Let $S' \subset S$ be the shift type set of the *coverage constraint* $cc_{d,S',k}$. Let $E'_d \subset E$ be the set of employees whose schedules are locked on day d. Let $JobTime(s)$ be the net job duration of shift type s. The required job time on day d for skill type k is calculated using Equation 5.3.

$$requiredJobTime_{d,k} = \\ max\left(0, \sum_{cc_{d,S',k} \in CC_{d,k}} \left(n(cc_{d,S',k}) - \sum_{s \in S'} \sum_{e \in E'_d} x_{e,d,s,k} \cdot JobTime(s)\right)\right) \tag{5.3}$$

Let $E''_{d,k} \subset E$ be the set of employees who have skill type k and whose schedules are not locked on day d. Let $hwc_{e,D'}$ be the *hours worked counter* of employee e, such that $d \in D'$. Let $m(hwc_{e,D'})$ be the *maximum threshold* of hours worked counter $hwc_{e,D'}$. The available job time on day d for skill type k is calculated using Equation 5.4.

$$availableJobTime_{d,k} = \sum_{e \in E''_{d,k}} \frac{m(hwc_{e,D'})}{|D'|} \tag{5.4}$$

At the first sight, the *tightness ratio* resembles to the *total coverage constrainedness* measure by Vanhoucke and Maenhout [68]. However, there are some differences between the two measures. First, the model proposed in this dissertation does not require a *coverage constraint* value for each <day, skill type, shift type> triple. Therefore only the *coverage constraints* that are defined explicitly are taken into account by the *tightness ratio* measure. Second, the *tightness ratio* is calculated using the *minimum number of required working hours* instead of the *number of required working shifts*, because the availability of an *employee* is mostly given as a *number of working hours per week* in his or her *contract* and because the *coverage constraints* are modelled with a threshold instead of an exact value. Another difference is the exclusion of the *locked days* of *employees* from the availabilities, similar to the practice in *MNRFA*. These extensions make the current model applicable to the real world problems whose requirements are different then the problem instances created in laboratory conditions.

5.4 Conclusions

Being able to cope with a variety of problem instances is one of the real world requirements that has rarely been addressed in scientific literature on personnel rostering. According to the *no free lunch* theorem, a single algorithm configuration cannot be expected to address a wide range of problem instances. Several algorithm configurations are necessary in order to increase the probability of obtaining good enough solutions on the targeted range of problem instances. In addition to that, an efficient and automated decision mechanism to choose between the algorithm configurations to solve a given problem instance is needed. Such a decision mechanism can be realised as a function between problem properties and the most promising configuration of an algorithmic toolbox.

Various hardness measures have been introduced in the scientific literature. However, several situations that arise in the real world practice have not been addressed by these measures. *Schedule locks* are used frequently in the real world practice. They reduce the size of the problem instance that an algorithm has to deal with. *Schedule locks* have also a direct impact on the hardness of a problem instance by preventing the algorithm to operate on certain parts of a *schedule*, which might be restricted by a number of constraints. The factors of the size of a problem instance, i.e., the number of *employees*, *days* and *shift types*, have been proposed as measures of a problem instance in the scientific literature. However, these factors are not an accurate measure of the overall size of a problem instance by themselves.

A set of fine-grained quantitative measures have been introduced in the present chapter. They take into account the real world requirements, such as the *schedule locks*, that have not been addressed by the hardness measures proposed in the scientific literature. The quantitative measures introduced in this chapter involve

compound calculations to determine the overall size of a problem instance, instead of relying on the number *employees*, *days* and *shift types*. These calculations take into account the total number of assignments to be made by the algorithm, *schedule locks* and situations where the workforce demand exceeds the supply, which occurs occasionally in the real world practice.

The quantitative measures introduced in the present chapter will be utilised in Chapter 6 to state the size and tightness of various nurse rostering benchmarks. The experimental results of the applications of the solution methods (Chapter 7) to the benchmarks (Chapter 6) will be discussed in Chapter 8. As a result, a relation between the algorithm performance and quantitative measure values will be derived and reported in Chapter 8. The derived relation is deployed in the automated personnel rostering software of our industrial partners to choose the most promising configuration of the algorithmic toolbox for a given problem instance.

Chapter 6

Nurse rostering benchmarks

Benchmarks are a collection of problem instances from a specific scientific field. Mainly, they are used to measure and compare the relative performance of the solution methods. Furthermore, they set standards in the problem model and constraint evaluation.

In order to serve their intended purposes, benchmarks have to fulfill a number of requirements. They must be a comprehensive representation of their field. That entails including instances with a wide range of problem properties. A solution method that performs well on such a benchmark set can be expected to perform well on a great variety of instances from the corresponding field.

Benchmarks have two main sources. They are either created in laboratory conditions under controlled design or they are collected from real world resources. There are benchmark sets from both sources in the nurse rostering literature. Vanhoucke and Maenhout [68] have introduced NSPLib, a nurse rostering benchmark set created in laboratory conditions under controlled design. Burke et al. [20] have introduced the Nottingham benchmarks, another nurse rostering benchmark set, derived from real world resources in Belgium, Canada, France, Greece, Hong Kong, Italy, Japan, the Netherlands, Norway, Saudi Arabia, Spain, the United Kingdom and the United States. The Nottingham benchmarks have been covered in greater detail in Chapter 2.

The differences between the scientific literature and real world practice have been discussed in the chapters about the problem model and constraint evaluation, Chapters 3 and 4, respectively. These differences have led to the extensions of the problem model and to a modified set of constraint evaluation rules. Consequently, a new benchmark set to supplement the extended problem model and modified constraint evaluation methods is necessary. Such a benchmark set, the KAHO

benchmarks, are presented in this chapter. There are significant differences in the structure and constraint evaluation methods of the KAHO benchmarks and the nurse rostering benchmarks in the literature.

The problem model and solution methods presented in this dissertation have been implemented and utilised in the decision support software provided by our industrial partners[1]. That has enabled the end users of these software, i.e., the planners, to enter their personnel rostering problems into the system using the graphical user interface components such as forms and dialogs. In turn, one of our industrial partners[2] has agreed to offer the problem data to us for academic purposes. This problem set, collected from six wards in two Belgian hospitals, has been named as the KAHO benchmarks. The intention of this practice is to test the capability of the problem models and solution methods to address the real world challenges that our industrial partners face on a daily basis.

6.1 The KAHO benchmarks

The problem properties and quantitative measures of the KAHO benchmarks (Tables 6.1 - 6.3) indicate that these problem instances are a diverse set from the personnel rostering problem domain. The *tightness*, *minimum number of required and free assignments* (MNRFA) and *schedule period* length values of the KAHO and Nottingham benchmark instances are presented in Table 6.1. The *tightness* value is the ratio of the *availability of the employees* to the *required assignments*. If the *tightness* value of a problem instance exceeds 1.0, then the availability of the *employees* cannot address the required *assignments* completely. The MNRFA value is an indicator of the problem size. The greater the MNRFA value, the greater is the number of *shifts* that need to be assigned by the algorithm. The *tightness* and MNRFA measures have been explained in greater detail in Chapter 5.

The *schedule period* length of the KAHO benchmarks varies between four and 13 weeks. The same value varies between one week and 29 days in the Nottingham benchmarks. The KAHO benchmarks can be grouped into three sets according to their MNRFA value, i.e., small, middle and large size instances. Although the Nottingham benchmarks have problem instances with small and middle sizes, a large size problem instance with an MNRFA value higher than 534 is not present in the Nottingham benchmarks. The KAHO benchmarks have been divided into two groups according to their *tightness* values, i.e., normal and high tightness. The Nottingham benchmarks do not involve any high tightness instances. The ratio between the workforce demand and supply does not exceed 1 in the Nottingham benchmarks. However, the data collected from our industrial partner points to

[1]SAGA Consulting, GPS NV Belgium
[2]SAGA Consulting

the fact that it can exceed 1, in the real world practice. In total, the KAHO benchmarks have been divided into four groups in order to find the most promising configuration of the algorithmic toolbox for each group.

As can be seen in Table 6.2, high numbers of *shift types* are common in Belgian hospitals. In some of the wards, the sum of the number of *employees* in each *skill* category exceeds the total number of *employees* in the ward (Table 6.2). In this case, there are *employees* with *multiple skill types* in that ward. The *skill* structure has a direct impact on the *employee*, *assignment* and *coverage constraint* elements of the problem model (Chapter 3). The *skill* structure has also been used as a hardness measure while deciding on the algorithmic configuration. Neighbourhoods and low level heuristics that operate on *multiple skill types* have been used in the solution method if the problem instance involves *employees* that have more than one *skill type* (Section 7.6.1).

The flexibility of the *employment contracts* in Belgian hospitals can be seen in Table 6.3. *Nurses* can opt for different working hours: full time, half time and various ratios in between (Table 6.3). Real world benchmarks require the ability to deal with *multiple serial contracts* per *employee* per *schedule period*. This happens when an *employee* switches from one *contract type* to another within the same *schedule period*. *Multiple serial contracts* are encountered in the *meal preparation* and *geriatrics* wards of the KAHO benchmarks. The sum of the number of nurses for each contract in the *meal preparation* and *geriatrics* wards (Table 6.3) are greater than the total number of nurses in those wards (Table 6.2), because of the nurses who have more than one contract.

For each ward in the KAHO benchmarks, there are three different scenarios: *normal*, *overload* and *absence*. These scenarios stem from situations that arise in the real world. The *normal* scenario represents the personnel rostering problem faced by the wards regularly.

In the real world practice, the workload of a ward is not stable over different *schedule periods* and varies depending on the external events such as epidemics and seasonal diseases. A significant increase in the workload must be matched by a proportional increase in the workforce demand in the related problem instance. The workforce demand has been modelled by the *coverage constraints* element in the problem model (Section 3.2.3). A higher workforce demand is represented with higher *threshold* values in the *coverage constraints*. Higher *coverage constraints* result in larger problems and higher tightness values. The *overload* scenario instances represent the cases where the workload demand is higher than usual. The *overload* scenario instances have been constructed by incrementing the *threshold* values of the *coverage constraints* of the *normal* scenario instances, with a single exception. The *threshold* values of the *coverage constraints* for the head nurse has not been incremented, because no more than one head nurse is needed in any situation.

In the real world practice, it is possible that a scheduled *employee* cannot work

her planned shifts due to an unforeseen event such as sickness. In such cases, the complete roster needs to be rescheduled to take into account the unforeseen absence of the *employee*. Several factors need to be taken into account when addressing this problem. In contrast to the *normal* and *overload* scenarios, the input roster is already published in the ward in this case. That means the *employees* have already adjusted their private plans according to it. Therefore the changes to the schedule must be kept at a minimum level. In order to ensure that, only the changed part, i.e., the days when the *employee* is scheduled but cannot be present, should be rescheduled. The remainder of the schedule needs to be locked using *schedule locks*. This practice reduces the size of the problem instance considerably. The *absence* scenario instances represent the case, where one week of a complete roster needs to be rescheduled due to the unforeseen absence of an *employee*.

The XSD for the model, the input data, a sample output and a penalty report for each problem instance in the KAHO benchmarks are available online[3].

6.2 Conclusions

The extensions to the standard problem models and the modifications to the constraint evaluation methods have made a new set of benchmark instances necessary, because the benchmarks in the scientific literature do not cover the extensions and modifications introduced in Chapters 3 and 4. The problem model, constraint evaluation methods and solution methods have been developed to satisfy the real world requirements our industrial partners had to deal with. Consequently, a real world benchmark set has been provided by one of our industrial partners[4] to test the capabilities of the proposed models and methods.

The real world benchmark set, the KAHO benchmarks, represent the challenges of the personnel rostering problem in the Belgian healthcare sector. An analysis of the problem properties reveals the complexity and the diversity of the KAHO benchmarks. It can be observed in Table 6.1 that the size (MNRFA) and tightness values of the KAHO benchmarks cover a broader range of values than the Nottingham benchmarks. The KAHO benchmarks include problems with various sizes. The ratio between the workforce supply and demand varies among the instances of the KAHO benchmarks. The problem instances expose numerous challenges encountered in the real world practice. Individual contracts, multiple skill types, multiple serial contracts in a schedule period and variable workforce demand are a few of the real world challenges that can be observed in the KAHO benchmarks. These elements are examples of how the extended problem model in Chapter 3 can be used to represent real world problems.

[3] http://ingenieur.kahosl.be/vakgroep/it/nurse/archive.htm
[4] SAGA Consulting

Problem size	Problem instance	Tightness	MNRFA	Period
Small size	Geriatrics Absence	0.77	40	4 weeks
	Reception Absence	0.69	45	6 weeks
	Psychiatry Absence	0.97	56	1 month
	Meal Preparation Absence	1.12	63	1 month
	Palliative Care Absence	1.69	64	13 weeks
	Emergency Absence	0.71	107	4 weeks
Midsize	Emergency Normal	0.66	429	4 weeks
	Emergency Overload	0.72	513	4 weeks
	Geriatrics Normal	0.75	160	4 weeks
	Geriatrics Overload	0.89	200	4 weeks
	Psychiatry Normal	0.92	241	1 month
	Psychiatry Overload	1.00	300	1 month
	Meal Preparation Normal	1.11	263	1 month
	Meal Preparation Overload	1.19	305	1 month
Midsize-high tightness	Reception Normal	1.35	234	6 weeks
	Reception Overload	1.66	327	6 weeks
Large size	Palliative Care Normal	0.71	846	13 weeks
	Palliative Care Overload	0.77	1031	13 weeks
Nottingham Instances-Midsize	BCV-3.46.2	0.78	534	26 days
	BCV-4.13.1	0.85	174	29 days
	SINTEF	0.92	198	3 weeks
	Valouxis-1	0.78	264	4 weeks
Nottingham Instances-Small size	GPost	1	112	4 weeks
	Millar1	1	56	2 weeks
	Millar1.1	1	56	2 weeks
	LLR	0.79	105	1 week
	Ozkarahan	1	47	1 week

Table 6.1: Tightness and MNRFA values for public problem instances

Ward	Shift Types	Skill 1	Skill 2	Skill 3	Skill 4	Total Employees
Emergency	27	1	15	4	26	27
Psychiatry	14	1	17	1	-	19
Reception	19	1	1	3	15	19
Meal P.	9	1	31	-	-	32
Geriatrics	9	4	20	-	-	21
P. Care	23	1	21	4	1	27

Table 6.2: Number of shift types and number of nurses with each skill type

Ward	38 hours (100%)	34.2 hours (90%)	30.4 hours (80%)	28.5 hours (75%)	22.8 hours (60%)	19 hours (50%)
Emergency	24	-	-	3	-	-
Psychiatry	13	-	-	2	-	4
Reception	5	-	-	7	-	7
Meal P.	3	2	-	1	-	28
Geriatrics	9	-	-	9	1	3
P. Care	13	-	2	4	1	7

Table 6.3: The weekly job time and the corresponding number of nurses

An input file and a sample output file are provided online for each of the instances to facilitate further research[5]. Furthermore, an XML Scheme Description (XSD) file has been provided online to unambiguously define the model structure. The KAHO benchmarks have been used in the experiments to determine the most promising configuration of the algorithmic toolbox for a given problem instance whose quantitative measure values are within a particular range (Chapter 8). In addition to the KAHO benchmarks, results of the experiments on ten instances from the Nottingham benchmarks have also been reported in Chapter 8.

