Aalto University
School of Science
Master's Programme in Mathematics and Operations Research

Jari Hast

Optimal work shift scheduling:

a heuristic approach

Master's Thesis
Espoo, April 19, 2017

Supervisors: Professor Harri Ehtamo, Aalto University
Advisor: M.Sc. Santtu Klemettilä, Haahtela HR

Aalto University
School of Science
Master's Programme in Mathematics and Operations Research

ABSTRACT OF MASTER'S THESIS

Author: Jari Hast

Title:
Optimal work shift scheduling: a heuristic approach

Date:	April 19, 2017	**Pages:**	54
Major:	Systems and Operations Research	**Code:**	SCI3055
Supervisors:	Professor Harri Ehtamo, Aalto University		
Advisor:	M.Sc. Santtu Klemettilä, Haahtela HR		

Work shift planning is a task of assigning work force to tasks in order to satisfy a business defined demand. The result of a planning task is a work shift schedule usually ranging over several weeks, sometimes called a roster. In addition to filling the demand, a roster has to satisfy certain varying criteria including but not limited to mandatory holidays, sufficient resting hours and employees' wishes.

Creating a roster by hand becomes an arduous task as the number of employees and shifts increase. Organizations in work force intensive industries often have to employ one or more work shift planners solely for this purpose. In Finland, the numerous rules defined by the Finnish collective agreements are particularly tricky to deal with. This leads to suboptimal work shift schedules that have a negative effect on the business as well as employee satisfaction.

For these reasons, an on-line optimization algorithm for scheduling work shifts has been created in Haahtela® Contactor, a software as a service product sold by the company Haahtela HR Oy. This algorithm is implemented as a linear model. While functional and in frequent production use, it suffers from inherent drawbacks associated with the method. For instance, the linear cost function is not an ideal way of conveying the actual preferences of work shift planners.

This thesis sets out to devise a novel approach to this problem by introducing a heuristic based algorithm. This method is compared with the incumbent linear model using a real world test case. While the heuristic is found to be inferior compared to the linear model, promising results are achieved. After the numerous enhancements suggested in this thesis are implemented to the new heuristic, it can be tested in production use.

Keywords: heuristic, metaheuristic, work shift scheduling, rostering, simulated annealing

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Matematiikan ja operaatiotutkimuksen maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Jari Hast		
Työn nimi:	Työvuorosuunnitelun optimointi heuristisin menetelmin		
Päiväys:	19. heinäkuuta 2017	**Sivumäärä:**	54
Pääaine:	Systeemi- ja operaatiotutkimus	**Koodi:**	SCI3055
Valvojat:	Professori Harri Ehtamo, Aalto University		
Ohjaaja:	Diplomi-insinööri Santtu Klemettilä, Haahtela HR		

Työvuorosuunnittelussa käytettävissä oleville työntekjöille jaetaan ennalta määriteltyjä tulevaisuuden työvuoroja tarkoituksena tyydyttää liiketoiminnallinen työvoiman tarve. Toiminnan tuloksena syntyvä työvuorolista on yleensä viikkojen mittainen. Työvuorolistan pitää myös täyttää useita vaatimuksia, kuten riittävää lepoaika, lomia ja työntekijöiden toiveita.

Kun työntekijöiden ja työvuorotarpeiden määrä kasvaa, käy työvuorosuunnittelu usein erittäin työlääksi. Työvoimaintensiivisillä aloilla on yleensä tarpeen palkata yksi tai useampi työvuorosuunnittelija pelkästään työvuorolistojen tuottamiseen. Erityisesti Suomen moninaiset yleissitovat työehtosopimukset asettavat työvuorosuunnittelua monimutkaistavia sääntöjä. Nämä seikat johtavat alioptimaalisiin työvuorolistoihin, joilla on negatiivisia vaikutuksia sekä yritysten liiketoimintaan että työntekijöiden työtyytyväisyyteen.

Haahtela HR Oy:n myymässä SaaS-ohjelmistossa Haahtela® Contactorissa työvuorolistojen optimointi on toteuttu lineaarisena ongelmana. Vaikka ratkaisu on todettu toimivaksi ja se on yleisesti tuotantokäytössä, on siinä useita varjopuolia. Esimerkiksi lineaarinen hyötyfunktio on puutteellinen tapa ilmaista työvuorosuunnittelijoiden toiveita työvuorolistasta.

Tämän työn tarkoituksena on luoda uusi heuristinen ratkaisutapa kyseiseen ongelmaan. Uutta metodia verrataan tämänhetkiseen ratkaisutapaan optimoimalla erään oikean organisaation todellista työvuorolistaa. Vaikka uusi algoritmi ei muodostakaan yhtä hyviä työvuorolistoja kuin nykyinen lineaarinen malli, ovat tulokset lupaavia. Kun kaikki tässä työssä esitetyt kehitysehdotukset uuteen algoritmiin toteutetaan, voidaan sen suorityskykyä testata tuotantokäytössä.

Asiasanat:	heuristiikka, metaheuristiikka, työvuorosuunnittelu, simuloitu jäähdytys
Kieli:	Englanti

Acknowledgements

First and foremost I want to thank my instructor Santtu Klemettilä who helped me a great deal along the way. While assisting me with technical problems and the writing process, he also provided me with some much needed refreshment on several occasions. In the same vein, I am grateful to my supervisor professor Ehtamo for giving me firm and straightforward guidance.

I also want to thank my family and friends for supporting me and listening to my worries. Especially I would like to mention my mother Satu and our family dog Spede, whose stress relieving talents are second to none. Two other specials mentions go to the participants of WhatsApp groups '0735MaKePe' and 'Dota draft'.

Lastly, I want to thank the folks at Haahtela HR, who were supportive of my efforts and gave me time off from work when it was necessary.

Helsinki, April 19, 2017

Jari Hast

Abbreviations and Acronyms

CP	Constraint programming
HR	Human resource
MILP	Mixed integer linear programming
NRP	Nurse rostering problem
SA	Simulated annealing
SaaS	Software as a Service
TDD	Test driven development
VNS	Variable neighbourhood search
VNC	Variable neighbourhood descent

Contents

Abbreviations and Acronyms **5**

1 Introduction **8**
1.1 Problem statement 8
1.2 Research problem 8
1.3 Structure 9

2 Background **11**
2.1 Work shift planning 11
2.2 The requirements 13
2.3 The current solution 14
2.4 Overview of the literature 15
2.5 Constraint programming 15
2.6 Mixed linear integer programming 18
2.7 Metaheuristics 19
2.8 Hybrid methods 21

3 Methods **22**
3.1 Heuristic approach 22
3.2 Justification for the choice of heuristics 23
3.3 Foreseeable challenges 23

4 Implementation **25**
4.1 Problem specific concepts 25
4.2 Hard constraints 26
4.3 Soft constraints 26
4.4 The implementation process 27

5 Evaluation **34**
5.1 Evaluation with a real world test set 34
5.2 Problem model 35

5.3 Results 40

6 Discussion **42**
6.1 Result analysis 42
6.2 Viability of the solution 43
6.3 Next steps of development 44

7 Conclusions **45**
7.1 Summary 45
7.2 Future research 45

A Example of typical constraints in NRP **51**

B Terms of employment in the commercial sector **53**

Chapter 1

Introduction

1.1 Problem statement

Work shift planning is a task of assigning work force to tasks in order to satisfy a business defined demand. The result of a planning task is a work shift schedule usually ranging over several weeks, sometimes called a roster. In addition to filling the demand, a roster has to satisfy certain varying criteria including but not limited to mandatory holidays, sufficient resting hours and employees' wishes.

Creating a roster by hand becomes an arduous task as the number of employees and shifts increase. Organizations in work force intensive industries often have to employ one or more work shift planners solely for this purpose. In Finland, the numerous rules defined by the Finnish collective agreements are particularly tricky to deal with. This leads to suboptimal work shift schedules that have a negative effect on the business as well as employee satisfaction.

1.2 Research problem

The aim of this thesis is to create an automated on-line optimization routine for work shift planning. This aim stems from a business need for this kind of feature to an existing human resources management software.

As of now, the optimization of the roster is implemented as a linear mixed integer problem. This approach suffers from inherent drawbacks associated

with the optimization method. This aspect is further elaborated in section 2.3. This thesis sets out to devise a proof-of-concept version of an alternative solving method.

When dealing with an on-line optimization service, i.e. the user can decide when to run the routine and expects to receive the results in a reasonable time, some properties of the algorithm such as its running time are critical for the end-product of this thesis. On the other hand, finding the global minimum of the problem is typically not regarded as an important feature. The reasoning for this argument is presented in section 2.2.

