Crew Rostering at NS using SAT Solvers

Janssen, Guido

Award date:
2021

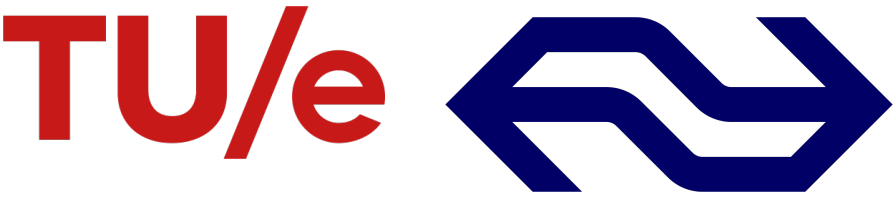
Eindhoven University of Technology

Crew Rostering at NS using SAT Solvers

Author:
Guido Janssen
0898236

Academic Supervisor:
Christopher Hojny

Company Supervisor:
Pieter-Jan Fioole



Contents

Chapter 1

Introduction

Nederlandse Spoorwegen (NS) is the largest railway company of the Netherlands, and is responsible for most of the passenger trains that travel across the country. To make this happen they have more than 20.000 employees. Around 3.000 of them are train drivers and 2.500 are conductors. This thesis will be about constructing the rosters of conductors, but with some minor adjustments the proposed methods could also be used to make rosters for train drivers.

In the Netherlands there are 29 crew bases, each with their own staff of drivers and conductors. These 29 crew bases are the locations at which trains start at the beginning of the day, and end at the end of the day. All duties for employees of a specific crew base start and end at the crew base. Therefore the crew bases can be considered as separate companies with respect to the Rostering Problem.

Rosters need to meet multiple requirements. First of all, they cannot be in conflict with the Collective Labour Agreement. In the CLA rules about minimal rest times and maximum working times are stated, which cannot be violated. However, only adhering to the CLA is not considered to be enough. In the past, multiple strikes occurred because the employees were not satisfied with the way the duties were constructed. They do not like to drive the same trajectory multiple times on the same day for example. Following these strikes, the sharing-sweet-and-sour rules were made [1]. However, even if duties are constructed following these rules, this does not guarantee that all employees receive equal work. Not all duties can be equal, and some will be more desirable than others. Employees do not like it when they have to do more unpopular work than others, and therefore the concept of fairness is important in the Rostering Problem. Duties need to be divided among employees, such that every employee has to perform an equal amount of popular and non-popular work. Finally, duties need to be rostered in an attractive way. One can imagine that, although it is allowed by the CLA, it is not desirable to have for example a week in which there are multiple switches between early and late duties. One would much rather have one week with only late duties and one with only early duties. The rules of the CLA should be interpreted as "it cannot be worse than this". This leads to the concept of attractiveness, in which it is tried to assign the duties to different days in an attractive pattern.

It has been shown that there is a trade-off between fairness and attractiveness [6]. Therefore, it is difficult to determine what solution would be optimal for the Rostering Problem. One must choose a weight for fairness and attractiveness. However, even when these weights are assigned, determining an optimal solution is computationally extremely expensive. Breugem [6] formulated the Rostering Problem as a mixed integer program, and used a branch and price approach that successfully returned optimal rosters for relatively small instances within reasonable time. However, when a full crew base is considered this approach is not applicable anymore as it needs too much time to solve.

Because we also want to make fair and attractive rosters for larger instances we must look at alternative solution approaches. Recently NS had success applying a satisfiability approach to the Timetabling Problem. Therefore the presumption is this could also be the case with the Rostering Problem. There are multiple similarities between both problems. In both problems tasks must be assigned to times. Furthermore, both problems are cyclic. However, there are also clear differences. For example fairness and attractiveness are not a part of the Timetabling Problem.

The goal of this thesis is to investigate whether it might be useful to use a satisfiability formulation to solve the Rostering Problem for these larger instances. Therefore we formulate a couple research questions that we want to answer.

- How complex is the Rostering Problem? Can we expect to develop an efficient algorithm?
- Is it possible to formulate the Rostering Problem as a SAT instance?
- Is it possible to integrate fairness and attractiveness into this SAT formulation?
- Is it useful to combine MIP and SAT techniques to solve the Rostering Problem?
- How does a SAT encoding compare to a mixed integer programming approach?
- Can we improve the MIP approach with the help of a SAT approach?

First we will look at the complexity of the problem and prove that a general version of the Rostering Problem is NP-Complete. In Section 6 the problem will be encoded as a Boolean satisfiability problem. Then it will be investigated whether it is possible to incorporate fairness and attractiveness to the model. In Section 7 multiple ways are presented to encode fairness in SAT clauses and we will investigate whether these methods are suitable for the Rostering Problem. In Section 8 a hybrid solving method is proposed that uses mixed integer programming to achieve fairness and a MAX-SAT approach to achieve attractiveness. We propose different heuristics to improve the performance of the Hybrid Model. Then we compare the approaches with a general MIP. Multiple formulations for the MIP are presented and we will look which one is best. Furthermore we will combine the hybrid approach and the MIP approach to improve the speed of the solver.

Chapter 2

Crew Rostering

2.1 Specifications of the Rostering Problem

To be able to develop solution methods for the Rostering Problem we first need a proper definition of the problem and its characteristics. In this section the details of the Rostering Problem will be explained.

Duties

The Crew Rostering Problem starts after the Crew Scheduling Problem is solved. In the Crew Scheduling Problem all tasks are divided in different duties, such that every train has a driver and an adequate number of conductors on it. These duties vary from six to nine and a half hours, and could be divided in early, late and night duties. Every duty has to start and end at the crew base to which it belongs. NS has a model that given the timetable with all specifications creates these duties, which will be the input for the Crew Rostering Problem.

The Crew Rostering Problem is to assign these duties to crew members, such that it is allowed according to the Collective Labour Agreement. Furthermore, the division has to be fair and attractive, which will be explained later on in this section.

Roster Groups

As stated in the introduction, every crew base can be considered as its own company with respect to rostering crew members. Each crew base has a certain amount of employees, who are divided in roster groups. All members of a roster group cycle through all the rosters that were assigned to their group. So if a roster group has n members, then n weeks will be rostered for this group, and in n weeks time every member of the roster group will perform all n different weeks that were rostered for this group. This way, some degree of fairness is already achieved, as everyone within one roster group performs all duties that were assigned to this group. However, duties still must be divided such that each roster group receives equal work.

Fairness is not the only reason for the use of roster groups. It also creates attractiveness, as group members perform different duties every week, which leads to a small amount of repetition. Also preferences could be incorporated within the groups. For example, it is possible to assign only early duties to one group, and only late duties to another group. This way people that like early duties can go in the first group, while people that like late duties can go in the second. This also leads to less switches between early and late duties, which helps creating an attractive roster.

Basic Schedule

At this point in time, crew rostering is done manually by a roster committee. Every crew base has its own roster committee, mostly consisting of experienced train drivers and conductors that have great knowledge about the entire train network. They start with making a Basic Schedule. In the Basic Schedule each day of the roster is assigned a type of work. This could be V ("vroeg") for an early duty, L ("laat") for a late duty or N ("nacht") for a night duty. Also rest days (R) are rostered in the Basic Schedule, as are reserve days (RES).

On a reserve day an employee has the day off, but could be called up to work. In practice this happens a lot, for example when someone is ill. It could also happen that someone cannot perform his rostered duty anymore due to some unforeseen events. One reason could be that a driver had a delay on the previous day, such that the time in between two duties would be less than twelve hours. This is not allowed by the CLA, and hence someone with a reserve shift will need to take over. An employee can only be called up for a task if performing that task would not violate any rules of the CLA. For the Rostering Problem, a reserve day counts as a day without a duty, but it also does not need to fulfill the requirements of a rest day. One reserve shift counts as eight hours of work. To regulate this, some reserve days are marked as CO/RES, which means compensation or reserve. When an employee has done enough reserve tasks, this becomes a compensation day and the employee cannot be called up this day.

Furthermore, some other types of days are specified, namely WTV days, Streepjesdagen, and RO days. Normal employees have a contract for 36 hour per week. When an employee reaches a certain age, they can choose to only work for 32 hours. Normal weeks consist of 40 hours of work. To compensate for this difference normal employees have 26 WTV days every year, so on average one every two weeks. These WTV days are shifts of eight hours, but without a specified start time and without the employee having to do any work. This is different to a rest day, as a rest day has additional requirements.

Older employees that work 32 hours do not receive WTV days. Instead, they have an RO day every week, which is just an additional rest day. Contrary to WTV days, these RO days are considered rest days and therefore they need to meet the additional requirements of a rest day.

Finally, there are Streepjesdagen which are also considered rest days. Streepjesdagen are used to fill up rosters of parttime employees who do not fall into the category of older employees.

After the Basic Schedule is made, duties are assigned to the roster such that their types match the types specified by the Basic Schedule.

The Collective Labour Agreement

The most important part of the Rostering Problem is to adhere to the rules of the Collective Labour Agreement. The rules that are applicable to the Rostering Problem are:

1. There must be a rest period of at least twelve hours between two duties on consecutive days.
2. The rest period after a night duty that ends after 2.00 must be at least fourteen hours.
3. One rest day is at least 30 hours long. Every additional rest day requires an additional 24 hours.
4. There must be two rest days per week on average.
5. There can be at most seven consecutive days without a rest day, RO day or WTV day.
6. Every period of 7x24 hours must have a period of at least 36 hours without duties, or every period of 14x24 hours must have at least 72 hours without duties, which can be divided in two periods of at least 32 hours.
7. There can be at most 36 night duties per sixteen weeks.
8. At least once per three weeks an employee has a Red Weekend, which is a period of at least 60 hours free time that must include the period from Saturday 0.00 until Monday 4.00.
9. On average, an employee cannot work more than 40 hours per week.

The Basic Schedule should enforce some of these rules, for example rule 4,5 and 7. But even when using a Basic Schedule it is not trivial to find a roster that satisfies all these rules.

Fairness

The concept of fairness is very important in the Rostering Problem. It requires the rosters to be such that all employees perform equal work on different factors. Fairness of the workload could even be considered as a feasibility constraint, as it would be really undesirable if some employees have an average workload per week that is higher than others. Other parameters are slightly more flexible, but still we want to make a roster that is fair in all specified factors as this is agreed upon with the labor union.

Besides the workload four other factors are defined that quantify the popularity of a duty. These factors are:

1. Repetition Within Duty
2. Percentage of work in A-trains
3. Percentage of work in Aggression trains
4. Percentage of work in Double deckers

Repetition Within Duty is a value that quantifies the amount of repetition within one single duty.

Duties with a lot of repetition are considered undesirable, as it is unpleasant to drive on the same trajectory multiple times in a row.

Trains that have longer trips and only stop at large stations are considered pleasant. These trains, mostly intercities, are called A-trains. As this work is considered nicer than other work, a fair distribution of the work on A-trains is desired.

On some trajectories employees experience more aggression of travelers than on other trajectories. NS has a list of those trajectories that on average have the most aggression. Employees obviously do not like to face aggression, and therefore working on aggression trains is considered undesirable. To limit the work on aggression trains for every employee, a fair distribution is required.

Working in double deckers is considered to be physically harder, and therefore less pleasant. For that reason, also the percentage of work on double deckers needs to be distributed fairly among the employees.

