Erasmus University Rotterdam
Erasmus School of Economics

Master Thesis

Solving Nurse Rostering Problems with Lagrangian Relaxation and Column Generation

Author:
J.J. Dopheide, BSc

Student Nº :
371175

Supervisor:
Dr. R. Spliet

Supervisor ORTEC:
L.M. Fijn van Draat, MSc

November 2, 2018

Abstract

In this thesis we try to fill the gap between the current nurse rostering research and the very complex real world. In reality, very complex and elaborate regulations hold. Therefore, methodology is proposed that is very flexible and robust to a variety of constraints. The nurse rostering problem is solved by applying a column generation approach that involves solving the Lagrangian dual problem with a subgradient algorithm. The algorithm is considered very robust to varying and difficult constraints, since the pricing problem is solved heuristically. This heuristic creates many patterns of shifts for every employee, which are evaluated on feasibility and solution value. In the restricted master problem, one of these patterns is selected for each employee, yielding an overall schedule.

For validation of our methods, we use benchmark instances of which the optimal solution is known. Next to that, we compare the results of our algorithm with a different algorithm currently used by ORTEC. Although the ORTEC algorithm has a different objective function, the optimal solutions to the benchmark instances is the same and therefore the two algorithms can be compared.

It is shown that for both algorithms it is very hard to find good quality schedules within a short amount of time. When exact methodology is incorporated in the algorithm described in this paper, it is shown for the smaller instances that the new algorithm is outperforming the algorithm of ORTEC. For the larger instances it is hard to conclude which algorithm performs best, due to the slight difference in objective functions.

It is concluded, that it is of interest to investigate whether it is necessary to be able to comply to all exceptions to the rules or that it is sufficient to comply to more strict constraints to obtain good solutions in limited time.

Contents

1 Introduction

In this paper, we will address the Nurse Rostering Problem (NRP) from a practical point of view. The NRP is originally constructed for hospitals, and is therefore called the NRP, but there are many other applications as well. Any situation where there is a set of shifts to be executed, and a set of employees that can execute these shifts, it can be considered as instances of the NRP

Wren (1995) defines rostering as: *the placing, subject to constraints, of employees into slots in a pattern. One may seek to minimise some objective, or simply obtain a feasible allocation.* In other words, the goal of the NRP is to assign the shifts to employees, where this assignment of shifts requires that a number of constraints is met. The objective of this problem is to find a feasible schedule with minimal costs or minimal penalties for unmet constraints.

This research is conducted in cooperation with ORTEC. ORTEC is one of the largest providers of optimisation software and data analytics solutions in the world. One of the departments within ORTEC is providing software solutions to companies that have to deal with the complex world of workforce management. These solutions increase the efficiency and effectiveness of the workforce, by providing a high-quality platform for every stakeholder within the workforce. One of the aspects within workforce management, is workforce scheduling, where the goal is to provide quality schedules that take into account all practical aspects, such as employee satisfaction and complex regulations. ORTEC already provides solutions for automated scheduling for over two decades, but continuously strives to improve its products and optimisation capabilities, of which this thesis is the result.

Although this problem is elaborately studied in the past, we belief that there is still a gap between the NRP studied in the literature and the NRP experienced in practice by companies that require solutions to this problem. This is in line with E. K. Burke, De Causmaecker, Berghe, and Van Landeghem (2004), that state that *very few approaches are suitable for directly solving difficult real world problems. [...] There is a definite gap between much of the current nurse scheduling research and the demanding and challenging requirements of today's hospital environments.* In the years after, more research in the field of NRP has been conducted, but Kellogg and Walczak (2007) still experience a gap a few years later. We belief this gap is still not covered and the request of ORTEC to conduct this research endorses this. To our knowledge there is no solution method provided in the literature that is as flexible as it is required to be for a company as ORTEC, except for the current ORTEC workforce schedule

optimisation software.

The flexibility of the solution approach is very important to ORTEC, since the same algorithm is used for many different instances. These instances differ in size and constraint presence. The general NRP is known to be a NP-hard problem, which in practice means that the problem is not always solvable to its optimal solution in tractable time. The time to solve the problem is of high importance to ORTEC, since it is experienced that users are not willing to wait long for a solution to be provided.

In this thesis we will try to contribute to the field of NRP with a reliable, flexible method, that is still fast and provides good solutions. To gain more understanding of the NRP in practice and to come up with such a method, we propose the following research objectives:

1. Extensive literature review of the NRP.
2. Review the current algorithm of ORTEC.

   We will use benchmark instances to establish this review. For these instances bounds on the optimal solution are known.
3. Develop a new algorithm for the NRP.
4. Evaluate the performance of the new algorithm.

   We will use the same benchmark instances as in 2. such that the a comparison between the currently used algorithm and the to be proposed algorithm can be made.

The outline of this thesis is as follows. In Section 2 we discuss the literature review. In Section 3 we provide an extensive problem description. In Section 4 we provide the methodology used in the constructed algorithm. In Section 5 we discuss the benchmark instances used for evaluating the algorithm. In Section 6 the results are presented of testing the performance of the new algorithm on the instances, and finally, we conclude this thesis in Section 7.

2 Literature Review

As mentioned the NRP is an intensively studied problem and has been studied for several decades. *Since the 1960's many papers have been published on various aspects of computerised health care personnel scheduling*, E. K. Burke et al. (2004). Cheang, Li, Lim, and Rodrigues (2003) made similar statements in their review, and also state that the many publications are due to the many different constraints and objective functions that are used within the NRP. Both reviews have a similar classification of solution approaches. Similar classifications are used in Van den Bergh, Beliën, De Bruecker, Demeulemeester, and De Boeck (2013), which is the most up to date review paper on nurse rostering papers we encountered. Both E. K. Burke et al. (2004) and Van den Bergh et al. (2013) provide a classification based on types of constraints. It is commonly known that the personnel scheduling world is highly constrained, and many different approaches are used to cope with these constraints. To give an example, most papers examined, state that under staffing hospitals should not be possible, but still use methodologies that consider the staffing constraints as soft constraints which are allowed to be violated.

The three review papers mention classes of solution methods, such as *optimisation* approaches, *multi-criteria* approaches, *artificial intelligence* methods, and *(meta-)heuristics*. We will discuss the different approaches briefly, and refer to the review papers for an extensive description.

Kellogg and Walczak (2007) provide a review that addresses many of the papers that are discussed in the first two reviews, but focus on reporting whether the models are applicable in practice. They state that there is a definite gap between the theoretical models and the demands from the hospitals, since many practical aspects are not considered. One of their recommendations is to fill this gap by collaborating with vendors.

Next to the review papers we also looked into the International Nurse Rostering Competition (INRC) that was held in 2010, in which fifteen different teams competed in solving NRP instances. In this competition three different tracks were present: the sprint, middle and long distance track. The tracks differed in instance sizes and in the maximum running time for computing solutions. Haspeslagh, De Causmaecker, Schaerf, and Stølevik (2014) address this competition, and discuss the five best performing teams in each track, to conclude on what kind of solutions work well for solving the NRP.

The *optimisation* approaches are used to find an optimal solution, and make use of linear and integer programming. Finding an optimal solution is obviously

desired, but will lead in the practical cases to intractable computation times due to the large and complex solution spaces, and due to the integrality constraints on the shift assignments. The *multi-criteria* approaches aim to reach multiple goals at the same time, which are often relaxations of the constraints, penalised with a certain weight per deviation. Such an objective function can be optimised with either exact or heuristic methods. *Artificial intelligence* methods consist of constraint programming, which can handle a wider class of constraints, given that the variables are non-continuous. *Heuristics* are approximation algorithms that are often derived from automating the sequence of steps that a planner would make when a planning is constructed manually. *Metaheuristics* on the other hand are the more general approximation methods that can be applied to a wider range of problems, and often make use of problem specific heuristics as well.

E. K. Burke et al. (2004) show that until the publication of their review, the methods that were implemented for use in practice, are often (hybrid) heuristic methods, suggesting that it is the best methodology for solving real life instances. Some of the suggestions they make regarding future research directions are to decompose the problem into smaller problems, to decrease the necessity of research expertise to be able to use the solution approaches in practice, such that planners do not need to understand difficult methodology to parameterise the models, and to look deeper into hybridisation of different solution approaches, due to its promising results. Furthermore, it is mentioned that flexibility of solutions approaches should be increased, a suggestion that Ernst, Jiang, Krishnamoorthy, and Sier (2004) mainly emphasises on. Van den Bergh et al. (2013) endorse this and note that researchers have given more attention to decomposition and hybridisation techniques, but state that the quality of the algorithms is only tested against basic algorithms.

Problem decomposition is promising, since it leads to smaller easier to solve sub problems, where each subproblem accounts for some of the aspects, but not all aspects of the total problem. This will make it possible to get a better grip on the highly constrained world of nurse rostering problems.

In general, decomposition of problems might lead to a big change in the objective when combining the results of the subproblems, or it even might lead to infeasibilities. Branch and price is one of the methods used for decomposition, as described in Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance (1998). Branch and price uses column generation in every node of the branching tree. The problem to be solved in each node is split into a master problem and a pricing problem. The master problem assigns values to the variables corresponding

to the columns added, minimising some objective function, while respecting a subset of constraints. The pricing problem constructs the columns that have negative reduced costs (in case of a minimisation problem) that will be added to the master problem. To compute the reduced costs of a column, the Linear Programming relaxation or a Lagrangian relaxation Caprara, Fischetti, and Toth (1999) can be applied.

Branch and price, has been proven to work for the NRP as well, which is shown in Eveborn and Rönnqvist (2004) and E. K. Burke and Curtois (2014). E. K. Burke and Curtois (2014) competed with their methodology in the INRC 2010, and performed best on all instances that were provided upfront, but not on the hidden test sets, which suggests over fitting. The former uses a *nurse-pattern* view for the decisions variables in the master problem and uses a set partitioning formulation, whereas the latter uses a *nurse-day* view with a set covering formulation. Here the *nurse-pattern* and *nurse-day* view are in line with the definitions in Cheang et al. (2003). Solving the master problem in both papers is done by solving the Linear Programming (LP) relaxation, using a solver. The solution approaches of the papers using branch and price, formulate the pricing problem as a shortest path problem.

Eveborn and Rönnqvist (2004) do also generate shifts in the subproblem, and therefore formulate a nested shortest path problem.

E. K. Burke and Curtois (2014) formulate it as an employee constrained shortest path problem. The downside of solving the subproblem as an employee constrained shortest path problem, is that when the nodes in the graph represent a shift on a certain day, it can only handle day-to-day constraints. For example, a constraint on the maximum number of weekends of work in a planning period, cannot be incorporated. E. K. Burke and Curtois (2014) overcome this by adding such constraints as soft constraints to the master problem. To calculate the total penalty of the soft constraints fast, they model the constraints as regular expression constraints, meaning that the constraints are modelled as sub-patterns that are required to exist a minimum or maximum number of times in the schedule. E. K. Burke and Curtois (2014) do admit that this still does not allow for modelling a wide range of constraints, e.g. constraints that restrict the minimum or maximum amount of work time.