In this thesis, the chosen approach is to use a heuristic optimization algorithm. Heuristics are suitable for large problems when a reasonably good local optimum suffices as a solution [Talbi, 2009]. There are already countless different heuristic algorithms devised for the work shift assignment problem and the process of choosing the appropriate one is a potentially time consuming task. Thus, simplicity is favoured in the evaluation of different approaches. The implementation process along with the justifications of using certain methods is documented in section 4.4.

Work shift optimization has been researched extensively. Probably the most visible occurrence of this is the international nurse scheduling competition [Haspeslagh et al., 2014]. The literature is focused on the more simple instances of the problem where, for instance, a working day contains only three fixed time slots for shifts.

1.3 Structure

In the following chapter 2, a literature review of different methods in solving the nurse rostering problem is conducted. The chapter also lists the qualities required from the chosen optimization method as well as presents the shortcomings of the incumbent linear model.

Chapter 3 gives an overview of the selected heuristic method and the justifications for the selection. In addition, the potential pitfalls of this approach are also reflected upon.

A more thorough review of the method is given chapter 4. In this chapter, the implementation process of the heuristic is also documented.

The algorithm is then tested on a real world dataset in chapter 5. The dataset is anonymized to a certain extent but the most important characteristics are

described. The performance of the novel method is compared with the linear model when applied on this set of data.

The results and their evaluation are given in chapter 6. The viability of this method in general is also discussed. As the devised method is currently at the proof-of-concept level, the next development steps to increase its performance and to complete it functionality-wise are listed.

In the last chapter 7, the whole thesis and its results are summarized. If the development of the algorithm is continued, this chapter offers several interesting long-term directions of research.

Chapter 2

Background

This thesis is written at Haahtela HR, which is a subsidiary company under Haahtela Oy. The company produces a SaaS-licenced HR-management software entitled Haahtela® Contactor that sets the context for this thesis.

Most of the literature available is focused on the nurse rostering problem. Although the problem is less complex, the same approaches present in the academia are applicable in this thesis' context as well.

2.1 Work shift planning

In the context of this thesis, rostering is a process of assigning workers to predefined work shifts in order to satisfy a demand created by the underlying business. The time span that contains these shifts is called a planning period. For each shift, a competence is defined along with the starting and ending times between which the shift is to be completed. For employees to be able to fill this shift, they must have the corresponding competence. A shift might also contain a need for a more specific skill or qualification, such as a hot work permit, that acts as an additional restriction for selecting a worker.

Work shift planning is done under certain rules and preferences, which can be described in hard and soft constraints. A solution or a roster will be considered feasible if it does not violate hard constraints. Fulfilling soft constraints on the other hand increases the quality of the solution. Hard constraints usually stem from legislation or physical constraints e.g. an employee can not or is not legally allowed to complete a 24h shift. Soft constraints may

aim to make the roster more ergonomic by, for instance, avoiding consecutive night and morning shifts. [Smet et al., 2013] [Deb, 2014]

This thesis concentrates on non-cyclic scheduling as opposed to cyclic scheduling. In cyclic scheduling, one roster is repeated in every scheduling period. While not guaranteeing an optimal work shift plan, cyclic scheduling was the only viable approach at the time when computers lacked the adequate processing power. In non-cyclic scheduling, a unique roster per se is created for every new scheduling period. [Cheang et al., 2003]

There are several ways of organizing the creation of rosters. [Silvestro and Silvestro, 2000] identify three different approaches of scheduling hospital staff. In *departmental rostering* a single person is responsible for creating rosters (although an authorization might be required from a senior manager). A more decentralized model is *team rostering*, where the staff is split up into teams. For each team, one member is appointed to create a roster for the team. The team leaders then evaluate if the entire roster is feasible and possibly acquire an approval from a senior manager before releasing the rosters. Lastly, in *self-rostering* the staff themselves plan a work shift schedule which is then typically authorized by a senior manager. [Silvestro and Silvestro, 2000]

[Silvestro and Silvestro, 2000] identify several characteristics that affect the complexity of rostering:

- The number of staff
- Predictability of demand measured in the ratio of planned and emergency operations
- Demand variability is based on the variability in the length of patient stay and the degree of variation in the manning requirement over a working week.
- Complexity of skill mix is measured in terms of the degree of variation in staff grades and the complexity of the manning requirement specification.

Typically, when the number of staff increases, the demand variability and skill mix complexity increase whereas the predictability of demand decreases. Thus, the complexity of the rostering problem correlates strongly with the number of workers. [Silvestro and Silvestro, 2000]

[Silvestro and Silvestro, 2000] found that from the perspective of the management the choice of optimal rostering method depends on the complexity

of the rostering problem. Small hospital wards with a staff count of 35 or less seem to benefit most from self-rostering, that empowers and increases staff motivation. However, this method becomes unmanageable with a higher number of workers. When the staff count is between 35 and 70, team rostering is seen as the best alternative whereas departmental rostering should be used for the most complex problems.

Work shift planning has a significant effect on the company's personnel costs especially in workforce intensive industries. Constant unergonomic work shift scheduling might lead to increased sick leave rates and decreased productivity among the personnel. While the Finnish law and the collective agreements define most of the constraints concerning the working hours, many organizations have a set of their own rules with which they try to increase job satisfaction. By adopting centralized work shift planning in a hospital environment, [Wright and Mahar, 2013] found the desirability of the schedules to rise approximately 34% while seeing a decrease of costs by more than 10%.

2.2 The requirements

There are certain criteria to be satisfied As the optimization routine is online, the results of the algorithm should be obtained in a relatively short time. The defining factor for the running time of the algorithm is the satisfaction of the end customer and the user of the software in question.

Another aspect to elaborate is the difficulty of defining an optimum roster i.e. choosing the objective function. Rostering can be seen as a multi-objective problem where there are several objective functions to optimize. However, typically the objectives are cast into a single function. [Deb, 2014]

Practice has shown that there are various different sets of rostering preferences among work shift planners. Even if it was possible to work with only one preference set, the elicitation of all the factors is exceedingly difficult because of the sheer amount of different scenarios. For example, is it better to fulfill one employee's wish to work on a certain day if another employee then needs to work on a weekend. Moreover, the elicitated weights might not always be static under, for instance, a variable number of available staff.

Therefore, the role the soft constraints is actually not that significant. The optimization of the roster is often thought more as an automation than actual optimization. If a roster with hundreds of shifts is fulfilled completely, it is

very difficult for a shift planner to tell if it is indeed optimal or not. In this kind of situation, where even producing any kind of a satisfactory schedule by hand is exceedingly laborious, the automation of creating a roster brings most of the added value of this feature to the user.

2.3 The current solution

In the software that sets the context for this thesis, the automatic rostering is currently modeled as a mixed integer problem. The modeling of the problem has been implemented in the software while the actual solving of the linear model is carried out by a third party solver. This manner of approach has proved itself functional but there are several drawbacks present.

Firstly, exact optimization methods (as opposed to approximate ones) are guaranteed to find optimal solutions, but typically suffer from long running times since the algorithms are non-polynomial-time. On the other hand, approximate methods are known to produce high-quality but non-optimal results in a relatively short running time [Talbi, 2009].

While the third party linear solver does have built-in heuristics, the search space simply grows too large to solve when the number of employees and shifts grow too large. As the optimal solution the rostering problem is ambiguous and fast running times are desirable, approximate algorithms are seemingly the better approach.

Another issue lies within the linearity. Many of the soft constraints are inherently non-linear. For example, the even distribution of shifts to employees is usually desirable. Not assigning any shifts to one employee is sometimes seen a considerably more severe deficiency than having a one or two shift differences among the pool of employees.

The software in question allows for manual work shift planning. When assigning a shift to an employee manually, the feasibility of the roster, i.e. the hard constraints have to be checked. However, these checks are not run through the linear solver for practical reasons. This means that the software has a duplicate set of constraints. Consequently all new specifications to the constraints have to be implemented twice. Moreover, all software engineers might not be educated in the field of linear programming making the maintenance of the software further burdensome. In addition, the license to the third party solver is seen as expensive from the business point of view.