[5]http://ingenieur.kahosl.be/vakgroep/it/nurse/archive.htm

Chapter 7

Solution methods

According to the literature surveys on personnel rostering, the solution methods presented in the scientific literature have not been broadly utilised in the real world practice (Chapter 2). Several reasons have been observed for the limited utilisation. The academic efforts have been focused on providing optimal solutions to a limited range of often simplified problem instances. The real world practitioners, on the other hand, demand good enough solutions for a great variety of problem instances within limited execution times. In addition, the problem instances in the real world evolve over time with new labour legislations, union demands and organisational changes. Consequently, the solution methods must be flexible enough to address the changes in the problem definitions without requiring significantly more human intervention and computation time. A generic problem model has been introduced in Chapter 3 to represent a great variety of problem instances. Experience in collaborating with software companies and discussions with practitioners has revealed that the presented model is rich enough for today's problems in Belgian hospitals and rest homes. The proposed problem model can be easily extended with new elements in case the problem instances evolve to a point that cannot be addressed with the current version of the model.

The real world practitioners demand to have a certain control over the solution method. They do not want a system that decides for them. They want a system that supports their decision process. Therefore, the solution methods presented in this dissertation have been developed to be utilised in decision support systems. They are required to provide satisfying solutions in different settings and time frames, and to be adaptable to the customer's ever changing preferences.

The execution time of the algorithm is required to be a parameter set by the end user. The real world planners expect a result mostly within one to ten minutes. This allows them to check the solutions, make adjustments based on their personal

judgement and rerun the algorithm if necessary. The planners need solutions quickly when a roster has to be rescheduled, for example because an *employee* cannot work his or her *assignment* due to an unexpected absence in case of sickness. The end users expect a complete solution no matter how long the execution takes. A solution is considered to be complete if all the required *shifts* have been assigned to *employees* to the extent allowed by the hard constraints. If that is not possible however, which happens occasionally in the real world practice, then the shortages are reported after the execution of the solution method.

The success of the algorithm depends to a large extent on the composition of the objective function. In Chapter 4, the objective function in the present approach has been defined as a linear combination of soft constraint violations. The weights of the soft constraints indicate the relative importance of the soft constraints to each other. Setting these weights in such a way that they reflect the actual importance of the constraints and at the same time effectively guide the algorithm towards good quality solutions is a tedious task. The users might need to run the algorithm several times until they reach a weight configuration that results in a satisfying roster. This is another motivation for the end users to favour shorter execution times.

Several factors have discouraged the utilisation of exact methods as solution methods for the problem at hand. The academic context in Chapter 2 provides a set of references to the proofs that rostering problems are NP-Hard. Furthermore, exact methods proposed in the scientific literature require the incorporation of instance specific information to perform well. This contradicts with the requirement to perform good enough on a broad range of problem instances, which is what practical systems require. Even with the problem specific implementations, the execution times of the exact mathematical optimisation methods are above the expectations of the real world practitioners. Complementary to that, the real world practitioners do not demand optimal solutions.

As an alternative to the exact methods, heuristic approaches have been chosen as the solution method for a number of reasons. In this dissertation, this category of solution methods have been concentrated on. The main structure of the solution method consists of two stages, a random initialisation method succeeded by an iterative improvement method. The random initialisation method assigns the *shifts* that are required by the *coverage constraints* to the *employees* randomly. This way, a complete roster is constructed in the shortest execution time possible. The resulting roster is improved by an iterative improvement method. Hard constraints are respected throughout the execution of the solution method so that the candidate solution is always feasible.

An algorithmic toolbox has been composed by the author of the present dissertation to be deployed as the iterative improvement method. The algorithmic toolbox consists of variable neighbourhood search (VNS), adaptive large neighbourhood

search (ALNS), hyperheuristics, six neighbourhoods to be utilised in VNS and ALNS and the low level heuristic variants of these neighbourhoods to be utilised in hyperheuristics. VNS, ALNS and hyperheuristics have been adopted from the scientific literature. These algorithms modify the roster in each iteration in order to improve its quality for a given execution time. This practice addresses the requirement of having the execution time as a parameter and providing a good quality solution relative to the execution time.

In order to explore the search space, VNS and ALNS utilise a set of neighbourhoods. Hyperheuristics use a set of low level heuristics for the same purpose. Each point in the search space corresponds to a solution, a roster in the present case, with a specific quality. The quality of a solution is adversely proportional to its objective function value. The search algorithms return the best solution they have found within the execution time they have been given.

Contrary to VNS, ALNS and hyperheuristics, which are independent from the problem definition, the neighbourhoods and the low level heuristics are based on the problem model. Most of the time, the neighbourhoods and the low level heuristics are inspired by typical manual modifications made by the human planners. They can be as simple as assigning or deleting a *shift* or more complicated, such as changing the *assignment* of an *employee* so that he has to work using a different *skill type*. Six neighbourhoods are utilised in the algorithmic toolbox. Three of these neighbourhoods have been adopted from the scientific literature: *assign shift*, *delete shift* and *single shift day*. The other three neighbourhoods have been developed by the author of the present dissertation: *change assignment shift*, *change assignment shift based on shift type set* and *change assignment skill*. Although the neighbourhoods utilised by the automated rostering algorithms are similar to the modifications implicitly applied by human planners, the automated algorithms outperform the human planners by leveraging the computation power of modern PCs and the algorithmic capabilities of VNS, ALNS and hyperheuristics.

Another advantage of using VNS, ALNS and hyperheuristics is their adaptability. These algorithms are problem independent. In order to apply VNS and ALNS to a problem, a problem model and several neighbourhoods should be implemented. The application of the hyperheuristics to a problem requires the implementation of a problem model and a set of low level heuristics. The interaction between VNS, ALNS and the neighbourhoods are carried out with a well-defined interface. A similar well-defined interface provides a structure for the interaction between the hyperheuristics and the low level heuristics. VNS and ALNS can interact with any neighbourhood that implements the aforementioned interface properly. Similarly, the hyperheuristics can interact with low level heuristics that implement the corresponding interface. This approach simplifies the maintenance of VNS, ALNS and the hyperheuristics.

As mentioned before, it is possible that the problem instances encountered in the

real world can evolve to a point, where the current problem model fails. In this case, the problem model can be modified without adjusting the hyperheuristic, VNS or ALNS implementation, because their implementations are independent from the problem model. Complementary to that, the solution methods can also be extended with new neighbourhoods and low level heuristics, to address the extensions in the problem model. Such extensions do not require significant changes in the solution method. The new neighbourhoods and low level heuristics can easily be plugged in to the system via their implementations of the corresponding interfaces.

According to the well-known *no free lunch theorem* [71], an algorithm configuration cannot be expected to perform well on every problem instance encountered. The solution method introduced in the present dissertation has been developed to tackle the real world personnel rostering problem. Real world personnel rostering problem instances, no matter how diverse, are still a subset of all the possible personnel rostering problem instances. The probability of finding good enough solutions in a subset of a problem domain can be increased by deploying various algorithm configurations. This option has been explored in the present dissertation with an algorithmic toolbox that allows for numerous configurations.

In the real world practice, all configurations of the algorithmic toolbox cannot be experimented with every time a problem instance has to be solved. Therefore, a decision method is needed to select the most promising configuration of the toolbox for a given problem instance. The algorithmic toolbox presented in this dissertation has been developed in a way that a decision support system can compose a fully functional solution method out of the toolbox and parametrise it at run time without any intervention by a human expert. In order to allow that, the set of *quantitative measures* proposed in Chapter 5, has been utilised. Based on the experimental results in Chapter 8, a decision support system can automatically select the most promising configuration of the algorithmic toolbox for a given problem instance. This fact broadens the application domain of the algorithmic toolbox presented in the current chapter.

The problem model introduced in Chapter 3 allows for multiple *skill types* per *employee*. Therefore, the *schedules* of the *employees* with different *skill types* are connected with each other via the *schedules* of the *employees* that have more than one *skill type*. This fact is exploited in the neighbourhoods and the low level heuristics in order to produce better schedules. Therefore, *employees* with different *skill types* are not scheduled separately as in [25].

The neighbourhoods and low level heuristics take into account the hard constraints and do not make any modifications that result in an infeasible *schedule*. That way the execution of the algorithm can be stopped at any given time and a feasible *schedule* can be obtained as a result. The termination criterion is the maximum execution time of the algorithm, without taking the random initialisation step into account. That means the end users can specify the execution time of the algorithm

and the algorithm will stop running when the specified execution time passes.

The remainder of the chapter is organised as the following. The random initialisation method is introduced in Section 7.1. The hyperheuristic approach within the algorithmic toolbox is discussed in Section 7.2. The utilisation of VNS and ALNS are presented in Sections 7.4 and 7.5, respectively. The details of the tabu list used in the VNS and ALNS approaches are discussed in Section 7.3. The neighbourhoods and the low level heuristics are discussed in Section 7.6. The chapter is concluded in Section 7.7.

7.1 Initialisation Method

The solution method accepts an input *schedule* from the user. The input *schedule* can be an empty, a partial or a complete *schedule*. The initialisation method tries to fulfill the minimum *coverage constraints* randomly until either all the minimum *coverage constraints* are satisfied or no more feasible *assignment* can be made. A roster that satisfies the minimum *coverage constraints* is perceived to be a complete roster by the planners. Therefore, the term *complete roster* will refer to a roster that satisfies the minimum *coverage constraints* to the extent possible by the hard constraints. The aim of the initialisation step is to provide a complete roster in the shortest execution time possible. The roster can be improved iteratively in the following steps of the search algorithm. This way the planner will receive a complete roster no matter how short the execution time is. The execution time will only influence the quality of the roster.

The pseudocode of the $RandomInitialisation(C)$ method is presented in Algorithm 9. In this algorithm, C_0 refers to the initial candidate solution received from the user. The initial candidate solution can be an empty, partial or a complete schedule. CC refers to the set of all *coverage constraints*. $cc_{d,S',K'}$ refers to the *coverage constraint* that is defined on day $d \in D$, shift type set $S' \subset S$ and skill type set $K' \subset K$. The $RandomPermutation(X)$ method returns a random sequence of the elements of the set X, so that the elements of X can be processed in a random order.

7.2 Hyperheuristic approach

A theoretical discussion of the hyperheuristics approach has been presented in the academic context (Section 2.6.4). Hyperheuristics are three layer combinatorial optimisation methods. The problem model and objective function reside on the lowest layer. Low level heuristics that operate directly on the problem model and

objective function are on the middle layer. A heuristic is positioned on the top layer and guides the low level heuristics throughout the search.

Algorithm 9 *RandomInitialisation*(C)

```
C_0 = input candidate solution
C = C_0
CC = the set of all coverage constraints
CC' = RandomPermutation(CC)
for all cc_{d,S',K'} ∈ CC' do
  S'' = RandomPermutation(S')
  K'' = RandomPermutation(K')
  E' = RandomPermutation(E)
  for all e ∈ E' do
    if ¬(e.IsAssigned(d) ∨ e.IsLocked(d)) then
      for all k ∈ K'' do
        if e.HasSkill(k) then
          for all s ∈ S'' do
            if ¬(cc_{d,S',K'}.IsMinFulfilled() ∨ e.IsOverlapping(d, k)) then
              C.Assign(e, d, s, k)
            end if
          end for
        end if
      end for
    end if
  end for
end for
return C
```

The hyperheuristic approach used in this work can be classified as a selection hyperheuristic according to the terminology in [21]. Several hyperheuristic approaches have been composed and experimented with for generating the results of the present dissertation. *Simple random* and *choice function* have been used as selection methods. Three approaches have been utilised as acceptance criteria, *improving and equal moves accepted*, *simulated annealing* and *great deluge*. The cooling schemes, *linear*, *quadratic* and *quartic*, have been applied in both *simulated annealing* and *great deluge*. The low level heuristics utilised in the hyperheuristics are discussed in Section 7.6.

7.3 Tabu Search

The theoretical background of tabu search has been reviewed in the academic context (Section 2.6.1). Tabu search is an iterative improvement method to address combinatorial optimisation problems. It carries out modifications on a single candidate solution at each iteration in order to improve it. The executed modifications, called moves, are kept in a short term memory, i.e., the tabu list, in order to avoid repeating them. Although tabu search is not directly utilised in the solution methods of the present dissertation, the concept of the tabu list is utilised within the VNS and ALNS algorithms. A tabu list is a short term memory that keeps track of the moves that are recently executed during the search. The moves that are in the tabu list are not repeated as long as they remain in the tabu list. The only exception of this practice is when a move results in an overall best solution. This practice is referred to as the aspiration criterion in the scientific literature. The function of the tabu list is to avoid cycles of the algorithm around local optima. The parameters of the executed moves are kept in the tabu list in a hashed way. That way instead of comparing a set of parameters each time a move is evaluated, only a hash value is compared with other hash values in the tabu list. This practice increases the efficiency of the tabu list and contributes to the overall performance of the tabu search.

The parameters of the executed moves in the solution methods presented in this dissertation are <employee, day, shift type, skill type>. They are hashed using the perfect hash function in Equation 7.1. D, S and K refer to the set of *days*, *shifts* and *skills* in the problem instance. The notation $|X|$ refers to the total number of elements in the set X. Each *employee*, *day*, *shift* and *skill* is assigned an unique index starting with 0. The indices e, d, s and k refer to the indices of the *employee*, *day*, *shift* and *skill* in the problem.

$$H(e, d, s, k) = ((e|D| + d)\,|S| + s)\,|K| + k \tag{7.1}$$

The length of the tabu list, often referred as the tabu tenure in the scientific literature, varies throughout the execution of the algorithm. Prime numbers are used as values for the tabu tenures, in order to avoid hash collisions and cycling. The tabu tenure is increased to the next prime number at each non-improving iteration and decreased to the previous prime number if there is an improvement.

Decreasing the tabu tenure has an intensifying effect on the search. The search process is focused in the area close to the candidate solution. Therefore, the tabu tenure is only decreased if the vicinity of the candidate solution is promising. The vicinity of the candidate solution is considered to be promising if there are better

candidate solutions in its immediate neighbourhood. Increasing the tabu tenure has a diversifying effect on the search. The search process is moved away from the vicinity of the candidate solution. Therefore, the tabu tenure is increased if the vicinity of the candidate solution is not promising. The vicinity of the candidate solution is considered to be not promising if there is not any better candidate solution in its immediate neighbourhood.

The tabu tenure is varied between a lower and an upper bound. This is to avoid that the tabu tenure converges to 0 rendering the tabu list useless. It also prevents that the tabu tenure is increased to a very high value after a series of non-improving moves. In the latter case, there is a risk that the tabu list will block the execution of a high number of moves, which will result in decreased performance of the tabu search. If the tabu tenure converges to the lower bound, then it is not decreased anymore, even if the algorithm keeps finding improving moves. Similarly, if the tabu tenure converges to the upper bound, then it is not increased anymore, no matter how many non-improving moves follow.

The lower bound is equal to 7 and the upper bound is a parameter of the algorithm. Based on the preliminary experiments, two values, 97 and 199, have been decided to be experimented with as the upper limit. The aspiration criterion holds so that tabu moves that result in overall best candidate solutions are allowed to be executed. Algorithms 10 and 11 present the rules for increasing and decreasing the tabu tenure, respectively. The application of the tabu list within VNS and ALNS are presented in Algorithms 12 and 13, respectively.

Algorithm 10 *IncreaseTabuTenure*(T)

U = *upper limit of the tabu tenure*
T {tabu tenure}
if $T < U$ **then**
  T = *the smallest prime number greater than* T
end if

Algorithm 11 *DecreaseTabuTenure*(T)

T {tabu tenure}
if $T > 7$ **then**
  T = *the greatest prime number less than* T
end if

7.4 Variable neighbourhood search

The theory of the VNS algorithm has been reviewed in the academic context (Section 2.6.2). The VNS algorithm used in the solution methods of the present dissertation involves a tabu list as described in Section 7.3.