Valouxis, Gogos, Goulas, Alefragis, and Housos (2012) competed in the INRC 2010 as well and showed that hybridisation of solution methods and decomposing the problem into smaller problems (as is suggested in E. K. Burke et al. (2004)) works, as they have won the competition in all tracks. The solution method they applied was a two-stage algorithm in which first the employees

were assigned to days, and given the assignment it was determined which shift had to be performed by each employee. The latter part was solved exactly solving a mathematical programming formulation using a solver.

Artificial intelligence methods were applied in the NRC 2010 as well. Nonobe (2010) used constraint programming with a powerful solver. Demeester, Bilgin, De Causmaecker, and Berghe (2012) provided a hyper heuristic, which is a self learning algorithm that chooses the use of a low level heuristic based on a tournament. Low level heuristics compete one against one each round and the heuristic that provides the best score wins and goes through to the next round. The overall winner is then executed. These artificial intelligence based solution methods performed well; Nonobe (2010) provided good results on the sprint track, but its performance quality diminished for the larger instances. Demeester et al. (2012) performed well on the largest instances.

Methods that are more flexible are the (meta-)heuristics, especially when problems are highly constrained. E. K. Burke et al. (2004) emphasise this and state that *metaheuristics are generally better suited than most other approaches for generating an acceptable solution in cases where the constraint load is extremely high.* Metaheuristics that have been used for the NRP are *Simulated Annealing* (SA), *Tabu Search* (TS) and the *Genetic Algorithm* (GA).

SA is based on the chemical process of annealing, in which the material is cooled in control, to make it stronger. This method uses a temperature function, which in general is highly parameterised, which makes it less desirable to use in terms of ease of use.

TS is a derivation of the hill climbing heuristic, where the idea is to get out of a local optimum by restricting a certain change to be undone for several iterations, which is kept track of on a tabu list. Deciding the length of a tabu list is problem (instance) specific, and may therefore be difficult to determine.

The GA is based on the evolution theory "survival of the fittest". For the GA the number of solutions that are considered in each iteration has to be determined upfront. TS and GA are thus far less parameterised than SA. Since ease of use is important E. K. Burke et al. (2004), the first two metaheuristics are more appropriate than SA.

As can be concluded from this literature review, splitting up the problem and using (meta-)heuristics for simplicity and flexibility, provides promising results. Combining them leads to hybrid solutions methods. According to E. K. Burke et al. (2004) and Van den Bergh et al. (2013) hybridising leads to good results in practice, and will therefore be a step in the direction of filling the research-application gap discussed by Kellogg and Walczak (2007).

3 Problem Description

The NRP in general is a problem where a number of shifts have to be assigned to a set of employees, under the condition that a number of constraints are met, while optimising an objective function. Solving the NRP will result in a schedule where the shifts are assigned to the employees. We describe all the aspects of the problem in the coming subsections, where we will first address some definitions, such as the schedule period, an employee, a department, and a shift, which is followed by a section about the constraints in the NRP. In Figure 1 an example schedule is provided with a description of the before mentioned aspects of the problem.

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Anna	D	D	D	D			N
Ben	N	N			D	D	D
Cindy			D	D	D		
Dirk			N	N	N	N	
D	1	1	2	2	2	1	1
N	1	1	1	1	1	1	1

Figure 1: Example schedule: The planning period is a week; The department consists of four employees Anna, Ben, Cindy, Dirk; There are two different shift types, a day (D) and a night (N) shift type; The demand per day for each of the shift types is provided in the bottom rows; Possible shift patterns are shown by the row for each employee in the schedule, where the shift pattern of Anna consists of four day shifts, and a night shift.

3.1 Definitions

Below we will explain some definitions. We tried to minimise the number of circular references, but some of the definitions are interdependent such that it was inevitable. For clarity's sake, we will therefore first name all the definitions, before elaborating on them:

1. Planning Period
2. Employee
3. Department
4. Shift Type
5. Shift
6. Shift Pattern

The planning period is a continuous interval in which shifts need to be assigned to employees, which is defined by a start time T_s and end time T_e. This does not mean that periods outside this interval are disregarded. We define the problem as such that the history, which is defined by everything before the start of the planning period T_s, has to be taken into account. We need to consider the history, since many constraints are defined over a period of time that is in general longer than the planning period.

In other words, interval dependent constraints and penalties are included, even though the interval partially covers the history. Intervals that entirely lie in the history are disregarded, since the problem is not influenced by restrictions on these intervals.

An employee is a person, to which shifts can be assigned. Employee i has a certain skill set $\mathcal{Q}_i$, that defines which activities it can execute.

A department denotes the set of all employees. We will not consider float nurses (Trivedi and Warner (1976)), i.e., personnel that can be scheduled to different departments to cope with fluctuations of demand in an efficient way.

A shift type d is defined by a time interval, in which an employee has to execute predetermined activities. The set of shift types $\mathcal{D}$ states all the different shift types.

Next to that, shift types may require certain skills, implying that it cannot be assigned to every employee. The skill set $\mathcal{R}_d$ is required to be able to perform shift type d, such that an employee i is able to perform shift d when $\mathcal{R}_d \subseteq \mathcal{Q}_i$.

A shift is defined by the combination of a shift type, and the day on which it has to be performed. We define $\mathcal{S}$ as the set of all possible shifts, and define the set $\mathcal{S}_i$ as the set of shifts that can be performed by employee i. The beginning of shift s is defined by b_s and its end by e_s. Both b_s and e_s are determined by the starting time of the shift type, and the date on which the shift type is defined. An employee can only execute one shift at the time. In contrast with most other problem descriptions in the literature, multiple shifts could be assigned to an employee per day.

A shift pattern is defined by a sequence of shifts that could be performed by an employee within the planning interval. In other words, a shift pattern is one of the many possible personal schedules of an employee. A subpattern is a shift

pattern, restricted to a subinterval of the planning period.

3.2 Constraints

The NRP instances in practice have many different types of constraints, as has been mentioned in Section 2. This is mainly due to the complex laws and regulations. These laws and regulations differ per sector which leads to a large variety of constraints. Therefore, we will provide a classification of constraints, which will be explained by some examples.

First of all, the constraints that are present within every instance of the NRP, are the **coverage constraints**. The coverage states how many shifts of a certain shift type on which day have to be performed. In other words, the coverage constraints tell how many employees are required to perform a certain shift type on each day in the planning interval to cover the demand.

Next, there is the class of **legislation rules**. Legislation rules are used to state how a shift pattern may look like. These legislation rules are imposed by the government by law or by the collective labour agreement of the sector to which they apply. There is a wide variety of constraints, that make this class of constraints hard to deal with. Especially, the exceptions to the rules make this class complex.

An example of a constraint that should hold in practice is a rule on rest times between two shifts. This rule is taken from the Dutch legislation rules. It states that within every 24 hours an employee should have at least 11 hours of uninterrupted rest. So far, the definition of the rule is not very hard to deal with, because it in general only imposes constraints on two consecutive shifts. But, the exception to this rule states that it is allowed to reduce this rest time to 8 hours, once every 7 times 24 hours. Therefore, the dependency from two consecutive shifts is extended to dependency for all shifts within an interval of 7 days.

The **skills** class is mostly covered in Section 3.1 already, and takes into account the requirements to perform a shift. For an employee to be able to perform a certain shift, it should have the skills to do so.

The next class of constraints consist of the **preferences and fixed assignments** of the employees. Employees might want a day off, such that no shift at all can be assigned, or rather not work certain shift types on certain days, such that specific shifts may not be assigned. On the other hand, employees might prefer to work certain shifts, such that those need to be assigned. Fixed assignments are shifts that are already assigned to an employee and may not

be changed. This may occur when an employee is working for multiple departments.

Below the summarised classification is shown:

1. Coverage constraints
2. Legislation rules
3. Required skills
4. Personal preferences and fixed assignments

We want to emphasise, that within a typical schedule as in Figure 1, the coverage constraints hold vertically, such that the constraints hold for every combination of day and shift type, resulting in interdependence of employees. The legislation rules and the personal preferences hold horizontally, which means the constraints only dependent on a single employee. The class of required skills limit the shift types that an employee is able to execute.

Furthermore, in general, constraints can be classified as either hard or soft constraints. The hard constraints need to be met and will result in a smaller solution space, whereas the soft constraints are encouraged to be met, but are not required to be met. The soft constraints result in (highly) penalised spaces in the entire solution space. Since many constraints need to be satisfied in the NRP, it is often hard to find a feasible solution. That is why many solution approaches use soft constraints. The choice of classifying constraints as either hard or soft is a modelling choice, that obviously influences the possible solution approaches.

4 Methodology

In this section we will discuss the currently used methodology by ORTEC, as well as the proposed methodology.

4.1 Current Algorithm ORTEC

As described in the introduction, ORTEC already has an algorithm that solves the NRP. This algorithm is developed over the years and parts of it are discussed in E. K. Burke, Curtois, Post, Qu, and Veltman (2008). In this section we will provide a more up to date description of the algorithm.

ORTEC considers the NRP as a hierarchical constraint satisfaction problem, where variables are modelled according to the *nurse-day* view Cheang et al. (2003) and the constraints in the class Legislation Rules are modelled as hard constraints, whereas all other constraints are modelled as soft constraints. This way of modelling allows it to initialise the algorithm with an empty schedule, which then can be adjusted iteratively, where non feasible solutions can be rejected and feasible ones are accepted. This is desired since this makes sure at all times a feasible solution is available. So every time a new schedule is obtained, the hard constraints are checked, and a schedule is accepted or not. For the soft constraints, ORTEC uses a hierarchy between the cover constraints, and all other soft constraints, i.e., assigning an extra shift to an employee is always preferred over all the other soft constraints.

The algorithm starts by applying a Greedy Insertion construction heuristic (GI) that sorts a list of to be scheduled shifts from difficult to easy, based on a score that is computed based on several criteria, such as whether a shift has to be planned in the weekend, and whether it has relative priority compared to other shifts. Then it is tried to assign each shift to employees in the order of the sorted list. The GI is applied on different subsets, to obtain a set of different solutions. These solutions are obtained to initialise the next phase in the algorithm, which is a Genetic Algorithm (GA).

Provided the different solutions, the GA uses five different operators to adjust the solutions in the initial set. Two of these operators are crossover operators, which combine two solutions into two new ones. These operators first unassign some shifts, since crossing two schedules will generally not lead to feasible schedules E. Burke, Cowling, De Causmaecker, and Berghe (2001), and after crossing a repair heuristic is applied (which is the same GI heuristic). The other three operators are mutation operators, which adjust one solution into a new one. The GA selects one of the five operators randomly. Depending on

the operator one or two parent solutions are selected and one or two children solutions are constructed. These children solutions are saved in a new set of solutions. On the new set of solutions the same steps are applied. These steps are applied iteratively, and when a stopping criterion is met, the best solution found so far is selected.