Attractiveness

The concept of attractiveness tries to quantify how nice duties are rostered. This is different from fairness, because fairness was only about the division of duties to roster groups, while attractiveness is about the quality of the roster for one specific individual. The Collective Labour Agreement states rules that must be obeyed. However, if every rule is followed with no margin a very unattractive roster might be the result. In some cases the CLA should be interpreted as "worst case". Train drivers and conductors have very irregular working times. Attractiveness also tries to reduce the irregularity in the rosters.

Attractiveness depends a lot on the preferences of employees of one specific crew base. Furthermore it could also differ between individuals. However, in this thesis we assume that attractiveness is the same for every employee. Examples of attractiveness preferences could be:

1. The time between duties should be more than twelve hours.
2. One rest day should be as long as possible, so more than 30 hours.
3. If there are multiple early duties in a row, then start times should be increasing.
4. If there are multiple late duties in a row, then end times should be decreasing.

The reason the first and second point increase attractiveness is because if there is a short rest period in one spot, then another spot will have a very long rest period. By trying to increase the duration of the short rest periods, more regularity is achieved. The third increases attractiveness because it creates some rhythm. Employees don't have to start at 5.00 one day, 10.00 the next day and then at 5.00 again on the third day. Instead they have to get up later every day. A same reasoning holds for the late duties.

These are only four examples, but many more could be thought of. An attractive roster is a roster that limits the violation of these preferences.

2.2 Literature Overview

In the literature multiple examples are given on how to achieve fairness and attractiveness in combinatorial optimization problems, especially in assignment problems. In this section we will start with a mathematical description of fairness in combinatorial optimization problems. Then we give examples of research that has been done regarding the Rostering Problem in various settings, including hospitals, airline companies and railway companies.

Definition 1 *A* Combinatorial Optimization problem *is a problem in which a solution must be chosen from a finite set of feasible solutions.*

Definition 2 *An* Assignment Problem *is a problem in which a minimum cost or maximum profit matching must be found in a weighted bipartite graph.*

A way of obtaining a fair solution is via a fairness scheme. To formally describe a fairness scheme we need the to introduce some concepts. Each assignment problem has a number of players n. Each assignment of resources to the n players carries some value representing the utility for each player. The set of all achievable utility allocations to the players is called the utility set $U \in \mathbb{R}^n$. The vector $f(U)$ is a fair allocation under utility set U.

Definition 3 *A* Fairness Scheme *is a function $f\colon U \to \mathbb{R}^n$ that maps a utility set to an element in the utility set.*

Bertsimas et al. [4] described axioms that a fairness scheme of a combinatorial optimization problem should ideally satisfy. They stated the following axioms for two player problems:

1. Pareto optimality
2. Symmetry
3. Affine invariance
4. Independece of irrelevant alternatives
5. Monotonicity

An allocation is Pareto optimal if there is no $u \in U$ such that $u_i \geq (f(U))_i$ for all i and $u \neq f(U)$. In other words, no one can improve their utility without someone else losing utility.

Symmetry means that a fair allocation to a permutation of the players is the same as the permutation of the allocation. So first permuting the players and then making a fair allocation should give the same end result as first making a fair allocation and then permuting the players. More formal: let $T\colon \mathbb{R}^2 \to \mathbb{R}^2$ be a function such that $T(x_1, x_2) = (x_2, x_1)$ and $f\colon U \to \mathbb{R}^2$ be a fairness scheme that allocates the resources among the players, (and thus determines the allocation of utility to the players), then $f(T(x_1, x_2)) = T(f(x_1, x_2))$.

An affine invariant allocation is one for which an affine transformation of the allocation is the same as the allocation to the affine transformation of the utility set. In mathematical terms, this means:

let $A = (A_1, A_2)$ be a two-dimensional affine function, which means $A_i(x) = c_i x + d$ for some constants $c_i > 0$ and $d \in \mathbb{R}$. Then $A(f(U)) = f(A(U))$. Here f is the fairness scheme and U is the utility set that consists of all feasible pairs.

When an allocation is independent of irrelevant alternatives then the following must hold: if $U \subset W$ and $f(W) \in U$ then $f(W) = f(U)$. This means that if an allocation is chosen by f in utility set W than it must also be chosen in a subset of W as long as this subset contains this allocation.

Finally, monotonocity is defined as follows. Let U and W be two utility sets and the maximum utility player one can achieve is equal under U and W, but the utility player two can derive, given the demand of player one, is larger or equal under W than under U. Then in a fair allocation player two should be given a higher or equal utility under W than under U.

Nash [15] proposed a unique scheme that satisfies the first four axioms. As he proved this solution is unique, and as it does not satisfy the fifth axiom, a scheme that satisfies all five axioms does not exist. Smorodinsky and Kalai [12] proposed a scheme that satisfies the first three and the fifth. They also gave a proof that this solution is the only one satisfying monotonicity.

Bertsimas [4] explains two fairness schemes that are justified by these axioms, namely proportional fairness and max-min fairness. Proportional fairness is a generalisation of the Nash solution to n players, while max-min fairness is a generalisation of the scheme proposed by Smorodinsky and Kalai. Then they introduced the concept price of fairness, which quantifies the loss of total utility because of requiring a fair solution. He gave bounds for these costs for the proportional and max-min fairness schemes.

Caprara et al. [7] modeled the Crew Rostering Problem for the Italian Railways as a directed graph. Every node in the graph represents a duty. If duty i can be followed by duty j then there is a directed arc from node i to node j. The goal is to find a set of paths with minimal costs that covers all nodes of the graph. Each arc has an associated cost. The arc costs can be used to express attractiveness. This is limited, as only attractiveness between two consecutive duties can be modeled. Further attractiveness constraints were modeled as hard constraints because the were seen as rostering rules, and therefore could not be violated. The problem is solved by a heuristic that creates the rosters one by one. When a roster is finished, the duties that were selected are removed from the graph. This procedure continues until all duties are part of a roster. A disadvantage of this approach is that the rosters that were created later tend to be worse.

Kohl and Karish [13] describe the Rostering Problem from an airline point of view. Airlines often choose for non-cyclic personal rosters instead of cyclic rosters. They divide the constraints in three types, namely horizontal, vertical and artificial. Horizontal constraints are rules about one specific roster, for example resting times between duties. Vertical constraints are about multiple rosters, for example certain types of duties can only be performed by a subset of the employees. Lastly, artificial rules are not needed for a feasible solution, but they do improve the quality of the solution. Attractiveness is named as one of the options to add via artificial rules. Another option they propose as artificial rule is robustness. This can be seen as attractiveness from the side of the company. The final option they give is adding rules that help the solver by eliminating patterns that never would be part of a good solution.

They propose multiple objectives that could be used to solve the Rostering Problem, and state that most often a combination of these objectives is used in practice. The first is an objective that relates to the real operational costs. Secondly, robustness can be added to the objective by penalising situations were there is only a small margin and a delay could cause problems. The third option is to use the objective to create fairness. Mainly in Europe, fairness is considered important. American airlines often choose to work via bidding. Senior employees will have first choice. Therefore, they like it when all undesired work is condensed in a few rosters that can be given to the junior crew members. Finally, personal preferences can be included in the objective function. Airlines mostly choose to first satisfy the preferences of senior employees.

They distinguish between two solution methods. The first is a constructive method that constructs the roster from scratch. The duties and rosters are viewed as a bipartite graph and the goal is to make a matching that assigns at most one duty to a crew member. Each iteration it adds to the rosters until all duties are assigned, or until no employee can get an additional duty. In the last case, the cost penalty will be high as either someone will have to work overtime, an additional crew member has to be found or a flight must be cancelled. After a roster is made using the constructive approach the second solution method is used, which is an improvement based method. The method they propose does lock a subset of the duties for each roster. Then it sequentially finishes the rosters via a depth-first search. One option they explain is to lock everything, except for all duties that are in a certain time frame, and then move this time frame every iteration.

Martin et al.[14] looked at fairness in nurse rostering. Every nurse has some types of work he or she likes most, or is most competent at. They model fairness as a set of soft constraints, and then propose multiple options to add the violation of these soft constraints to the objective. The most natural way seems to be to minimize the (weighted) sum of violations, but this might lead to a solution in which one nurse has a big violation, and the others a very small, while a solution in which everyone has an equal amount of violation would be more fair even though the total violation might be higher. They give four alternatives to achieve more fair solutions: minimize the maximum violation, minimize the total deviation from the average violation, minimize the difference between the minimum and maximum violation and finally minimize the total squared violation. All of these alternative objectives are non-linear and therefore are difficult to solve compared to a linear objective. They developed heuristics to solve the problem and used the Jains fairness index metric to determine the fairness of a solution. The Jains Fairness Index is calculated as follows:

$$\frac{(\sum_{n \in N} q_{ros}(n))^2}{|N| \sum_{n \in N} q_{ros}(n)^2} \tag{2.1}$$

Here $q_{ros}(n)$ is the total violation of fairness constraints by nurse n. If the roster is completely fair the index is 1. In the worst case the index takes on the value $\frac{1}{|N|}$.

Their solution method was an agent-based cooperative metaheuristic search. An initial solution is shared with multiple search agents that then will try different metaheuristics to improve the solution in parallel. These metaheuristics were Tabu-search, Simulated Annealing and Variable Neighbourhood Search, which are all established search heuristics. Furthermore, they developed a method that allowed the different agents to communicate nice patterns to each other to help the other agents. The algorithm finished after a fixed number of iterations. Then the fairest roster

among all agents is the final solution. They concluded that the minimum deviations objective resulted in the most fair solutions in the case of the Nurse Rostering Problem, closely followed by the min-max objective. The other options performed clearly worse, with the weighted sum of violations giving the least fair allocations.

Abbink et al.[2] developed the crew scheduling model for NS. In this model the duties are created and divided among the crew bases. These duties become the input for the crew rostering model. In the Crew Scheduling Problem fairness also plays an important role, as a fair distribution of duties among crew bases is desired. They enforce this fairness by setting an upper bound for the amount of undesirable work, and a lower bound on the amount of desirable work. Furthermore they impose an upper bound on the standard deviation of the division of attributes. This does not give a perfectly fair division, but it does guarantee a certain degree of fairness. In practice a perfectly fair division would imply an extreme increase in costs, as for example some parts of the country have a lot of aggression (e.g. the Randstad), while other parts have a very low amount of aggression. In a perfectly fair allocation, employees would have to travel across the country to perform an aggression trip, which leads to lost time. This lost time creates the need for more duties, which in the end leads to a higher number of employees that is needed to perform all duties.

Breugem [6] solved the Rostering Problem for NS on relatively small instances using mixed integer programming. Just like Abbink et al. [2] he modeled fairness by using bounds. For all attributes he introduced a variable that represents the minimum average value among the roster groups. For the maximum average value he introduced another variable. Both the minimum and maximum average value are bounded by a chosen parameter. Then he added a constraint that forced the the weighted sum of the difference between the minimum and maximum of all attributes to be smaller than some chosen parameter. This way, some budget for violating perfect fairness is created.

Furthermore, he showed that there is a trade-off between fairness and attractiveness, meaning that a high degree of fairness would lead to less attractive rosters. Intuitively the reason is that when a high degree of fairness is demanded, the solution space is shrinked which causes less room to manoeuvre to get attractive solutions. This manoeuvring room is created by varying the fairness budget parameter. Breugem modeled the attractiveness by using soft constraints. He added the violations of these soft constraints as penalties in the objective function.