2.4 Overview of the literature

Although there are numerous variations of the nurse rostering problem, it is proven to be a difficult optimization problem in addition to being NP-complete. This is true for most of the scheduling problems. Computers have been used to solve scheduling problems since the dawn of first commercial computers in 1950. However, only the recent decades have seen applications that fit for real-world use. [Kyngäs, 2011] [Goos et al., 2000] [Ásgeirsson et al., 2011]

Most methods found in the literature can be grouped in three different categories. *Exact methods*, such as linear integer and constraint programming, allow finding of optimal solutions but are costly in terms of running time and complexity. *Heuristic methods*, on the other hand, yield close to optimal solutions in a relatively short time. Heuristic methods include but are not limited to problem specific heuristics and a number of different metaheuristics. Finally, *hybrid methods* (or matheuristics [Boschetti et al., 2009]) are a mixture of the two former. A hybrid approach might, for instance, try to optimize an easier sub-problem with an exact algorithm and improve the solution with a heuristic local search. Other, more uncommon, approaches to the nurse rostering problem are AI-based planning [Spyropoulos, 2000] and simulation [Sajadi et al., 2016].

2.5 Constraint programming

Rostering problems can be modeled as a constraint satisfaction problem alias CSP. A CSP is defined as

$$(V, D, C), \tag{2.1}$$

where $V = \{v_i,\ \ i \in [1, n]\}$ is the set of variables. For each variable, there is a domain $d_i \in D$ that defines the possible values with corresponding index i. Lastly $C = \{c_j,\ \ j \in [1, m]\}$ defines a set of constraints that all have to be satisfied in order for the solution to be viable. The triplet (2.1) is also called a constraint network. A solution to this problem is a complete assignment of variables while satisfying all the constraints.

A domain might be defined as the set of all natural numbers or even the set $d_1 = \{\text{coffee}, \text{tea}\}$. A Constraint on the other hand could simply be stated

as a linear constraint such that $v_2 > v_3$.

Much like other methods in this section, the principle in constraint programming is to express the problem declarative and let a general purpose solver generate a solution. Constraints can be thought as relations between the variables.

There are a number of methods for solving constraint satisfaction problems. An essential method is called *backtracking*. This method is a depth-first search for the correct assignment of variables such that it is guaranteed to find a solution in the case of there being any. In the most basic form, backtracking constructs a solution by selecting one variable at a time and assigns a value to it. If the algorithm encounters a situation where no value can be assigned without violating a constraint, it retracts the last step made and tries again with a different step. [Rossi et al., 2006] [Dechter, 2003]

Constraint propagation is a broad concept that incorporates any reasoning that restricts values or combinations of them for some variables. It is used to reduce the search space of the problem in order for it to be more easily explored by a search algorithm. [Rossi et al., 2006]

A clarifying example of constraint propagation is presented in [Rossi et al., 2006]:

> In a problem containing two variables x_1 and x_2 taking integer values in 1..10, and a constraint specifying that $|x_1 - x_2| > 5$, by propagating this constraint we can forbid values 5 and 6 for both x_1 and x_2

The above quote is an example of *rules iteration*. Along with the most well-known method of *local consistency*, these are two ways of formalizing constraint propagation. Rules iteration specify the operations to be executed on the problem whereas local consistencies only define the properties that are to be satisfied after constraint propagation has taken place. [Rossi et al., 2006]

An example of local consistency is arc-consistency. Take a constraint network of two variables x and y with their respective domains D_x and D_y along with a set of constraints C. The two variables are arc-consistent iff for every assignment of a value for variable x in the domain of D_x there is value for y in the domain D_y so that every constraint is C is satisfied. Because the whole network consisted of only these two variables, the whole network is arc-consistent and also globally consistent.

Figure 2.1 represents a network of three variables and a set of constraints

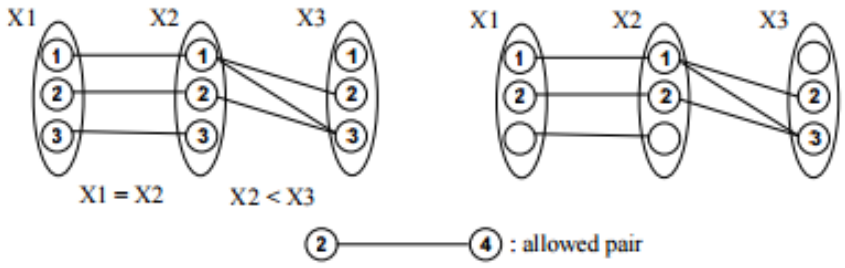

Figure 2.1: A matching diagram for three variables [Rossi et al., 2006]

related to them. The network on the right side pictures the network on the left after arc consistency has been applied. This type of graph is called a matching diagram by [Dechter, 2003] although the original source [Rossi et al., 2006] does not use this term.

Local search algorithms such as min-conflict presented by [Minton et al., 1992] have similar properties with heuristics. They cannot guarantee finding a solution but can outperform systematic procedures such as backtrack [Jussien and Lhomme, 2002]. While systematic methods start from an empty solution and construct a complete one (if possible), local search algorithms try to repair an infeasible complete assignment by partial exploration of the search space [Jussien and Lhomme, 2002].

Although CP is flexible and an easily extendable method, standard techniques are often not enough to solve NP-hard problems [Wong and Chun, 2004]. That is why in [Wong and Chun, 2004] the authors present a technique dubbed meta-level reasoning and probability-based programming (MR and PO respectively).

Meta-level reasoning (MR) is a process that deduces new implicit and redundant constraints from the already defined ones [Wong and Chun, 2004]. To give an example of how MR works, let there be three shift types: morning (M), evening (E), night (N) and day-off (DO). Suppose a rule stating that night shift has to be followed by another night shift or a day-off. This rule can be expressed in patterns [N,N] and [N,DO], which describe the activities in the range of two working days. The basic idea of MR, is to extend on these patterns. For example, from these two patterns one could introduce two new patterns [N,N,N] and [N,N,DO].

While MR is executed before the search, probability-based programming (PO) is used during it. In PO heuristic, each constraint is associated with

a scoring function to evaluate the probability of an assignment of a certain value to a variable. [Wong and Chun, 2004]

Both techniques exhibit a similarity in the sense that they consider the properties of the constraints before the actual constraint propagation. This helps to prevent situations, where backtrack must be initiated from deep within the search tree. [Wong and Chun, 2004]

2.6 Mixed linear integer programming

Constraint programming and linear/integer programming are two very classic approaches in solving optimization problems. Both methods aim at an optimal solution, but suffer from long execution times. Thus, the actual solving procedure usually utilizes some kind of heuristics. For example, integer decision variables could be modeled as continuous and then rounded to the nearest integer instead of solving a full branch and bound algorithm. [Aickelin, 2010]

There are several possible ways of modeling the decision variables of the linear problem [Aickelin, 2010]. Perhaps the most straightforward approach is to use binary variables x_{ij} to indicate if nurse i works on shift j. Other examples mentioned by [Aickelin, 2010] are to expand the binary variable to make it cover whole shift patterns or to convert it to an integer one in order to represent the number of workers working on a particular shift. The most suitable approach depends on the problem and the objectives of the research problem.

An approach typically preferred when modelling a scheduling problem into a linear of integer problem is to use the set covering formulation (referring to the set covering problem). While this method is flexible, it tends to generate a vast number of variables. To resolve this issue, several decomposing techniques and heuristic algorithms should be applied. [den Bergh et al., 2013]

Decomposing methods typically split the problem into two halves. The other half contains the "easier" constraints while the other has the more "complicated" constraints. [den Bergh et al., 2013]

Column generation allows for large linear programming models to be solved to optimality while holding only a subset of the original problems decision variables. In each iteration step, a *restricted master problem* containing this

subset is constructed and solved. The dual vector of the current solution is then used to generate a pricing sub-problem to identify a new variable with a reduced cost. This variable is then added to the master problem. [den Bergh et al., 2013]

2.7 Metaheuristics

Heuristics can be roughly divided into two categories: specific heuristics and metaheuristics. Specific heuristics are designed to solve a single specific problem or instance whereas metaheuristics can be practically tailored to solve any kind of problem. In general, heuristics cannot guarantee any solution quality or run-time bounds. However, they have a potential to produce "good enough" solutions with small computing costs. [Talbi, 2009]

There are also several other drawbacks associated with metaheuristics. They can't reduce the the search space nor do they have well defined stopping criteria. Furthermore, metaheuristics have difficulties in operating in a search space where feasible regions are disconnected i.e separated by an infeasible area, which is exactly the case with the highly constrained scheduling problems. [Burke et al., 2010]

Metaheuristics are approximate optimization algorithm frameworks that are often inspired by natural processes. A metaheuristic can be described as a high-level strategy or as a general algorithmic framework that can be applied to a variety of problems with only a few modifications. They are commonly used when faced by a complex problem that cannot be solved by exact algorithms in reasonable time. [Talbi, 2009]

Simulated annealing (SA) is probably one of the most known metaheuristics. It is a local search algorithm that is based on the physical analogy of annealing solids slowly to achieve the minimal energy configuration of the particles [Eglese, 1990]. It is typically used for solving combinatorial optimization problems. SA can be viewed as an extension to the hill climbing algorithm or the descent algorithm as it is dubbed in [Eglese, 1990].