Given the best schedule (individual) found, a local search heuristic is applied. This local search heuristic is based on planning shifts manually. For every (series of) shift(s) it is tried to assign it somewhere else or it is swapped with another (series of) shift(s). When such a move results in a better solution value, it is applied.

Finally, a variable neighbourhood search is applied, that first ruins and then recreates the schedule. For ruining the schedule, two different operators are used. The first operator selects a predefined number of employees randomly according to a roulette wheel, which has probabilities based on a score assigned to every employee, stating how good his schedule is. Employees that have a higher score assigned, have a lower probability to be selected. The shifts assigned to the selected employees are unassigned, and new shifts are assigned by the GI. The other operator unassigns a random block of shifts in the schedule, and the GI assigns shifts again. The operators are selected randomly, but the probabilities of selecting the operators are adjusted based on the success rate of creating an improved schedule.

When there is no time left to compute new solutions, the algorithm stops and the best found solution is returned.

4.2 Proposed Methodology

Based on the literature review and the problem description, we think it can be of great value to model the problem in such a way that we are able to split the problem into separate parts. We will exploit the classification proposed in Section 3.2 by decomposing the problem in a part where shift patterns are generated, and a part that combines these patterns into a schedule. This decomposition is chosen, such that the pattern generation part can deal with the horizontal aspects of a roster, i.e., the difficult legislation rules, the skill requirements, and the personal preferences of the employee. The covering constraints that remain make the patterns interdependent, and are therefore dealt with separately.

Recall that $\mathcal{N}$ denotes the set of employees, and $\mathcal{S}$ denotes the set of possible shifts. Let us define the sets $\mathcal{P}_i$, $\forall i \in \mathcal{N}$, that consist of all possible patterns of shifts that can be executed by employee i. Note that these sets contain exponentially many patterns. Next, let us define the decision variable

$$x_{pi} = \begin{cases} 1 & \text{if shift pattern } p \text{ is assigned to employee } i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Let c_{pi} denote the penalty costs of pattern p. Furthermore, a_{sp} is a parameter that equals 1 if shift s is included in pattern p and let d_s be the demand for shift s. Now we can formulate the integer programming problem P_{IP} as follows:

$$\begin{aligned} P_{IP} = \quad & \min \quad \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} c_{pi} x_{pi} \\ & \text{s.t} \quad \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} a_{sp} x_{pi} = d_s \qquad \forall s \in \mathcal{S} \\ & \qquad \sum_{p \in \mathcal{P}_i} x_{pi} = 1 \qquad \forall i \in \mathcal{N} \\ & \qquad x_{pi} \in \{0,1\} \qquad \forall p \in \mathcal{P}_i, \forall i \in \mathcal{N} \end{aligned} \tag{2}$$

The problem is modelled in this way since this set partitioning formulation allows us to use column generation, where we choose the right patterns in the master problem and construct patterns in the pricing problem. The pricing problem can be split up in $|\mathcal{N}|$ independent subproblems. Each subproblem is solved to construct a feasible pattern for that employee that has negative reduced costs. One advantage is that the subproblems are independent, so that they can be solved in parallel.

In the next sections the algorithm is explained in detail, but first we provide a general outline of the algorithm. The idea behind the proposed algorithm is to iteratively create shift patterns for employees, that are combined into a schedule. Based on the schedule, penalties are computed. These penalties are included in the objective for constructing new patterns.

To make sure that the patterns that are generated will increase the quality of the schedule, the Lagrangian relaxation (Section 4.2.1) is solved to obtain information for each shift to know whether it is important to include or exclude it in the pattern. The pattern creation is equivalent to column generation for which we define the restricted master problem in Section 4.2.2. The dual information is gathered by applying the subgradient algorithm (Section 4.2.3) to the Lagrangian relaxation of P_{IP}. We apply a Lagrangian heuristic in order to obtain feasible solutions to the original problem P_{IP} (Section 4.2.4). The dual information is used in the objective function of the construction of the pattern (Section 4.2.5). The pattern creation problem is solved heuristically (Section

4.2.6). If a created pattern has negative reduced costs, it is added to the restricted master problem, and column fixing is applied for convergence (Section 4.2.7). After the algorithm is terminated, the best found schedule is returned. In Figure 2 we present the flow diagram of the algorithm.

Although the Lagrangian relaxation of the problem is solved, instead of solving the original problem, it is believed that, by using large enough artificial penalties on deviations from the demand, the pricing problem will produce sufficiently good patterns to obtain a good solution to the original problem P_{IP}. This will be checked by solving P_{IP} given in Formulation 2 exactly with a solver.

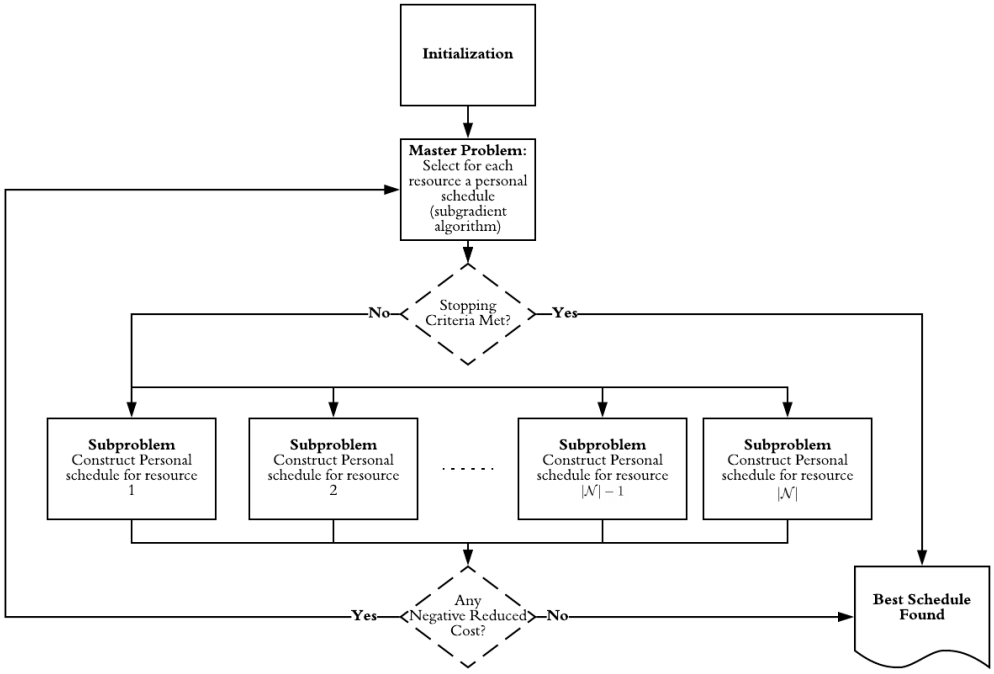

Figure 2: Flow diagram that shows the general outline of the proposed algorithm

4.2.1 Lagrangian Relaxation

We have formulated the master problem as a set partitioning formulation. The set partitioning problem is known to be NP-hard Karp (1972). This means that solving it exactly can result in intractable computation times. Therefore we suggest to solve the Lagrangian relaxation of the problem. We have formu-

lated the master problem as a set partitioning problem, with the side constraint that an employee may only be assigned one pattern. We apply the Lagrangian relaxation only to the cover constraints. Therefore, we introduce a variable $\lambda_s, \forall s \in \mathcal{S}$, which represents the penalty for violating coverage constraint s. This results in the following objective function:

$$\theta(\lambda) = \min_x \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} c_{pi} x_{pi} - \sum_{s \in \mathcal{S}} \lambda_s \left(\sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} a_{sp} x_{pi} - d_s \right) \tag{3}$$

that is subject to the constraint that exactly one pattern must be assigned to every employee, and the binary constraint on the decision variables. Since the cover constraint is an equality constraint, the penalty variables are unbounded such that $\lambda_s \in \mathbb{R}, \forall s \in \mathcal{S}$. Equation 3 can be rewritten into

$$\theta(\lambda) = \min_x \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} \left(c_{pi} - \sum_{s \in \mathcal{S}} \lambda_s a_{sp} \right) x_{pi} + \sum_{s \in \mathcal{S}} \lambda_s d_s \tag{4}$$

such that the term between the brackets in Equation 4 are the Lagrangian costs of selecting pattern p for employee i. The Lagrangian costs are denoted by c_{pi}^{LR}.

4.2.2 Restricted Master Problem

As mentioned, there are exponentially many patterns in $\mathcal{P}_i$, and therefore we propose to use column generation. This means that only a subset of all the patterns are included in the master problem, such that the master problem is restricted. Given that the number of columns added to the restricted master problem (RMP) is reasonable, solving the relaxed problem for a given vector of Lagrangian penalties λ is simple, i.e., select for each employee i, the pattern $p \in \bar{\mathcal{P}}_i \subset \mathcal{P}_i$ with minimal Lagrangian costs c_{pi}^{LR}. Here, $\bar{\mathcal{P}}_i$ is the set of patterns that have been added to the RMP for employee i.

It still remains to obtain the vector of Lagrangian penalties λ. Due to the relaxation, we have that $\theta(\lambda) \leq \theta^*$, where θ^* is the optimal value to the original problem with the currently added patterns $\bar{\mathcal{P}}_i, \forall i \in \mathcal{N}$. Therefore we want to choose λ in such a way that we solve $P_{LR} = \max_{\lambda \in \mathbb{R}^{|\mathcal{S}|}} \theta(\lambda)$, implying that the lower bound comes as close as possible to the solution of the original problem. This is called the Lagrangian dual problem. It has been proven by Geoffrion (1974) that the bound of the Lagrangian relaxation is as tight as the linear programming relaxation when the *Integrality property* holds, and tighter otherwise. We know that the property holds since the constraint matrix of the relaxed problem is totally unimodular. This is true since every column in the

constraint matrix consists of only one non-zero element that is equal to 1. This means that the determinant of every square sub-matrix is in the set $\{-1, 0, 1\}$, which defines total unimodularity.

4.2.3 Subgradient Algorithm

To solve the Lagrangian dual problem P_{LR}, we apply the near optimal subgradient algorithm for computing the Lagrangian penalties. The Lagrangian penalties are computed in an iterative way, where in each iteration a step in the direction of the optimum is taken. The penalties are computed as follows:

$$\lambda_s^{n+1} = \lambda_s^n + \delta_n y_s^n \tag{5}$$

Here, λ_s^n is the s-th element of the vector of Lagrangian penalties in the n-th iteration, δ_n is a step size, and y_s^n is the sub-gradient. The sub-gradient is obtained by taking the derivative with respect to λ_s. This subgradient equals the deviation from the demand:

$$y_s^n = d_s - \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} a_{sp} x_{pi}^n \tag{6}$$

where x_{pi}^n corresponds to whether employee i is assigned pattern p in the solution of P_{LR} in the n-th iteration of the subgradient algorithm.