Chapter 3

Notation

Throughout this thesis we will use the following notation:

D	Set of duties
S	Set of sequences
R	Set of roster groups
B	Basic Schedule
A	Set of Attributes
ϵ	Fairness margin of workload
δ_a	Fairness margin of attribute a
Δ	Set of δ_a for all attributes a
x_{is}	Boolean variable that is True if duty $i \in D$ is assigned to sequence $s \in S$ and False else
y_{ir}	Boolean variable that is True if duty $i \in D$ is assigned to roster group $r \in R$ and False else
z_{sd}	Boolean variable that is True if there is a rest day on weekday d in sequence s
v^c_{sd}	Variable that is used to model the violation of a soft constraint c at day d in sequence s
v^c_{ij}	Boolean variable that is True if soft constraint c is violated by duties i and j

Chapter 4

Complexity

In this section we will answer the first of our research questions. We will look into the complexity of the Rostering Problem. If the problem is NP-Complete we cannot expect to find an efficient algorithm unless $P = NP$, but if it is not then we might be able to find an algorithm that runs in polynomial time.

For the proof we will focus on a more general version of Rostering Problem.

We define the Rostering Problem as follows:

Definition 4 *The* General Rostering Problem with Basic Schedule *is as follows:*

Let D be a list of duties with for each duty $d \in D$ a start time, end time, type, weekday and a set A of attributes. Let R be the set of roster groups and B a Basic Schedule in which for each roster group $r \in R$ at each day the type of duty is specified. Let Δ be the set with for each attribute $a \in A$ a fairness margin δ_a. Let ϵ be the fairness margin for the average workload. Let U be a set of unattractive patterns. An unattractive pattern is a pattern of specific duties to specific spots in the Basic Schedule that cannot appear in the solution. Given an instance $(D, R, B, A, \Delta, \epsilon, U)$ is it possible to find a roster that satisfies the following constraints:

- *All duties are assigned to exactly one spot on the Basic Schedule.*
- *The type of a duty that is assigned to day d of roster group r matches the type that was specified in the Basic Schedule at day d of roster group r.*
- *If two duties are rostered on consecutive days there must be at least 12 hours rest between the end of the first duty and the start of the second duty.*
- *If n consecutive rest days are rostered then the start of the first duty after the rest period is at least $24n + 6$ hours after the end of the last duty before the rest period.*

- *The solution is fair.*
- *The solution is attractive.*

A fair solution is a solution in which for every roster group the average value of each of the attributes $a \in A$ is within a δ_a percent margin of the overall average value of that attribute and the average workload per week for every group is within ϵ percent from the total average workload.

A solution is attractive if none of the unattractive patterns in U appear in the solution.

Normally, one would model attractiveness using soft constraints. This means that attractiveness constraints might be violated at the cost of a predefined penalty, but we want to minimize the sum of the penalties. However, the first observation we do is that any problem with soft constraints is at least as hard as the same problem when all constraints would be hard constraints. As the problem with soft-constraints is a minimization problem we look at it as a decision problem that decides whether there is a solution with penalty at most K for some K. To justify this claim we follow the proof of Den Hartog [9].

Lemma 1 *Let Π be a problem with constraint set $C = C_1 \cup C_2$ and $C_1 \neq \emptyset$ such that the constraints in C_1 are hard constraints and the constraints in C_2 are soft constraints. Let the problem Π' have constraint set $C' = C'_1 \cup C'_2$ such that C'_1 are hard constraints, C'_2 are soft constraints, $C'_1 = C \backslash c$ for some constraint c and $C'_2 = C_2 \cup c$. Then if Π is NP-Complete, also Π' is NP-Complete.*

Proof. Let Π be a problem with constraint set $C = C_1 \cup C_2$ and $C_1 \neq \emptyset$. Now construct the problem Π' by picking one of the constraints $c \in C_1$ and transform this constraint into a soft constraint with penalty $K + 1$ such that the new constraint set is $C' = (C_1 \backslash c) \cup (C_2 \cup c)$ and solve the problem.

There are multiple possibilities. If Π had a solution with penalty $P < K$ then Π' will have a solution with penalty exactly P, as the solution of Π does not violate c.

If Π was a no-instance then two things can happen. It could be that the solution with lowest violation for Π' does violate c, or it does not. If it does violate c the penalty will be larger than K as c had penalty $K + 1$. If it does not violate c then the solution with lowest penalty is equal to the solution with lowest penalty of Π. This means that the penalty is the same as for Π, which is larger than K as Π was a no-instance.

Therefore, Π is a yes-instance if and only if Π' is a yes-instance. Transforming the problem can be done in polynomial time, so if Π was NP-Complete, then Π' will be NP-Complete too.

□

As a result, we can assume that all constraints are hard constraints to prove that the Rostering Problem is NP-Complete, even though in practice attractiveness is modeled using soft constraints.

In the literature the complexity of the Rostering Problem has been studied in the context various versions of the Nurse Rostering Problem.

Definition 5 *The* Basic Nurse Rostering Problem *is as follows:*

Given a set N of nurses, a set D of days and a set S of shifts, roster shifts to the nurses such that each nurse is given one shift every day.

Here rest days are assumed to be a specific type of shift. Then different types of constraints can be added to the problem. The Basic Nurse Rostering Problem is easy to solve, but when additional constraints are added the problem becomes NP-Complete. To prove that the Rostering Problem is NP-Complete, we will reduce from the Nurse Rostering Problem with strict shift constraints, forbidden sequences and day-off requests.

The strict shift constraint means that each shift s must be assigned to exactly k_{ds} nurses on day d, where k_{ds} is the number of times shift $s \in s$ is rostered to day $d \in D$. This means $\sum_{s \in S} k_{ds} = |N|$ for each $d \in D$. So for any given day it is known exactly how many nurses should be assigned to a specific shift type.

The forbidden sequences constraint means that specific types of shifts cannot follow up each other. Here the order does matter. A pratical example of such a constraint could be that a night duty cannot be followed by an early duty. In theory however, every sequence could be forbidden.

The day-off requests constraint means that all nurses can request which days they have a rest day. In the Nurse Rostering Problem, rest days are seen as a shift type.

Now we can define the Nurse Rostering Problem with strict shift constraints, forbidden sequences and day-off requests.

Definition 6 *The* Nurse Rostering Problem with strict shift constraints, forbidden sequences and day-off requests *is as follows:*

Let N be a set of nurses, $\tilde{D}$ be a set of days and $\tilde{S}$ a set of shifts. Let K be a set with for all $d \in \tilde{D}$ and $s \in \tilde{S}$ the number k_{ds} such that $\sum_{s \in \tilde{S}} k_{ds} = |N|$ for each $d \in \tilde{D}$. Let F be a list of pairs of types that cannot appear on consecutive days and let R_n be the set of requested rest days for nurse n. Let $\tilde{R}$ be $\bigcup_{n \in N} R_n$. Given an instance $(N, \tilde{D}, \tilde{S}, K, \tilde{R})$ is it possible to make a roster such that:

- *Each nurse is given one shift every day.*
- *Each shift type s must be assigned to exactly k_{ds} nurses on day d.*
- *For each pair $(s_1, s_2) \in F$ and each nurse n there are no two consecutive days d_1, d_2 such that shift s_1 is assigned to day d_1 and shift s_2 is assigned to day d_2.*
- *Nurse n has a rest day on each day $d \in R_n$.*

Den Hartog [9] showed that this problem is NP-Complete. In his version of the problem not all rest days were requested necessarily. However, we will assume that all rest days are requested. This is still NP-Complete. To see this we split up the shift with type rest day into two types, namely requested rest days and unrequested rest days which have the same constraints, with the exception that requested rest days must be at the assigned day.

As far as we know there has not yet been a proof of NP-completeness of a version of the Rostering

Problem that includes fairness and attractiveness.

We will prove that the General Rostering Problem is NP-Complete with a reduction from the Nurse Rostering Problem with strict shift constraints, forbidden sequences and day-off requests.

Lemma 2 *The General Rostering Problem with Basic Schedule is NP-Complete.*

Proof. Consider an instance $(N, \tilde{D}, S, K, \tilde{R})$ of the Nurse Rostering Problem with strict shift constraints, forbidden sequences and day-off requests. We will create an instance $(D, R, B, A, \Delta, \epsilon, U)$ of the General Rostering Problem the following way. For each nurse in N create a roster group of $7 \cdot \lceil \frac{|\tilde{D}|+1}{7} \rceil$ days, where $\tilde{D}$ is the set of days in the Nurse Rostering Problem and N the set of nurses. The total number of shifts in the Nurse Rostering Problem is $|N||\tilde{D}|$. Due to the strict shift constraints we know exactly how many working shifts and how many rest days there are.

The list of duties D is created as follows: for every non-rest shift add k_{ds} duties to the Rostering Problem that start at 14.00 and end at 14.00 and have weekday $d \mod 7$. The type of each duty is equal to the week of the corresponding shift in the Nurse Rostering Problem, so duties representing the first seven days get type 1, the next type 2 and so on. Take $A = \emptyset$ and $\Delta = \emptyset$. Take an arbitrary nonnegative value for ϵ.

We will first take care of the first $|\tilde{D}|$ days of the Basic Schedule for each roster group. Add a rest day in roster group n on each requested rest day of nurse n. On all other days add type i if the spot is in the i^{th} week. Now if the number of days in D was not a multiple of 7 finish off the Basic Schedule by adding fixed duties that start and end at 14.00, and also add these duties to the list of duties. If the number of days was divisible by 7, add an entire week of these fixed duties. This is needed to get rid of the cyclic character of the Rostering Problem to prevent problems with constraints between the first and last day.

For each forbidden sequence in F add unattractive patterns to U for the Rostering Problem that forbid all combinations of duties that correspond to a pair of shifts in the Nurse Rostering Problem that form a forbidden sequence. Normally one would use soft constraints to do this, but since we already proved that if the problem using hard constraints is NP-Complete then so will be the problem using soft constraints it suffices to use hard constraints.

Now if the Rostering Problem returns "yes" then we can create a solution for the Nurse Rostering Problem by picking the first $|\tilde{D}|$ slots of the roster and assign the shift corresponding to the duty to the nurse. As the duty was given the same weekday as in the Nurse Rostering Problem, and the type of the duty made sure they were assigned to the right week this will indeed assign k_{ds} shifts of each type in S to a nurse at all $|\tilde{D}|$ different days. As rest days were assigned to those days in the Basic Schedule that were requested the day off requests are respected. There also will not be forbidden sequences as these were forbidden by the attractiveness constraints. Therefore, the Nurse Rostering Problem is a yes-instance too.

If the Nurse Rostering Problem is a yes-instance than this solution could be transformed to a solution of the Rostering Problem by assigning roster group n the duties corresponding to the shifts of nurse n and adding the fixed duties. By construction, these duties fit the Basic Schedule. As all duties start and end at 14.00 no rest time constraints will be violated. Since the workload is zero

for all duties and there were no other attributes the solution is fair. Since there were no forbidden sequences, the solution is attractive too.

So the Nurse Rostering Problem is a yes-instance if and only if the Rostering Problem is a yes-instance. Furthermore, the transformation could be done in polynomial time, as creating the duties and Basic Schedules can be done in $O(|N||\tilde{D}|)$ time. Since we know the Nurse Rostering Problem is NP-Complete, this means that the general version of the Rostering Problem with a Basic Schedule is also NP-Complete.