The VNS algorithm utilises several neighbourhoods and holds the parameters of the executed moves in a tabu list. The utilised neighbourhoods share the same tabu list, so that they do not reverse the recent moves made by each other. At each iteration of the algorithm, a single neighbourhood is searched.

A move in the neighbourhood is considered for execution only if it satisfies all the hard constraints. Consequently, the *schedule* remains feasible throughout the execution of the algorithm. In addition to that, the move must not be tabu or it must result in an overall best solution. This practice is referred to as the aspiration criterion in the scientific literature. The best move in the neighbourhood is executed given that it satisfies these conditions.

The token-ring approach is used to switch between the neighbourhoods in the VNS algorithm. In this approach, the neighbourhoods are held in a circular queue that determines their application sequence [32]. If the applied neighbourhood does not result in an improving move, the algorithm switches to the next neighbourhood in the queue. The pseudocode of the VNS variant deployed in the solution method is given in Algorithm 12. The neighbourhoods utilised in VNS are discussed in Section 7.6.

The VNS algorithm utilised in the solution methods of the present dissertation is an adaptation of the existing methods from the scientific literature. The contribution of the present dissertation lies in the utilisation of the VNS in an algorithmic toolbox to address the personnel rostering problem and combining it with the extended problem model and with the existing and novel neighbourhoods that are introduced to the literature in the present dissertation.

7.5 Adaptive large neighbourhood search

The theoretical background of the ALNS method has been discussed in the academic context (Section 2.6.3). A set of neighbourhoods are deployed in ALNS [62], similar to VNS. At each iteration, a neighbourhood is selected stochastically. Each neighbourhood has a score that is used as its probability of selection. The scores of the neighbourhoods are updated regularly based on their performance in recent iterations.

Algorithm 12 The solution method utilising VNS

```
C_0 = input schedule
T = 7 {tabu tenure}
CQ = circular queue of the neighbourhoods
F(C) = objective function
C = RandomInitialisation(C_0)
{Neighbourhood to be explored in the current iteration}
N = first neighbourhood from CQ
BC = C {Variable to hold the best solution throughout the search}
while Termination criterion not met do
  C' = N(C)
  if F(C') < F(BC) then
    DecreaseTabuTenure(T)
    BC = C'
  else
    IncreaseTabuTenure(T)
    if F(C') ≥ F(C) then
      N = next neighbourhood in CQ
    end if
  end if
  C = C'
end while
return BC
```

The ALNS algorithm used in the solution method deploys the same set of neighbourhoods and the same tabu list approach as the VNS algorithm. The only difference between both algorithms is the way they select the next neighbourhood when they switch between neighbourhoods. The VNS algorithm uses a circular queue to switch between neighbourhoods, a practice referred as token-ring search in the scientific literature. ALNS uses a roulette-wheel selection variant. In the roulette-wheel method, each item in a set has a specific probability. The selection method chooses an item randomly based on these probabilities. The probabilities of the items are subject to change according to their performance or quality.

The pseudocode of the ALNS algorithm is presented in Algorithm 13. In this algorithm, SN refers to the set of all neighbourhoods, $P(N)$ to the roulette wheel probability of the neighbourhood N, $Q(N)$ to the temporary number of times N is called, $R(N)$ to the score of the neighbourhood N, $R'(N)$ to the temporary score of the neighbourhood N, and $F(C)$ to the objective function of the problem, respectively.

Algorithm 13 The solution method utilising ALNS

$C_0 = input\ schedule$
$T = 7$ {tabu tenure}
$\rho = 0.8$
$\alpha = 10$ {best solution found reward}
$\beta = 10$ {current solution improvement reward}
$\gamma = 1$ {solution found reward}
for all $N \in SN$ **do**
  $P(N) = 1/|SN|, Q(N) = 0, R(N) = 0, R'(N) = 0$
end for
$C = RandomInitialisation(C_0)$
$BC = C$ {variable to hold the best solution throughout the search}
$I = 0$ {temporary iteration counter}
while $Termination\ criterion\ not\ met$ **do**
  {Neighbourhood to be explored in the current iteration}
  $N = RouletteWheelSelect(SN, P)$
  $Q(N) = Q(N) + 1$
  $C' = N(C)$
  if $IsFeasible(C')$ **then**
    $R'(N) = R'(N) + \gamma$
    if $F(C') < F(BC)$ **then**
      $DecreaseTabuTenure(T)$
      $BC = C'$
      $R'(N) = R'(N) + \alpha$
    else
      $IncreaseTabuTenure(T)$
      if $F(C') < F(C)$ **then**
        $R'(N) = R'(N) + \beta$
      end if
    end if
    $C = C'$
  else
    $IncreaseTabuTenure(T)$
  end if
  $I = I + 1$
  if $I == 100$ **then**
    $I = 0$
    $UpdateScores(SN, P(N), Q(N), R(N), R'(N))$
  end if
end while
return BC

Algorithm 14 $UpdateScores(SN, P(N), Q(N), R(N), R'(N))$

```
Z = 0
for all N ∈ SN do
  R(N) = (1 − ρ) ∗ R(N)
  if Q(N) > 0 then
    R(N) = R(N) + ρ ∗ R'(N)/Q(N)
  end if
  Z = Z + R(N)
end for
for all N ∈ SN do
  P(N) = R(N)/Z
end for
```

7.6 Neighbourhoods and heuristics

VNS, ALNS and hyperheuristics are problem independent solution methods. They require neighbourhoods and low level heuristics to work on particular problems. The problem specific neighbourhoods searched by the VNS and ALNS are *assign shift*, *delete shift*, *single shift day*, *change assignment shift*, *change assignment shift based on shift type set* and *change assignment skill* (Section 7.6.1). The *assign shift*, *delete shift* and *single shift day* neighbourhoods have been adapted from the scientific literature [25]. The *change assignment shift*, *change assignment shift based on shift type set* and *change assignment skill* neighbourhoods have been developed by the author of the present dissertation. The latter three neighbourhoods have been developed considering the extended problem model (Chapter 3) and the modifications that the end users do to improve the rosters. The low level heuristics used by the hyperheuristics are based on the same neighbourhoods using a *tournament strategy* (Section 7.6.2).

7.6.1 Neighbourhoods

Only feasible moves, i.e., moves that satisfy all the hard constraints, are considered when searching the neighbourhoods. The tabu list is shared among all the neighbourhoods. This way, it is avoided that two neighbourhoods reverse the moves made by each other, which would result in a cycle. The parameters of the recently executed moves are kept in the tabu list. These parameters are the *employee*, *day*, *shift type* and *skill type*. They are held in the tabu list in a hashed way. The details of the hash function and the tabu list have been discussed in Section 7.3.

Several sets and methods are common among the neighbourhoods. D refers to the

set of all days in the schedule period, S to the set of all shift types, K to the set of all skill types and E to the set of all employees, respectively.

$C.Assign(e, d, s, k)$ method assigns the shift type s and skill type k on day d to the employee e in the current state of the candidate solution C. Similarly, $C.Delete(e, d, s, k)$ removes the assignment with the shift type s and skill type k on day d from the schedule of the employee e in the current state of the candidate solution C.

$IsTabu(e, d, s, k)$ method returns *true* if the move parameters employee e, day d, shift type s and skill type k are in the tabu list. It returns false otherwise. $IsDefined(d, s, k)$ method returns *true* if there is at least one *coverage constraint* with the parameters day d, shift type s and skill type k. It returns false otherwise. $e.HasSkill(k)$ returns *true* if the employee e has the skill type k, and false otherwise. $e.IsOverlapping(d, s)$ returns *true* if the assignment of the shift type s on day d overlaps with any assignment in the schedule of the employee e. It returns false otherwise. $e.IsAssigned(d)$ returns *true* if the employee e is assigned on day d, and false otherwise. $e.IsLocked(d)$ returns *true* if the schedule of the employee e is locked on day d, and false otherwise.

Assign shift

An *assignment* is defined as a quadruple of <employee, day, shift type, skill type> (Section 3.2.4). The *assign shift* neighbourhood is constructed using these quadruples. When an *assignment* is made, not only a *shift type* is assigned, but also the associated *skill type* for that *assignment*.

The number of shift types in the scientific literature have mostly been limited to three, early, late and night shifts. Contrary to the scientific literature, Table 6.2 in Section 6.1 about the KAHO benchmarks shows that the number of *shift types* are high in Belgian hospitals. This fact results in a large *assign shift* neighbourhood. Evaluating the *assign shift* neighbourhood of the candidate solution completely in each iteration consumes long spans of CPU time and results in inefficiencies of the algorithm. Therefore, the *assign shift* neighbourhood has been adapted so that it will perform efficiently on the real world instances. The adaptation is an exception to the steepest descent approach of exploring the *assign shift neighbourhood*. Only one random *shift type* for each <employee, day, skill type> triple is evaluated when searching the *assign shift* neighbourhood. This is the only exception to the steepest descent manner of searching the *assign shift* neighbourhood. A move of the *assign shift* neighbourhood is encoded as a hash value of the <employee, day, shift type, skill type> quadruple and inserted into the tabu list.

Algorithm 15 The assign shift neighbourhood

```
C := candidate solution
BC := best candidate solution found so far
F(C) := objective function
BC' := null
BF := ∞
for all d ∈ D do
  for all k ∈ K do
    for all e ∈ E do
      if ¬(e.IsAssigned(d) ∨ e.IsLocked(d)) ∧ e.HasSkill(k) then
        S' = RandomPermutation(S)
        b = true
        for all s ∈ S' do
          if b then
            if IsDefined(d, s, k) ∧ ¬e.IsOverlapping(d, s) then
              C' = C.Assign(e, d, s, k)
              if F(C') < F(BC) ∨ ¬IsTabu(e, d, s, k) then
                b = false
                if F(C') < BF then
                  BF = F(C')
                  BC' = C'
                end if
              end if
            end if
          end if
        end for
      end if
    end for
  end for
end for
return BC'
```

Delete shift

The deletion of an *assignment* is feasible only if two conditions are met. First, the *assignment* <day, shift type, skill type> must correspond to a *coverage constraint*, i.e., a coverage must be required for the *day*, *shift type* and *skill type* (see Section 4.2.4). In the real world practice, if an *employee* is assigned a *shift type* from another ward of the institution, then this *assignment* will not correspond to any *coverage constraint* of the ward to be planned and the *assignment* is not allowed to be altered by the algorithm. Second, the *assignment* should not have been locked (see Section 3.2.3). Since the *coverage constraints* are not defined as hard

constraints, deletions that violate *coverage constraints* are considered to be feasible as well (see Section 3.2.3), in contrast to most papers on academic progress.

A *delete shift* move not only deletes the assigned *shift type*, but also the *skill type* in the same day, because the *skill type* is a property of the *assignment* as well. Since the *delete shift* neighbourhood is relatively small, no exception is made to the steepest descent fashion of searching the neighbourhood. Similar to the *assign shift* neighbourhood, a move of the *delete shift* neighbourhood is encoded as a hash value of the <employee, day, shift type, skill type> quadruple and inserted into the tabu list.

Algorithm 16 The delete shift neighbourhood

```
C := candidate solution
BC := best candidate solution found so far
F(C) := objective function
BC' := null
BF := ∞
for all e ∈ E do
  for all d ∈ D do
    if e.IsAssigned(d) ∧ ¬e.IsLocked(d) then
      s = e.schedule[d].shift
      k = e.schedule[d].skill
      if IsDefined(d, s, k) then
        C' = C.Delete(e, d, s, k)
        if F(C') < F(BC) ∨ ¬IsTabu(e, d, s, k) then
          if F(C') < BF then
            BF = F(C')
            BC' = C'
          end if
        end if
      end if
    end if
  end for
end for
return BC'
```

Single shift day

A *single shift day* move involves deleting an *assignment* from the *schedule* of an *employee* and adding the same *assignment*, i.e., the same *shift type* and *skill type*, at the same date to another *employee* who is not assigned on that date. Suppose that, an *employee* is assigned an early shift as a caregiver on a particular date and another *employee* that has the *skill type* caregiver is idle on the same date. In

this case, a *single shift day* move deletes the early shift *assignment* from the first *employee* and assigns it to the second *employee* on the same date. This way, the *coverage constraints* are not affected, but there is a probability that the *schedules* of the related *employees* are improved. A move of the *single shift day* neighbourhood consists of two moves: a *delete* and an *assign* move. Therefore, the parameters of both the *delete* and *assign* moves, are separately encoded as hash values and inserted into the tabu list. The *single shift day* neighbourhood is searched in a steepest descent fashion without any exceptions.

Algorithm 17 The single shift day neighbourhood

```
C := candidate solution
BC := best candidate solution found so far
F(C) := objective function
BC' := null
BF := ∞
for all d ∈ D do
  for all e ∈ E do
    if e.IsAssigned(d) ∧ ¬e.IsLocked(d) then
      s = e.schedule[d].shift
      k = e.schedule[d].skill
      if IsDefined(d, s, k) then
        for all e' ∈ E | e ≠ e' do
          if ¬e'.IsAssigned(d) ∧ ¬e'.IsLocked(d) ∧ ¬e'.IsOverlapping(d, s) ∧
          e'.HasSkill(k) then
            C' = C.Delete(e, d, s, k)
            C'' = C'.Assign(e', d, s, k)
            if F(C'') < F(BC) ∨ ¬(IsTabu(e, d, s, k) ∨ IsTabu(e', d, s, k))
            then
              if F(C'') < BF then
                BF = F(C'')
                BC' = C''
              end if
            end if
          end if
        end for
      end if
    end if
  end for
end for
return BC'
```

Change assignment shift

Change assignment shift refers to a move where the assigned *shift type* is replaced with another *shift type* while the *skill type* remains the same. Suppose that a head nurse is assigned to a late shift on Monday. The effect of a *change assignment shift* move can be that instead of a late shift, she is scheduled in the early shift. Depending on her *assignments* during the preceding weekend and the days from Tuesday onwards, the new shift *assignment* can result in a better, worse or equally good *schedule*. Similar to the *assign shift* neighbourhood, a subset of the complete neighbourhood is considered. The subset consists of a single alternative random *shift type* for each *assignment*. This is an exception to the steepest descent fashion of searching the *change assignment shift* neighbourhood. Similar to the *single shift day*, the *change assignment shift* is also composed of two moves, a *delete* and an *assign* move. Consequently, its parameters are encoded and inserted into the tabu list in the same way as *single shift day*.

Change assignment shift based on shift type set

Change assignment shift based on shift type set is a variation of the *change assignment shift* neighbourhood. In the *change assignment shift neighbourhood*, the assigned shift s is replaced with a shift s' selected from the set of all the *shift types* in the problem instance, i.e., S. In the *change assignment shift based on shift type set* neighbourhood, the new shift s' is selected from the *shift type set* S' of the *coverage constraint* $cc_{d,S',K'}$ that covers the actual *assignment*, $x_{e,d,s,k} = 1$. That means the day of the *coverage constraint* and the actual *assignment* are the same, and the skill type k and shift type s of the actual *assignment* are elements of the *skill type set* K' and *shift type set* S' of the *coverage constraint* $cc_{d,S',K'}$, respectively. Another small difference is that in the *change assignment shift based on shift type set* neighbourhood, the new *shift type* s' does not need to be checked whether it is defined in a *coverage constraint* or not, because it is already extracted from a *coverage constraint*, namely $cc_{d,S',K'}$.