We will use the well-known Held-Karp method Held and Karp (1971) for updating the step size in Equation 5:

$$\delta_n = \alpha^n \frac{UB - \theta(\lambda^n)}{\sum_{s' \in S} (y_{s'}^n)^2} \tag{7}$$

where α^n is a step size parameter that decreases after a number of iterations to reduce the step-size and UB is an upper bound to the original problem. The step size parameter α^n is halved if within b iterations the lower bound $\theta(\lambda^n)$ has not been improved, provided that the step size parameter has not been changed.

4.2.4 Big-M method and Lagrangian Heuristic

As has been mentioned, an upper bound UB is required to determine the step size during the execution of the subgradient algorithm. Finding a feasible solution is quite hard already, meaning that finding an upper bound is hard as well. Therefore, we use the *big-M* method to evaluate infeasible solutions, such that we can still use the subgradient algorithm with this step size. We use the *big-M* in such a way, that after constructing any solution, we compute the solution

value of it by evaluating the objective function of the original problem in Formulation 2, and then add a large value M per unit of deviation from demand y_s^n.

This *big-M* method is applied to every solution of P_{LR} obtained in the subgradient iterations to obtain an artificial upper bound. When no deviations from demand are established in a solution, the bound is an upper bound to the original problem. The artificial upper bound is computed to steer the penalties in such a way that emphasis is put on creating new patterns that include shifts that are understaffed in the current solution, and will therefore contribute to creating good patterns for an employee in the pricing problem.

Next to just applying the *big-M* method to the solution obtained by solving the Lagrangian relaxation, we also propose a simple Lagrangian heuristic with the goal to find tighter (artificial) upper bounds. This Lagrangian heuristic is applied in each iteration of the subgradient phase. It is similar to what is proposed in Caprara et al. (1999), where the Lagrangian relaxation is applied to the set covering problem, instead of the set partitioning problem. Caprara et al. (1999) perform an iterative procedure that selects the pattern with minimal reduced costs per unit of cover in each iteration, set the corresponding variable to one and set the penalties corresponding to the rows covered by this pattern equal to zero. These steps are repeated until all demand is covered. Different to Caprara et al. (1999) we have that $d_s \in \mathbb{N}_0$, instead of $d_s = 1, \forall s \in \mathcal{S}$. This means we cannot set the penalty λ_s to zero after including a pattern including shift s in the solution, since not all demand for shift s is necessarily covered by that pattern. Although the dependence between the penalty and the units of covered demand is not necessarily linear, we depreciate the penalty linearly for each added unit of covered demand. This means that if a pattern is selected which includes a certain shift s, the penalty for this shift λ_s is reduced by $-\frac{1}{d_s^*}\lambda_s$, where d_s^* is the deviation of the demand before adding this pattern.

This method is easy, but still computationally quite expensive. This is due to the fact that after assigning a pattern to an employee, every pattern in the RMP for every employee that has not been assigned a pattern yet $\mathcal{N}^* \subset \mathcal{N}$, all patterns in $\mathcal{P}_i$ need their Lagrangian costs to be updated. This reduction is different for each pattern, since the reduction of Lagrangian costs depends on the union of shifts in the pattern assigned and the considered pattern.

To evaluate the quality of applying this Lagrangian heuristic, we also calculate the *big-M* score for the solution of every solution of the Lagrangian relaxation. The best (artificial) upper bound is saved and used in determining the step-size in the subgradient algorithm. The entire subgradient phase in pseudo

code is provided below:

Algorithm subgradient phase

```
Initialize variables
CONTINUE := TRUE
while CONTINUE do
    Construct Lagrangian relaxed solution
    Apply Lagrangian heuristic to obtain solution
    Calculate big-M on both solutions
    Update step size scalar if needed
    Update Lagrangian penalties
    if Stopping criteria are met then
        CONTINUE := FALSE
    end if
end while
```

The subgradient phase stops when one of the stopping criteria is met. The stopping criteria are (1) when a maximum number of iterations is reached; (2) the upper- and lower bound coincide; (3) the step size is smaller than some threshold ϵ. The subgradient phase then returns converged Lagrangian penalties that then can be used for the generation of new patterns for each employee i to be added to the RMP in its respective set of patterns $\bar{\mathcal{P}}_i$.

4.2.5 Pattern Generation

Given the penalties corresponding to the solution of P_{LR}, these penalties can be used for constructing new patterns to add to the master problem. For column generation it is known that only columns that have negative reduced costs need to be added to the master problem, and moreover, columns that have non-negative reduced costs can be left out, such that the master problem is kept small. The reduced costs for including an extra pattern in the master problem are given by:

$$r_{pi} = c_{pi} - \sum_{s \in \mathcal{S}} a_{sp} \lambda_s - \mu_i \tag{8}$$

This means that the value of μ_i is still to be determined. Here, μ_i equals the shadow price of the constraint that makes sure employee i is assigned exactly one pattern. Let P_{LP} be the linear programming relaxation of Formulation 2, let D_{LP} be the dual problem of P_{LP} and let $v(.)$ denote the solution value of

a problem. Now it is clear that $v(P_{LR}) = v(P_{LP}) = v(D_{LP})$, where the first equality holds due to total unimodularity and the second holds by the strong law of duality, where we assume that a solution exists to both P_{LP} and D_{LP}. Below D_{LP} is shown.

$$\begin{aligned} D_{LP} = \quad & \max \quad \sum_{s\in\mathcal{S}} d_s\lambda_s + \sum_{i\in\mathcal{N}} \mu_i \\ & \text{s.t} \quad \sum_{s\in\mathcal{S}} a_{sp}\lambda_s + \mu_i \leq c_{pi} \qquad \forall p \in \mathcal{P}_i, \forall i \in \mathcal{N} \\ & \qquad\qquad \lambda_s \in \mathbb{R} \qquad\qquad \forall s \in \mathcal{S} \\ & \qquad\qquad \mu_i \in \mathbb{R} \qquad\qquad \forall i \in \mathcal{N} \end{aligned} \tag{9}$$

Since λ is known from executing the subgradient algorithm, and $v(P_{LR}) = v(D_{LP})$, λ is also an solution to D_{LP}. Therefore, it is now possible to obtain the shadow prices of the assignment constraints from Formulation 2 by taking $\mu_i = \min_{p\in\bar{\mathcal{P}}_i} \left(c_{pi} - \sum_{s\in\mathcal{S}} a_{sp}\lambda_s\right)$, which holds in optimality of the dual problem. This means that μ_i equals the minimum Lagrangian costs c_{pi}^{LR} of all the patterns currently included in the RMP for employee i.

4.2.6 Subproblem

In the restricted master problem we take care of the cover constraints, which means that all other constraints have not been accounted for yet. These other constraints should therefore be dealt with in the subproblem. These constraints consist of legislation rules, personal preferences and the required skills. Since these constraints all hold per employee, the subproblems are independent of each other.

These subproblems are not equal for all employees, since employees have different contracts, different preferences and, maybe most importantly, a different history. This means that we have $|\mathcal{N}|$ different subproblems, which can be solved independently. In practice many different constraints hold, such that we propose a method that is very generic and does not require an explicit mathematical formulation of the constraints. Since the goal is to find new patterns with negative reduced costs, in general the objective function equals the reduced costs function. Since the solution of each subproblem i is independent of μ_i, we will minimise the Lagrangian cost function instead. The Lagrangian cost function is dependent on which shifts are included in the pattern:

$$c_i^{LR}(p) = c_i(p) - \lambda^T \mathbf{a}_p \qquad \forall i \in \mathcal{N} \tag{10}$$

where p is the constructed pattern, i.e, the set of shifts that are assigned in the pattern, λ is the penalty vector generated by applying the subgradient algorithm, $\mathbf{a}_p$ is a binary vector stating which shift is contained in p, and $c_i(p)$ are the costs that correspond to the pattern p. The function $c_i(p)$ consists of all the penalties invoked by violating any soft constraints for employee i in pattern p.

The function $c_i^{LR}(p)$ and the solution space determine how difficult it is to solve the subproblem. If the constraint set is fixed, a solution method optimised for this particular set can be developed. The goal of this research is to find a solution method that can deal with many combinations of constraints, and therefore must be robust in finding good enough patterns in tractable time. The downside is that this reduces the possible solution methods to general metaheuristics.

The solution method we will apply is an accept-reject heuristic that evaluates many different patterns. We have decided to create such an heuristic, since checking a pattern on feasibility and score is relatively easy compared to taking into account all constraints prior to constructing a pattern. This makes sure that the algorithm is robust and flexible to the varying constraints.

We propose a local search heuristic that constructs a pattern in an iterative way. The subproblem algorithm is initialised with $p = \emptyset$, such that no shifts are assigned to the pattern. Then the shifts that can be assigned to this employee based on their skills, are sorted in a descending order of the penalties λ_s. Next, the shifts are inserted one-by-one and the solution is accepted if it is feasible and leads to a Lagrangian costs improvement, compared to the previous solution.

The shifts are sorted based on the Lagrangian penalties, such that emphasis is put on the inclusion of shifts that are required to be inserted based on the Lagrangian penalties. In other words, the neighbourhood which is considered is biased to the inclusion of desired shifts.

After no more shifts are inserted, a transition from a greedy heuristic to a local optimisation heuristic is made. In this phase subpatterns of shifts are tried to be swapped with already assigned shifts. A subpattern is defined by a shift sequence of fixed length. The maximum time between two consecutive shifts in a pattern is bounded by t_{max}. Finding subpatterns within a pattern is easy. Just select a starting shift s, and look for the first shift s^* that is after the selected shift. If the time between does not exceed t_{max}, this shift s^* belongs to the subpattern. Creating a subpattern from unassigned shifts is a little more

difficult, since many possibilities are possible.