□

Given the General Rostering Problem with Basic Schedule is NP-Complete, it is very unlikely the full Rostering Problem is not. It would be very unexpected if for example adding the Red Weekend requirement would make the problem polynomial time solvable. Therefore, we do not expect to be able to find an optimal solution to the Rostering Problem efficiently.

Chapter 5

Boolean Satisfiablity

The goal of this thesis is to investigate whether a satisfiability approach is suitable for the Rostering Problem. To be able to do this we will introduce the most important definitions and concepts within Boolean satisfiability.

5.1 Fundamentals of Boolean Satisfiability

Boolean satisfiability is a way to describe a mathematical model. It makes use of Boolean variables and literals.

Definition 7 *A* Boolean variable *is a variable that can take on either the value True or the value False.*

Definition 8 *The* negation *of a Boolean variable is True if and only if the variable itself is False, and is False if and only if the variable itself is True.*

Definition 9 *A* literal *is a boolean variable or its negation.*

To link literals two operators are used, namely the "and"-operator ($\wedge$), which is also called conjunction, and the "or"-operator ($\vee$), which is also called disjunction. Together with parentheses these can be used to make propositional formulas.

Definition 10 *A string of literals is a* propositional formula *if and only if it fulfills one of these criteria:*

- $F = l$ *for some literal* l
- $F = (G \odot H)$ *for* G *and* H *propositional formulas and* $\odot \in \{\vee, \wedge\}$

A formula $F = (G \wedge H)$ is True if and only if both G and H are True, and False if this is not the case. $F = (G \vee H)$ is True if and only if at least one of G and H is True, else it is False.

Definition 11 *A propositional formula is called a* clause *if it is a disjunction of literals.*

Definition 12 *A propositional formula is said to be in* Conjunctive Normal Form (CNF) *if it is a conjunction of clauses.*

Most practical uses of satisfiability require the formula to be in Conjunctive Normal Form. To satisfy a CNF-formula one must assign the variables such that every clause is satisfied. This means that of every clause at least one literal must be set to True.

5.2 SAT Solving Techniques

Most state of the art SAT solvers use the conflict driven clause learning approach to find a solution [5] [11]. The solver starts with unit propagation and resolving pure literals, which are the fastest techniques of a SAT solver. During the unit propagation step all unit clauses are resolved. A unit clause is a clause consisting of only one literal. This literal must be True, because otherwise the clause is not satisfied. Therefore, the solver sets this literal to True, which means the corresponding variable is set to True if the literal was not a negation, and to False if the literal was a negation.

Pure literals are literals that only appear positive or only appear as a negation. These literals can be set to True, as this can never result in a conflict.

The basis of SAT solving is the Davis-Putnam-Logemann-Loveland algorithm [8], which combines those two techniques with a branch and bound tree in which there will be branched on the value of a non-assigned variable.

If a variable is set to True we remove all clauses that contain this variable, as they are satisfied regardless of the assignments of the other literals in the clause. All clauses that contain the negation of the variable are made smaller by deleting the variable. If the variable was set to False, the opposite happens. In a conflict driven clause learning solver the solver continues with unit propagation and pure literal resolving until one of the following things happens:

- A solution has been found.
- No solution has been found and no conflict has been found.
- A conflict has been found.

When a solution has been found the solver outputs this solution and it is finished.

When no solution has been found and no conflict has been found the solver must make a guess about one of the variables that has not yet been assigned to True or False. After the solver makes a guess, it can again start with unit propagation and pure literal resolving.

A conflict is found if unit propagation results in a variable having to be True and False at the same time. This means that the combination of guesses made so far cannot lead to a feasible solution. The solver can learn from this conflict. It can backtrack which assignments of variables have led to this conflict. Then it can add a clause to the problem that prevents the solver from assigning these

variables to the same values again in following runs.

To identify the conflict the implication graph is used, which is determined by the guesses and unit propagation. In the implication graph we must find a cut that led to the conflict. Then the clause that must be added is the negation of the implications in this cut. There are often multiple options for cuts that lead to the conflict as we can cut at multiple different levels in the implication graph. The most common way to generate the conflict clause is via the first unique implication point [5]. A unique implication point is a node in the implication graph such that each path from the conflict to the guess must pass through this node. The first unique implication point is the first UIP that appears on a path from the conflict to the guessed variable. All nodes after the first UIP will be cut of.

After we added the conflict clause we must backjump to the level on which the first of the variables in the conflict clause was assigned. It is generally accepted that adding conflict clauses drastically improves the performance of the solver.

If a conflict has been found and no guesses could be reversed anymore, there is no feasible solution and the solver returns UNSAT.

Chapter 6

SAT Model for Feasibility

One of our goals was to formulate the Rostering Problem as a SAT instance. The constraints of the Rostering Problem can be divided in three groups: feasibility constraints, fairness constraints and attractiveness constraints. At first a model is proposed to get a feasible roster, without looking at fairness and attractiveness. Later, constraints will be added to try to incorporate these.

The Rostering Problem can be described as a SAT formula in CNF. Every constraint of the problem will be encoded as a set of clauses. The conjunction of all these clauses will represent the problem. A satisfying assignment of the formula then leads to a feasible roster.

As input for the model there is a list of all duties and a Basic Schedule. The list of duties provides all duties with their respective start time, end time, duration and weekday. Furthermore it contains for each duty the repetition value, percentage of work on A-trains (popular work), double deckers (non-popular work) and aggression trains (non-popular work). These last values are not important when only looking at feasibility, but they will be important later, as in the end we want that the average of all these quantities for each employee is as equal as possible.

The Basic Schedule specifies the number of roster groups, the number of members each roster group has, and the type of work at each day of the roster.

Every week in the roster is called a sequence. The number of sequences is equal to the number of employees, which means that the model rosters one week of work for every employee. If an employee performed sequence s^* one week, then the next week he will perform sequence $s^* + 1$, unless s^* was the last week that belonged to his roster group. In that case he would continue with the first week of the roster group, which is sequence $s^* + 1 - |S_r|$, where $|S_r|$ is the number of members that roster group r has.

6.1 Constraints when using a Basic Schedule

The constraints that need to be formulated when modeling the Rostering Problem as a SAT instance are:

- Every duty must be in exactly one sequence.
- Duties on the same day cannot be in the same sequence.
- In between two duties there must be a rest period of at least twelve hours.
- If a night duty ends after 2.00, the rest periods must be at least fourteen hours.

When a Basic Schedule is used as input for the model there are constraints that make sure the roster follows the rules set by the Basic Schedule.

- Fixed duties have to be planned as fixed in the Basic Schedule.
- The type of duty in the roster must match the Basic Schedule.
- n consecutive rest days need to be at least $24n + 6$ hours long.
- When a Red Weekend is planned, it must be at least 60 hours and include the period from Saturday 0.00 until Monday 4.00.

Since a Basic Schedule is used, the other constraints formulated in the previous section are satisfied automatically. In theory it might happen that for some Basic Schedule the constraint that every period of 7x24 hours must include a rest period of at least 36 hours, or every period of 14x24 hours must have at least 72 hours divided in two blocks of at least 32 hours will be violated by the solution of the model. However, this will be extremely unlikely. In the rare case that the model would output such a roster we can add a clause that explicitly forbids this roster and run the model again. This way we do not have to deal with this constraint, that will be very difficult to model as a set of clauses.

In the model the variables x_{is} are used. x_{is} is True if duty i is assigned to sequence s, and False if this is not the case.

Every Duty must be in exactly one sequence

To make sure that every duty is placed in exactly one sequence two constraints are used, namely:

- Every duty is placed in at least one sequence.
- Every duty is placed in at most one sequence.

If both constraints are satisfied each duty must be placed in exactly one sequence.

Every duty is placed in at least one sequence.

Let D be the set of duties and S be the set of sequences. Then the constraint is formulated as follows:

$$\bigwedge_{i \in D} \bigvee_{s \in S} x_{is} \tag{6.1}$$

To satisfy $\bigvee_{s \in S} x_{is}$ when i is fixed to i^*, x_{i^*s} must be True for at least one $s \in S$. This means that duty i^* is assigned to at least one sequence. The clause is added for all duties i in D.

Every duty is placed in at most one sequence.

To make sure that a duty can be assigned to at most one sequence the following constraint is formulated:

$$\bigwedge_{i \in D} \bigwedge_{s,k \in S, s \neq k} (\neg x_{is} \vee \neg x_{ik}) \tag{6.2}$$

To satisfy $(\neg x_{is} \vee \neg x_{ik})$, x_{is} or x_{ik} must be False, or both. This means that duty i cannot be in sequence s and k at the same time, since then both would be True. The conjunction of this clause for every duty and every combination of two sequences forms the constraint.

Two duties on the same day cannot be in the same sequence.

An employee can only perform one duty every day. Therefore, two duties that are on the same day cannot be in the same sequence. To achieve this the set D_d is introduced for every d in $[0, 6]$. D_d is the set of all duties that take place on day d. So D_0 is the set of all duties that take place on Monday, while D_5 is the set of all duties that take place on Saturday. Then the constraint looks as follows:

$$\bigwedge_{d \in [0,6]} \bigwedge_{i,j \in D_d, i \neq j} \bigwedge_{s \in S} (\neg x_{is} \vee \neg x_{js}) \tag{6.3}$$

This formula is False if there exist two duties i^* and j^* that are both on day $d^* \in [0, 6]$ and that are both assigned to sequence s^* for some $s^* \in S$. If the formula is True this is not the case, and there is no sequence that contains two duties that are on the same day.

When a Basic Schedule is used, this constraint becomes redundant. A Basic Schedule is designed such that every type of duty is scheduled exactly the amount of times it appears in the list of duties. Therefore, if you require that the Basic Schedule is followed there is no possibility to plan two duties on the same day anymore. However, even when a Basic Schedule is used it still is useful to add this constraint as it helps the SAT solver to propagate faster, and therefore find a solution more quickly.

At least twelve hours rest after a duty

The Collective Labour Agreement states that an employee needs to get a break of at least twelve hours after he finished his duty. Therefore, if duties are on consecutive days, they can only be in the same sequence if the starting time of the second duty is at least twelve hours later than the end time of the first duty. Furthermore, it must be noticed that when the first duty is on a Sunday in sequence n the next duty the employee will have to perform is on Monday in sequence $n+1$, or in sequence $n+1-|S_r|$, where $|S_r|$ is the size of the roster group in the case that sequence n was the last sequence of the roster group.

First $start(i)$ is defined as the time at which duty i starts, and $end(i)$ as the time at which duty i ends. Now the time between two duties i and j is the difference between $end(i)$ and $start(j)$. We define $start(j) - end(i)$ as the amount of time in hours it takes from the end of duty i to the start of duty j, including days that are in between. If for example duty i finishes on Saturday at 19.00 and duty j starts on Tuesday at 10.15 then $start(j) - end(i) = 63.25$.

To formulate this constraint as a set of SAT clauses the set $D_{(d,d')}$ is introduced. $D_{(d,d')}$ is the set of tuples of duties such that the first duty is on day d, and the second on day d'.

Furthermore, the set S' is initialized as the set of sequences that are the last of their respective roster group.