Algorithm 18 The change assignment shift neighbourhood

```
C := candidate solution
BC := best candidate solution found so far
F(C) := objective function
BC' := null
BF := ∞
for all d ∈ D do
  for all e ∈ E do
    if e.IsAssigned(d) ∧ ¬e.IsLocked(d) then
      s = e.schedule[d].shift
      k = e.schedule[d].skill
      if IsDefined(d, s, k) then
        S' = RandomPermutation(S)
        b = true
        for all s' ∈ S' do
          if b then
            if s ≠ s' ∧ IsDefined(d, s', k) ∧ ¬e.IsOverlapping(d, s') then
              C' = C.Delete(e, d, s, k)
              C'' = C'.Assign(e, d, s', k)
              if F(C'') < F(BC) ∨ ¬(IsTabu(e, d, s, k) ∨ IsTabu(e, d, s', k))
              then
                b = false
                if F(C'') < BF then
                  BF = F(C'')
                  BC' = C''
                end if
              end if
            end if
          end if
        end for
      end if
    end if
  end for
end for
return BC'
```

Algorithm 19 The change assignment shift based on shift type set neighbourhood

```
C := candidate solution
BC := best candidate solution found so far
F(C) := objective function
BC' := null
BF := ∞
for all d ∈ D do
  for all e ∈ E do
    if e.IsAssigned(d) ∧ ¬e.IsLocked(d) then
      s = e.schedule[d].shift
      k = e.schedule[d].skill
      if IsDefined(d, s, k) then
        S'' = RandomPermutation(S')|cc_{d,S',K'} ∈ CC|s ∈ S' ∧ k ∈ K'
        b = true
        for all s' ∈ S'' do
          if b then
            if s ≠ s' ∧ ¬e.IsOverlapping(d, s') then
              C' = C.Delete(e, d, s, k)
              C'' = C'.Assign(e, d, s', k)
              if F(C'') < F(BC) ∨ ¬(IsTabu(e, d, s, k) ∨ IsTabu(e, d, s', k))
              then
                b = false
                if F(C'') < BF then
                  BF = F(C'')
                  BC' = C''
                end if
              end if
            end if
          end if
        end for
      end if
    end if
  end for
end for
return BC'
```

Change assignment skill

The *change assignment skill* neighbourhood is defined for the *schedules* of *employees* that have at least two different *skill types*. A move in this neighbourhood involves deleting an *assignment* and adding another *assignment* to the same *employee*, on the same day, but with another one of the *employee's skill types* and a corresponding

shift type. Suppose that an *employee* has the *skill types* caregiver and regular nurse and she is assigned to a late shift as a caregiver on a particular date. A *change assignment skill* move can result in her being assigned as a regular nurse, in the early shift, on the same date. Such a move can improve, worsen or have no effect on the *schedule*. The *change assignment skill* neigbourhood is searched in a steepest descent fashion without any exceptions.

Algorithm 20 The change assignment skill neighbourhood

```
C := candidate solution
BC := best candidate solution found so far
F(C) := objective function
BC' := null
BF := ∞
for all d ∈ D do
  for all e with multiple skills ∈ E do
    if e.IsAssigned(d) ∧ ¬e.IsLocked(d) then
      s = e.schedule[d].shift
      k = e.schedule[d].skill
      if IsDefined(d, s, k) then
        C' = C.Delete(e, d, s, k)
        for all k' ∈ K | k' ≠ k do
          if e.HasSkill(k') then
            for all s' ∈ S do
              if IsDefined(d, s', k') ∧ ¬e.IsOverlapping(d, s') then
                C'' = C'.Assign(e, d, s', k')
                if F(C'') < F(BC)∨¬(IsTabu(e, d, s, k)∨IsTabu(e, d, s', k'))
                then
                  if F(C'') < BF then
                    BF = F(C'')
                    BC' = C''
                  end if
                end if
              end if
            end for
          end if
        end for
      end if
    end if
  end for
end for
return BC'
```

7.6.2 Heuristics

The heuristics that are deployed by the hyperheuristics are constructed according to a tournament strategy (Algorithm 21). The motivation behind the tournament strategy is to increase the efficiency of the algorithms. This is accomplished by searching a limited random sample of a given neighbourhood instead of searching the complete neighbourhood in a steepest descent fashion.

In this study, three tournament factors have been experimented with: 128, 256 and 512. The tournament factors have been determined based on preliminary experimentation. A tournament heuristic has been implemented on each of the neighbourhoods introduced in the previous section: *assign shift*, *delete shift*, *single shift-day*, *change assignment shift*, *change assignment shift based on shift type set* and *change assignment skill*.

The *tournament heuristics* do not maintain a *tabu list*. The hyperheuristics deployed in the solution methods do not utilise a *tabu list*. The *tabu list* is utilised to avoid cycling during the optimisation process. Cycling happens when two complementary moves such as *assign shift* and *delete shift* with the same parameters are executed back and forth in the vicinity of a local optimum. The risk of cycling is miniscule when using the *tournament heuristics*, because they explore only a sample of a given neighbourhood. Therefore, the *tournament heuristics* are not expected to use the steepest descending path to the local optimum and they are not expected to remain around that local optimum.

Algorithm 21 The pseudocode of the tournament heuristics

Parameters
TF = Tournament Factor
N = Neighbourhood
Algorithm
Select TF random solutions in neighbourhood N of the candidate solution
return the solution with the best objective value

7.7 Conclusions

The common approach in the scientific literature has been to target the optimal solution of a single problem instance. The solution methods presented in this chapter however, have been developed to provide good enough solutions to a broad range of real world personnel rostering problem instances. The motivation behind this approach is the real world requirement for the solution methods to be capable of coping with a great variety of problem instances. The solution methods can be further extended to deal with the extensions in the underlying problem model.

The underlying problem model, as explained in Chapter 3, can be extended if the problem description changes, which happens frequently in the real world practice. Another demand by the real world practitioners is the construction of a complete solution as soon as possible and the quality of the end result to be a function of the execution time. This way, the practitioners can receive a complete solution no matter how short the execution time is. On the other hand, they can receive higher quality solutions in case their situation allows for longer execution times.

A random initialisation method has been developed to provide the users with a complete solution as quick as possible (Section 7.1). A complete solution refers to a schedule where the minimum of the coverage constraints are satisfied to the extent allowed by the hard constraints. The result of the initialisation method is also used as the starting point in the iterational improvement methods that follow the initialisation.

An algorithmic toolbox that consists of a variety of iterational improvement methods have been developed to work within the framework of the solution method. The common property of these methods is the termination criterion, which is the execution time. They have been developed to improve the schedule incrementally so that the quality of the schedule increases relative to the execution time. Variable neighbourhood search (VNS), adaptive large neighbourhood search (ALNS) and hyperheuristics have been used as iterational improvement methods. VNS, ALNS and hyperheuristics can be customised with different parameters, components, neighbourhood and low level heuristic sets.

ALNS and VNS have common parameters, the set of *neighbourhoods* utilised and the upper limit for the *tabu tenure*. The hyperheuristics deployed as solution methods are composed of two parts, a *selection method* and an *acceptance criterion*. *Simple random* and *choice function* are used as *selection methods*. *Improving or equal moves accepted*, *simulated annealing* and *great deluge* are used as *acceptance criterion*. *Linear*, *quadratic* and *quartic* are used as *cooling schemes* in both *simulated annealing* and *great deluge*. The utilisation of various algorithmic elements allows for a great range of algorithm configurations. That in turn allows for more specific customisation of the algorithmic toolbox for a given problem instance.

Six neighbourhoods are utilised in VNS and ALNS. These are *assign shift*, *delete shift*, *single shift day*, *change assignment shift*, *change assignment shift based on shift type set* and *change assignment skill*. They are all based on different properties of the extended problem model and explore the solution space in different ways. This way the solution space will be traversed more efficiently in order to find better solutions in less execution time. *Tournament heuristics* have been developed to explore these neighbourhoods within a hyperheuristic framework. At each iteration, a *tournament heuristic* searches a sample of a neighbourhood instead of searching the neighbourhood completely. That way the CPU time spent at each iteration is reduced and more iterations are made throughout the execution of the algorithm.

An algorithmic toolbox and three novel low level neighbourhoods to address the personnel rostering problem is the main contribution of the present chapter. The algorithmic toolbox has been designed to be configured automatically during the execution time by the deploying system without the intervention by a human expert. The algorithmic toolbox has been composed with components adopted from the scientific literature except three novel low level neighbourhoods. The low level neighbourhoods introduced by the author of the present dissertation are *change assignment shift*, *change assignment shift based on shift type set* and *change assignment skill*.

The customisation of the algorithmic toolbox plays an important role when addressing a broad range of problem instances. Quantitative measures have been introduced in Chapter 5 to measure the properties of the problem instances. Experiments have been conducted to find out the relations between the performance of the solution method settings and the quantifiable problem properties (Chapter 8). Consequently, with the set of algorithmic tools that has been presented in this chapter, a decision support system can select the most promising configuration of the algorithmic toolbox based on the quantitative measures of the problem instance and the experimental results. This practice is expected to increase the probability of finding good enough solutions accross a great variety of real world personnel rostering problem instances.

Chapter 8

Experiments

The objective of the present dissertation is to introduce a solution method to address a wide range of real world personnel rostering problems. According to the "no free lunch" theorems discussed by Wolpert and Macread in [71], a single algorithm configuration cannot be expected to outperform other algorithm configurations on all problem instances. Therefore, instead of targeting all personnel rostering problem instances, the research in the present dissertation has been limited to the real world personnel rostering problems. Furthermore, an algorithmic toolbox that can be configured in numerous ways has been utilised in the solution method (Chapter 7), in order to increase the probability of finding good enough solutions on a broad range of real world personnel rostering problems. However, in a real world environment, where a satisfactory solution is expected in a limited amount of execution time, it is not practical to experiment with each configuration of the algorithmic toolbox every time a new problem instance is encountered. Therefore, a decision method to select the most promising configuration of the algorithmic toolbox for a given problem instance is needed for practical purposes.

A comprehensive hardness analysis of the personnel rostering problems as proposed by Leyton-Brown et al. in [43] is beyond the scope of this dissertation. In such an analysis, a problem instance distribution is determined and the performance of the algorithm configurations at hand is measured among this distribution. Based on a statistical analysis of the experimental results, a performance prediction function is derived on problem instance - algorithm configuration pairs. Instead of departing from a problem instance distribution, the experiments in the present chapter are executed on the KAHO benchmarks introduced in Chapter 6. Consequently, the experimental results in the present chapter do not provide sufficient statistical evidence to generalise the derived relation between the problem instances and the algorithm configurations to all the personnel rostering problem instances. However,

the experimental results provide sufficient statistical evidence to derive a relation between the real world problem instances whose quantitative measure values are within a particular range and the most promising configuration of the algorithmic toolbox among the configurations experimented with. Such a relation, although cannot be generalised to all the personnel rostering problems and algorithms in its current form, provides a significant value to the developers who implement automated personnel rostering decision support systems and to the planners who use those systems. An extended empirical and statistical study of the relation between all the personnel rostering problems and the most promising configurations of the algorithmic toolbox is pointed out as future work.

In this dissertation, a two step decision method is proposed to select the most promising configuration of the algorithmic toolbox for a given problem instance. The first step is to determine the properties of the problem instance. The quantitative measures introduced in Chapter 5 have been developed to measure the size and hardness of a problem instance in a short amount of time that is negligible compared to the execution time of the solution method. The second step is to use a relation between the problem properties and the most promising configuration of the algorithmic toolbox. The purpose of the experiments presented in this chapter is to find such a relation. The discovered relation will then be used as the second step of the decision method to select the most promising configuration for a given problem instance.

Several decisions need to be made to configure the algorithmic toolbox introduced in Chapter 7. First, one of the variable neighbourhood search (VNS), adaptive large neighbourhood search (ALNS) or hyperheuristics needs to be chosen. Each of these algorithms involve a number of configuration decisions. The maximum tabu tenure must be decided when using VNS and ALNS. The selection method, acceptance criterion and the tournament factor needs to be determined when using the hyperheuristics. Furthermore, the set of neighbourhoods or low level heuristics must also be decided upon. To determine the most promising set of neighbourhoods, experiments have been carried out on five different neighbourhood sets deployed in several VNS variants.

The experimental results have been analysed in two steps. First the most promising VNS and ALNS configurations have been determined. Then, the best performing hyperheuristic variants have been compared with the most promising VNS and ALNS configurations. As a result, the most promising algorithm configuration for each problem group has been determined. Three problem properties have been used to divide the problem instances into specific groups. These properties are presented in the following list.

- Existence of employees with multiple skill types
- Minimum number of required and free assignments (MNRFA)

- Tightness ratio

The combination of the MNRFA and tightness ratio have been used to divide the problem instances into four groups to determine the most promising overall configuration of the algorithmic toolbox for each group. The KAHO benchmarks, introduced in Chapter 6, have been used as test data. The Nottingham benchmarks, surveyed in Chapter 2, have been used to verify the findings of the experiments.

The decision variables of the algorithm configurations, such as the tournament factor and the maximum tabu tenure, are discrete. Therefore a set of candidate values have been used to find out the most promising one. Preliminary experimentation has been used to decide upon the sets of these candidate values.

8.1 Experimental settings

There are two decisions to be made for setting the VNS algorithm: the composition of the neighbourhood set and the maximum tabu tenure. The neighbourhood sets that are applied in the experiments are presented in Table 8.1. As mentioned in Chapter 7, the basic neighbourhood set, i.e., *assign shift*, *delete shift* and *single shift day*, has been adopted from the scientific literature. The other three neighbourhoods, *change assignment shift based on shift type set*, *change assignment shift* and *change assignment skill*, have been developed by the author of the present dissertation. Different neighbourhood sets have been experimented with in order to measure the contribution of the neighbourhoods developed by the author. The *assign shift* and *delete shift* neighbourhoods are complementary. They have been utilised consecutively in the VNS algorithm. The remaining neighbourhoods are special compositions of *delete shift* and *assign shift* neighbourhoods, each with a specific rule to restrict the *delete shift-assign shift* sequence. In the VNS algorithm, the search starts with the *assign shift* and *delete shift* neighbourhoods, and continues with the *single shift-day* neighbourhood. The additional neighbourhoods follow the basic set in the order of appearance in Table 8.1.

The rationale behind the ALNS algorithm is based on the dynamic selection of the neighbourhood at each iteration. Therefore, a specific order of the neighbourhoods is not needed. The most comprehensive neighbourhood set, Set 5, has been utilised in the ALNS algorithm, because ALNS is expected to select relevant neighbourhoods with higher probability.

One of the parameters of the VNS and ALNS is the upper bound for the tabu tenure. The upper bound for the tabu tenure must be a positive integer value, and consequently, it can take an unlimited number of values. Therefore, a limited number of values, 97 and 199, have been determined as the upper bounds for the tabu tenure using preliminary experimentation. The upper bounds for the tabu

Set 1	assign shift, delete shift, single shift day
Set 2	Set 1 + change assignment based on shift type set
Set 3	Set 1 + change assignment skill
Set 4	Set 1 + change assignment shift
Set 5	Set 1 + change assignment shift + change assignment skill

Table 8.1: Neighbourhood sets

Problem property	No employee with multiple skill types	At least one employee with multiple skill types
Most suitable heuristic set	Assign shift Delete shift Single shift day Change assignment shift	Assign shift Delete shift Single shift day Change assignment shift Change assignment skill

Table 8.2: Heuristic sets

tenure, 97 and 199, have been experimented with in both, VNS and ALNS. As a result, 10 different VNS and 2 different ALNS variants have been experimented with for comparison purposes.