The function selects the possible first shifts in a subpattern, and those will depend on the start interval I that is provided as an input parameter. The output is a set Φ of all possible subpatterns:

ConstructSubpatterns$(I) : \Phi$

1: Get all unassigned shifts $\mathcal{S}^{Unassigned}$ sorted on start time
2: Create empty list of Subpatterns Φ
3: $i := 0$
4: **for** $i := 0$ to $|\mathcal{S}^{Unassigned}|$ **do**
5: $\quad s := \mathcal{S}_i^{Unassigned}$
6: $\quad$ **if** $b_s \in I$ **then**
7: $\quad\quad$ Create empty Subpattern ϕ
8: $\quad\quad$ AddShiftsToSubpattern(ϕ, Φ, $\mathcal{S}^{Unassigned}$, i)
9: $\quad$ **end if**
10: **end for**

The rest of the shifts in the subpattern need to start in an interval that is dependent on the previous shift in the sequence. This interval is dependent on a minimum time t_{min} and a maximum time t_{max} between two consecutive shifts in a subpattern. Taking $t_{min} = 0$ and $t_{max} = \infty$ means that any shift after a given shift is allowed as a next shift in the subpattern. Choosing t_{min} and t_{max} is a modelling choice where the convergence rate of the algorithm and solution quality need to be balanced. The ordering of shifts in time is exploited by starting the search for a possible successive shift at the index of the current shift, and the search is terminated when a shift is encountered that has a starting time too far away from the current shift. Below pseudo code for the recurrent function that adds all shifts is provided:

AddShiftToSubpattern(ϕ, Φ, $\mathcal{S}^{Unassigned}$, i)

1: $\phi^* := \phi$.Copy()
2: Add shift $\mathcal{S}_i^{Unassigned}$ to ϕ^*
3: **if** ϕ^* is of required length **then**
4: $\quad$ Add complete subpattern ϕ^* to Φ
5: $\quad$ Exit function
6: **end if**
7: **for** $j := i$ to $|\mathcal{S}^{Unassigned}|$ **do**
8: $\quad s := \mathcal{S}_j^{Unassigned}$
9: $\quad s^* :=$ last shift in ϕ^*

10: **if** $b_s - e_{s^*} \geq t_{max}$ **then**
11: Break loop
12: **else if** $b_s - e_{s^*} > t_{min}$ **then**
13: AddShiftToSubPattern(ϕ^*, Φ, $\mathcal{S}^{Unassigned}$, j)
14: **end if**
15: **end for**

Given a subpattern from the assigned shifts, it is tried to swap this subpattern with one of the created possible alternative subpatterns in Φ. First a fast check is executed to check whether a possible alternative does not overlap with one of the remaining assigned shifts. Next, the score is computed when the alternative subpattern is inserted instead of the currently assigned subpattern. If this leads to a score improvement, it is checked whether this assignment is also feasible. This is done for all possible alternatives in Φ, and the best improvement is executed if an improvement is found. Otherwise, no swap is executed. Since the swapping process is not too intuitive, we support it with an example in the following section:

Swap Example

To support our explanation on how the subpatterns are swapped, let us provide an example. The example is based on real events that happen during executing the algortihm on the benchmark cases that will be introduced in Section 5. Let us assume there are four shift types: an early (E), day (D), late (L) and a night (N) shift type, with positive demand on every day in the scheduling period, such that a shift of every type might be scheduled on each day. In Table 1 the shift types are defined.

Shift Type	Begin Time (b_s)	End Time (e_s)	Duration ($e_s - b_s$)
Early	06:00	14:00	8h
Day	09:00	17:00	8h
Late	14:00	22:00	8h
Night	22:00	06:00	8h

Table 1: Swap Example: The shift types present in the problem.

Now let there be an employee for which the subproblem is going to be solved, which does not have the skill to perform a N shift. This means that only subpatterns have to be created of shifts that are of type E, D or L. After the construction phase has been finished, let there be within the entire schedule of this employee a sequence of three type E shifts in a row as is shown in Table 3.

Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon
$\cdots$			E	E	E			L	L	$\cdots$

Figure 3: Swap Example: Partial schedule of an employee.

For this shift sequence of length three with its starting shift s of shift type E on Monday, we want to check if a suitable swap can be made. Let us have defined that possible alternative subpatterns may start within the open interval $I = (b_s - 24, b_s + 24)$, where this interval is defined in hours. Possible alternative starting shifts then become the shifts of shift types D and L on both Sunday and Monday.

Next to that, let us define $t_{min} = 14$ and $t_{max} = 32$ hours. Based on these parameters the possible successor shifts for each shift s of a certain shift type on a day number t are shown in Table 2.

Shift s	Possible Successors		
E_t	E_{t+1}	D_{t+1}	L_{t+1}
D_t	D_{t+1}	L_{t+1}	E_{t+2}
L_t	L_{t+1}	E_{t+2}	E_{t+2}

Table 2: Swap Example: For each shift it is defined what the possible successor might be in the subpattern based on the interval $[e_s + t_{min}, e_s + t_{max})$.

Since there are four possible starting shifts according to the chosen interval I, and for each shift there are three possible successors, there are in total $4 \times 3^2 = 36$ combinations that can be considered as possible alternative subpatterns of length three. For each of these subpatterns, it is calculated if it leads to a Lagrangian cost reduction, and if so whether the pattern is feasible regarding the hard constraints. The Lagrangian penalty matrix for the shifts that have to be taken into account for creating alternative subpatterns is provided in Table 3. The Legislation Rules that hold for this employee, are that the employee has to be at least two days off in a row, it has to work at least two days in a row, and that, provided that he already works the second weekend in the provided interval, it cannot work the first as well. Next to that, for the simplicity of the example, it is assumed that the employee has no personal preferences on these days, such that $c_i(p) = 0$ for any p, and $c_i^{LR}(p) = -\lambda^T \mathbf{a}_p$.

Based on the Legislation Rules that should hold for this employee, the employee cannot have any shifts assigned on Sunday (too much weekends of work). Next to that, many combinations will be rejected on feasibility, because it will lead to violations of one of the other two constraints. For example, assigning a Monday shift, with rather high Lagrangian penalty, will mandate that a

Shift Type	Sun	Mon	Tue	Wed	Thu	Fri
E	280.287	77.758	-34.104	81.579	-1.752	25.522
D	280.250	**81.634**	**-36.563**	**81.565**	-3.642	25.400
L	278.422	79.779	-36.535	76.466	2.558	26.843

Table 3: Swap Example: The Lagrangian penalty matrix stating the Lagrangian penalty for every shift type on a certain day. Note that a day and shift type combination defines a shift, such that the matrix contains the Lagrangian penalties per shift. The bold-faced numbers correspond to the Lagrangian penalties of the shifts that will be assigned after the swap.

Tuesday shift is assigned as well, with negative Lagrangian penalty, due to the constraint that the employee has to work at least two days in a row. Therefore, all subpatterns that are created with a shift on Monday but not on Tuesday will be rejected. The swap that leads to the maximum score improvement and is feasible after checking the Legislation Rules is given by the sequence of D shifts on Monday, Tuesday and Wednesday. The assigned subpattern of E shifts is then replaced by the created subpattern, resulting in the schedule provided in Figure 4.

Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon
$\cdots$			D	D	D			L	L	$\cdots$

Figure 4: Swap Example: Improved schedule of an employee.

If a swap has been successfully executed, we continue by applying a very simple move heuristic. This heuristic tries to move all shifts in $\mathcal{S}_i$ to the pattern if it is not yet assigned, and if a shift is assigned already, it is tried to unassign this shift. This is done since a swap can make room in the pattern for shifts that have a positive effect on the solution value of the subproblem, as well make it possible to unassign redundant shifts that have been assigned to make the pattern feasible in the first place. Both findings were seen during experiments, and therefore this addition was made to the algorithm.

It is explained that we generate subpatterns of different lengths. The number of alternative subpatterns increases exponentially in the length of the subpattern. Therefore, it is undesirable to evaluate many subpatterns of increased length. Taking this into account, it was decided to perform swaps in increasing length, such that first it is tried to swap subpatterns of length one, and then increase the subpattern length by one every time no swaps of the current length can be found anymore. If a subpattern of a length greater than one has been successful, the heuristic resets and first tries to find swaps of length one again.

Next to that, the starting interval I has a great influence on the number of

alternative subpatterns as well. Therefore it is proposed to choose I small at first. Next, after no more swaps can be performed to improve the quality of the schedule, it is checked whether the constructed pattern has negative reduced costs. If the pattern has negative reduced costs, we return the pattern, and terminate the subproblem phase. If this is not the case, the length of interval I is increased, such that a larger neighbourhood is explored, with the goal to further reduce the Lagrangian costs and eventually end up with a pattern that does have negative reduced costs.

The global outline of the subproblem algorithm is as follows:

SubproblemAlgorithm: p

```
1: Order shifts on Lagrangian penalties
2: Insert shifts in a greedy manner
3: for I ∈ [I_small, I_large] do
4:    while ImprovementFound do
5:       ImprovementFound := False
6:       for i := 1 to MaxSubpatternLength do
7:          ImprovementFound := SwapSubpatternsOfLength(i, I)
8:          if ImprovementFound then
9:             TryMoveShifts
10:            Break
11:         end if
12:      end for
13:   end while
14:   if Reduced costs < 0 then
15:      Break
16:   end if
17: end for
```

4.2.7 Column management

The patterns that are created are only added to the RMP when they are of added value. In general it holds that a column (pattern) will contribute to lower solution value when it has negative reduced costs, i.e., $r_{pi} < 0$, where r_{pi} is provided in Equation 8. From D_{LP} it is derived that the value of μ_i equals the Lagrangian costs of the column selected for employee i in the solution of P_{LR}. This means that after rewriting the reduced costs function a pattern p' is only added to the master problem when:

$$c_{p'i}^{LR} < \min_{p \in \bar{\mathcal{P}}_i} c_{pi}^{LR} \tag{11}$$

This inequality intuitively makes sense, since the solution to $\theta(\lambda)$ is obtained by selecting for each employee i the best pattern. Therefore, the new pattern p' will only contribute, when it is better than the currently best found column for fixed Lagrangian penalties λ.

In the proposed algorithm multiple subproblems are solved at the same time, where between zero and $|\mathcal{N}|$ patterns with negative reduced costs are produced. Therefore, it has to be decided which ones will be added to the master problem. The simple rule is applied that all patterns with negative reduced costs are added to the problem. The downside of this rule is that (at most) $|\mathcal{N}|$ patterns will be added to the master problem, where only one of those patterns might be used and making the others redundant. On the other hand, choosing only one of the columns, might result in reconstructing earlier found patterns in later iterations. This is expensive, especially because it is decided that new patterns are always constructed from scratch. This means that quite some iterations will be part of the construction phase, where in every move the pattern has to be checked on feasibility, and will make reconstructing already constructed patterns particularly expensive. Therefore it is decided to add all constructed patterns with negative reduced costs.

In column generation in general, when only one new column with negative reduced costs is added in each iteration, it is known that this column will be included in the basis of the next solution of the master problem when it is solved exactly. In that case it is therefore always of use to set the variable corresponding to the pattern that is constructed equal to one to speed up computation of the master problem.

As has been discussed, we add multiple patterns in each iteration, instead of just one. Therefore, we have to decide which patterns to fix. Fixing a pattern is done by setting the value of the binary variable corresponding to this pattern to one, such that it is included in the schedule for sure. Fixing more than one of the newly created patterns at the same time does not make sense in general, since it will lead to bad solutions. Although, it is not necessary that the pattern with the most negative reduced costs will lead to the optimal solution of P_{LR}, it is decided to fix that pattern. This means that we set the variable $x_{p'i} = 1$, where i is the employee for which the created pattern p' fulfils

$$r_{p'i} = \min_{p \in \mathcal{P}_C} r_{pi} \tag{12}$$

where $\mathcal{P}_C$ is the set of constructed patterns with negative reduced costs that will be added to the master problem in the current iteration.