Now the constraint is split up in multiple cases. In the first case the duties on Monday to Saturday are handled. In the second and third case the duties on Sunday are handled. The second case deals with the sequences that are not the last of their respective roster group. The third case looks at those sequences that are the last of their roster group. If sequence s is the last of its group then instead of looking at sequence $s+1$, one must look at sequence $s+1-|S_r|$, where $|S_r|$ is the size of the roster group of which sequence s is a part.

$$\bigwedge_{d\in(0,5)} \bigwedge_{(i,j)\in D_{(d,d+1)}:start(j)-end(i)<12} \bigwedge_{s\in S} (\neg x_{is} \vee \neg x_{js}) \tag{6.4}$$

$$\bigwedge_{(i,j)\in D_{(6,0)}:start(j)-end(i)<12} \bigwedge_{s\in S\setminus S'} (\neg x_{is} \vee \neg x_{j(s+1)}) \tag{6.5}$$

$$\bigwedge_{(i,j)\in D_{(6,0)}:start(j)-end(i)<12} \bigwedge_{s\in S'} (\neg x_{is} \vee \neg x_{j(s+1-|S_r|)}) \tag{6.6}$$

At least fourteen hours rest after a night duty that ends after 02.00.

This constraint is created equivalently to the last one, with the only difference being the set of duties for which the constraints need to hold. The set $D^N_{(d,d')}$ consists of all tuples of duties (i,j) such that duty i is a night duty on day d and j is a duty on day d'.

$$\bigwedge_{d\in(0,5)} \bigwedge_{(i,j)\in D^N_{(d,d+1)}:start(j)-end(i)<14} \bigwedge_{s\in S} (\neg x_{is} \vee \neg x_{js}) \tag{6.7}$$

$$\bigwedge_{(i,j)\in D^N_{(6,0)}:start(j)-end(i)<14} \bigwedge_{s\in S\setminus S'} (\neg x_{is} \vee \neg x_{j(s+1)}) \tag{6.8}$$

$$\bigwedge_{(i,j)\in D^N_{(6,0)}:start(j)-end(i)<14} \bigwedge_{s\in S'} (\neg x_{is} \vee \neg x_{j(s+1-|S_r|)}) \tag{6.9}$$

Because this constraint is more restrictive than the twelve hour constraint we could also exclude the night duties from that constraint to reduce the number of clauses.

Fixed duties have to be planned as fixed in the Basic Schedule.

In the Basic Schedule it can be the case that some duties are already filled in. So in that case there is no choice. If in a Basic Schedule a duty i^* is fixed in sequence s^* then the clause consisting of only $x_{i^*s^*}$ is added. This means that the formula can only be satisfied if this variable is set to True. The other constraints automatically imply that duty i^* cannot be assigned to another sequence, and also that no other duty that has the same day as duty i^* can be assigned to sequence s^*.

The type of duty must match the Basic Schedule

In the Basic Schedule each day of each sequence has a type assigned to it. In the SAT model, clauses must be added that enforce this. For example, if a late duty is assigned to day d of sequence s then this leads to the following clause:

$$\bigvee_{i\in D^L_d} x_{is} \tag{6.10}$$

Here D^L_d is the set of late duties on day d. This formula is satisfied when at least one of the duties of D^L_d is assigned to sequence s. The other constraints make sure that this has to be exactly one duty. This means that for every day that needs a duty assigned to it such a constraint is formulated. In general it will look like:

$$\bigvee_{i\in D^{type}_d} x_{is} \tag{6.11}$$

If no duty is specified in the Basic Schedule, no duty can be assigned

On days that no duty is specified, meaning there is a rest day, reserve day, WTV day, Streepjesdag or RO-day, we add a clause that no duty should be assigned. So if there is no duty type specified at day d in sequence s we add the following clauses:

$$\bigwedge_{i \in D_d} \neg x_{is} \tag{6.12}$$

Strictly speaking, these clauses are not needed as the number of duties exactly matches the number of slots in the Basic Schedule on which a duty is specified. Therefore, the only way to get a feasible solution is to keep the days on which no duty is specified free. However, adding the constraint is beneficial as it means the solver can propagate this information from the start. If these unit clauses are not added the solver must find out that there cannot be a duty on those days by making guesses and reaching conflicts.

n consecutive rest days need to be at least $24n + 6$ hours long.

In the Collective Labour Agreement it is stated that a rest day needs to be at least 30 hours long, and every additional rest day requires an additional 24 hours. In general this means that n consecutive rest days need to be at least $24n + 6$ hours long. For example, this means that if duty i finishes on Monday at 23.30, Tuesday is a rest day and duty j starts on Wednesday at 5.00, it is not allowed to put both duties together in this sequence, even though the entire Tuesday is free.

This constraint will be very similar to the constraint that there has to be at least twelve hours in between duties. The difference is that now we look at duties on day d and day $d + n + 1$, where n is the number of rest days, and that the required time difference between the end of the first duty and the start of the second duty is different.

However, since this constraint applies to more days (namely two plus n) there are much more opportunities to cross over to the next week. When one of the rest days is on Monday or Sunday the period on which the constraint applies is spread over two sequences. These situations need careful administration.

When the rest days are not on Monday and/or Sunday the formula is straightforward. The trick again is to look at all possible pairs of duties on the day before and day after the rest period. If the difference between the end time of the duty before the rest period and the start time of the duty on the day after the rest period is too small, we force the model to not take both duties in this sequence by adding a clause that will only be False if both duties are chosen.

When in the Basic Schedule in sequence s there are n rest days on days $d+1...d+n$, where $d+n+1 \mod 7 > d$ the following is added to the model:

$$\bigwedge_{(i,j) \in D_{(d,d+n+1)} : start(j) - end(i) < 24n+6} (\neg x_{is} \vee \neg x_{js}) \tag{6.13}$$

Here we assume that it cannot happen that there are seven or more rest days in a row, because

then $d + n + 1 \mod 7 > d$ could be True even though d and $d + n + 1$ are not in the same week. As this will never occur in practice in a Basic Schedule this assumption can be made.

When at least one of the rest days is on Monday or Sunday, a careful look is needed. If this is the case, then $d + n + 1 \mod 7 \leq d$. Again, the assumption is made that there cannot be seven or more rest days in a row. Now, it is again needed to distinguish between two cases. The first case is the case that day d, which is the day before the rest period, is in a sequence $s \in S \backslash S'$. The second case is the case in which $d \in S'$. Remember that S' is the set of sequences that are the last of their roster group. Now let s be the sequence of day d, let n be the length of the rest period and let $|S_r|$ be the size of the roster group of which sequence s is a part. Then the following formulas represent the constraint:

If $s \in S \backslash S'$:

$$\bigwedge_{(i,j) \in D_{(d,d+n+1-7)} : start(j) - end(i) < 24n+6} (\neg x_{is} \vee \neg x_{j(s+1)}) \tag{6.14}$$

If $s \in S'$:

$$\bigwedge_{(i,j) \in D_{(d,d+n+1-7)} : start(j) - end(i) < 24n+6} (\neg x_{is} \vee \neg x_{j(s+1-|S_r|)}) \tag{6.15}$$

Red Weekend

The Collective Labour Agreement requires that at least once per three weeks a Red Weekend is rostered. A Red Weekend is defined as a free period of at least 60 hours that contains the period from Saturday 00.00 until Monday 04.00. This period is only 52 hours long, so there need to be an additional eight free hours. The constraints will be added when a Red Weekend is scheduled in the Basic Schedule and there are duties planned on Friday and on Monday. If there is also a rest day on Friday then all requirements are met automatically, since there are no duties on Monday that start before 04.00. If there is no duty on Monday we must still watch out that we do not roster a duty that ends after midnight on Friday. Therefore the Red Weekend requirement is split up into two separate constraints.

When a Red Weekend is planned in sequence s of the Basic Schedule, there cannot be a job on Friday that finishes after midnight in sequence s.

$$\bigwedge_{i \in D_4 : i \text{ ends after } 0.00} (\neg x_{is}) \tag{6.16}$$

Additionally, there must be at least 60 hours between the end of the duty on Friday and the start of the duty on Monday. Again, we must be careful here, since the constraint is about days in different sequences. Hence, we must handle two cases again. The case in which sequence s was the last of its roster group, and the case in which it is not.

If $s \in S \backslash S'$:

$$\bigwedge_{(i,j) \in D_{(4,0)} : start(j) - end(i) < 60} (\neg x_{is} \vee \neg x_{j(s+1)}) \tag{6.17}$$

If $k \in S'$:

$$\bigwedge_{(i,j)\in D_{(4,0)}:start(j)-end(i)<60} (\neg x_{is} \vee \neg x_{j(s+1-|S_r|)}) \tag{6.18}$$

6.2 Model without the Basic Schedule

Another option we will investigate is to not use the Basic Schedule. This means we need extra feasibility constraints to make up for the constraints that were implicitly created by the Basic Schedule. All constraints that were described earlier in this chapter remain, although some of them need some adaptation. This will be discussed in this section.

First, new variables are needed to indicate whether a day has a rest day. So for all sequences s and days d the variable z_{sd} is True if in sequence s there is a rest day on day d. Then with the use of constraints it will be forced that no duty can be rostered on a day if z_{sd} is True. The constraint will be:

$$\bigwedge_{s\in S} \bigwedge_{d\in(0,6)} \bigwedge_{i\in D_d} (\neg x_{is} \vee \neg z_{sd}) \tag{6.19}$$

If there is a rest day at day d in sequence s then $\neg x_{is}$ is forced to True, meaning that the duty cannot be assigned to sequence s. If there is no rest day, the clause is satisfied regardless of $\neg x_{is}$.

Furthermore, a linear constraint will be added to ensure that the sum of all z_{sd} is equal to the number of rest days that were assigned to this group. This linear constraint is translated to clauses using a binary decision diagram translation, which will be explained in Section A.1. In this case, this does not lead to a problem. Firstly because only one roster group is looked at. Secondly because there is only one z-variable per slot in the roster group. This means the size of the BDD will be reasonably small and it will not cause issues for the solver.

Because the Red Weekends are very restrictive, and any Basic Schedule will plan them optimally (as this is part from the instruction on making Basic Schedules), they are the only thing that are copied from the Basic Schedule. This means the corresponding z variables are forced to True, and the general Red Weekend constraints are used as stated earlier in this section.

In the end, there will be days to which nothing is assigned. These days will become reserve days and WTV days. Which days would become reserve and which WTV can be determined in different ways. Later in this report a method will be proposed that looks at the usability of the reserve shifts.

Rest Constraints

When using the Basic Schedule all other constraints about rest days were added only for those spots in the Basic Schedule on which a rest day was placed. But now it is not known beforehand whether a day contains a rest day or not. To make up for this, we make use of the variables z_{sd}. As an example the constraint n consecutive rest days need to be at least $24 + n$ hours long is used, where $n = 1$. So we look at one single rest day. Remember that this constraint was:

$$\bigwedge_{(i,j)\in D_{(d,d+n+1)}:start(j)-end(i)<24n+6} (\neg x_{is} \vee \neg x_{js}) \tag{6.20}$$

For the example, we only focus on rest days that are not on Sunday or Monday, but the principle will be the same.