In descent algorithm, one starts with an initial, possibly empty or random, solution. By some means, a neighbor to this solution is generated and the utility of it is calculated. If the utility is greater than the current solution's, the current solution is replaced by the neighbor. This process is repeated until a neighbor with better utility cannot be found. While this algorithm

is simple, its greatest weakness is getting stuck at a local minimum. [Eglese, 1990]

In simulated annealing, this flaw is mitigated by adding randomness to the acceptance of neighboring solutions with lesser utility. This probability is typically set to $exp(-\delta/T)$, where δ is the decrease in utility and T is the "temperature" parameter, that is set to decrease over time. This acceptance function is constructed so that smaller decreases in utility are more likely to be accepted and that the probability to accept worse solutions is highest at the first steps of the algorithm. Typically, several neighboring solutions are tested for each temperature level. [Eglese, 1990]

Variable neighbourhood search (VNS) is also a non-deterministic metaheuristic that differs from other local search based search heuristics by changing the explored neighbourhood over the course of the execution. [Hansen and Mladenovič, 2001]

There are many variations to VNS, but the basic one can be described as follows. First, let us denote a set of pre-selected neighbourhood structures as $N_k(k = 1, ..., k_{max})$ and the set of solutions in the kth neighbourhood of x as $N_k(x)$. Before the actual execution of the algorithm, the set of neighbourhood structures along with the stopping condition have to be selected and an initial solution x is to be found. [Hansen and Mladenovič, 2001]

The first step of basic VNS is *shaking*. In the shaking phase a random solution x' is generated from the current neighbourhood N_i, $i \in [1, k_{max}]$, where i equals 1 in the in the beginning of the algorithm. Then a local search is applied generating a solution x''. If x'' is a better local optimum than the incumbent solution x, it is selected as the current solution and search is continued in the same neighbourhood. If not the search continues with shaking in another neighbourhood $i \leftarrow i + 1$. These steps are continued until the stopping condition is reached. [Hansen and Mladenovič, 2001]

VNS explores increasingly distant solutions from the starting point and changes neighbourhoods only if an improvement is possible. This aspect is advantageous because it preserves most of the variables that are at their optimal value. Roughly speaking, the systematic exploration of the different neighbourhoods is practically the only aspect that sets this algorithm apart from a basic local search. Also note how the undeterministic nature of procedure prevents cycles. In order to generate the different neighbourhoods, a metric has to be introduced to measure the distance between different solution. [Hansen and Mladenovič, 2001]

2.8 Hybrid methods

A good example of a hybrid method for solving the nurse rostering problem is presented in [Burke et al., 2010]. In this approach, a sub-problem containing all the hard constraints and a subset of soft constraints is solved with integer programming. The solution is further enhanced with variable neighbourhood search. The subset of the soft constraints included in the sub-problem is determined by constraint complexity, importance or their trade-off. Complexity is defined as the number of variables and constraints that would be added to the problem while the importance factor is determined by the hospital staff.

The constraints presented in this article are a good example of the typical constraint set in the literature. They are presented in appendix A.

The authors in [Burke et al., 2010] argue that the choice of the heuristic algorithm should be simple. This is the reason for selecting VNS. They select the neighbourhood structure as a group of consecutive shifts of two nurses. The time frame of consecutive shifts ranges from one day to the whole scheduling period. The longer the time frame of the consecutive shifts is, the further the neighbourhood is regarded to be. As in the integer problem, this heuristic respects the hard constraints of the problem.

Chapter 3

Methods

3.1 Heuristic approach

The scheduling algorithm deployed in the context of this thesis is a two-stage heuristic. In the first phase, a greedy search is conducted to attain the initial solution. Then, simple swap heuristics are used to enhance the solution.

The simple nature of the solving method yields at least two tangible benefits: the ease of development and rapid prototyping. These aspects are important because in the literature there are no examples of this exact problem type.

To be exact, most of the literature focuses on the NRP sub-problem where shift types can be classified to a fixed number of categories i.e. morning and evening shifts. In the context of this thesis, shifts can begin and end at arbitrary times, which changes the nature of the problem radically. Therefore, in order to find an adequate scheduling heuristic, several iterations of trial and error are most likely needed, since there are virtually no ready-made solutions for this kind of task. Moreover, most scientific articles implement their solution in the context of one hospital ward or other specific problems instance. In this case, the algorithm should be applicable to many different collective agreements that are defined in Haahtela® Contactor.

The emphasis on the straightforwardness of the heuristic can also be based on the fact that the increased complexity of the algorithm does not imply better results. Even in the case where a complex algorithm would successfully be implemented, it should be benchmarked against a simpler one in order to

validate the benefits. In the same vein, this thesis should at least accomplish the tasks of establishing a valid benchmark for possible future iterations of the scheduling algorithm.

3.2 Justification for the choice of heuristics

The promises of (meta)heuristics, namely fast solving times and solutions of adequate quality, are a good fit for the work shift scheduling problem. Metaheuristics are very commonly used in nurse rostering problems [Burke et al., 2008] [Asta et al., 2016].

There are also other benefits in the context of software development. If the scheduling algorithm was written in the C# language as the rest of the software, the coherence and maintainability of the software can be increased. The solving procedure could use the common routines and methods already available eliminating the need for duplicate logic.

As metaheuristics are at their best when they are simple but powerful, there is a chance that developers not endorsed in any formal mathematical education could be empowered to maintain the scheduling algorithm. This would diversify the risks for the company because there would be no need to employ mathematically oriented programmers, who are a more scarce resource in the job market.

3.3 Foreseeable challenges

While C# and .NET framework are battle tested and mature, they are not particularly designed for computationally intensive code. This might represent a challenge as the search space is large. The metaheuristics need to be implemented especially well because of this.

Furthermore, the choice of the metaheuristic must also match the problem structure and the structure of search space. As there are several possible collective agreements which define different rules for the algorithm, there is a possibility that one algorithm is not sufficient for each problem case. Even the demand for the workforce in itself might be totally different along with the business need in separate problem instances. For example, in some cases there could an oversupply of the workforce whereas in another instance scarce employee resources might prevent the total fulfilment of demand.

Since there are several different algorithms available, one must somehow identify the problem structure and match it with an appropriate algorithm, which is not a trivial task. Even in the context of one algorithm, one might need to tune the implementation of the metaheuristic. For instance, in the case of the Artificial bee colony metaheuristic there are several parameters, such as the number of different types of bees, to be defined [Karaboga and Basturk, 2007].

Chapter 4

Implementation

4.1 Problem specific concepts

The planning period, also known as planning horizon, is the interval of time, in which the the shifts are to be assigned to employees, when creating a future roster. In a typical scenario, the length of the planning period is fixed.

In Finnish collective agreements, the concepts of *period* and *leveling period* usually define the planning period, since the work shift schedules are planned for one period at a time. The length of a period is typically one or several weeks. The levering periods, which usually contain several periods, can span to even six months.

Usually, most of the various rules and constraints of a roster are associated with periods and leveling periods. For example, take the following fictional rule: the maximum working hours of a regular employee during one week can reach a maximum of 40 hours, but when averaged for the whole period, the number of working hours in a week can't exceed 37.5.

Although the planning period is typically the length of a period, the automatic scheduling tool should also be able to function on an arbitrary time interval. This is because of the normal fluctuations in work shift scheduling such as sick leaves and other unpredictable changes in either the demand of the business or employees schedules. Although such changes might sometimes be easy to compensate manually, the convenience of a one button automatic scheduling brings about better usability and user satisfaction.

Wishes and *blocks* are employee's way of conveying the positive and negative

preferences, respectively, of a certain time slot in the planning period to the person in charge of the rostering as well for the automatic scheduling algorithm. Each wish and block has a starting and ending time and can span up to 24 hours. They can be created in an ad-hoc fashion for a certain time interval or they can be set to be recurring, for example, from midday to midnight for every Monday. The employees' preferences are considered to match the shift if the shift overlaps with either a wish or block.