Solving the Lagrangian dual problem then involves the union of $|\mathcal{N}| - 1$ sets of patterns $\bar{\mathcal{P}}_i$ that need to be considered. We suggest to further reduce the problem size by fixing the patterns for multiple iterations. Let n denote the number of iterations pattern p' is fixed for. Note that, it only makes sense to fix a pattern for at most $n = |\mathcal{N}|$ iterations.

Fixing a pattern for $n = |\mathcal{N}|$ iterations will lead to a problem were in every iteration only one subproblem is solved. When the subproblem heuristic is not able to find a pattern with negative reduced costs, the master problem is enlarged, since in the next iteration another pattern that was fixed is freed again. If a different solution to the new Lagrangian subproblem is found, new patterns might be found. For any n we say the algorithm is converged, when no patterns are fixed anymore, which occurs when the subproblem heuristic was not able to construct any patterns with negative reduced costs for n iterations.

5 Data

To be able to test the performance of the newly proposed algorithm, we make use of benchmark instances created by Curtois (2014). The creators made a "diverse and challenging collection of benchmark test instances from various sources including industrial collaborators and scientific publications". The 24 instances cover a wide variety of input sizes, where the planning period differs from 2 weeks, to one year, the number of employees differ from 8 to 150, and the number of shift types differ from 1 to 32. For some of these instances the optimal solution and solution value is known. For the larger instances only bounds on the optimal solution value are known. We decided that we will test the methodology on the first half of the instances, since we believe that those instances are relevant in practice. In Table 4 a brief description for those instances is provided.

Inst.	#Days	#Employees	#Shift Types	Optimal Score	Penalty On Preferences	Ave. # Shifts per Employee per Day
1	14	8	1	607	6	0.59
2	14	14	2	828	28	0.51
3	14	20	3	1001	1	0.51
4	28	10	2	1716	15	0.59
5	28	16	2	1143	42	0.62
6	28	18	3	1950	46	0.56
7	28	20	3	1056	56	0.54
8	28	30	4	1300	99	0.56
9	28	36	4	439	39	0.40
10	28	40	5	4631	29	0.58
11	28	50	6	3443	20	0.57
12	28	60	10	4040	40	0.58

Table 4: A brief description of the first 12 instances. In the optimal solution a penalty of 100 and 1 is respectively added for a unit of under and over staffing. Rest of the penalties are induced for not satisfying personal preferences (note that column *penalty on personal preferences* coincides with the optimal score of the problem with adjusted demand, see Section 5.1). The last column shows how many shifts per employee per day are assigned in the best found solution.

The created benchmark instances include all the 4 classes of constraints that are mentioned in Section 3. For these instances the optimal value is known, which includes penalties on the deviations from demand, since the covering constraints are modelled as soft constraints. The provided solutions to these instances contain a penalty of 100 for a unit of under-staffing and a penalty of 1 for a unit of over-staffing with 1. All constraints in the class Legislation Rules are modelled as hard constraints. The following nine constraints are included

in this class of constraints:

Legislation Rules included in benchmark instances

An employee ...

1. may only be assigned one shift a day
2. is required to have minimum amount of rest time between two consecutive shifts
3. may only be assigned maximum number of shifts of each shift type
4. may only work a maximum amount of time in the planning period
5. is required to work a minimum amount of time in the planning period
6. is only allowed to work for a maximum number of days in a row
7. is required to work a minimum number of days in a row
8. is required to have a minimum number of days off in a row
9. may only work a maximum number of weekends in the planning period

Constraint 1 and 2 are the only two constraints that are independent of the employee, whereas all other constraints are employee dependent. The constraints class of Required Skills are for these instances translated to which shifts can and which shifts cannot be performed by an employee. These constraints are, just as the legislation rules, modelled as hard constraints. Finally, the class of Personal Preferences constraints are modelled partially as hard constraints and partially as soft constraints. This means that some requests are required to be accepted, whereas others are encouraged to be accepted, or discouraged to be rejected. The personal preferences that are modelled as soft constraints, consist of requests to work a certain shift, not to work a certain shift, or as a request to not work an entire day. These soft constraints have a penalty 1, 2 or 3.

The solution value z of the schedules is then determined by the following equation, where the first term equals the total penalty for under staffing, the second term equals the penalty for over staffing, and the third term includes all penalties regarding the unmet personal preferences:

$$
\begin{aligned}
z \quad = \quad & 100 \cdot \sum_{s \in \mathcal{S}} \max \left\{ d_s - \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} a_{sp} x_{pi}, 0 \right\} \\
& + \quad 1 \cdot \sum_{s \in \mathcal{S}} \max \left\{ \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} a_{sp} x_{pi} - d_s, 0 \right\} \quad + \quad \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} c_{ip} x_{ip}
\end{aligned} \tag{13}
$$

5.1 ORTEC algorithm on benchmark instances

The modelling of the NRP instances by Curtois (2014) deviates in some ways from the one of ORTEC in their current algorithm. Therefore, to be able to test the ORTEC algorithm on the benchmark instances some adjustments to the data had to be made.

First of all, some of the hard constraints that hold in the benchmark instances, cannot be modelled as such by ORTEC. For example, Constraint 5 about the minimum workload assignment cannot be modelled as a hard constraint, since the algorithm cannot cope with an infeasible schedule to start with. Also, Constraint 7 being modelled as a hard constraint cannot be applied, an insertion of one shift at the time cannot be accomplished. Therefore, we adjusted those constraints from hard to soft constraints, such that it is allowed to generate schedules that lie in the infeasible space according to Curtois (2014). Violations of the soft constraints are highly penalised, such that generating schedules inside these spaces is highly discouraged, but allowed when required.

Another downside to these instances is that the algorithm of ORTEC is based on the assumption that assigning more shifts is always better than complying to soft constraints (see Section 4.1). Since we have relaxed some of the hard constraints to soft constraints, the algorithm automatically assumes that inserting an extra shift is better than complying to these relaxed hard constraints. Running the current algorithm to the instances then would result in higher services levels (percentage of assigned demand) at the cost of violating constraints. Since for all benchmark instances it is known that demand cannot be covered on all days, the ORTEC algorithm will never perform well, as long as relaxed hard constraints are present, and over staffing is allowed.

At first it was believed that changing the ORTEC algorithm would be the best option to overcome the problem, but after testing it was concluded that the changes had a lot of impact on the performance of the algorithm, and therefore it would not be fair to compare this algorithm with the adjusted

ORTEC algorithm. Since Curtois (2014) also provided the solutions next to the solution values, it was decided to adjust the demands for every instance to the demand covered in the optimal solution, such that the benchmarks are (more) applicable to how ORTEC currently tackles the NRP, since allowing for over staffing is not necessary anymore. This means that d_s is derived from the optimal solution of the benchmark instances instead of from the instances itself. Therefore, we also omit the penalties for scheduling over and under staffing, and the function for evaluating a schedule becomes in line with the objective function of the mathematical formulation P_{IP} provided in Section 4:

$$z = \sum_{i \in \mathcal{N}} \sum_{p \in \mathcal{P}_i} c_{ip} x_{ip} \tag{14}$$

Here, c_{ip} includes the penalties for unmet personal preferences, as well as penalties on relaxed hard constraints, which have been assigned a value of 10000.

Although we are now able to test both the current ORTEC algorithm and the newly proposed algorithm on the benchmark instances, the downside is that these changes to the original modelling result in a change in difficulty of the problem. It might be that initially many optimal solutions with the same solution value are present, such that we restrict ourselves to fewer optimal solutions by adjusting the demand (1). On the other hand, the size of the solution space is reduced (2), since the ORTEC algorithm will, after adjustment of the instances, be initialised in such a way that it will come up with solutions that include over staffing.

5.2 Proposed algorithm on benchmark instances

To be able to compare the algorithms, we will run the newly proposed algorithm on the same adjusted instances as the ORTEC algorithm. The proposed methodology, contrary to the ORTEC algorithm, does not exploit the demand in such way that no solutions are able to be created that include over staffing. This means that although it is known that over staffing is not allowed, the algorithm will still explore solutions in the infeasible space. Therefore, one might argue that making these changes give a head start to the ORTEC algorithm, since the downside of the adjustment (1) applies to both the ORTEC and the proposed algorithm, whereas the upside (2) only applies to the ORTEC algorithm.

6 Results

In this section it is addressed how the algorithm performs. The algorithm has been implemented within the OWS product of ORTEC, and has been written in Delphi. When we refer to CPLEX, we refer to a CPLEX solver version 12.1.

In this section, first of all, the results of applying the subgradient algorithm to the Lagrangian relaxation will be discussed. This is followed by the analysis on the subproblem heuristic. Afterwards, the solutions computed by the proposed algorithm are compared with the solutions of the ORTEC algorithm.

6.1 Lagrangian Relaxation

The subgradient algorithm with Held-Karp step size was implemented to solve the Lagrangian dual problem P_{LR}, as explained in the Section 4. Experiments showed that applying a step size parameter of $b = 10$, showed best results in general. In Figure 5 it is shown how the solution value of the Lagrangian relaxation behaves during execution of the subgradient algorithm on Instance02. During execution of the algorithm on this particular instance, 68 column generation iterations are executed, after which the algorithm was not able to find any more patterns with negative reduced costs. In each column generation iteration the subgradient algorithm is executed.

The horizontal line in the graph shows the optimal value given that all patterns are added to the master problem. This value is obtained from the published benchmark instances by Curtois (2014). The dots show the value of $\theta(\lambda)$ in each subgradient iteration, the lower bound value of the problem with the patterns added so far. This explains why the dots can lie above the line of optimality, since not all necessary patterns to be able to reach optimality or even feasibility are added to the problem at that time.

Each column generation iteration can be recognised by the ray of dots that curves to convergence. Note that the rays corresponding to the first 17 column generation iterations, converge outside the range of the figure. It can be seen that over the column generation iterations the end point of the rays converge. In other words, the solution of the Lagrangian dual problem P_{LR} converges. The reason that the end points of the rays do not decrease monotonically is that some patterns are fixed in each iteration, as is described in Section 4.2.7. Therefore the solution space of the RMP is different per column generation iteration and a non-monotonic decrease of the solutions of the Lagrangian dual problem are possible.