Two selection methods have been experimented with in hyperheuristics: *simple random* and *choice function*. Three acceptance criteria have been utilised: *improving and equal moves accepted*, *simulated annealing* and *great deluge*. Three different cooling schedules have been used in *simulated annealing* and *great deluge*: linear where $n = 1$ in Equations 2.6 and 2.7, quadratic where $n = 2$, and quartic where $n = 4$. Although greater values for n are possible, that will only result in the behaviour of *simulated annealing* and *great deluge* approaching asymptotically to the behaviour of *improving and equal moves accepted*, and it is not expected to add any algorithmic diversity to the experiments.

Similar to the upper bound for the tabu tenure, the tournament factors can take any positive integer value, resulting in infinite number of candidate parameters. Therefore, a limited number of tournament factors have been determined as a result of preliminary experiments. Three tournament factors have been used in the tournament heuristics: 128, 256 and 512. As a result, the experiments have involved 42 hyperheuristic variants. The heuristic sets have been constructed using the two most successful neighbourhood sets in the experiments on the VNS algorithms (Table 8.2).

The problem model and solution methods have been implemented in C#. The operating system of the experimentation platform was MS Windows Server 2003 Enterprise Edition SP 2. The hardware was Intel Pentium 4 CPU with 2.40 GHz and 2.00 GB of RAM. For the KAHO benchmarks, the termination criterion was ten minutes of execution time for each run for normal and overload scenarios, and one minute of execution time for the absence scenario. For the Nottingham benchmarks, the termination criterion was ten minutes of execution time, except for Millar1.1 and Ozkarahan, where the execution time was one minute. The termination criterion has been selected to reflect the expectations of the end users.

8.2 Experimental results

Each algorithm variant has been executed ten times on each problem instance to allow statistical analysis. The results of experiments have been processed in two groups, first the results on VNS and ALNS and then the results on the hyperheuristics. In each group and for each problem instance, the best performing algorithm according to the average objective value over ten runs has been selected. The Wilcoxon test with 95% confidence level has been used to assess the statistical significance of the difference between the results of the best performing algorithm and the remaining algorithms. If there is no significant performance variance between an algorithm and the best performing one, this algorithm has also been considered in the best performing group.

8.2.1 Results on VNS and ALNS variants

The experimental results for each VNS, ALNS variant-problem instance couple are presented in Tables 8.3 - 8.8. The values in these tables are the average objective values of the best solutions found and their standard deviations. In these tables, the algorithm settings that performed significantly better than the rest are highlighted with bold characters. Tables 8.3 - 8.8 also depict the best results of the normal scenarios of the KAHO benchmarks achieved manually by the human planners. Table 8.9 presents the results by VNS variants with 97 as the *maximum tabu tenure* to give an overall view.

The experimental results for VNS and ALNS indicate that the solution methods with 97 as the *maximum tabu tenure* have performed better than the ones with 199 on the KAHO benchmarks. The VNS, ALNS variants with 97 as the *maximum tabu tenure* have been among the best performers for 17 problem instances, while the ones with 199 have been among the best performers only for seven of the KAHO benchmarks. The only exception to this observation is the *emergency overload* scenario. The VNS, ALNS variants with neighbourhood sets 3, 4 and 5 and the

maximum tabu tenure 199 have been the best performing solution methods for the *emergency overload* scenario (Table 8.3).

Among the VNS, ALNS variants with 97 as the *maximum tabu tenure*, VNS with neighbourhood set 4 has been among the best performers for 13 instances, and neighbourhood set 5 for 12 instances. The remaining VNS, ALNS variants could not compete with this performance. The exceptions to this observation are the experiments for the normal scenarios of the *psychiatry*, *meal preparation* and *geriatrics* wards. ALNS has performed better than VNS on the *psychiatry* normal scenario. The best performing VNS, ALNS variants for *geriatrics* normal scenario have been VNS with neighbourhood set 1 and 2, and for *meal preparation* normal scenario VNS with neighbourhood set 1, 2 and 3.

VNS with neighbourhood set 5 has outperformed VNS with neighbourhood set 4 only on the *emergency* absence scenario. This result can be explained by the fact that the nurse rostering problem in the *emergency* ward involves numerous nurses with secondary *skill types* and neighbourhood set 5 involves the *change assignment skill* neighbourhood. As a conclusion, VNS with neighbourhood set 4 and *maximum tabu tenure* 97 has been the most promising one among the VNS and ALNS settings tested on the KAHO benchmarks.

The basic neighbourhood set, Set 1, has been among the best performing neighbourhood sets on eight out of 18 problem instances. For these instances, the basic neighbourhood set has not been the unique best performer. The contribution of the problem specific neighbourhoods that take advantage of the problem properties like *secondary skill types* and *compatible shift types* has been emphasised by this result.

8.2.2 Results on hyperheuristics variants

In Table 6.1, the *minimum number of required and free assignments* (MNRFA), *tightness* and *period* values of each problem instance are given. Further details on how these values have been calculated can be found in Chapter 5. The problem instances from the KAHO benchmarks have been divided into four groups according to their MNRFA and tightness values (Table 6.1). The performance of the algorithm variants on each group has been analysed.

The experimental results of a selected subset of the hyperheuristic variants and the results of the best performing VNS variant have been reported in Tables 8.10 to 8.12. In Tables 8.10 to 8.12, CF refers to *choice function*, SR to *simple random*, SA to *simulated annealing*, L to *linear*, Q4 to *quartic*, while the numbers denote the tournament factors. In the same tables, the first number after VNS is the *maximum tabu tenure* and the second one the neighbourhood set; the number after ALNS denotes the *maximum tabu tenure*. The values in Tables 8.10 to 8.12 refer to

the best objective values obtained by the algorithms. The results of the algorithm variants that are in the best performing group are indicated in boldface characters.

The small size problem instances, i.e., the instances with absence scenarios, have an MNRFA between 40 and 107 (Table 6.1). The hyperheuristic variant with *choice function*, *simulated annealing* with *linear cooling*, and 128 as *tournament factor* has been the best performing algorithm variant for this group, among the algorithm configurations experimented with. This algorithm variant has been among the best performing group on all of the small size problem instances. On three out of six small size problem instances, this variant has found the best objective value found by the algorithms experimented with, in all of the ten executions (Table 8.10).

The midsize problem instances with normal tightness ratio have an MNRFA between 200 and 513, and their *tightness ratios* vary between 0.66 and 1.19 (Table 6.1). For this group, the hyperheuristic configuration with *choice function*, *simulated annealing* with *linear cooling*, and 256 as *tournament factor* has been the best performing algorithm configuration among the algorithm configurations experimented with. This algorithm variant has been in the best performing group on all of the midsize problem instances with normal tightness. On two out of eight problem instances in this group, this variant has generated the best overall objective value found by the algorithms experimented with, in all of the ten experiments (Tables 8.11 and 8.12).

Although *reception normal* and *reception overload* belong to the group of midsize problems, they are distinguished from this group by their high *tightness ratios* (Table 6.1). This fact is also reflected by the algorithm performance, because the best performing algorithms for the midsize-normal tightness instances have not been among the best performing group for midsize-high tightness instances. On midsize-high tightness instances, there are seven algorithm variants that have been in the best performing group among the algorithm variants experimented with. Among these seven variants, the following hyperheuristic configurations stand out: the hyperheuristic composed of *simple random*, *simulated annealing* with *quartic cooling scheme*, and 256 as *tournament factor*, and the hyperheuristic composed of *choice function*, *simulated annealing* with *quartic cooling scheme*, and 128 as *tournament factor*. The first variant has returned the best average objective value on *reception normal*, and the second the best average objective value on *reception overload* (Tables 8.11 and 8.12).

The *palliative care normal* and *overload* instances form the group of large size problem instances with an MNRFA of 846 and 1031, respectively (Table 6.1). VNS methods with the *maximum tabu tenure* of 97 have been in the best performing group for these problem instances among the algorithms experimented with (Tables 8.11 and 8.12). The only problem group where VNS has outperformed the hyperheuristics is the group of large size problem instances.

An overall view on the experimental results is given in Table 8.13. The problems

are grouped according to their MNRFA and *tightness values*. The values in Table 8.13 denote how well the algorithms have performed on the given problem instance. Only the algorithm settings that are considered as the most suitable ones for at least one of the problem instance groups have been reported on this table.

In addition to the KAHO benchmarks, ten instances from the Nottingham benchmarks[1] have been used in the experiments. The algorithms in the best performing group on the KAHO benchmarks have been applied to the Nottingham benchmarks. These are the hyperheuristic algorithms with *choice function* and *simulated annealing* with *linear cooling scheme*. In this setting, the *tournament factors* 128 and 256 have been the most suitable for the small and midsize-normal tightness instances, respectively.

As it has been explained in detail in Chapter 4, the results of the experiments on the Nottingham benchmarks have been severely influenced by handling the continuity between two *schedule periods*. The experimental results on the Nottingham benchmarks need to be interpreted considering the fact that the model presented here has a different problem definition, namely one that does not conflict with the continuous nature of the problem.

8.3 Conclusions

The objective of the present chapter is to find a relation between the real world problem instances whose quantitative measures are within a particular range and the most promising configuration of the algorithmic toolbox introduced in Chapter 7. The discovered relation can be used in automated personnel rostering software to select the most promising algorithm configuration of the solution method in Chapter 7 for a given problem instance. A series of experiments have been carried out and the results of the experiments have been analysed using statistical methods to find out the desired relation. Three problem properties have been taken into account, the existence of employees with multiple skill types, the minimum number of required and free assignments (MNRFA) and the tightness ratio. The experiments have been carried out on the KAHO benchmarks. The Nottingham benchmarks have been used to verify the results of the experiments.

The instances of the KAHO benchmarks have been divided into two groups according to the existence of employees with multiple skill types. A specific low level heuristic set has been suggested for each group (Table 8.2). Another division is based on the instances' MNRFA and tightness ratio values so that the most promising algorithm configuration for each group can be investigated. Ten variants of the variable neighbourhood search (VNS), two variants of the adaptive

[1] http://www.cs.nott.ac.uk/∼tec/NRP/

large neighbourhood search (ALNS) and 42 hyperheuristic variants have been experimented with.

Among the configurations experimented with, the hyperheuristic configuration with *choice function* as the selection method, *simulated annealing with linear cooling schedule* as the acceptance criterion, and 128 as the tournament factor has been found to be the most promising algorithm configuration for small size problem instances. The same hyperheuristic configuration but with 256 as tournament factor has been found to be the most promising configuration for midsize problem instances with normal tightness ratios. Two hyperheuristic configurations, first, *simple random* as selection method and 256 as tournament factor, and second, *choice function* as selection method and 128 as tournament factor have been found to be the most promising configuration for midsize problem instances with high level tightness ratios. The acceptance criterion in both configurations is *simulated annealing with quartic cooling scheme*. For large size problem instances, variable neighbourhood search with 97 as the maximum tabu tenure and neighbourhood set 4 has been determined to be the most promising algorithm configuration. An overview of these results can be found in Table 8.13. The contribution of the problem specific neighbourhoods introduced in Chapter 7 has been emphasised through the better performance of the algorithm configurations that deploy these neighbourhoods.

The results of the experiments with the best performing algorithm variations on ten instances from the Nottingham benchmarks have also been reported. However, these results have been distorted by the fact that the constraint evaluation methods of the Nottingham benchmarks and the solution methods presented in this dissertation do not match.

The experimental results in the present chapter do not represent a comprehensive hardness analysis of the personnel rostering problems as proposed by Leyton-Brown et al. in [43]. Instead of constructing an experimental design on a problem instance distribution to represent all personnel rostering problems, the focus of the experiments has been the real world personnel rostering problems whose quantitative measures vary in a particular range. The experimental results in the present chapter cannot be generalised to all the personnel rostering problems, which was not the objective of the present chapter. Nevertheless, the derived relation is of substantial value in the real world practice for the developers who have to provide automated personnel rostering solutions and for the planners who utilise the provided solutions. A generalisation of the experimental results in the present chapter using the methodology in [43] is pointed out as future work.

A. Setting		Normal		Overload		Absence	
N. Set	T. L.	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
VNS - 1	97	**11410.50**	**386.46**	27441.33	332.37	21849.67	88.86
VNS - 1	199	14020.66	1566.69	27052.50	138.65	21864.67	45.72
VNS - 2	97	11817.33	655.30	29231.67	2461.29	21862.17	38.55
VNS - 2	199	13310.66	714.66	27055.33	203.29	21882.17	91.24
VNS - 3	97	**11295.83**	**275.42**	27813.66	733.73	21412.67	246.77
VNS - 3	199	11801.66	376.20	**26836.17**	**227.36**	21843.67	38.09
VNS - 4	97	**11201.17**	**308.12**	27130.00	314.70	21532.67	186.01
VNS - 4	199	12014.00	561.92	**26699.67**	**161.17**	21711.17	268.75
VNS - 5	97	**11361.17**	**239.20**	27483.17	370.90	**21175.17**	**20.82**
VNS - 5	199	**11285.50**	**201.21**	**26731.84**	**185.23**	21753.84	213.04
ALNS - 5	97	11753.33	191.95	27679.17	216.29	21327.67	137.90
ALNS - 5	199	11858.83	163.41	27325.17	123.33	24121.33	1598.55
Human Planner		49236.00		-			

Table 8.3: **Hospital 1 Emergency Results**. A. Setting, N. Set, T. L., St. Dev. denote algorithm setting, neighbourhood set, upper bound for the tabu list length and standard deviation, respectively.

A. Setting		Normal		Overload		Absence	
N. Set	T. L.	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
VNS - 1	97	9079.00	191.51	12339.00	311.11	13109.00	210.42
VNS - 1	199	10236.00	255.13	14687.00	557.77	**12906.00**	**203.26**
VNS - 2	97	9074.00	223.67	12429.00	234.92	12993.00	99.34
VNS - 2	199	10160.00	430.86	14492.00	656.94	**12874.00**	**235.33**
VNS - 3	97	9041.00	173.36	12207.00	205.32	**12946.00**	**206.30**
VNS - 3	199	10363.00	218.84	14930.00	570.28	13022.00	124.26
VNS - 4	97	8751.00	127.93	**10914.00**	**115.59**	**13001.00**	**201.19**
VNS - 4	199	9184.00	256.31	12166.00	393.62	**12888.00**	**337.96**
VNS - 5	97	8774.00	146.98	**10966.00**	**215.83**	**12850.00**	**172.43**
VNS - 5	199	9292.00	208.74	12196.00	240.15	**13019.00**	**263.67**
ALNS - 5	97	**8649.00**	**136.42**	**10916.00**	**238.15**	13790.00	473.15
ALNS - 5	199	9210.00	219.95	11929.00	297.97	14247.00	1653.78
Human Planner		35480.00		-			

Table 8.4: **Hospital 1 Psychiatry Results**. A. Setting, N. Set, T. L., St. Dev. denote algorithm setting, neighbourhood set, upper bound for the tabu list length and standard deviation, respectively.