Now the negation of the variable $z_{s(d+1)}$ is added to each clause. This results in:

$$\bigwedge_{(i,j)\in D_{(d,d+2)}:start(j)-end(i)<30} (\neg x_{is} \vee \neg x_{js} \vee \neg z_{s(d+1)}) \tag{6.21}$$

If no rest day is assigned to day $d+1$ in sequence s then $z_{s(d+1)}$ is False. Therefore its negation is True. If this negation is True, the clause will be satisfied regardless of the assignments of x_{is} and x_{js}. However, if there is a rest day on day $d+1$ in sequence s then the negated variable is False and the constraint must be satisfied to satisfy the clause. It basically says that the constraint must be satisfied, or there must be no rest day. When the constraint would be about more rest days more variables would be added. Then it says that the constraint must be satisfied, or at least one of the days has to be no rest day.

Additional constraints

All constraints that were implicitly satisfied by the Basic Schedule must now be added to the model. These are:

- No more than seven days in a row without a rest day.
- After three consecutive duties that end after 01.00 at least two rest days.

Strictly speaking the second constraint should say that there are at least 46 hours rest after three consecutive night duties. Two rest days mean at least 54 hours rest. However, due to the nature of the duties in practice the only way to achieve the minimum of 46 hours is to put a second rest day in the roster. The reason is that if a duty ends at 01.00, then 46 hours later is 23.00. Therefore only duties that start after 23.00 are allowed, but there are none of them. Theoretically it would be allowed to put a reserve day in this spot, but this would be very undesirable for NS as this employee would have an extremely limited availability for this reserve shift. Therefore there is chosen to require a second rest day.

The first constraint could be achieved by adding a clause for every set of eight consecutive days, stating that at least one of them has to be a rest day. This is just a disjunction of the eight z variables corresponding to these days.

$$\bigwedge_{i\in[0,6],j\in[0,|S_r|]} \bigvee_{\substack{d\in[i,i+7] \\ s:j \text{ if } d<7, \text{ else } j+1 \bmod |S_r|}} z_{s(d \bmod 7)} \tag{6.22}$$

The second is a little bit trickier. This needs to be split up in two parts. The first part saying that if there are duties ending after 1.00 on days $d, d+1$ and $d+2$ then on day $d+3$ there must be a rest day. The second constraint states that there must be a rest day on day $d+4$. Assuming everything takes place within the same week the clauses look like this:

$$(\neg x_{is} \vee \neg x_{js} \vee \neg x_{ks} \vee z_{s(d+3)}) \tag{6.23}$$

$$(\neg x_{is} \vee \neg x_{js} \vee \neg x_{ks} \vee z_{s(d+4)}) \tag{6.24}$$

Here duty i, j and k are all night duties and duty i is on day d, duty j is on day $d+1$ and duty k is on day $d+2$. Again, one must be careful with the administration if the series of duties and rest days crosses the end of a week. If the three night duties are all placed after each other the x variables will be True, meaning their negations are all False. Then the z variable is forced to True for the clause to be satisfied. Hence, a rest day will be scheduled for day $d+3$ and $d+4$.

Chapter 7

Incorporating Fairness

7.1 Fairness using SAT Clauses

Fairness is an important part of the Rostering Problem. The SAT model of the previous section returns a feasible roster, assuming one exists, but it does not provide any guarantees about fairness. To model the entire Rostering Problem as a SAT instance we must find a way to encode fairness as SAT clauses. In this section we will explore three options to achieve this.

One of the easiest ways to achieve fairness is by describing the model as a pseudo-boolean problem.

Definition 13 *A* pseudo-boolean *problem is a problem with only Boolean variables, but with linear constraints.*

Linear constraints can be used to formulate bounds on the sum of variables. To achieve fairness the following constraints could be added for every roster group r:

$$Lowerbound < \sum_{i \in D} \sum_{s \in S_r} q_i x_{is} < Upperbound \tag{7.1}$$

Here q_i represents a quantity on which fairness must be achieved, such as the workload.

Een and Sorensen [10] provide three ways to translate linear constraints to SAT clauses:

1. Binary Decision Diagrams
2. Adder Networks
3. Sorters

All these encodings make use of logic gates.

Definition 14 *A* Logic Gate *consists of electronically controlled switches that implement Boolean logic processes. A logic gate has one or more logical inputs, called the input signals, that generate a single logic output, called the output signal.*

A circuit consists of multiple logic gates. All logical gates, and therefore all circuits can be translated to clauses using the Tseitin transformation.

Definition 15 *The* Tseitin Transformation *is a transformation that takes as input a logical circuit and transforms it to a formula in Conjunctive Normal Form.*

This transformation introduces new variables that represent the value of each gate. Finally it adds a unit clause that forces the outcome of the circuit to True. The size of the transformation is linear in the size of the circuit.

The laws of De Morgan and the Distributive Property are used to make the translation of each gate.

Theorem 1 De Morgan's laws *state that the following expressions are equivalent:*

$$\begin{aligned} \neg(P \vee Q) &\iff \neg P \wedge \neg Q \\ \neg(P \wedge Q) &\iff \neg P \vee \neg Q \end{aligned}$$

Theorem 2 *The* Distributive Property *states that the following expressions are equivalent:*

$$\begin{aligned} P \wedge (Q \vee R) &\iff (P \wedge Q) \vee (P \wedge R) \\ P \vee (Q \wedge R) &\iff (P \vee Q) \wedge (P \vee R) \end{aligned}$$

Each gate is translated following the same idea. All gates can be described as two implications. Every gate has a set of inputs that causes the gate to be True, and another set that causes the gate to be False. To illustrate how the Tseitin transformation works, the AND-gate is used. Let x and y be the inputs of an AND-gate, then the Boolean variable z is added to the problem. z represents the output signal of the AND-gate. The AND-gate is defined as the following implications:

$$\begin{aligned} (x \wedge y) &\implies z \\ \neg(x \wedge y) &\implies \neg z \end{aligned}$$

These implications can be turned into the following Boolean expression:

$$(\neg(x \wedge y) \vee z) \wedge ((x \wedge y) \vee \neg z)$$

Now if $(x \wedge y)$ is True then z must be True, because otherwise the statement will be False. So setting $(x \wedge y)$ to True implies that z must be True as well. If $(x \wedge y)$ is False, then z is forced to False as well. Therefore, this expression represents the implications stated above.

Using De Morgan's Law this is equivalent to:

$$(\neg x \vee \neg y \vee z) \wedge ((x \wedge y) \vee \neg z)$$

Finally, using the distributive property the gate is turned into Conjunctive Normal Form:

$$(\neg x \vee \neg y \vee z) \wedge (x \vee \neg z) \wedge (y \vee \neg z)$$

If x and y are True then z is forced to True by the first clause, while x and y force the second and third clause to True. If x is False then z is forced to False by the second clause and equivalently if y is False then z is forced to False by the third clause. If the circuit would consist of only this AND-gate, then the unit clause z is added, which forces z to True. If z is True, then x is forced to True by the second clause, and y is forced to True by the third clause. If the circuit would consist of multiple gates then z could just be used as an input for another gate, which means z could potentially be False in the final assignment.

Using this same procedure any gate can be translated by describing the truth table as two implications and translating these into clauses using De Morgan's Laws and the distributive property.

The three methods provided by Een and Sorensen [10] all express each linear constraint in as a circuit of logic gates. These circuits are then translated to clauses with the use of the Tseitin transformation. In Appendix A the methods are explained in detail.

7.2 Solving the Pseudo-Boolean problem

All three methods are implemented in the Minisat+ solver made by Een and Sorensen [10], and for this thesis all methods have been applied to the Rostering Problem.

The method using BDDs is the strongest formulation, as it preserves arc-consistency.

Definition 16 *A translation of a linear constraint into a set of SAT clauses is* arc-consistent *if and only if whenever an assignment could be propagated on the original constraint, unit propagation on the translation of the constraint also finds this assignment.*

This property is very useful, as unit propagation is the fastest technique of a SAT solver. Every variable that can be assigned by unit propagation does not have to be a part in a guess later on. However, in the case of BDDs this comes at a cost as the number of clauses needed to translate a pseudo-boolean constraint into clauses is exponentially large in the number of variables [3]. The BDD encoding proved to be the best encoding for a lot of the benchmark problems of the Pseudo-Boolean competition Een and Sorensen [10] used to judge the performance of their solver. However, the size of these problems was much smaller than our Rostering Problem.

The fact the size of the BDD encoding is exponential in the number of variables is problematic, as for the Rostering Problem the number of variables is equal to the number of employees times the

number of duties. To limit the size of the fairness constraints, it is important to only include those variables in the constraint that could potentially be set to True. To do this, a variable x_{is} is only included in the fairness constraint if the type of the duty i matches the type in the slot of the Basic Schedule at Weekday(i) in sequence s.

The number of clauses needed in the test instance when using BDDs was still in the order of hundred thousands, or in some cases even millions per constraint. In total there are five attributes and seven roster groups, meaning that 35 of these constraints must be added. The model without fairness constraints had about two million clauses. Even when only fairness of working time is required, the size of the problem becomes approximately ten times as big as it was. As a result, solving it within reasonable time became impossible.

Another approach proposed by Een and Sorensen makes use of sorters. This encoding proved to be the best for most of the benchmark problems of the pseudo-boolean competition were the BDD encoding did not perform well. Sorters do preserve arc-consistency for cardinality constraints ($q_i = 1$ for all i), but lose this property in general pseudo-boolean constraints. The size of the encoding via sorters is $O(n\ log^2\ n)$, where n is the total number of digits needed to express all the coefficients in the chosen base. This shows the importance of choosing a smart base.

The number of clauses needed for the sorter-encoding was in the order of tens of thousands per constraint. Therefore, the size of the problem does not explode as with the BDDs. However, the solver still did not find a solution within reasonable time when only fairness of workload was considered. The reason most likely is the loss of arc-consistency in the fairness constraints. Sorters did preserve arc-consistency for cardinality constraints, but not when the coefficients are not equal to one, which is the case with the fairness constraints. Every time unit propagation does not take place while the value of the variable could have been known an extra guess is needed, meaning an extra opportunity to guess wrong.

This leaves the adder encoding as final option. In Minisat+, a heuristic is build in that, unless specified differently by the user, chooses for each linear constraint which encoding is most likely to be the most suitable one. This heuristic chose the adder encoding for the fairness constraints. In general, the heuristic takes the BDD-encoding as long as it doesn't blow up, else it looks if the sorter encoding is not too large. If both are too large it takes the adder encoding.

The big advantage of this encoding is its size, which is $O(n)$, where n again is the total number of digits needed to express all the coefficients. However, the big disadvantage is the loss of arc-consistency for all types of linear constraints. Even when only one fairness constraint was added the problem was not solved in reasonable time, although it reached a much higher progress than the other two options. Een and Sorensen already concluded in their paper that the adder encoding performs bad and that in the case of SAT the strength of propagation seems to be more important than the size, as long as the size does not blow up completely.

These findings led to believe that encoding the entire problem, including fairness, as a SAT instance would not result in a good algorithm. Therefore, adding attractiveness to this model has not been tried. Instead, a hybrid approach will be proposed in the following section.

Chapter 8

Hybrid Approach

From the previous sections it could be concluded that a SAT encoding is very suitable to model the feasibility part of the Rostering Problem, but not suitable to model the fairness part. In this section a hybrid approach is proposed that uses mixed integer programming techniques to solve the fairness part, and SAT techniques to solve feasibility. Furthermore, attractiveness is integrated in the feasibility part by formulating the problem as a MAX-SAT instance, instead of a general SAT instance.