This figure also immediately shows the major disadvantage of the method-

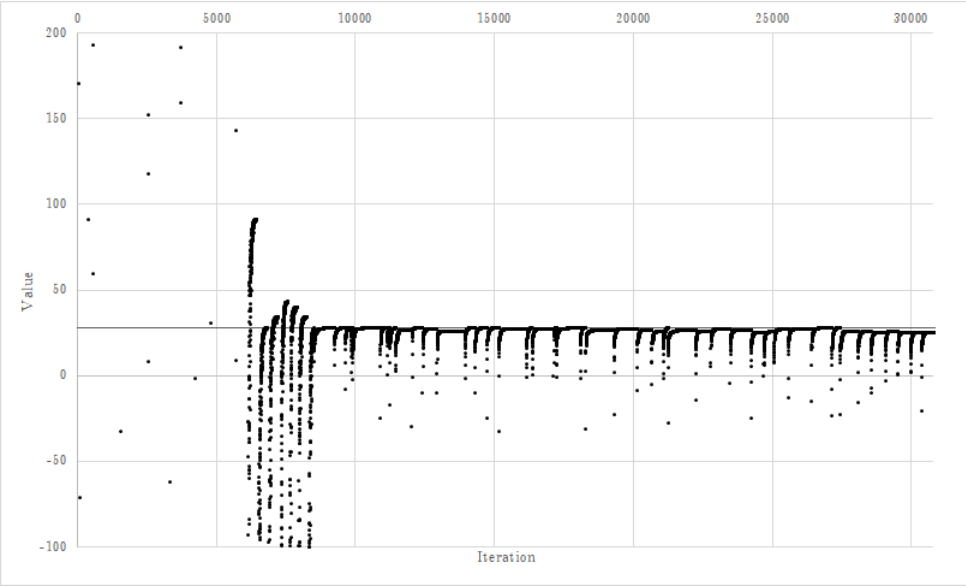

Figure 5: Subgradient algorithm on benchmark Instance02

ology, which is that the solution of P_{LR} in any column generation iteration does not give any guarantees on the original problem P_{IP} including all patterns. This is due to the fact that the we solve the restricted master problem, so the solution value the Lagrangian dual problem P_{LR} only provides a bound on P_{IP} with the subset of all possible patterns, i.e., those patterns that have been generated by solving the subproblem.

The Lagrangian heuristic based on Caprara et al. (1999) is evaluated by comparing its *big-M* score to the *big-M* score of the solutions of the Lagrangian relaxed solutions, were we have used $M = 1000$ for all instances. Experiments showed that the constructed heuristic performs on average better, i.e., the average solution value after applying the heuristic in each iteration is lower than the average solution value of the solution of the Lagrangian relaxation. The Lagrangian relaxed solutions on the other hand provide in general the most tight (artificial) upper bound on the optimal solution. Since only the best solution so far is of use in this algorithm, it is decided that the heuristic is not of sufficient competition, mainly because it is more computationally expensive, as discussed in Section 4.2.4, and the solutions are not of the desired quality.

The Lagrangian heuristic does not provide very good solutions, due to the fact that it is based on an algorithm constructed for the set covering problem, whereas we are applying it to the set partitioning problem. Next to that, in our

case we also have to deal with the side constraint that only a limited number (the number of employees) of patterns can be selected. The difference in results for the two different problems can be explained best by two examples:

Assume that there are two employees, and a scheduling period of 10 days. For the first example, assume that there is one constraint for each employee and it states that an employee has to work at least five days. Now selecting a pattern that contains more than five shifts for the first employee, will result in a schedule that includes over staffing when selecting a pattern for the second employee. This example shows that having a set partitioning problem instead of a set covering problem leads to bad quality results, since a set covering problem is equivalent to a problem where over staffing is not penalised.

For the second example, assume that the same setting, but the constraint states that instead of working at least five days, each employee can only work at most five days. Now selecting a pattern with less than five shifts included for the first employee, will result in a schedule with under staffed demand. This is a result of the side constraint that only a limited number of patterns can be selected.

The examples show that the Lagrangian heuristic does not take into account possible over- or under staffing in the first pattern assignments, which will have a bad effect on the final schedule. In other words, the heuristic is too greedy for applying it to a set partitioning problem with a side constraints that limits the number of assignments.

This could have been overcome by taking into account information on other employees when assigning a pattern to the current employee. An adjustment that takes into account such aspects, will have great impact on the computational complexity of the heuristic. The goal of the Lagrangian heuristic was to construct a good enough schedule fast, but it is shown that this heuristic is not capable of this. On the other hand, it will be shown in the coming section that the algorithm is capable of constructing good patterns without a good Lagrangian heuristic. Constructing a good final schedule can then be done by choosing the patterns in an exact manner.

6.2 Subproblem heuristic

In Section 4.2.6 a heuristic method has been explained that optimises the reduced costs function of a pattern. This heuristic consists of a greedy construction phase, after which shifts are moved and swapped to improve the quality of the pattern. A swap is performed on two subpatterns; one that consists of shifts that are assigned to the employee, and one that consists of unassigned shifts.

As mentioned in Section 4.2.6, many different subpatterns can be created from the set of unassigned shifts. Therefore, it was decided to limit this number of subpatterns, by selecting a subset of these subpatterns. The subset is selected based on a maximum deviation between the start times of the first shift in the subpattern of assigned shifts, and the subpattern of unassigned shifts.

In Figure 6 we present the performance of the swaps performed on running the algorithm on Instance 4 that includes 28 days in the planning period. During this experiment we did not limit the deviation of starting times of the subpatterns, such that number of days between the start of the subpatterns can be at most 27. The results shown, are based on the aggregated data of the 10 employees in the instance, each having a separate subproblem. In the figure we show the number of days between the starts of the two subpatterns, for different subpattern lengths. From this figure for swaps of subpatterns with a length greater than one, that most swaps are performed within the same day. Furthermore, a great part of the successful swaps are performed within a week. This is emphasised by Table 5.

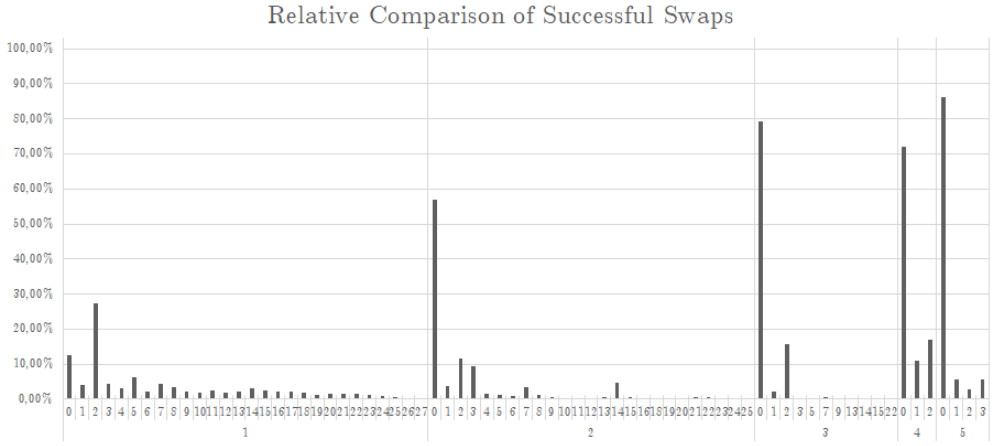

Figure 6: The figure shows for each deviation between the start of two successfully swapped subpatterns its percentage of the total succesfully swapped subpatterns. This is shown for swaps between subpatterns of length 1 to 5.

Table 5 also shows that for swaps of subpatterns of length one, these findings do not hold, and that there is still a relative large part of the successful swaps that are performed with more than a week between the starts of the two subpatterns. Based on these findings it is decided that for swaps of subpatterns of length one, that we will always consider the entire planning interval. For swaps with a length greater than one, we first try to find a pattern with negative re-

duced costs by only considering swaps with a deviation of one day, i.e., we search for a subpattern with a starting shift s^*, that is an alternative for a subpattern with starting shift s, such that $b_{s^*} \in I = (b_s - 24, b_s + 24)$. If no pattern with negative reduced costs is found, the interval is extended such that the deviation becomes a week, i.e., the interval is extended to $I = (b_s - 7 \times 24, b_s + 7 \times 24)$, such that it possible to further improve the pattern.

		Length Subpattern				
		1	2	3	4	5
Days Between Subpatterns	0	12,57%	57,02%	79,39%	72,08%	86,11%
	1 to 7	51,78%	31,96%	19,36%	27,92%	13,89%
	8 to 28	35,65%	11,02%	1,24%	0,00%	0,00%

Table 5: The percentage of successful swaps with a certain deviation between starting times of the subpatterns. This table summarises the results shown in Figure 6.

To test whether the subproblem is able to come up with good quality patterns, it was decided to run the algorithm and after either the time-limit was reached or the algorithm could not find any new patterns. Afterwards, the original problem P_{IP} was solved in CPLEX including all the generated patterns $\bar{P}_i \ \forall i \in \mathcal{N}$. Solving the original problem, makes sure that the measurement of the quality of the patterns is not influenced by the Lagrangian heuristic that is implemented. All runs that have been executed have been parameterised with a minimum and maximum time between shifts within a subpattern $t_{min} = 8$ and $t_{max} = 32$ hours, according to the Dutch Regulation rules.

The results of solving P_{IP} exactly after running the algorithm for a maximum time of 1 hour, where the time that CPLEX may use is limited by 15 minutes. It can be seen for instances 1 to 5, for which the algorithm converges before reaching the time limit, the created patterns are of good quality, such that it is possible to create a solution in which all demands are met exactly. For instances 1 to 3 it is shown that optimality can be reached given the created patterns. For instances 6 to 12, CPLEX could not find a solution in which there is no deviation from demand. On the other hand, the algorithm did not converge within an hour for these instances, so the heuristic was still able to find patterns with negative reduced costs. Based on these findings, we conclude that the subproblem is of sufficient quality to generate good patterns based on the penalties provided by the RMP.

Inst.	# Shifts	Run Time	CPLEX Run Time	Total Staff Deviation	Score
1	66	00:04	00:00	**0**	**6**
2	100	00:51	00:00	**0**	**28**
3	144	02:25	00:00	**0**	**1**
4	166	40:26	00:01	**0**	21
5	278	38:33	00:09	**0**	71
6	284	01:00:00	15:00	5	60
7	305	01:00:00	15:00	7	120
8	471	01:00:00	15:00	20	246
9	406	01:00:00	15:00	16	124
10	649	01:00:00	15:00	30	118
11	800	01:00:00	15:00	48	61
12	967	01:00:00	15:00	119	505

Table 6: Results of applying CPLEX for solving P_{IP} to the patterns that have been created by the new algorithm (fixed max. run times of 1 hour for the implemented algorithm and 15 minutes for CPLEX)

6.3 Solution quality

The solution method that is proposed in this paper is chosen to be very flexible and can be applied to varying sets of NRP instances. Expecting to be able to obtain similar results as have been reported on the benchmark instances from Curtois (2014) is irrational, since the branch-and-price algorithms that they have implemented are specialised for solving the NRP that contains the nine constraints mentioned in Section 5. Also, the algorithm has been implemented within the ORTEC product, which causes a lot of overhead in computing. Therefore, it is more suitable to test the new methodology against the current algorithm of ORTEC. Furthermore, since we have adjusted the problem instances to test the current algorithm of ORTEC fairly, it is also not possible to compare the solutions on the adjusted instances with the branch-and-price algorithms.