4.2 Hard constraints

Hard constraints are not a part of the implementation this thesis deals with. This is because the rules and constraints defined by various collective agreements are already implemented in the software for the sake of manual work shift scheduling. The algorithm devised in this thesis merely communicates with an interface in the software checking for any constraint violations. Thus, the algorithm has actually no information about which constraint it might have violated. An example of a rule set derived from a collective agreement is presented in appendix B. This particular set of rules concerns the employees in private sector services.

The utility function i.e. soft constraints for the heuristic algorithm are identical to the linear optimization model. Even though the heuristic algorithm allows for more complicated utility functions, for instance non-linear ones, the current one is validated and iterated with the input from the client. As the new algorithm devised in this thesis is not yet in production use, there has been no chance to design novel and rivaling utility measures with any client.

4.3 Soft constraints

The essential role of soft constraints, i.e. the utility function, is to guide the associated algorithm to assign as many shifts to employees as possible. For this reason, every assigned minute is valued as one utility unit. This baseline is, however, customizable.

Presently, there are a total of four different types of soft constraint types available with different options to apply them. The first type of utility function is the preference unit type. A preference unit can be chosen to be or not

be defined for an employee. The units in this case are usually the different companies or sections of them, who have the demand for different kind of workers. In Haahtela® Contactor, the software this algorithm is implemented in, every shift is assigned to a single unit. With the preference unit utility function, we can define a constant coefficient to increase or decrease the utility gained from assigning an employee to fill shift with his or her preference unit by multiplying the baseline with the coefficient.

Preference occupation utility function type works in the same fashion. Employees can have several different competences and a preference occupation or title can be set to an employee. A coefficient can then be defined to either penalize or reward matching occupational assignments, rewarding being the more natural choice.

Lastly, a coefficient can be defined for shifts which overlap with a wish or a block conveying the employees preferences to the heuristic. However, it needs to be noted that there is a way for an employee to take advantage of this preference utility function. Since rostering often involves part-timers, it is possible for them to fill their calendar full of wishes which makes them more likely to get a shift. Full-timers on the other hand get payed a monthly wage and thus they are typically the primary work force. In their case, the wish-block system works appropriately. To avoid this pitfall, a limit must be placed for the number of wishes on part-time workers and they should be advised to use all of their wishes in order to ensure a fair shift assignment.

These utility functions can be further specified. Every utility function can be set to apply only for a certain contract type, for instance part-timers. In addition, the applicable shifts for each function can be constraint by the day of week and starting time.

4.4 The implementation process

The creation of the algorithm was executed by means of test driven development. In TDD, a test case for the wanted solution is written before any implementation takes place. Then, implementation is iterated until the test case has been successfully passed. Then, a more demanding test is written.

First, a test ensuring the assignment of a single shift was created. In this test set, there is only one employee with a blank schedule and a shift that is applicable to her. To pass this test case, a greedy search was implemented.

It is a simple way to pass this test and also generates a good starting solution for more complex cases. The greedy search terminates, when it can't find any shifts that can be assigned to workers.

For the second test case, two fictional employees were created. The first, named Tauno, has two competences: inventory making and packing. The other, Tero, can only do packing. In this test case, there are two hard constraints concerning working hours:

- maximum workings hours during a week are 40
- minimum number of combined Friday-Saturday and Saturday-Sunday day-offs is one.

The objective is now to assign Tauno and Tero to the shifts presented in figure 4.1. In this visualization, the shifts requiring different competences are color-coded. There are no additional utility functions at play. Only the total number of minutes of the assigned shifts, i.e. the baseline the utility, are regarded desirable. The optimal solution to this test case is presented in the schedule visualization 4.2. To be exact, there are actually two optimal solutions, because the shifts scheduled on Tuesday can be switched between the two employees. Note that all the shifts in question are 8 hours long except for the the other Monday and Friday shifts.

For this test case, it felt appropriate to devise a shift swap operation. This operation searches for two assigned shifts that can be swapped between two employees. The employees must have competences for both of the shifts and the swap cant violate any hard constraints. It is executed several times after the greedy search. To determine the number of execution times, a terminal condition had to be chosen. For this purpose simulated annealing was implemented. Now, the algorithm would first greedily assign all the shifts it can, and then swap the shifts under the context of SA.

The swap heuristic could now potentially improve the utility of the schedule. Yet there are occasions where swapping two shifts opens up a slot for a new shift for the employee to have. In order to take advantage of this fact, a greedy search was set to run every time a successful swap was made. That is, when the utility of the solution is greater after the swap or when simulation annealing allowed it anyway.

However, the test case was still not passed. The algorithm got stuck in a situation described in figure 4.3. In this particular scenario, no combination of swap move could optimize the schedule without the violation of the hard constraints. This is due to the fact that the greedy search happened to assign

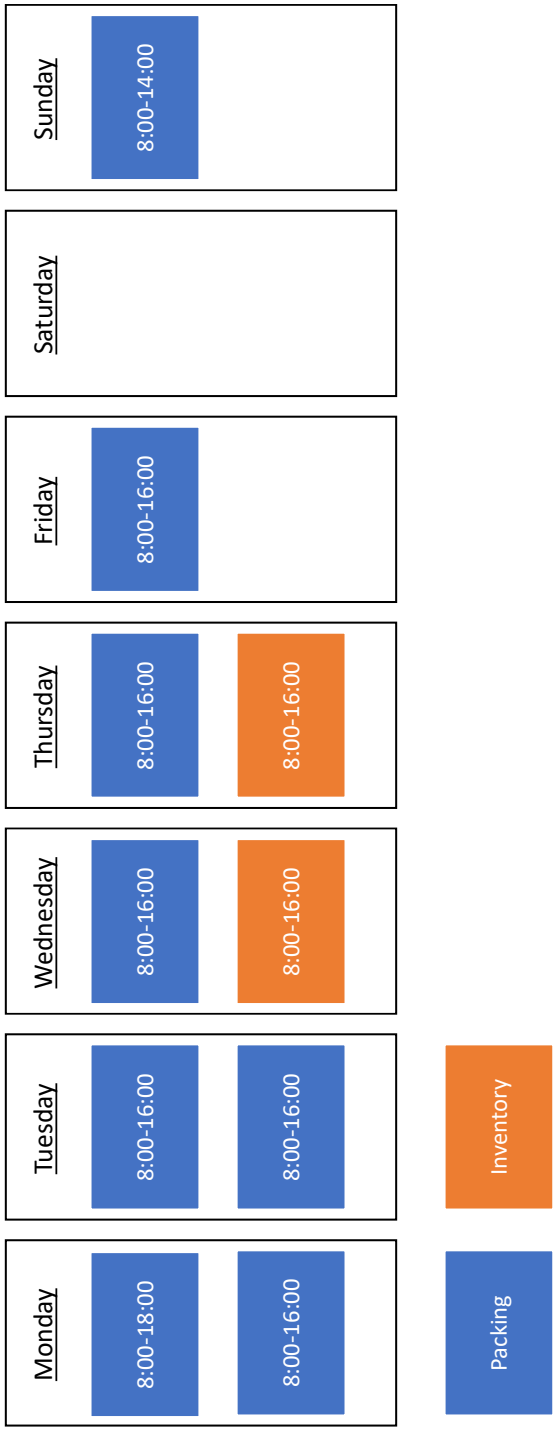

Figure 4.1: The demand for the second test case

Figure 4.2: The optimal assignment of shifts for the second test case

the packing shifts to Tauno, as they, in fact, contribute equally to the utility function.

The most simple solution to this scenario was to define another heuristic to move shifts from one employee to another. Let the name of this heuristic be the switch heuristic. The case for this heuristic is similar to the swap heuristic in the sense of new slots opening up in the schedule after the operation. Thus, greedy search was again implemented to run if this operation was successful. The decision of picking the switch or the swap heuristic was implemented as a random choice and the switch heuristic was also added to the SA context. With these modifications the test case was passed in approximately 500 milliseconds. All test were run on a setup with 32GB of RAM and an Intel processor i7-4470 on clock speed of 3,4 GHz.

The final form of the algorithm is presented in pseudo code in algorithm 1. The *annealing OK* phrase in the algorithm refers to the acceptance function of simulated annealing. To elaborate, a heuristic operation is to be approved if it increases the total utility of the solution or if it is randomly accepted by the simulated annealing even though the utility would decrease.