A. Setting		Normal		Overload		Absence	
N. Set	T. L.	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
VNS - 1	97	22582.17	219.75	56302.17	606.76	28730.67	223.07
VNS - 1	199	23786.67	238.13	56348.17	387.77	28821.17	120.93
VNS - 2	97	**21720.67**	**173.39**	54122.67	378.08	28699.67	233.65
VNS - 2	199	22555.67	183.41	53873.67	133.67	28778.17	105.38
VNS - 3	97	22525.67	242.67	56336.67	382.11	28838.17	192.46
VNS - 3	199	23884.67	289.83	56274.17	179.97	28850.67	205.76
VNS - 4	97	**21739.67**	**139.77**	**53036.17**	**120.17**	28673.67	187.00
VNS - 4	199	22698.17	238.81	53605.17	189.69	28768.17	137.22
VNS - 5	97	**21812.17**	**136.88**	53122.67	145.43	28729.17	128.80
VNS - 5	199	22624.67	215.23	53563.17	226.91	28786.17	117.70
ALNS - 5	97	**21774.17**	**199.63**	**52975.17**	**120.72**	28786.17	364.21
ALNS - 5	199	22534.67	264.43	53536.67	199.51	28658.67	372.63
Human Planner		48358.00		-			

Table 8.5: **Hospital 1 Reception Results**. A. Setting, N. Set, T. L., St. Dev. denote algorithm setting, neighbourhood set, upper bound for the tabu list length and standard deviation, respectively.

A. Setting		Normal		Overload		Absence	
N. Set	T. L.	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
VNS - 1	97	**2904.10**	**25.40**	10949.10	22.49	**5342.83**	**185.31**
VNS - 1	199	3048.80	31.76	11238.20	72.98	**5408.50**	**157.95**
VNS - 2	97	**2894.67**	**21.82**	10942.30	36.74	**5348.67**	**157.48**
VNS - 2	199	3064.63	53.68	11245.60	68.49	**5401.33**	**177.21**
VNS - 3	97	**2896.33**	**14.08**	10944.80	37.97	**5346.33**	**186.36**
VNS - 3	199	3049.70	40.02	11230.90	64.73	**5425.00**	**157.11**
VNS - 4	97	3162.60	37.78	**10867.90**	**29.75**	**5326.33**	**99.71**
VNS - 4	199	3121.50	51.75	10978.30	68.29	5516.17	133.61
VNS - 5	97	3133.33	19.07	**10881.60**	**25.54**	**5350.17**	**82.24**
VNS - 5	199	3104.83	59.53	10987.00	46.31	**5405.00**	**155.49**
ALNS - 5	97	3107.87	28.20	11058.00	49.10	**5338.33**	**52.90**
ALNS - 5	199	3124.27	39.96	11120.60	42.98	7193.17	460.98
Human Planner		22100.00		-			

Table 8.6: **Hospital 1 Meal Preparation Results**. A. Setting, N. Set, T. L., St. Dev. denote algorithm setting, neighbourhood set, upper bound for the tabu list length and standard deviation, respectively.

A. Setting		Normal		Overload		Absence	
N. Set	T. L.	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
VNS - 1	97	**4301.00**	**134.88**	10898.50	303.58	**9194.83**	**260.34**
VNS - 1	199	5295.33	633.63	12659.67	935.36	**9035.67**	**447.74**
VNS - 2	97	**4208.67**	**117.93**	10914.33	285.87	**9138.83**	**204.83**
VNS - 2	199	5170.67	364.91	13539.67	643.76	**9286.83**	**299.59**
VNS - 3	97	4612.67	175.90	11417.00	414.39	9451.33	247.29
VNS - 3	199	6177.50	1212.17	15225.50	1202.69	**9425.67**	**395.41**
VNS - 4	97	4343.67	110.91	**10705.50**	**146.81**	**9180.17**	**358.61**
VNS - 4	199	5647.67	381.44	12192.83	510.15	**9153.17**	**410.28**
VNS - 5	97	4788.33	209.88	11131.17	257.38	**9400.17**	**390.50**
VNS - 5	199	6038.00	642.87	13302.67	889.10	**9300.17**	**320.23**
ALNS - 5	97	4657.67	199.62	**10897.00**	**330.73**	9984.33	554.00
ALNS - 5	199	6945.50	805.99	13685.83	871.46	10282.00	706.24
Human Planner		28594.00		-			

Table 8.7: **Hospital 1 Geriatrics Results**. A. Setting, N. Set, T. L., St. Dev. denote algorithm setting, neighbourhood set, upper bound for the tabu list length and standard deviation, respectively.

A. Setting		Normal		Overload		Absence	
N. Set	T. L.	Average	St. Dev.	Average	St. Dev.	Average	St. Dev.
VNS - 1	97	**44392.50**	**1090.53**	55675.00	1169.25	**56388.00**	**363.93**
VNS - 1	199	46348.25	1053.51	54399.00	1632.47	**56481.50**	**438.73**
VNS - 2	97	**44515.75**	**973.50**	55553.25	965.17	56957.50	351.71
VNS - 2	199	46539.25	721.09	55346.75	1711.53	57139.00	444.82
VNS - 3	97	**44555.75**	**967.81**	55916.25	1828.05	**56659.50**	**642.87**
VNS - 3	199	46662.25	856.09	54812.50	1198.47	**56355.00**	**323.37**
VNS - 4	97	**44951.50**	**791.85**	**50632.25**	**647.25**	**56356.50**	**445.51**
VNS - 4	199	46325.50	1004.18	51638.75	858.53	**56676.00**	**420.49**
VNS - 5	97	**44952.75**	**725.61**	**50177.75**	**528.20**	**56655.50**	**443.67**
VNS - 5	199	46145.25	917.36	51736.75	816.13	**56658.50**	**420.48**
ALNS - 5	97	**44155.25**	**840.44**	51612.25	829.23	57713.75	721.77
ALNS - 5	199	45646.75	607.29	52746.00	1032.53	61334.25	2523.93
Human Planner		183859.00		-			

Table 8.8: **Hospital 2 Palliative Care**. A. Setting, N. Set, T. L., St. Dev. denote algorithm setting, neighbourhood set, upper bound for the tabu list length and standard deviation, respectively.

		VNS-1-97	VNS-2-97	VNS-3-97	VNS-4-97	VNS-5-97	ALNS-5-97
Emergency	N.	**11410.50**	11817.33	**11295.83**	**11201.17**	**11361.17**	11753.33
		386.46	655.30	**275.42**	**308.12**	**239.20**	191.95
	O.	27441.33	29231.67	27813.66	27130.00	27483.17	27679.17
		332.37	2461.29	733.73	314.70	370.90	216.29
	A.	21849.67	21862.17	21412.67	21532.67	**21175.17**	21327.67
		88.86	38.55	246.77	186.01	**20.82**	137.90
Psychiatry	N.	9079.00	9074.00	9041.00	8751.00	8774.00	**8649.00**
		191.51	223.67	173.36	127.93	146.98	**136.42**
	O.	12339.00	12429.00	12207.00	**10914.00**	**10966.00**	**10916.00**
		311.11	234.92	205.32	**115.59**	**215.83**	**238.15**
	A.	13109.00	12993.00	**12946.00**	**13001.00**	**12850.00**	13790.00
		210.42	99.34	**206.30**	**201.19**	**172.43**	473.15
Reception	N.	22582.17	**21720.67**	22525.67	**21739.67**	**21812.17**	**21774.17**
		219.75	**173.39**	242.67	**139.77**	**136.88**	**199.63**
	O.	56302.17	54122.67	56336.67	**53036.17**	53122.67	**52975.17**
		606.76	378.08	382.11	**120.17**	145.43	**120.72**
	A.	28730.67	28699.67	28838.17	28673.67	28729.17	28786.17
		223.07	233.65	192.46	187.00	128.80	364.21
Meal P.	N.	**2904.10**	**2894.67**	**2896.33**	3162.60	3133.33	3107.87
		25.40	**21.82**	**14.08**	37.78	19.07	28.20
	O.	10949.10	10942.30	10944.80	**10867.90**	**10881.60**	11058.00
		22.49	36.74	37.97	**29.75**	**25.54**	49.10
	A.	**5342.83**	**5348.67**	**5346.33**	**5326.33**	**5350.17**	**5338.33**
		185.31	**157.48**	**186.36**	**99.71**	**82.24**	**52.90**
Geriatrics	N.	**4301.00**	**4208.67**	4612.67	4343.67	4788.33	4657.67
		134.88	**117.93**	175.90	110.91	209.88	199.62
	O.	10898.50	10914.33	11417.00	**10705.50**	11131.17	**10897.00**
		303.58	285.87	414.39	**146.81**	257.38	**330.73**
	A.	**9194.83**	**9138.83**	9451.33	**9180.17**	**9400.17**	9984.33
		260.34	**204.83**	247.29	**358.61**	**390.50**	554.00
P. Care	N.	**44392.50**	**44515.75**	**44555.75**	**44951.50**	**44952.75**	**44155.25**
		1090.53	**973.50**	**967.81**	**791.85**	**725.61**	**840.44**
	O.	55675.00	55553.25	55916.25	**50632.25**	**50177.75**	51612.25
		1169.25	965.17	1828.05	**647.25**	**528.20**	829.23
	A.	**56388.00**	56957.50	**56659.50**	**56356.50**	**56655.50**	57713.75
		363.93	351.71	**642.87**	**445.51**	**443.67**	721.77

Table 8.9: **Overall Results on the KAHO benchmarks**. The first row for each problem instance setting is the average objective value of the solutions, the second row the standard deviation. N., O. and A. refer to the normal, overload and absence scenarios, respectively.

Geriatrics-Absence	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-199-1
Average	**8140.00**	**8140.00**	**8140.00**	**8140.00**	**8140.00**	9035.67
St. Dev.	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	447.74
Min.	**8140.00**	**8140.00**	**8140.00**	**8140.00**	**8140.00**	8140.00
Max.	**8140.00**	**8140.00**	**8140.00**	**8140.00**	**8140.00**	9695.00

Psychiatry-Absence	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-5
Average	**11868.00**	**11724.00**	**11764.00**	**11848.00**	**11810.00**	12850.00
St. Dev.	**150.39**	**180.63**	**166.81**	**160.13**	**201.11**	172.43
Min.	**11590.00**	**11590.00**	**11590.00**	**11590.00**	**11590.00**	12470.00
Max.	**11970.00**	**11970.00**	**11970.00**	**11970.00**	**12070.00**	13090.00

Meal P.-Absence	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-4
Average	**4828.33**	**4829.33**	4897.17	4842.83	4835.83	5326.33
St. Dev.	**9.33**	**11.09**	41.24	9.13	10.22	99.71
Min.	**4820.00**	**4820.00**	4841.67	4825.00	4820.00	5163.33
Max.	**4841.67**	**4843.33**	4945.00	4863.33	4843.33	5455.00

Emergency-Absence	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-5
Average	**21146.67**	**21147.67**	**21147.67**	**21147.67**	21200.17	21175.17
St. Dev.	**0.00**	**2.11**	**2.11**	**2.11**	157.08	20.82
Min.	**21146.67**	**21146.67**	**21146.67**	**21146.67**	21146.67	21151.67
Max.	**21146.67**	**21151.67**	**21151.67**	**21151.67**	21646.67	21221.67

Reception-Absence	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	ALNS-199
Average	**28126.67**	**28126.67**	**28126.67**	**28126.67**	**28126.67**	28658.67
St. Dev.	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	372.63
Min.	**28126.67**	**28126.67**	**28126.67**	**28126.67**	**28126.67**	28206.67
Max.	**28126.67**	**28126.67**	**28126.67**	**28126.67**	**28126.67**	29296.67

Palliative C.-Absence	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-199-3
Average	**55061.00**	**55037.00**	**55042.50**	55169.00	**55050.00**	56355.00
St. Dev.	**151.57**	**183.26**	**171.22**	218.54	**186.94**	323.37
Min.	**54872.50**	**54727.50**	**54847.50**	54872.50	**54842.50**	55937.50
Max.	**55317.50**	**55327.50**	**55402.50**	55637.50	**55382.50**	56767.50

Table 8.10: Experimental Results of the Absence Scenarios

Geriatrics-Normal	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-2
Average	**3660.00**	**3660.00**	3779.33	**3660.00**	**3660.00**	4208.67
St. Dev.	**0.00**	**0.00**	102.73	**0.00**	**0.00**	117.93
Min.	**3660.00**	**3660.00**	3660.00	**3660.00**	**3660.00**	4060.00
Max.	**3660.00**	**3660.00**	3860.00	**3660.00**	**3660.00**	4426.67
Psychiatry-Normal	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	ALNS-97
Average	**8120.00**	**8068.00**	**8159.00**	**8170.00**	**8145.00**	8649.00
St. Dev.	**91.89**	**147.56**	**81.30**	**115.95**	**189.93**	136.42
Min.	**8000.00**	**7900.00**	**8000.00**	**8000.00**	**7900.00**	8490.00
Max.	**8200.00**	**8390.00**	**8200.00**	**8400.00**	**8580.00**	8880.00
Meal P.-Normal	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-2
Average	2893.90	**2796.90**	2962.60	2909.10	2806.37	2894.67
St. Dev.	25.08	**26.62**	40.69	23.94	21.80	21.82
Min.	2848.50	**2760.17**	2879.17	2874.17	2771.83	2849.17
Max.	2923.50	**2835.17**	3018.50	2941.83	2842.50	2925.83
Emergency-Normal	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-4
Average	11370.67	**10753.33**	11077.83	11151.33	11074.50	11201.17
St. Dev.	142.29	**175.40**	176.63	255.35	205.66	308.12
Min.	11173.33	**10543.33**	10811.67	10798.33	10833.33	10736.67
Max.	11571.67	**11138.33**	11403.33	11583.33	11488.33	11818.33
Reception Normal	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-2
Average	21783.67	21890.17	**21562.67**	**21598.67**	22126.17	**21720.67**
St. Dev.	143.45	322.84	**202.37**	**190.85**	442.38	**173.39**
Min.	21616.67	21431.67	**21321.67**	**21336.67**	21336.67	**21416.67**
Max.	22126.67	22406.67	**21921.67**	**21916.67**	22726.67	**21931.67**
Palliative C.-Normal	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	ALNS-97
Average	50021.25	48773.75	50067.00	48340.00	48139.25	**44155.25**
St. Dev.	1236.05	459.81	811.59	696.94	744.77	**840.44**
Min.	47365.00	47875.00	49220.00	47350.00	46785.00	**42965.00**
Max.	51675.00	49380.00	51360.00	49430.00	49385.00	**45450.00**

Table 8.11: Experimental Results of the Normal Scenarios

Geriatrics-Overload	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-4
Average	9718.50	**9591.67**	9647.67	9623.67	**9591.67**	10705.50
St. Dev.	60.56	**0.00**	54.00	41.31	**0.00**	146.81
Min.	9671.67	**9591.67**	9591.67	9591.67	**9591.67**	10380.00
Max.	9791.67	**9591.67**	9751.67	9671.67	**9591.67**	10860.00

Psychiatry-Overload	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-4
Average	**9978.00**	**10112.00**	10069.00	10080.00	10218.00	10914.00
St. Dev.	**102.94**	**300.03**	168.75	175.12	260.80	115.59
Min.	**9890.00**	**9700.00**	9700.00	9900.00	9900.00	10680.00
Max.	**10100.00**	**10570.00**	10280.00	10500.00	10680.00	11080.00

Meal P.-Overload	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-4
Average	10937.90	**10811.00**	10931.90	10927.30	10818.30	10867.90
St. Dev.	23.98	**29.91**	26.13	37.85	23.46	29.75
Min.	10876.00	**10746.00**	10896.00	10856.00	10775.00	10821.00
Max.	10961.00	**10856.00**	10976.00	10961.00	10851.00	10906.00