First we will show the performance of the current algorithm of ORTEC. As has been discussed in Section 5, some constraints are modelled as soft constraints. These constraints are penalised with a cost of 10000 per unit of violation. This value is chosen such that it does not influences the other soft constraints on personal preferences. Furthermore, we decided to let the maximum run time depend on the instance size. The maximum run time is set to the multiplication of the number of days, the number of times the number of employees and a time unit. This decision is based on the assumption that the average demand per day aggregated over the shift types is stable, for differing

instances. This assumption holds well as is shown in Table 4, where in general an employee is on average assigned a bit more than one shift per two days. At last, ORTEC algorithm consists partially of a GA (see Section 4), which requires to define a number of individuals that is used. It was decided to test the algorithm without the GA and with the GA where every generation contains 10 individuals. Since the algorithm including the genetic phase, outperformed the algorithm without, only the results including the genetic phase are provided. In Table 7 the results of running the ORTEC algorithm are shown.

The ORTEC algorithm has as a main criterion to minimise the staff deviation as the main criterion, even at the expense of violations of relaxed hard constraints. When no staffing deviations nor violations are present, minimising the score becomes important. The results show that the current algorithm is not working very well on the instances, since except for instance 1, the algorithm is not capable of finding any solution that has no staffing deviation and no violations.

Although the adjustment of the demand made it possible to obtain the optimal solution to the instances, it is seen that only for the smallest instance the algorithm was able to find this optimum. Furthermore, it is seen that the spread within the obtained results is very large. This means that the algorithm is not very stable, and finding good results is rather random.

The proposed algorithm was given more time than the ORTEC algorithm. This decision was made, to simulate the effect of solving the subproblems in parallel. The proposed algorithm is designed to exploit the property of parallelising, but it was found out that it was very difficult to implement this efficiently in Delphi, without considering memory management and allocation. These aspects were not forseen at the start, and are believed to be outside the scope of this thesis. The factor 4 has been chosen, since often a computer is armed with 4 processing cores. We acknowledge that this way of testing the effect of parallelising does not come close to using a platform in which parallelising is possible.

In Table 8 it can be seen how the algorithm proposed in this thesis performs. In the case of the new algorithm, highly penalised relaxed hard constraints were never violated, and therefore that column is omitted in Table 8. The main criterion is to minimise the total staff deviation, since staffing deviation is considered an infeasibility. This means that we want to find the most feasible solution, after which the score is optimised. Therefore, we evaluate the algorithm in the same way as the ORTEC algorithm has been evaluated.

The results show that the Lagrangian relaxation solution of the first two

			Average	Best Solution			Worst Solution		
Inst.	# Shifts	Max. Run Time	Total Staff Deviation	Total Staff Deviation	# Violations	Score	Total Staff Deviation	# Violations	Score
1	66	00:56	0	**0**	**0**	**6**	**0**	**0**	**6**
2	100	01:38	0,1	0	2	26	1	0	31
3	144	02:20	0,3	0	2	8	1	3	24
4	166	02:20	0,1	0	3	36	1	7	26
5	278	03:44	0,8	0	14	65	2	14	66
6	284	04:12	2,3	0	27	85	4	19	63
7	305	04:40	2,8	2	13	92	5	12	102
8	471	07:00	9,5	8	66	190	11	73	197
9	406	08:24	0,1	0	5	143	1	12	130
10	649	09:20	10,9	9	23	155	14	31	161
11	800	11:40	8	6	247	200	10	386	181
12	967	14:00	19,1	16	89	297	24	70	264

Table 7: Results of ORTEC algorithm (with Genetic phase, including 10 individuals). Due to the high spread in results, we show results from 10 runs.

			Solution			CPLEX Solution		
Inst.	# Shifts	Max. Run Time	Real Run Time	Total Staff Deviation	Score	Run Time	Total Staff Deviation	Score
1	66	4 × 00:56	00:04	0	9	00:00	**0**	**6**
2	100	4 × 01:38	00:53	0	29	00:00	**0**	**28**
3	144	4 × 02:20	02:25	12	13	00:00	**0**	**1**
4	166	4 × 02:20	09:20	12	66	04:46	1	77
5	278	4 × 03:44	14:56	12	110	01:36	0	117
6	284	4 × 04:12	16:48	29	132	15:00	10	110
7	305	4 × 04:40	18:40	38	184	15:00	11	141
8	471	4 × 07:00	28:00	56	293	15:00	23	286
9	406	4 × 08:24	33:36	56	115	15:00	21	130
10	649	4 × 09:20	37:20	88	378	15:00	41	146
11	800	4 × 11:40	46:40	99	357	15:00	54	71
12	967	4 × 14:00	56:00	189	509	15:00	119	505

Table 8: Results of the new algorithm (relative run times)

instances provide near optimal solutions to the original problem. The algorithm is relatively fast for the first three instances, but slows down drastically when the number of days in the planning interval is doubled, which is the case for the other instances. As has been shown in Section 6.2, the algorithm performs much better when the patterns that are created, are combined in an exact manner, after the algorithm has been terminated, and the created patterns are used as input.

Again we have used CPLEX in which we have implemented the original problem P_{IP} in Formulation 2. The results of CPLEX are also shown in Table 8. It can be seen that the results significantly improve, and that for the first three instances optimal solutions have been obtained. Furthermore, a feasible solution is obtained for Instance 5.

If we compare the results of the ORTEC algorithm in Table 7 and the results of this algorithm in Table 8, we see that indeed constructing patterns per employee and combining those patterns leads to an increased solution quality. Where the ORTEC algorithm provides solutions with infeasiblities in all cases except for the first instance, this algorithm finds near optimal solutions for the first two instances. It especially outperforms the ORTEC algorithm when CPLEX is used to obtain the solution to the original problem P_{IP}, after the algorithm has converged. For the larger instances, both algorithms are not able to come up with good solutions. The ORTEC algorithm provides solutions where many constraints on Legislations Rules are violated and staff deviations are present, whereas the algorithm discussed in this thesis provides large staff deviations.

7 Conclusion

In this thesis we tried to construct a very generic algorithm that is robust to varying instances of the NRP. A generic algorithm is required to able to cope with the different rules and regulations that hold in different sectors and countries. Furthermore, based on the literature research, it was decided to decompose the problem. Since the diverse rules and regulations hold per employee, we chose to decompose the problem in such a way that schedules per employee are constructed, which will be combined in a different stadium in the algorithm. The schedules for all the employees are created with an accept-reject heuristic, such that the algorithm satisfies the requirement to be generic.

From this research it can be concluded, that this way of decomposing leads to good results for smaller instances. Especially, when exact methodology is applied for combining the shift patterns constructed for the employees, it is shown that the algorithm developed for this research, outperforms the ORTEC algorithm. For larger instances it is experienced that it is difficult to find good schedules within a limited amount of time, with such a generic algorithm.

For the proposed subproblem heuristic we see that it is good enough to generate good quality shift patterns, but since the heuristic does not take into account any explicit information from the imposed constraints, many different solutions have to be evaluated. Since the larger instances have a larger planning period and contain many different shift types, the heuristic becomes computationally much more expensive. This becomes clear from the results as well.

Therefore, we conclude that it is very difficult to make a generic algorithm that is very robust to differing constraints. We accomplished to develop such a method, but the lack of constraint information taken into account in the algorithm, limits the possibilities to find a good schedule fast.

For further research, we therefore recommend to not try to construct a methodology that is able of coping with all the different constraints, but instead look into all the constraints, and find out how to they can be combined or even omitted. If it is possible to reduce the variety of constraints, and one can come up with abstract constraints that the original constraints can be translated to, we think that it would be of great value to this field of research.

Next to that, it should be investigated, whether all the exceptions in the laws and regulations, are of added value in the first place. By including all the exceptions to the feasible space, the space becomes much larger and more complex to explore. This limits the possible solution approaches, such as exact solution approaches. Omitting any exceptions allows for methodology that is

capable of solving these problems faster, but might also result in previously optimal solutions to the problem being infeasible, or even make the entire problem infeasible. In this thesis, we noticed that for the larger instances, the proposed algorithm as well as the ORTEC algorithm do not come close to feasibility. Therefore, it might not even make sense to believe that reducing the size of the feasible space to be a problem.

So, to conclude, this thesis demonstrates that it is very hard to obtain good results when generic and robust methodology is applied to nurse rostering problems. Further research should investigate whether making the solution approach less robust, but allowing for faster approaches, will influence the results in a negative way when time is limited.

References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., & Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations research*, *46*(3), 316–329.

Burke, E., Cowling, P., De Causmaecker, P., & Berghe, G. V. (2001). A memetic approach to the nurse rostering problem. *Applied intelligence*, *15*(3), 199–214.

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

Burke, E. K., Curtois, T., Post, G., Qu, R., & Veltman, B. (2008). A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European journal of operational research*, *188*(2), 330–341.

Burke, E. K., De Causmaecker, P., Berghe, G. V., & Van Landeghem, H. (2004). The state of the art of nurse rostering. *Journal of scheduling*, *7*(6), 441–499.

Caprara, A., Fischetti, M., & Toth, P. (1999). A heuristic method for the set covering problem. *Operations research*, *47*(5), 730–743.

Cheang, B., Li, H., Lim, A., & Rodrigues, B. (2003). Nurse rostering problems—a bibliographic survey. *European Journal of Operational Research*, *151*(3), 447 - 460. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0377221703000213` doi: https://doi.org/10.1016/S0377-2217(03)00021-3

Curtois, T. (2014). Employee shift scheduling benchmark data sets. *School of Computer Science, The University of Nottingham, Nottingham, UK, Tech. Rep*.

Demeester, P., Bilgin, B., De Causmaecker, P., & Berghe, G. V. (2012). A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice. *Journal of Scheduling*, *15*(1), 83–103.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, *153*(1), 3–27.

Eveborn, P., & Rönnqvist, M. (2004). Scheduler–a system for staff planning.

Annals of Operations Research, *128*(1-4), 21–45.

Geoffrion, A. M. (1974). Lagrangean relaxation for integer programming. In *Approaches to integer programming* (pp. 82–114). Springer.

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

Held, M., & Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical programming*, *1*(1), 6–25.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85–103). Springer.

Kellogg, D. L., & Walczak, S. (2007). Nurse scheduling: from academia to implementation or not? *Interfaces*, *37*(4), 355–369.

Nonobe, K. (2010). Inrc2010: An approach using a general constraint optimization solver. *The First International Nurse Rostering Competition (INRC 2010)*.

Trivedi, V. M., & Warner, D. M. (1976). A branch and bound algorithm for optimum allocation of float nurses. *Management Science*, *22*(9), 972–981.

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

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., & De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, *226*(3), 367–385.

Wren, A. (1995). Scheduling, timetabling and rostering—a special relationship? In *International conference on the practice and theory of automated timetabling* (pp. 46–75).