Figure 4.3: A sub-optimal assignment of shifts for the second test case

```
Data: assignable shifts, employees and competences, hard and soft
      constraints
Result: shifts assigned to employees

greedy search;
while annealing is not finished do
    pick random shift;
    pick swap heuristic randomly;
    if swap heuristic then
        find suitable shift to swap with the random shift;
        if possible swap found and annealing OK then
            swap shifts;
        end
    end
    else if switch heuristic then
        find suitable employee for the random shift;
        if suitable employee was found and annealing OK then
            assign the random shift to the new employee;
        end
    end
    if switched or swapped then
        greedy assignment;
    end
end
```

Algorithm 1: The shift assignment algorithm

Chapter 5

Evaluation

5.1 Evaluation with a real world test set

One real world optimization case was selected for evaluation. The test case was run with the heuristic algorithm as well as with the incumbent linear optimization model.

This test case includes 422 open shifts all of which have the same competency requirement. The shifts range over a three-week period starting from the 20th day of January. Each of the 84 employees in this test case have the corresponding competence. There are 28 employees, who have a full-time contract with differentiating weekly hours for this time period. The rest have a so-called zero-hour contract. Simply put, the working hours stated in the contract are guaranteed to the employee by the employer. The distribution of employees and their contract hours is presented in table 5.1.

Contract hours	Number of employees
0	56
15	8
22.5	6
30	5
37.5	9

Table 5.1: Distribution of employees and their contract hours in the real world test case

All the shifts are 7 hours 30 minutes long with an additional 30-minute lunch break. There are a total of five different starting and ending time

combinations among the shifts:

- 9:00 - 17:00
- 8:00 - 16:00
- 7:00 - 15:00
- 14:00 - 22:00
- 22:00 - 06:00

The customer's process in this particular test case is to first assign shifts to the full-time employees and then use the part-timer work force to fulfill the roster. The optimization feature of Haahtela® Contactor is used in both of these steps.

5.2 Problem model

To formulate the problem, let

D := the set of days in the planning horizon
D_y^{nat} := the set of national holidays during year y
$D_d^{c(n)}$:= the set of consecutive days starting from day d
W := the set of weeks in the planning horizon
W_p := the set of weeks contained by leveling period p
W_y := the set of weeks contained by year y
$W_w^{c(n)}$:= the set of n consecutive weeks starting from week
I := the set of workers
J := the set of shifts
J_w := shifts that start during week $w \in W$
J_p := shifts that start during leveling period $p \in P$
J_y := shifts that start during year $y \in Y$
$J_i^{c(n)}$:= the set of n consecutive shifts starting from shift i
$J_i^{c_t(n)}$:= the set of shifts that start within n hours from e_i
P := the set of leveling periods overlapping the planning horizon
Y := the set of years overlapping with the planning horizon
S_d := shifts that start during day $d \in D$
l_j := the length of shift $j \in J$ in hours
s_i := the starting time of shift i
e_i := the ending time of shift j
q_i := weekly contract hours of employee i

The problem has the following utility functions along with their coefficients:

- Preference occupation for full-timers, 5
- Preference occupation for part-timers, 0.5
- Preference unit, 2
- Wish, 1.7
- Block, 0.3

Let x_{ij} be a binary variable indicating if a worker $i \in I$ works on a shift $j \in J$. To formulate the wishes and blocks utility functions into an objective function let

$$p_{ij} = \begin{cases} 1.7 & \text{if shift } j \text{ overlaps with the wishes of worker } i \\ 0.3 & \text{if shift } j \text{ overlaps with the blocks of worker } i \\ 1 & \text{otherwise.} \end{cases} \tag{5.1}$$

Also to express preference unit and preference occupation utility functions, let

$$u_{ij} = \begin{cases} 2 & \text{if shift } j \text{ is assigned to the preference unit of worker } i \\ 1 & \text{otherwise} \end{cases} \tag{5.2}$$

and

$$o_{ij} = \begin{cases} 5 & \text{if } i \text{ is full-timer and preference occupation matches } j \\ 0.5 & \text{if } i \text{ is part-timer and preference occupation matches } j \\ 1 & \text{otherwise.} \end{cases} \tag{5.3}$$

The function to be maximized is

$$\sum_{I} \sum_{J} x_{ij} p_{ij} u_{ij} o_{ij} \tag{5.4}$$

subject to the following hard constraints.

There can be only one employee per shift:

$$\sum_{I} x_{ij} \leq 1 \quad \forall j \in J \tag{5.5}$$

The maximum length of a working day is 10 hours:

$$\sum_{J^d} x_{ij} l_i \leq 10 \quad \forall d \in D, \forall i \in I. \tag{5.6}$$

The minimum length of a working day is 4 hours:

$$\sum_{J^d} x_{ij} l_i \geq 4 \quad \forall d \in D, \forall i \in I. \tag{5.7}$$

The maximum hours during a week are equal to the hours stated in the employees contract:

$$\sum_{J_w} x_{ij} l_i \leq q_i \quad \forall w \in W, \forall i \in I. \tag{5.8}$$

The maximum number of working hours in a leveling period is 975:

$$\sum_{J_p} x_{ij} l_i < 975 \quad \forall p \in P, \forall i \in I. \tag{5.9}$$

The maximum number of working days in a week is 6. Note that in this problem, an employee can only work one shift per day which simplifies this constraint to the form:

$$\sum_{J_w} x_{ij} \leq 6 \quad \forall w \in W, \forall i \in I. \tag{5.10}$$

The maximum number of working days in a week when averaged for the leveling period is 5:

$$\frac{\sum_{J_p} x_{ij}}{|W_p|} \leq 5 \quad \forall p \in P, \forall i \in I, \tag{5.11}$$

where $|W_p|$ denotes the number of weeks during a leveling period p.

The maximum number of consecutive working days is 9. Again, note the fact that there can only be one shifts per employee per day:

$$\sum_{J^{D_d^{c(10)}}} x_{ij} \leq 9 \quad \forall j \in J, \forall i \in I, \tag{5.12}$$

To help formulate the next constraints, let

$$v(x_{ij}, a) = \begin{cases} 1 & \text{if shift } j \text{ starts on day } a \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

and

$$g_{iw}(a, b) = \sum_{J_w} v(x_{ij}, a)x_{ij} + v(x_{ij}, b)x_{ij} \tag{5.14}$$

$$h_{iw}(a, b) = \sum_{J_w} v(x_{ij}, a)(1 - x_{ij}) \cdot v(x_{ij}, b)(1 - x_{ij}) \tag{5.15}$$

Additionally, let us define an auxiliary combined day-off binary decision variable:

$$z_{iw} \leq 1 - \frac{g_{iw}(\text{fri}, \text{sat}) \cdot g_{iw}(\text{sat}, \text{sun}) \cdot g_{iw}(\text{sun}, \text{mon})}{M}, \tag{5.16}$$

where $M \geq 8$.

Now, the minimum number of Friday-Saturday, Saturday-Sunday or Sunday-Monday combined day-offs during a three week period is 1:

$$\sum_{W_w^{c(3)}} z_{iw} \geq 3 \quad \forall w \in W, \forall i \in I \tag{5.17}$$

The minimum number of Friday-Saturday, Saturday-Sunday or Sunday-Monday combined day-offs during a year is 17:

$$\sum_{W_y} z_{iw} \geq 17 \quad \forall y \in Y, \forall i \in I, \tag{5.18}$$

The minimum number of Friday-Saturday combined day-offs during a year is 9:

$$\sum_{J_y} h_{ij}(\text{fri}, \text{sat}) \geq 9 \quad \forall y \in Y, \forall i \in I \tag{5.19}$$

The minimum number of Sunday day-offs during a year is 22:

$$\sum_{J_y} v(x_{ij}, \text{sun})(1 - x_{ij}) >= 22 \quad \forall y \in Y, \forall i \in I \tag{5.20}$$

The minimum number of Sunday day-offs during two weeks is 1:

$$\sum_{J^{W_w^{c(2)}}} v(x_{ij}, \text{sun})(1 - x_{ij}) >= 1 \quad \forall w \in W, \forall i \in I \tag{5.21}$$

The minimum number of day-offs on eves of national holidays during a year is 2:

$$\sum_{J^{D_y^{nat}}} x_{ij} \leq |D_y^{nat}| - 2 \quad \forall y \in Y, \forall i \in I, \tag{5.22}$$

where $|D_y^{nat}|$ is the number of national holidays during year y.