Emergency-Overload	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-199-4
Average	26636.17	**26906.33**	26577.33	26622.50	26988.67	26699.67
St. Dev.	181.01	**561.09**	101.85	193.92	689.24	161.17
Min.	26186.67	**26331.67**	26386.67	26351.67	26086.67	26496.67
Max.	26866.67	**27716.67**	26716.67	26963.33	28301.67	27013.33

Reception Overload	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	ALNS-97
Average	**52888.17**	53270.17	**52953.17**	**52806.17**	53381.17	52975.17
St. Dev.	**186.38**	237.63	**186.10**	**132.15**	339.40	120.72
Min.	**52571.67**	52971.67	**52651.67**	**52586.67**	52931.67	52826.67
Max.	**53081.67**	53696.67	**53291.67**	**52971.67**	54031.67	53156.67

Palliative C.-Overload	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	CF-SA-Q4-256	VNS-97-5
Average	53063.00	51386.75	**50836.00**	50797.25	**50610.00**	**50177.75**
St. Dev.	841.53	597.54	**818.57**	612.15	**384.20**	**528.20**
Min.	51172.50	50535.00	**49810.00**	49985.00	**50075.00**	**49475.00**
Max.	53775.00	52475.00	**52295.00**	51865.00	**51460.00**	**50930.00**

Table 8.12: Experimental Results of the Overload Scenarios

	CF-SA-L-128	CF-SA-L-256	SR-SA-Q4-256	CF-SA-Q4-128	VNS-97-4
Geriatrics Absence	**2**	2	2	2	
Reception Absence	**2**	2	2	2	
Psychiatry Absence	**1**	1	1	1	
Meal Absence	**1**	1			
Palliative Absence	**1**	1	1		
Emergency Absence	**2**	1	1	1	
Geriatrics Normal	2	**2**		2	
Geriatrics Overload		**2**			
Psychiatry Normal	1	**1**	1	1	
Psychiatry Overload	1	**1**			
Meal Normal		**1**			
Meal Overload		**1**			
Emergency Normal		**1**			
Emergency Overload		**1**			
Reception Normal			**1**	**1**	1
Reception Overload	1		**1**	**1**	
Palliative Normal		1		1	**1**
Palliative Overload			1	1	**1**

Table 8.13: Overall Results: 2 denotes that the algorithm has found the best objective value found by all the algorithms experimented with, in all of ten executions. 1 denotes that the algorithm has been in the best performing group for that instance. If no value is given, the algorithm has not been in the best performing group. The values of the most suitable algorithm variants for each group are indicated with boldface characters.

	BCV-3.46.2		BCV-4.13.1		Valouxis		SINTEF
	(N)	(K)	(N)	(K)	(N)	(K)	(N&K)
Average	943.00	942.20	156.30	156.20	284.00	562.00	26.10
St. Dev.	29.75	29.93	80.64	80.16	41.95	38.24	3.75
Min.	901.00	899.00	39.00	38.00	220.00	500.00	22.00
Max.	998.00	997.00	280.00	279.00	340.00	600.00	33.00
Literature	894.00		10.00		20.00		0.00

Table 8.14: Results on the Nottingham Benchmarks 1: N and K denote the results obtained from the evaluation methods of the Nottingham and KAHO benchmarks, respectively.

	LLR		Millar1	Millar1.1	Ozkarahan	GPost	Gpost-B
	(N)	(K)	(N&K)	(N&K)	(N&K)	(N&K)	(N&K)
Average	343.40	349.80	200.00	0.00	0.00	250.40	246.40
St. Dev.	17.20	15.25	210.82	0.00	0.00	42.51	104.60
Min.	322.00	334.00	0.00	0.00	0.00	138.00	82.00
Max.	381.00	378.00	400.00	0.00	0.00	292.00	453.00
Literature	301.00		0.00	0.00	0.00	5.00	3.00

Table 8.15: Results on the Nottingham Benchmarks 2: N and K denote the results obtained from the evaluation methods of the Nottingham and KAHO benchmarks, respectively.

Chapter 9

Conclusions

The objective of this dissertation was to present a complete solution ready to be implemented to address the real world personnel rostering problem. The presented solution involves an extended problem model, quantitative measures of problem properties, a solution method that can be configured in numerous ways and a decision method to select the most promising configuration of the solution method for a given real world problem instance.

Literature surveys on personnel rostering have revealed that only a small percentage of the academic methods have been used in practice. There are several reasons for the low impact of personnel rostering research on the real world. The focus within the research domain has been on the algorithms and little specific effort has been reported on the problem models. Simplified models have been utilised to prove the usefulness of certain algorithms. The real world personnel rostering problems cannot be represented accurately with simplified problem models. Solutions based on incomplete representations will not be satisfactory in the real world practice, even if they are optimal with respect to their own problem definitions. An accurate problem definition and an extended problem model was needed to capture the complexity of the real world problems.

A common approach reported on the personnel rostering literature has been to develop and finetune solution methods for a single problem instance under study. The real world problem instances, however, vary significantly over countries, sectors and institutions. Even the problem instances of different departments in the same institution have significant distinctions. For example, the meal preparation and the emergency wards in a hospital do not have the same working regimes. Furthermore, the personnel rostering problem of the same institution varies over time. In the short term, the workload of institutions like hospitals have seasonal variances. In the long term, structural changes in the organisation, evolution of the labour

legislation and union demands result in modified personnel rostering problems. All these factors contribute to the diversity of the personnel rostering instances encountered in the real world practice. As a result, a solution method developed and finetuned to address a single problem instance does not answer the needs of the real world practice.

In order to be applicable in practice, a solution method must be able to cope with the complexity and diversity of the problem instances encountered. Such a method may not be able to come up with an optimal solution for every problem instance encountered. On the other hand, optimality is not the main concern of the real world practitioners. What they expect from a method is to find good enough solutions with minimum human effort in a limited amount of execution time. It is economically not sustainable for real world institutions to hire a group of specialists for several months to work on their problem in order to solve it to optimality.

This dissertation contributes by presenting a problem model and corresponding solution methods to solve a broad range of real world personnel rostering problems. The research has been carried out with two industrial partners to ensure the real world relevance[1]. The first step of the present research has been an extensive study of the problem model. The real world problem instances that our industrial partners encounter have been analysed in-depth and the standard academic models have been extended with various elements. The resulting problem model, reported in Chapter 3, has a higher potential to deal with the complexity and diversity of the real world problem instances. The extended model has been presented with thorough explanations and examples to guide developers wanting to implement a personnel rostering solution for real world problems. The model has been presented stressing its relations with standard academic models so that researchers with an academic background can easily understand and deploy it.

The problem model is a structured way of representing the problem data. In addition to the problem model, the rules of constraint evaluation are also needed for a complete problem description. The constraint evaluation mechanism utilised in the solution methods of this dissertation has been presented in Chapter 4. In order to be useful in practice, the constraint evaluation mechanism of a solution method must be accurate and consistent with the expectations of the end users. Even the smallest deviation or inconsistency in constraint evaluation can render the resulting solution method unacceptable in the real world.

The inconsistencies between the constraint evaluation of the real world practice and scientific literature have been studied in depth in Chapter 4. Most of these inconsistencies stem from ignoring the previous and upcoming schedule periods and trying to solve the problem in an isolated schedule period. Ignoring the enclosing schedule periods results in implicit assumptions about the situation in those periods.

[1]SAGA Consulting, GPS NV Belgium

Such assumptions cannot reflect the actual schedules accurately all the time. More important than that, these implicit assumptions are not consistent with each other in several cases. As a result, even the solutions claimed to be optimal based on assumptions may be unacceptable in practice according to the actual schedules in the previous and upcoming schedule periods.

It can be derived from the *no free lunch theorem* [71], that a single algorithm configuration cannot be expected to cope with all personnel rostering problems. In order to increase the probability of finding good enough solutions, the solution method in the present dissertation has been developed to deal with the real world personnel rostering problems and it can be configured in numerous ways. That entails the need for a decision mechanism to select the most promising configuration of the solution method on a given problem instance. Such a decision mechanism has been constructed as the combination of two steps, measuring the properties of the problem instance and using a mapping between the problem properties and the most promising configuration of the solution method.

Complexity indicators to measure problem properties have been introduced to academia [68]. However, the extended problem model requires more specific measures than the ones reported in the literature. Three quantitative measures have been developed (Chapter 5) to ensure an accurate measure of the size and constrainedness of a problem instance expressed in the extended problem model.

In the personnel rostering domain, researchers have mostly resorted to experiments on a set of problem instances to demonstrate the usefulness of the solution methods they propose. The test data utilised in the experiments have not always been published, which prevents the comprehension, verification and the reproduction of the results reported. The benchmark data sets have proven to be a useful tool as an entry point to a research field. They allow the verification and reproduction of the experimental results and the comparison of the solution methods reported. Moreover, they enforce a set of standards regarding the problem definition and structure.

The Nottingham benchmarks [20] are a collection of personnel rostering instances gathered from various resources and published online[2]. The diversity of its resources makes the Nottingham benchmarks a valuable sample of international personnel rostering problems. The researchers are also invited to add their own test data to the existing body of the Nottingham benchmarks.

A real world benchmark data set, the KAHO benchmarks, has been introduced in Chapter 6 and published online[3]. The KAHO benchmarks consist of test data provided by one of our industrial partners[4]. They originate from the Belgian

[2]http://www.cs.nott.ac.uk/∼tec/NRP/
[3]http://ingenieur.kahosl.be/vakgroep/it/nurse/archive.htm
[4]SAGA Consulting

healthcare sector. Although their origin can be perceived as focused, the measures of their problem properties (Table 6.1) indicate their complexity and diversity.

The solution methods reported in the present dissertation utilise heuristic methods. The NP-hard nature of the personnel rostering problems, the limited execution times allowed in the real world practice, and the complexity and the diversity of the real world problems have been the main motivations behind this choice. As it is pointed out earlier in the chapter, a single algorithm configuration cannot be expected to deal with all personnel rostering problems. However, limiting the targeted problem instances to the real world instances and utilising a variety of algorithms can increase the probability of finding good enough solutions. The solution method presented in Chapter 7 has been designed to utilise an algorithmic toolbox that can be configured in numerous ways. The algorithmic toolbox consists of variable neighbourhood search (VNS), adaptive large neighbourhood search (ALNS), hyperheuristics, and a set of neighbourhoods and low level heuristics. Basic neighbourhoods and low level heuristics have been utilised among the new ones that explore the structure of the extended problem model in various ways.

An extensive set of experiments have been carried out to find a significant relationship between the problem properties and the performance of various configurations of the algorithmic toolbox. The KAHO benchmarks have been used as test data for the experiments. In total 42 hyperheuristic, ten VNS and two ALNS configurations have been experimented with. The experimental results have shown that the algorithm settings deploying problem specific neighbourhoods in addition to the basic neighbourhood set perform significantly better than the algorithm settings with the basic neighbourhood set only. The most promising configuration of the algorithmic toolbox for a particular range of problem sizes and levels of constrainedness have been reported in Chapter 8. Except on the problem instances with large sizes, the hyperheuristic variations presented in this study have outperformed the VNS and ALNS variants. The resulting relations can be utilised in a decision support software to select the most promising configuration of the algorithmic toolbox to solve a given problem instance. The experimental results have been verified on the Nottingham benchmarks.

The problem model, evaluation method, quantitative measures and solution methods documented in the presented dissertation have been implemented in the personnel management software of the industrial partners of the research project. At the time of writing, these personnel management systems were deployed in more than 20 institutions in Belgium, mostly hospitals and rest homes, but also a retail store and a municipality including its fire department. The utilisation of the automated personnel rostering system in these institutions result in time and cost savings, because they do not need a human expert planner who has to analyse the problem instances, experiment with several algorithm configurations and utilise the most promising one. All of these tasks are automated in the solution proposed in the present dissertation. The resulting software provides a significant economic

value, because it can be duplicated and deployed easily in multiple sites, whereas this is not the case with a human planning expert.

The content of the present dissertation has been the subject of two journal papers [14, 66], a conference full paper [13], five extended conference abstracts [7, 9, 10, 56, 67], nine conference abstracts [6, 8, 11, 12, 51, 53, 54, 58, 70].

9.1 Future research

The personnel rostering problem is only one phase of the personnel planning problem, staffing and task planning being the other phases. Staffing refers to determining the size of the staff needed. It can be perceived as a forecasting practice. The solution methods presented in this dissertation can be used as a simulation tool in a staffing solution. Different workload scenarios can be tested against the actual workforce composition using the personnel rostering solution methods. Possible shortcomings or excesses of the workforce composition can be detected in different scenarios. Based on the staffing analysis, the size of the staff can be adjusted for a robust workforce composition. The third phase of the personnel planning problem, i.e., task planning, can be integrated with the personnel rostering approach reported. A set of task requirements can be included in the problem instance to be assigned to the employees. Upon calculation of the rosters of the employees, an automated task planning solution can assign these tasks to the schedules of the employees.

Although the experimental results of various algorithm configurations have been reported in the present dissertation, numerous algorithms can be applied to the personnel rostering problem. The KAHO benchmarks have been published online for encouraging other researchers to apply new or existing solution methods to the problem model introduced in the present dissertation[5]. Similar to the algorithm configurations, the quantitative measures of problem properties is also open to the contributions of the researchers in the field. Countless measures and algorithm configurations can be experimented with, in order to search for more accurate relations between the problem properties and algorithm performance. The test data for the experiments in the present dissertation has been the KAHO benchmarks. In order to generalise the experimental results to all personnel rostering problems, experiments based on a problem instance distribution can be executed. Such an experimental methodology to reach more general conclusions has been provided by Leyton-Brown et al. [43].

Personnel rostering is a dynamic research domain. The evolutions in the organisational structures, labour legislations and union demands will pose new challenges to personnel rostering. Future challenges can exceed the potential of the

[5] http://ingenieur.kahosl.be/vakgroep/it/nurse/archive.htm

research results reported in the present dissertation or in the scientific literature. As a consequence, modified problem models and solution methods will be necessary to address the new versions of the personnel rostering problems encountered. The author hopes that the present dissertation will be a valuable resource for the research and development activities on the future challenges of the personnel rostering domain.

Journal Papers

[1] Burak Bilgin, Peter Demeester, Mustafa Mısır, Wim Vancroonenburg, and Greet Vanden Berghe. One hyperheuristic approach to two timetabling problems in health care. *Journal of Heuristics*, accepted for publication, 2011.

[2] Burak Bilgin, Patrick De Causmaecker, Benoît Rossie, and Greet Vanden Berghe. Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research - Patat Special Issue*, 194(1):33–57, 2010.

[3] Peter Demeester, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice. *Journal of Scheduling*, 15:83–103, 2012.

[4] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

[5] Pieter Smet, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Modelling and evaluation issues in nurse rostering. *Annals of Operations Research*, 2012. Accepted for publication.

International Conference Papers

[1] Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. A hyperheuristic approach to Belgian nurse rostering problems. In *4th Multidisciplinary International Scheduling Conference, Dublin, Ireland*, pages 683–689, 2009.

[2] Burak Bilgin, Patrick De Causmaecker, Benoît Rossie, and Greet Vanden Berghe. Local search neighbourhoods to deal with a novel nurse rostering model. In *The 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal*, page 20p, 2008.

[3] Tommy Messelis, Stefaan Haspeslagh, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Towards prediction of algorithm performance in real world optimisation problems. In *Proceedings of the 21st Benelux conference on Artificial Intelligence (BNAIC). Eindhoven 29-30 October 2009*, pages 177–183, 2009.