In the first part of the algorithm the duties will be divided in a fair way among the roster groups using a mixed integer program. In the second part of the algorithm the duties are rostered as attractive as possible in the roster group they were assigned to. The second part uses a MAX-SAT approach. The third part of the algorithm will evaluate the solution and either output it, or give feedback to the first part and ask it for a different assignment of duties to the roster groups. The algorithm will loop through these three parts until either a solution is found that satisfies the fairness and attractiveness requirements, or until no solution can be found anymore. A very brief outline of the hybrid approach is given in the algorithm below.

Algorithm 1: Hybrid Algorithm

```
while Solution is not good enough do
    Run Fairness
    if Fairness is infeasible then
        return Best solution so far
    else
        for i in NumberOfRosterGroups do
            Run Attractiveness(i)
    if Solution is good enough then
        return solution
    else
        Give feedback to Fairness
```

8.1 Fairness

The first part of the method will make a fair division of the duties, such that every roster group receives duties that match the slots of the Basic Schedule. However, nothing is decided yet about the specific place in the roster group the duty will be assigned to. For example, if one roster group has three sequences that have an early duty on Monday, then three early duties with weekday Monday need to be assigned to this group. The second part of the algorithm will then decide which one will be at each specific Monday. Note that the specific assignment does not influence fairness, as all members of the roster group will perform all of the duties assigned to that roster group.

8.1.1 Mathematical Formulation

The first part of the model is formulated as a mixed integer program. The constraints will enforce that each group receives duties matching the Basic Schedule. Furthermore, constraints are added to enforce fairness.

To match the slots of the Basic Schedule we basically create a roster, but without adding any feasibility constraints other than those that enforce the model to match the Basic Schedule.

First we add the constraint that makes sure that every duty is assigned to exactly one sequence.

$$\sum_{s \in S} x_{is} = 1 \quad \forall i \in D \tag{8.1}$$

Then for every slot in the roster the constraint is added that requires the model to assign exactly one duty that matches the type of the Basic Schedule to it.

For every sequence s that has a duty of type T assigned to it on day d we add the following constraint:

$$\sum_{i \in D_d^T} x_{is} = 1 \tag{8.2}$$

For the fairness constraint we split in two cases, namely the average workload and the other quantities. The reason is that the average workload consists not only of days on which duties are scheduled. Reserve days, RO-days, WTV-days and streepjesdagen all count as eight hours of work. Let R_t be the total number of reserve days, RO-days, WTV-days and streepjesdagen, and R_r be the number of those days in roster group r. Let the length of duty i be denoted as l_i.Then for all roster groups the following constraint is added:

$$\frac{1}{|S|}(\sum_{i \in D} l_i + 8R_t) \cdot (1 - \epsilon) < \frac{1}{|S_r|}(\sum_{i \in D} \sum_{s \in S_r} l_i \cdot x_{is} + 8R_r) < \frac{1}{|S|}(\sum_{i \in D} l_i + 8R_t) \cdot (1 + \epsilon) \tag{8.3}$$

For the other quantities we only consider the days on which a duty takes places. Let $D(r)$ be the number of duties in roster group r then for every roster group the following constraint is added:

$$\frac{1}{|D|}\sum_{i\in D} a_i \cdot (1-\delta_a) < \frac{1}{D(r)}\sum_{i\in D}\sum_{s\in S_r} a_i \cdot x_{is} < \frac{1}{|D|}\sum_{i\in D} a_i \cdot (1+\delta_a) \tag{8.4}$$

Bounds are added for ϵ and all δ_a to ensure a minimum fairness level for all quantities. The constraints need for this are:

$$0 \leq \epsilon \leq M \tag{8.5}$$

$$0 \leq \delta_a \leq M_a \tag{8.6}$$

The model will minimize the sum of ϵ and δ_q for all attributes a. It is possible to add weights in the objective function to put more emphasis on certain attributes. In practice one probably wants to put more importance on the workload. It would be very undesirable if one roster group on average works an hour less than another roster group, as this would result in the fact that after a year one roster group has worked more than a week more than the other group. This would already be the case if ϵ is around 1.5%. On the other side, if such a difference would occur in for example the percentage double deckers this would be fine.

This method of imposing fairness is slightly different than most ways proposed by the literature, but it is close. These constraints ensure that every roster group gets assigned each attribute within a δ_a margin of the average. In other words, the difference between the minimum and maximum assigned value of a certain attribute to the groups is at most $2\delta_a$ times the average value.

The main reason to choose for this approach is because the number of extra variables that are needed is small, only one per attribute. As the solving method will consist of multiple iterations and all iterations have to solve the fairness part, speed is a key factor. By limiting the number of variables in the objective function it is tried to also limit the running time of the algorithm. We saw in the literature [7][13][14] that often heuristics were needed to find the fair solution when for example the max-min or minimum deviation objectives were used. As a consequence these solution methods will most probably not give an optimal solution either. If a fast algorithm can find a solution that could be considered "fair enough" then in practice this will be a very good alternative.

An even faster alternative is to use a meaningless objective function and rely on the bounds M and M_a. This way fairness is not optimized, but there is a guarantee that fairness is between acceptable bounds. The advantage is that solving the problem will be quicker.

The last step of the first part is to extract which duties were assigned to which group. This can be done with new variables y_{ir} that are 1 if duty i is assigned to roster group r. To enforce this we add the following constraint for every roster group r:

$$\sum_{a\in S_r} x_{ia} = y_{ir} \tag{8.7}$$

If duty i is assigned to one of the sequences of roster group r then the sum is equal to one, which forces y_{ir} to one. If duty i is not assigned to one of the sequences of roster group r the sum is zero and therefore y_{ir} is also zero.

The fairness model is stated below. Here the type of duty that is assigned to day d in week s of the Basic Schedule is denoted as B_{sd}.

Fairness

$$\begin{aligned}
\text{minimize} \quad & w\epsilon + \sum_a w_a \delta_a & \\
\text{subject to} \quad & \sum_{s \in S} x_{is} = 1 & \forall i \in D \\
& \sum_{i \in D_d^T} x_{is} = 1 & \forall d \in [0,6], T = B_{sd} \\
& \sum_{s \in S_r} x_{is} = y_{ir} & \forall i \in D \\
& x_{is} \in \{0,1\} & \forall i \in D, \forall s \in S \\
& y_{ir} \in \{0,1\} & \forall i \in D, \forall r \in R \\
& 0 \leq \epsilon \leq M & \\
& 0 \leq \delta_a \leq M_q & \forall a \\
& \text{Fairness of workload (8.3)} & \forall r \in R \\
& \text{Fairness of other attributes (8.4)} & \forall r \in R, \forall a \in A
\end{aligned}$$

8.2 Feasibility and Attractiveness

In the second part of the algorithm the variables y_{ir} are used to determine which duties will be assigned to each roster group. Then for each roster group a separate MAX-SAT instance is formulated and solved.

8.2.1 MAX-SAT

MAX-SAT is a variation of a SAT problem in Conjunctive Normal Form which has two types of clauses. Hard clauses are clauses that must be satisfied, just like in a normal SAT instance. Soft clauses are clauses that may be violated if this is needed to satisfy the hard clauses. The goal of solving a MAX-SAT instance is to find an assignment of the variables such that all hard clauses are satisfied, and as many soft clauses as possible. If a hard clause is violated, the problem is unsatisfiable.

An extension of MAX-SAT is weighted MAX-SAT. Here every clause has a weight associated with it. The goal now becomes to minimize the total weight of the soft clauses that are violated.

MAX-SAT is used to model the feasibility of the Rostering Problem including the attractiveness. Hard clauses are used to model feasibility, as no violation of these constraints is allowed. The clauses that are necessary to get a feasible solution are described in Section 6. Attractiveness is modeled as soft clauses, as violating attractiveness constraints is undesirable but allowed. It will most likely be impossible to find a roster that obeys all attractiveness constraints. Using weights can help to quantify the importance of a soft clause.

Attractiveness Constraints

Some attractiveness constraints are equivalent to feasibility constraints. For example the constraint that the time between two duties ideally is longer than twelve hours. One way to formulate this constraint is: the time between duties must be at least thirteen hours. Weights can be used to quantify how bad it is to violate the constraint. In this case it is reasonable to make the weight equal to the number of minutes the constraint is violated. This would result in a weight between one and 60. If the weight would be less than one the clause would not be added. Weights of more than 60 can occur, but in that case there is a hard clause that prevents this soft clause from being violated. One could choose for more than thirteen hours. Breugem [6] chose sixteen hours in his model. This means that there will be multiple violations, because having every rest time at least sixteen hours is impossible.

In the feasibility part in Section 6.1 there is already a constraint that enforces the time between duties to be at least twelve hours, namely (ignoring Sundays):

$$\bigwedge_{d\in(0,5)} \quad \bigwedge_{(i,j)\in D_{(d,d+1)}:start(j)-end(i)<12} \quad \bigwedge_{s\in S} (\neg x_{is} \vee \neg x_{js}) \tag{8.8}$$

The attractiveness constraints will be:

$$\bigwedge_{d\in(0,5)} \quad \bigwedge_{(i,j)\in D_{(d,d+1)}:start(j)-end(i)<13} \quad \bigwedge_{s\in S} (\neg x_{is} \vee \neg x_{js}) \tag{8.9}$$

There are more preferences that are just a less strict version of a feasibility constraint. In those cases, the soft clauses are created equivalently.

There are also preferences that are not directly linked to feasibility constraints. One example is the constraint that consecutive early duties have increasing starting times. To model this a set of clauses is added for every spot in the Basic Schedule where two early duties appear on consecutive days:

$$\bigwedge_{d\in(0,5)} \quad \bigwedge_{(i,j)\in D^E_{(d,d+1)}:start(j)<start(i)} (\neg x_{is} \vee \neg x_{js}) \tag{8.10}$$

If the first day is a Sunday this must be changed to for $s \in S \backslash S'$:

$$\bigwedge_{(i,j) \in D^E_{(6,0)} : start(j) < start(i)} (\neg x_{is} \vee \neg x_{j(s+1)}) \tag{8.11}$$

And for $s \in S'$:

$$\bigwedge_{(i,j) \in D^E_{(6,0)} : start(j) < start(i)} (\neg x_{is} \vee \neg x_{j(s+1-|S_r|)}) \tag{8.12}$$

Again the weight could be formulated as the number of minutes the constraint is violated by the two duties.

The constraint that late duties should have decreasing end times is very similar:

$$\bigwedge_{d \in (0,5)} \bigwedge_{(i,j) \in D^L_{(d,d+1)} : end(i) < end(j)} (\neg x_{ik} \vee \neg x_{jk}) \tag{8.13}$$

If the first day is a Sunday this must be changed to for $k \in S \backslash S'$:

$$\bigwedge_{(i,j) \in D^E_{(6,0)} : end(i) < end(j)} (\neg x_{ik} \vee \neg x_{j(k+1)}) \tag{8.14}$$

And for $k \in S'$:

$$\bigwedge_{(i,j) \in D^E_{(6,0)} : end(i) < end(j)} (\neg x_{is} \vee \neg x_{j(s+1-|S_r|)}) \tag{8.15}$$

The clauses are added when two consecutive late duties appear in the Basic Schedule.

In this thesis we use four attractiveness constraints, namely:

- At least thirteen hours between duties
- One single rest day is at least 34 hours
- Early duties must have increasing starting times
- Late duties must have decreasing end times

However, SAT provides the possibility to mark any combination of duties as unattractive. So if one defines attractiveness differently this approach can still be used.