The minimum time between two consecutive shifts is at least 11 hours:

$$x_{ik} + x_{il} \leq 1 + \frac{e_k - s_l}{11\text{h}} \quad \forall k \in J, \forall l \in J, \forall i \in I \tag{5.23}$$

The maximum difference in hours between the starting times of shifts during a week is 2:

$$x_{ik} + x_{il} \leq 2 - H(s_k - s_l - 2\text{h}) \quad \forall k \in J, \forall l \in J, \forall i \in I, \tag{5.24}$$

where

$$H(n) = \begin{cases} 0 & n \leq 0 \\ 1 & n > 0 \end{cases} \tag{5.25}$$

Lastly, the minimum hours of an uninterrupted time span without any shifts during a week is 35. To formulate this constraint, let an auxiliary decision variable

$$a_{ij} \geq \sum_{J_j^{c_t(35)}} x_{ij} \quad \forall i \in I. \tag{5.26}$$

Now, the constraint is of form

$$\prod_{J_w} a_{ij} \leq 0 \quad \forall w \in W, \forall i \in I. \tag{5.27}$$

The length of a leveling period in this problem is six months. Note the auxiliary weekly day-off constraints that ensure that the yearly constraints are not violated. These constraints would otherwise not be considered by either optimization method until the end of the year. Also note that for the part-timer optimization round, the maximum hours during a working week is overridden to equal 15 hours.

The employees do have shifts before the starting time of this particular roster. However, the earlier shifts mainly affect this roster only via the "Friday-Saturday, Saturday-Sunday or Sunday-Monday combined day-offs" constraints.

5.3 Results

The linear optimization took approximately 3 seconds to complete the full-timer round assigning 176 shifts and accumulating 599280 units of utility. The part-timer round was completed in 8 seconds. All but four of the remaining shifts were assigned during this round. The final utility of the roster totaled 1493280.

The results for the heuristic algorithm for the full-timer round are presented in table 5.2. The difference between the two different steps of the algorithm is notable. The swapping phase takes a time two magnitudes greater than the greedy phase. It manages to increase the utility by only less than one percent. The notation "#" is to be read "the number of". The column "#Assigned shifts" denotes the number of assigned shifts by the respective method. It is not a cumulative measure.

Method	#Assigned shifts	Utility increase	Elapsed time (seconds)
Greedy	174	579900	2
Swap	0	1680	105

Table 5.2: Results of the heuristic algorithm for the full-timer round

The second round for the part-timers is described similarly in table 5.3. In this case, the swap heuristics enhanced the solution considerably. The total running time of this phase was also considerably faster. The total utility of the heuristic approach was 1394760 which is more than 6% less than what was achieved with the linear model.

Method	#Assigned shifts	Utility increase	Elapsed time (seconds)
Greedy	236	496764	1
Swap	15	329694	8

Table 5.3: Results of the heuristic algorithm for the part-timer round

As both methods are non-deterministic, these results vary slightly across different runs. However, these changes are insignificant. Note that the durations of each solving method are rounded to the nearest second as the precise length of the time span is irrelevant in the context these methods are used.

Chapter 6

Discussion

6.1 Result analysis

In short, even though the algorithm developed in this thesis underperformed the incumbent one, it showed promising results. When taken into account that it was in fact a proof of concept and the very first implementation, this approach is valid for further exploration and research. Even though the test case presented surprises in how the automatic scheduling functionality was used by the customer, the new heuristic still proved functional.

In the test case, the customer had a small group of full-time employees and a large pool on-call workers. This pool of employees was used to fill a demand much larger than the full-time employees could cover. The way this customer utilizes the automatic scheduling feature is to first use it on full-time employees. Once these employees had their schedules filled, the rest of shifts are then automatically scheduled to the part-timers. The reasoning behind this course of action is the fact that full-time employees are paid a monthly salary regardless of their working hours, while part-timers only receive wage for the hours they have worked.

The disproportionate number of shifts compared to the number of employees presented a problem with the greedy search, when the heuristic was used to assign shifts to the full-timers. Since the initial greedy search can only assign a fraction of the total number of shifts, most of them are then used in the greedy searches after the switch and swap operations. Consequently, a huge number of shifts are tested for compatibility with the employees in the majority of the iterations, which number in thousands.

This approach taken by the customer is of course sub-optimal. Two optimization rounds can't produce the global optimum of the problem. The reasons for this course of action are unclear but this fact confirms one of the premises of this thesis. Namely, the optimality of the roster is not seen as important as the fact that the process is automated. This observation is valuable in itself for future development.

By taking this aspect into account, the greedy search phase in itself might be something of value to the customer even if it is inferior to the linear optimization method. In purely business sense, it is easy and inexpensive to put into operation and it does not require an expensive third party linear solver.

6.2 Viability of the solution

The algorithm is still not ready for production use where a performance close to the linear solver is expected. Ultimately it is a proof of concept in which there are bound to be weaknesses that can, however, be mitigated.

This single test case, however, does not ascertain if this approach is general enough to cover all the possible scenarios that are possible in the context of Haahtela® Contactor. This potential shortcoming mentioned in section 3.3 is thus still undetermined.

If indeed this solution method was determined to be unfit to generalize, the concept of using heuristics for this problem would become fruitless. If the solution needed maintenance and re-implementation for every new problem instance, it would be too cumbersome to maintain. Even in the case of having a range different heuristics for different problem instances, the challenge of choosing the right one remains.

Still, there are dozens of aspects of the algorithm that can be further enhanced. For instance, the algorithm could be fed more information about the constraints it violated or is about to violate. Moreover, the straightforwardness was heavily emphasized in the implementation. There is still room for more complex operations without compromising the maintainability of the code.

6.3 Next steps of development

There are still some functionalities missing from the heuristic implementation that are present in the linear optimization model. For instance, employees can be set to be eligible for shifts in certain time interval during a day. Also there are some more complicated utility functions that were not included this thesis. As an example, one of these functions add utility with respect to how close the full-timers' hours are filled to the maximum.

For the heuristic itself, a method must be found to overcome the situation of disproportionate number of shifts and employees. Before making changes to the heuristic itself, one option could be to modify the problem. If all the shifts could be assignable in a single optimization round, the initial greedy search could assign more shifts which would make the subsequent searches less intensive. Whether or not this step would prove functional, the frequency of greedy searches could be lowered or the otherwise limited.

Probably the most efficient development step would be to increase the knowledge of the violated constraints for the heuristic. For instance, if an employee has full weekly hours, the heuristic wouldn't go through the trouble of checking all the constraints in an attempt to assign a shift to the employee during the week in question. As there are a considerable number of constraints, the violation check is an expensive operation in terms of computation.

To test these improvements more robustly, a test case containing at least the mass the real world test case has should be created. This would facilitate the testing of performance improvements considerably.

Chapter 7

Conclusions

7.1 Summary

This thesis set out to develop a proof-of-concept of an alternative way of optimizing work shift schedules. This goal was met, even though the novel method did not perform as well as the incumbent one. There are still, however, numerous ways of enhancing the algorithm.

It is still unclear, if this approach will be sufficient for production use in Contactor. Even on the single schedule this algorithm was tested against, a surprising way of utilizing the automatic rostering feature was discovered. Moreover, one can find problems an order of magnitude larger from different real world cases. Of course, full coverage is probably not going to be achieved in the near future with any method as the problem is np-hard.

However, the linear model is a result of years of development and it is backed by a top-of-the-line third party linear solver. Against this background, the results achieved in this thesis are promising. The results serve as a good proxy in search of a better rostering method. Lastly, the novel algorithm is cheaper with regard to the requirement of a third party solver and it is easy to deploy for testing in the production environment.

7.2 Future research

There are numerous interesting directions to take when considering the long term development of what was achieved in this thesis. The most technical

of which would be to parallelize the solving process. This would require a substantial new work to the algorithm. Moreover, not all of the algorithm can be made parallel as every operation modifies the global solution. That is, if in one thread a shift is assigned to an employee, other threads need to know about it. Thus, the search operation for suitable shifts for an employee suitable for a certain shift could prove to be an appropriate candidate for evaluating this method.

Hybrid algorithms are another intriguing target of development. There are multiple examples of such successful implementations, for instance by [Burke et al., 2010] and [Rahimian et al., 2017]. Since the linear solver is already in use in Haahtela® Contactor, it would be relatively easy to harness as such. However, this would require a substantial amount of researching and testing to discover which parts of the solving process would be solved by which method.

As machine learning has gained popularity in solving various types of problems, it has also been applied to the nurse rostering problem [Asta et al., 2016]. The concept of machine learning is close to the concept of hyper heuristic if a learning component is present in it. Hypothetically, this would solve the problem of deciding the suitable set of different low level heuristics for each problem instance.