Abstracts International Conferences

[1] Burak Bilgin, Peter Demeester, Greet Vanden Berghe, and Tony Wauters. A model and a hyperheuristic approach for automated assignment of patients to beds in a hospital. In *International Conference on Metaheuristics and Nature Inspired Computing (Meta 2008), Hammamet, Tunisia, 29-31 October 2008 2008*, 2008.

[2] Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Application of hyperheuristics to the nurse rostering problem in Belgian hospitals. In *23rd Annual Conference of the Belgian Operations Research Society (ORBEL 23), 05/02/2009, Leuven, Belgium*, 2009.

[3] Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Modelling questions in nurse rostering. In *24th Annual Conference of the Belgian Operations Research Society (ORBEL 24), 28/01/2010, Liege, Belgium*, pages 139–140, 2010.

[4] Burak Bilgin, Patrick De Causmaecker, Benoît Rossie, and Greet Vanden Berghe. The problem description and a solution method for the nurse rostering problem in Belgian hospitals. In *22nd Annual Conference of the Belgian Operations Research Society (ORBEL 22), Brussels, Belgium*, pages 37–38, 2008.

[5] Greet Vanden Berghe and Burak Bilgin. Constraint and precondition definitions for nurse rostering problems. In *EURO XXIV Lisbon 24th European Conference on Operational Research, Lisbon, 11-14 July 2010*, 2010.

[6] Tommy Messelis, Stefaan Haspeslagh, Burak Bilgin, and Patrick De Causmaecker. Algorithm performance prediction in a real world environment. In *Belgian-French-German Conference on Optimization Leuven 14-18 September 2009*, 2009.

[7] Tommy Messelis, Stefaan Haspeslagh, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Hardness studies for nurse rostering problems. In *ORBEL 23 Leuven 5-6 January 2009*, 2009.

[8] Mustafa Mısır, Burak Bilgin, Peter Demeester, Katja Verbeeck, and Greet Vanden Berghe. A hyper-heuristic approach to the patient admission scheduling problem. In *the 35th International Conference of Operational Research Applied to Health Services (ORAHS'09), Leuven,Belgium, 12-17 July 2009*, 2009.

[9] Tony Wauters, Katja Verbeeck, Burak Bilgin, Peter Demeester, and Greet Vanden Berghe. Learning automata for hyperheuristic selection. In *ORBEL 23 Leuven 5-6 February 2009*, 2009.

Technical Reports

[1] Burak Bilgin, Pieter Smet, and Greet Vanden Berghe. A mathematical model for a generic nurse rostering problem. Technical report, KAHO St.-Lieven, Information Technology, 2011.

[2] Tommy Messelis, Stefaan Haspeslagh, Burak Bilgin, Greet Vanden Berghe, and Patrick De Causmaecker. Algorithm performance prediction for a real world optimisation problem. Technical report, K.U. Leuven Campus Kortrijk, Department of Computer Science, KAHO St.-Lieven, Information Technology, 2010.

Bibliography

[1] Uwe Aickelin, Edmund K. Burke, and Jingpeng Li. An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. *Journal of the Operational Research Society*, 58(12):1574–1585, 2007.

[2] Mohamed N. Azaiez and S.S. Al Sharif. A 0-1 goal programming model for nurse scheduling. *Computers & Operations Research*, 32:491–507, 2005.

[3] Lotte Bailyn, Robin Collins, and Yand Song. Self-scheduling for hospital nurses: an attempt and its difficulties. *Journal of Nursing Management*, 15(1):72–77, January 2007.

[4] Jonathan F. Bard and Hadi W. Purnomo. Cyclic preference scheduling of nurses using a lagrangian-based heuristic. *Journal of Scheduling*, 10(1):5–23, February 2007.

[5] F. Bellanti, Giuliana Carello, Frederico Della Croce, and Roberto Tadei. A greedy-based neighborhood search approach to a nurse rostering problem. *European Journal of Operational Research*, 127(1):28–40, February 2004.

[6] Burak Bilgin, Peter Demeester, Greet Vanden Berghe, and Tony Wauters. A model and a hyperheuristic approach for automated assignment of patients to beds in a hospital. In *International Conference on Metaheuristics and Nature Inspired Computing (Meta 2008), Hammamet, Tunisia, 29-31 October 2008 2008*, 2008.

[7] Burak Bilgin, Patrick De Causmaecker, Stefaan Haspeslagh, Tommy Messelis, and Greet Vanden Berghe. Hardness studies for nurse rostering problems. In *Learning and Intelligent OptimizatioN LION 3, 14/01/2009, Trento, Italy*, 2009.

[8] Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Application of hyperheuristics to the nurse rostering problem in Belgian hospitals. In *23rd Annual Conference of the Belgian Operations Research Society (ORBEL 23), 05/02/2009, Leuven, Belgium*, 2009.

[9] Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. A hyperheuristic approach to Belgian nurse rostering problems. In *4th Multidisciplinary International Scheduling Conference, Dublin, Ireland*, 2009.

[10] Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Modelling issues in nurse rostering. In *Proceedings of the 8th International Conference on the Practice and theory of automated timetabling, Belfast, 10-13 August 2010*, volume 8, pages 477–480, 2010.

[11] Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Modelling questions in nurse rostering. In *24th Annual Conference of the Belgian Operations Research Society (ORBEL 24), 28/01/2010, Liege, Belgium*, pages 139–140, 2010.

[12] Burak Bilgin, Patrick De Causmaecker, Benoît Rossie, and Greet Vanden Berghe. The problem description and a solution method for the nurse rostering problem in Belgian hospitals. In *22nd Annual Conference of the Belgian Operations Research Society (ORBEL 22), Brussels, Belgium*, pages 37–38, 2008.

[13] Burak Bilgin, Patrick De Causmaecker, Benoît Rossie, and Greet Vanden Berghe. Local search neighbourhoods to deal with a novel nurse rostering model. In *The 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal*, page 20p, 2008.

[14] Burak Bilgin, Patrick De Causmaecker, Benoît Rossie, and Greet Vanden Berghe. Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research - Patat Special Issue*, 194(1):33–57, 2010.

[15] Burak Bilgin, Pieter Smet, and Greet Vanden Berghe. A mathematical model for a generic nurse rostering problem. Technical report, KAHO St.-Lieven, Information Technology, 2011.

[16] Peter Brucker, Edmund K. Burke, Tim Curtois, Rong Qu, and Greet Vanden Berghe. A shift sequence based approach for nurse scheduling and a new benchmark dataset. *Journal of Heuristics*, 16:559–573, 2010.

[17] Peter Brucker, Rong Qu, Edmund K. Burke, and Gerhard Post. A decomposition, construction and post-processing approach for a specific nurse rostering problem. In *Proceedings of Multidisciplinary International Conference on Scheduling : Theory and Applications, Aug, 2005, New York*, pages 397–406, 2005.

[18] Edmund K. Burke, Timothy Curtois, Gerhard Post, Rong Qu, and Bart Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2):330–341, 2008.

[19] Edmund K. Burke, Timothy Curtois, Rong Qu, and Greet Vanden Berghe. A time predefined variable depth search for nurse rostering. Technical report, School of Computer Science and IT, University of Nottingham, 2007.

[20] Edmund K. Burke, Timothy Curtois, Rong Qu, and Greet Vanden Berghe. A scatter search methodology for the nurse rostering problem. *Journal of the Operational Research Society*, 61:1667–1679, November 2010.

[21] Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. A classification of hyper-heuristics approaches. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer, 2010.

[22] Edmund K. Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In Fred Glover and Gary Kochenberger, editors, *Handbook of Metaheuristics*, volume 57, pages 457–474. Springer New York, 2003.

[23] Edmund K. Burke, Graham Kendall, and Eric Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, December 2003.

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

[25] Edmund K. Burke, Patrick De Causmaecker, and Greet Vanden Berghe. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, chapter Novel Metaheuristic Approaches to Nurse Rostering Problems in Belgian Hospitals, pages 44.1–44.18. CRC Press, 2004.

[26] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The State of the Art of Nurse Rostering. *Journal of Scheduling*, 7(6):441–499, 2004.

[27] Edmund K. Burke, Patrick De Causmaecker, Sanja Petrovic, and Greet Vanden Berghe. Fitness evaluation for nurse scheduling problems. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 1139–1146. IEEE Press, 27-30 2001.

[28] Peter I. Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach for scheduling a sales summit. In *PATAT 2000: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, volume 2079, pages 176–190, London, UK, 2001. Springer-Verlag.

[29] Kathryn A. Dowsland, Eric Soubeiga, and Edmund Burke. A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research*, 179(3):759–774, June 2007.

[30] Andreas T. Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, February 2004.

[31] Helle Frøyseth, Martin Stølevik, and Atle Riise. A heuristic approach for solving real world nurse rostering. *The 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal*, page 5p, 2008.

[32] Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In Edmund K. Burke and Patrick De Causmaecker, editors, *PATAT*, volume 2740 of *Lecture Notes in Computer Science*, pages 262–275. Springer, 2002.

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

[34] Fred Glover and Gary A. Kochenberger, editors. *Handbook of Metaheuristics*. Springer, 2003.

[35] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[36] Pierre Hansen and Nenad Mladenović. *Handbook of Metaheuristics*, chapter Variable Neighborhood Search, pages 145–184. Springer, 2003.

[37] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 1995, Montréal, Québec, Canada*, pages 607–614, 1995.

[38] Atsuko Ikegami and Akira Niwa. A subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3):517–541, 2003.

[39] Mark W. Isken. An implicit tour scheduling model with applications in healthcare. *Annals of Operations Research*, 128:91–109, 2004.

[40] Deborah L. Kellogg and Steven Walczak. Nurse Scheduling: From Academia to Implementation or Not? *INTERFACES*, 37(4):355–369, 2007.

[41] Graham Kendall and Mazlan Mohamad. Channel assignment in cellular communication using a great deluge hyper-heuristic. In *Proc. of the 2004 IEEE International Conference on Network (ICON2004)*, 2004.

[42] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942 –1948 vol.4, nov/dec 1995.

[43] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 91–100. Springer Berlin / Heidelberg, 2006.

[44] Broos Maenhout and Mario Vanhoucke. An electromagnetic meta-heuristic for the nurse scheduling problem. *Journal of Heuristics*, 13(4):359–385, August 2007.

[45] Broos Maenhout and Mario Vanhoucke. Comparison and hybridization of crossover operators for the nurse scheduling problem. *Annals of Operations Research*, 159:333–353, 2008.

[46] Patrick De Causmaecker. Towards a reference model for timetabling and rostering. In *The 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal*, page 10p, 2008.

[47] Patrick De Causmaecker and Greet Vanden Berghe. Relaxation of coverage constraints in hospital personnel rostering. *Lecture Notes in Computer Science*, 2740:129–147, 2003.

[48] Patrick De Causmaecker and Greet Vanden Berghe. A categorisation of nurse rostering problems. *Journal of Scheduling*, 14(1):3–16, 2011.

[49] Patrick De Causmaecker and Greet Vanden Berghe. Towards a reference model for timetabling and rostering. *Annals of Operations Research*, 194:167–176, 2012.

[50] Melanie L. De Grano, Deborah J. Medeiros, and David Eitel. Accommodating individual preferences in nurse scheduling via auctions and optimization. *Health Care Management Science*, 12:228–242, 2009.

[51] Greet Vanden Berghe and Burak Bilgin. Constraint and precondition definitions for nurse rostering problems. In *EURO XXIV Lisbon 24th European Conference on Operational Research, Lisbon, 11-14 July 2010*, 2010.

[52] Tommy Messelis, Stefaan Haspeslagh, Burak Bilgin, Greet Vanden Berghe, and Patrick De Causmaecker. Algorithm performance prediction for a real world optimisation problem. Technical report, K.U. Leuven Campus Kortrijk, Department of Computer Science, KAHO St.-Lieven, Information Technology, 2010.

[53] Tommy Messelis, Stefaan Haspeslagh, Burak Bilgin, and Patrick De Causmaecker. Algorithm performance prediction in a real world environment. In *Belgian-French-German Conference on Optimization Leuven 14-18 September 2009*, 2009.

[54] Tommy Messelis, Stefaan Haspeslagh, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Hardness studies for nurse rostering problems. In *ORBEL 23 Leuven 5-6 January 2009*, 2009.

[55] Tommy Messelis, Stefaan Haspeslagh, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Towards prediction of algorithm performance in real world optimisation problems. In *Proceedings of the 21st Benelux Conference on Artificial Intelligence*, number 21 in BNAIC, Eindhoven, October 2009.

[56] Tommy Messelis, Stefaan Haspeslagh, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Towards prediction of algorithm performance in real world optimisation problems. In *Proceedings of the 21st Benelux conference on Artificial Intelligence (BNAIC). Eindhoven 29-30 October 2009*, pages 177–183, 2009.

[57] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. *Solving Nurse Rostering Problems Using Soft Global Constraints*, pages 73–87. Springer Berlin / Heidelberg, 2009.

[58] Mustafa Mısır, Burak Bilgin, Peter Demeester, Katja Verbeeck, and Greet Vanden Berghe. A hyper-heuristic approach to the patient admission scheduling problem. In *the 35th International Conference of Operational Research Applied to Health Services (ORAHS'09), Leuven,Belgium, 12-17 July 2009*, 2009.

[59] S. Philip Morgan. Is low fertility a twenty-first-century demographic crisis? *Demography*, 40(4):589–603, November 2003.

[60] Takayuki Osogami and Hiroshi Imai. Classification of various neighborhood operations for the nurse scheduling problem. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, D. Lee, and Shang-Hua Teng, editors, *Algorithms and Computation*, volume 1969 of *Lecture Notes in Computer Science*, pages 72–83. Springer Berlin / Heidelberg, 2000.

[61] Sanja Petrovic, Gareth Beddoe, and Greet Vanden Berghe. Storing and adapting repair experiences in personnel rostering. *Lecture Notes in Computer Science*, 2740:148–165, 2003.

[62] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

[63] Gerhard Post and Bart Veltman. Harmonious personnel scheduling. In *Proceedings of the Fifth International Conference on Practice and Theory of Automated Timetabling (PATAT)*, pages 557–559, 2004.

[64] Elina Rönnberg and Torbjörn Larsson. Automating the self-scheduling process of nurses in Swedish healthcare: a pilot study. *Health Care Management Science*, 13:35–53, 2010.

[65] Rhian Silvestro and Claudio Silvestro. An evaluation of nurse rostering practices in the national health service. *Journal of Advanced Nursing*, 32(3):525–535, 2000.

[66] Pieter Smet, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Modelling and evaluation issues in nurse rostering. *Annals of Operations Research*, 2012. Accepted for publication.

[67] Wim Vancroonenburg, Mustafa Mısır, Burak Bilgin, Peter Demeester, and Greet Vanden Berghe. A hyper-heuristic approach for assigning patients to hospital rooms. In *Proceedings of the 8th International Conference on the Practice and theory of automated timetabling, Belfast, 10-13 August 2010*, Belfast, Northern Ireland, 10/08/2010 2010.

[68] Mario Vanhoucke and Broos Maenhout. On the characterization and generation of nurse scheduling problem instances. *European Journal of Operational Research*, 196(2):457–467, 2009.

[69] Zhiguo Wang and Chun Wang. Automating nurse self-rostering: A multiagent systems model. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 4422–4425, 2009.

[70] Tony Wauters, Katja Verbeeck, Burak Bilgin, Peter Demeester, and Greet Vanden Berghe. Learning automata for hyperheuristic selection. In *ORBEL 23 Leuven 5-6 February 2009*, 2009.

[71] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Science
Department of Computer Science
Research group CODES
Etienne Sabbelaan 53, 8500 Kortrijk