Bibliography

Uwe Aickelin. Genetic algorithms for multiple-choice problems. *CoRR*, abs/1004.3147, 2010. URL http://arxiv.org/abs/1004.3147.

Shahriar Asta, Ender Özcan, and Tim Curtois. A tensor based hyper-heuristic for nurse rostering. *Knowledge-Based Systems*, 98:185 – 199, 2016. ISSN 0950-7051. doi: http://dx.doi.org/10.1016/j.knosys.2016.01.031. URL http://www.sciencedirect.com/science/article/pii/S0950705116000514.

Marco A. Boschetti, Vittorio Maniezzo, Matteo Roffilli, and Antonio Bolufé Röhler. *Matheuristics: Optimization, Simulation and Control*, pages 171–177. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-04918-7. doi: 10.1007/978-3-642-04918-7_13. URL http://dx.doi.org/10.1007/978-3-642-04918-7_13.

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. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2007.04.030. URL http://www.sciencedirect.com/science/article/pii/S0377221707004390.

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. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2009.07.036. URL http://www.sciencedirect.com/science/article/pii/S0377221709005396.

B Cheang, H Li, A Lim, and B Rodrigues. Nurse rostering problems – a bibliographic survey. *European Journal of Operational Research*, 151 (3):447 – 460, 2003. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/

S0377-2217(03)00021-3. URL http://www.sciencedirect.com/science/article/pii/S0377221703000213.

Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer, 2014.

R. Dechter. *Constraint Processing*. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science, 2003. ISBN 9780080502953. URL https://books.google.fi/books?id=U_6G5txE8_MC.

Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367 – 385, 2013. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2012.11.029. URL http://www.sciencedirect.com/science/article/pii/S0377221712008776.

R.W. Eglese. Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 46(3):271 – 281, 1990. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/0377-2217(90)90001-R. URL http://www.sciencedirect.com/science/article/pii/037722179090001R.

Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, D. T. Lee, and Shang-Hua Teng, editors. *Classification of Various Neighborhood Operations for the Nurse Scheduling Problem*, pages 72–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. ISBN 978-3-540-40996-0. doi: 10.1007/3-540-40996-3_7. URL http://dx.doi.org/10.1007/3-540-40996-3_7.

Pierre Hansen and Nenad Mladenovìc. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130 (3):449 – 467, 2001. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/S0377-2217(00)00100-4. URL http://www.sciencedirect.com/science/article/pii/S0377221700001004.

Stefaan Haspeslagh, Patrick De Causmaecker, Andrea Schaerf, and Martin Stølevik. The first international nurse rostering competition 2010. *Annals of Operations Research*, 218(1):221–236, 2014. ISSN 1572-9338. doi: 10.1007/s10479-012-1062-0. URL http://dx.doi.org/10.1007/s10479-012-1062-0.

Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139 (1):21 – 45, 2002. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(02)00221-7. URL http://www.sciencedirect.com/science/article/pii/S0004370202002217.

Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007. ISSN 0925-5001. doi: 10.1007/s10898-007-9149-x. URL http://dx.doi.org/10.1007/s10898-007-9149-x.

Jari Kyngäs. *Solving Challenging Real-World Scheduling Problems*. PhD thesis, University of Turku, 2011.

Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1):161 – 205, 1992. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(92)90007-K. URL http://www.sciencedirect.com/science/article/pii/000437029290007K.

PAM. Terms of employment in the commercial sector, 2016. URL http://netpaper.lonnberg.fi/pam/tes-fi/kaupan/.

Erfan Rahimian, Kerem Akartunali, and John Levine. A hybrid integer and constraint programming approach to solve nurse rostering problems. *Computers I& Operations Research*, 82:83 – 94, 2017. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2017.01.016. URL http://www.sciencedirect.com/science/article/pii/S0305054817300163.

Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

Seyed Mojtaba Sajadi, Shima Ghasemi, and Hashem Vahdani. Simulation optimisation for nurse scheduling in a hospital emergency department (case study: Shahid beheshti hospital). *International Journal of Industrial and Systems Engineering*, 23(4):405–419, 2016. doi: 10.1504/IJISE.2016.077691. URL http://www.inderscienceonline.com/doi/abs/10.1504/IJISE.2016.077691.

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. ISSN 1365-2648. doi: 10.1046/j.1365-2648.2000.01512.x. URL http://dx.doi.org/10.1046/j.1365-2648.2000.01512.x.

Pieter Smet, Patrick De Causmaecker, Burak Bilgin, and Greet Vanden Berghe. *Automated Scheduling and Planning: From Theory to Practice*, chapter Nurse Rostering: A Complex Example of Personnel Scheduling

with Perspectives, pages 129–153. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-39304-4. doi: 10.1007/978-3-642-39304-4_6. URL http://dx.doi.org/10.1007/978-3-642-39304-4_6.

Constantine D Spyropoulos. {AI} planning and scheduling in the medical hospital environment. *Artificial Intelligence in Medicine*, 20(2): 101 – 111, 2000. ISSN 0933-3657. doi: http://dx.doi.org/10.1016/S0933-3657(00)00059-2. URL http://www.sciencedirect.com/science/article/pii/S0933365700000592. Planning and Scheduling in the Hospital.

E.G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. Wiley, 2009. ISBN 9780470496909. URL http://books.google.fi/books?id=SIsa6zi5XV8C.

G.Y.C. Wong and Andy Hon Wai Chun. Constraint-based rostering using meta-level reasoning and probability-based ordering. *Engineering Applications of Artificial Intelligence*, 17(6):599 – 610, 2004. ISSN 0952-1976. doi: http://dx.doi.org/10.1016/j.engappai.2004.08.001. URL http://www.sciencedirect.com/science/article/pii/S0952197604000831.

P. Daniel Wright and Stephen Mahar. Centralized nurse scheduling to simultaneously improve schedule cost and nurse satisfaction. *Omega*, 41(6):1042 – 1052, 2013. ISSN 0305-0483. doi: http://dx.doi.org/10.1016/j.omega.2012.08.004. URL http://www.sciencedirect.com/science/article/pii/S0305048312001119.

E. I. Ásgeirsson, J. Kyngäs, K. Nurmi, and M. Stølevik. A framework for implementation-oriented staff scheduling. In J. Fowler, G. Kendall, and B. McCollum, editors, *In proceedings of the 5th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2011), 9-11 August 2011, Phoenix, Arizona, USA*, pages 308–321, 2011. Paper.

Appendix A

Example of typical constraints in NRP

The hard constraints presented in [Burke et al., 2010]:

- daily coverage requirement of each shift type
- for each day, a nurse may not start more than one shift
- maximum number of total working days during the scheduling period
- maximum number of on-duty weekends during the scheduling period
- maximum number of night shifts during the scheduling period
- no stand-alone night shift (i.e. no night shift between two non-night shifts)
- minimum two free days after a series of night shifts
- maximum number of consecutive night shifts
- maximum number of consecutive working days
- no late shifts for one particular nurse

The soft constrains identified in [Burke et al., 2010] are:

- complete weekends (i.e. either no shifts or two shifts in weekends)
- avoiding any stand-alone shift (i.e. a single day between 2 days off)
- minimum number of free days after a series of shifts

- maximum/minimum number of consecutive assignments of early and late shifts
- maximum/minimum number of weekly working days
- maximum number of consecutive working days for part-time nurses
- avoiding certain shift type successions (e.g. a day shift followed by an early one, etc.)

Appendix B

Terms of employment in the commercial sector

In Finland, employees in private service sectors, such as retail and wholesale trade, travel and restaurant sectors, belong under the collective agreements negotiated by Service Union United (PAM).

The rules concerning working hours can be extracted from the collective agreement that Service Union United has negotiated for the workers in the commercial sector for the year 2017. Note that according to the collective agreement [PAM, 2016], the rules may vary depending on the chosen work time balancing system i.e. are there periods in use and other circumstances. Also, the agreement only defines the minimum terms that the employer must abide to. A single employment contract can be negotiated to be more beneficial for the employee.

The following simplified rule set, that is of importance to this thesis, is one possible application of the terms defined by the collective agreement:

- maximum length of a working day is 10 hours
- minimum length of a working day is 4 hours
- maximum working hours in a week are agreed on in an employment contract
- the minimum time between two consecutive shifts is at least 11 hours
- maximum number of workdays in week is 6 days
- maximum number of consecutive working days is 9

- minimum number of Sunday day-offs during a year is 22
- minimum number of Friday-Saturday, Saturday-Sunday or Sunday-Monday combined day-offs during a year is 17
- minimum number of Friday-Saturday combined day-offs during a year is 9