8.2.2 Evaluating the Solution

After the MAX-SAT problems are solved a couple of things can happen. The first possibility is that at least one of the MAX-SAT problems returns UNSAT, which means that for this roster group

the given distribution of duties was infeasible. The model then gives this feedback to the first part. This is done by adding a constraint to the fairness part that explicitly forbids this assignment to that specific roster group.

If there is no MAX-SAT problem that returns UNSAT, a feasible and fair roster has been found. Now the model must decide if it is satisfied with this roster. Given the distribution of duties by the fairness part this roster is optimal with respect to attractiveness, but another division of duties may result in a much better roster. A threshold value is needed to indicate how much penalty is considered acceptable for a roster group.

If the attractiveness penalty is higher than the chosen threshold value the model does the same as when one group was UNSAT, which is forbidding this exact assignment of duties to that group. Furthermore, we can deduce which specific duties cause the attractiveness was very high. This information could be used to give more feedback than only explicitly forbidding this composition of the roster group. However, this could become dangerous as the optimal solution might get lost. Later in this report an investigation is done regarding multiple heuristic ways to give feedback and their results regarding the quality of solutions they create. In practical instances, a huge amount of feasible rosters exist. Therefore, even if the optimal solution is lost it is very likely there are multiple solutions that are very close to this optimal solution in quality.Therefore in practice it might not be a big problem if the optimal solution is lost.

The final case is when the MAX-SAT problems all return a solution which is well enough. A global threshold value will be used to quantify "well enough". Then if a solution is found that has a weight below the threshold value, the model returns this roster and terminates.

The model keeps looping through the three parts until a solution has been found that has a total weight below the threshold value and in which each group has a weight below the group threshold, or until it times out, or until the fairness part becomes infeasible. One could keep track of the best found solution so far and output this when the model did not find a solution with weight below the threshold value.

Unsatisfiable groups

It turns out that it happens a lot in practice that groups are UNSAT. To limit this it is possible to add some feasibility constraints to the fairness part. It makes sense to add the most restrictive constraints, such as the twelve hour rest between duties. For example, there might be a spot in the Basic Schedule where a late duty is followed by an early duty such that only a couple of combinations of duties fit. Also red weekends are likely to cause problems, certainly when there is a late duty on Friday. A lot of late duties end after midnight and can therefore not be put in front of a red weekend. If this information is added to the fairness part we know that for these constraints at least one option is in the group that does satisfy it. It still can happen that the group turns out to be UNSAT, but it has become less likely. More feasibility constraints to the fairness part probably come at the expense of losing flexibility in the MAX-SAT part. However, it cannot happen that feasible solutions get lost. In the implementation part we will investigate whether adding feasibility constraints to the fairness part will improve the algorithm.

Creating linear feasibility constraints from the SAT model

The feasibility constraints are build from the SAT clauses described in Section 6. Each clause is represented by a cardinality constraint that says that the sum of the variables must be at least one. If a variable appears negated in the SAT clause it appears as 1 minus the variable in the cardinality constraint.

8.3 Feedback

When the model in part two does not find a solution with small enough weight it gives feedback to part one. The first feedback it gives is adding a constraint to the fairness part that explicitly forbids the assignments of roster groups that were either UNSAT, or had to much weight. Let D' be the set of duties that were assigned to roster group r such that the model in part two returned either UNSAT or a too large weight, then the following constraint is added:

$$\sum_{i \in D'} y_{ir} < |D'| \tag{8.16}$$

This feedback is very limited. As there is probably a huge amount of feasible solutions, only preventing the model from taking the same distribution again does not achieve a lot. This feedback only forbids infeasible assignments, and therefore it cannot happen that good solutions are lost.

8.4 Heuristic Feedback Methods

Only forbidding one assignment each iteration is very limited. The number of feasible assignments of duties to groups is very large, and increases exponentially when the instance becomes larger. This means that the Hybrid Model will probably find a lot of very bad solutions. Therefore, we want to look further. In this section four heuristic methods are proposed to give additional feedback. These feedback methods look at the solution and try to identify what causes the penalty to be big. Then they will forbid all assignments that contain this bottleneck. The risk of forbidding more than just the entire assignment is that feasible, and possibly very good solutions could get thrown away. Let's say duty i and j cause a very high weight in the optimal MAX-SAT solution for some roster group. There could be another assignment of duties to this group, such that it still contains both duty i and duty j and the MAX-SAT solution will be very good, because in this case duty i and j were placed in different spots of the roster. Therefore, we must watch out with stricter feedback rules. The heuristic methods do not solve the problem exactly, but the aim is to speed up the algorithm and still get a good solution.

8.4.1 High Weighted Pairs

The first heuristic we propose looks at the pairs of duties that cause the high penalties. When a solution is found by the MAX-SAT solver one can extract which soft clauses are violated, and the weight associated with this violation. If some pair of duties causes a very high penalty, a constraint is added that prevents the fairness model from putting these two duties together in this roster group again.

Let i and j be the duties that caused a violation larger than the threshold value T in group r the following constraint can be added to the fairness part:

$$y_{ir} + y_{jr} \leq 1 \tag{8.17}$$

This way, in the following iterations duty i and j cannot both be assigned to roster group r again.

Although this might throw away good solutions it seems reasonable to do if the weight is really large. This can be accomplished by setting the threshold T to a large number. As stated before, there is a huge amount of feasible solutions, and therefore throwing away a small amount of good solutions is probably not the end of the world. That said, if T is set to a low value, this might result in the entire model becoming infeasible within a small amount of iterations. Another possible danger is that there might be spots in the Basic Schedule where it is impossible to roster two duties without getting a high violation. Then, forbidding a high weighted assignment might directly lead to infeasibility.

8.4.2 High Weighted Groups

The method of high weighted pairs had some risks, as possibly very good solutions are thrown away, and infeasibility might be reached within a small amount of iterations. To limit this risk, this heuristic looks at all pairs that cause a high weight and adds a constraint to the fairness part that asks to not put all these duties together in that group again. So instead of breaking apart all pairs with high weight, the model is forced to break up at least one pair. The constraint that is needed to achieve this is very similar to the constraint that forbids the model to pick the same assignment again to a roster group. However, now we use the set $\tilde{D}$, which is the set of all duties that cause a penalty larger than T in roster group r.

$$\sum_{i \in \tilde{D}} y_{ir} < |\tilde{D}| \tag{8.18}$$

When a penalty larger than T happens at only one spot, the constraint becomes the same as the high weighted pairs constraint, so the potential dangers are the same. However, when there are multiple pairs of duties that cause a high penalty it is very unlikely that a very good solution contains all these duties, so this limits the risk of such dangers actually becoming a problem. The question is if this feedback is strong enough to improve the model considerably.

8.4.3 High Weighted Days

The first two methods proposed focused on pairs of duties that caused a very high penalty. Another option is to look at the days in the solution that cause these high penalties. Remember that the division of duties to roster groups was made by filling in a Basic Schedule. If in the solution two duties are rostered such that they cause a high penalty, we will forbid the fairness part to put these two duties on this spot in its roster. This means that the two duties could still end up in this group when the fairness part rosters them to other days within this group. Therefore they could even

end up in the same spot again in the MAX-SAT solution. However, by giving this restriction we reduced the number of options of this happening. The advantage is that when in a good solution these two duties are scheduled at other days in this group, that solution does not get lost.

Let i and j be two duties that cause the high penalty in sequence s. The constraint that must be added to the fairness part to achieve this goal is:

$$x_{is} + x_{js} \leq 1 \tag{8.19}$$

Notice that in this case the x-variables are used and not the y-variables, since we want to restrict the assignment of the duties on a sequence level instead of a roster group level.

8.4.4 High Weighted Cuts

The last heuristic we propose also focuses on the specific spots in the roster. The idea is to add cuts. If in the solution two duties violate some soft constraint with a too big margin, a cut will be added to the fairness part that requires the two duties that are rostered on these days to not violate that specific constraint more. This does not prevent the model from assigning these duties to that group and also does not guarantee that in the MAX-SAT solution the constraint is not violated worse. However, the likelihood of this happening is reduced, which hopefully makes the algorithm find good solutions quicker. This heuristic forbids many assignments, because also other combinations of duties than the one that caused the violation are forbidden.

The cuts are specific for each constraint. As example we take the constraint that the time between two duties on consecutive days needs to be at least 13 hours. Now in the MAX-SAT solution two duties are assigned that have only 12.5 hours rest in between. Let duty i be on day d and duty j be on day $d+1$. Let the violaton happen in sequence s. Then the cut that would be added is:

$$\sum_{j \in D_{d+1}} x_{js} \cdot start(j) - \sum_{i \in D_d} x_{is} \cdot end(i) \geq 12.5 \tag{8.20}$$

The other constraints make sure that only one of the x_{is} and only one of the x_{js} is equal to 1. In general if duty i on day d_1 in sequence s_1 and duty j on day d_2 in sequence s_2 cause a penalty of P than the cut will look like:

$$\pm \sum_{i \in D_{d_1}} x_{is_1} \cdot q_i \pm \sum_{j \in D_{d_2}} x_{js_2} \cdot q_j \odot P \tag{8.21}$$

Here q_i is the quantity of importance in the constraint and $\odot \in \{\leq, \geq\}$ depends on the type of constraint. Using strict inequalities is also possible, but this carries more risk of forbidding good solutions as the current solution will also be forbidden. In theory, attractiveness constraints could also be about more than two days. In that case, one can just add terms to achieve this.

Adding such a cut only makes sense when the penalty depends on the amount of violation of the constraint. If a unit penalty, or other fixed penalty is used, adding such a constraint only prevents certain divisions that would lead to an equal penalty for this constraint. Therefore you would risk losing divisions that would have an equal penalty on this constraint, but less penalty on other constraints.

8.5 Attractiveness Constraints without the Basic Schedule

Now we will look at attractiveness when we do not make use of the Basic Schedule. The model we proposed will give a feasible roster with respect to the CLA requirements, but the Basic Schedule also had some attractiveness build into it. Therefore, additional soft constraints are needed to get a better roster. Constraints that could be added are:

- No more than 6 (or even 5) consecutive days without a rest day.
- No Rest - Duty - Rest.
- Early and Late/Night duties must be divided by at least one rest day.
- Try to avoid one single rest day.

Employees like short stints of work of the same type, but only one day is not preferred. Furthermore, it is preferred that there is at least one rest day between the switch from duty type. Since the difference between a night duty and a late duty is not so big this constraint is most important for the switch from and to an early duty. The first three constraints all help to create such a short cyclic roster without asking too much.

The last constraint is very restrictive, partly because it contradicts the other three in a way. This could slow down the solver considerably. To create the short cyclic roster single rest days would be very useful. In the Basic Schedule single rest days happen quite regularly. However, employees also like longer rest periods. Therefore an option could also be to only try to avoid certain patterns. For example forbid a late duty in front of a single rest day, or an early duty directly after a single rest day. This way you do not avoid Early - Rest - Late, but that pattern is way better than for example Late - Rest - Early.

No more than 6 consecutive days without a rest day

The first constraint is equal to constraint 6.22, with the only exception that the clauses will all be only about 5 or 6 days instead of 7.

No Rest - Duty - Rest

The second constraint could be enforced by looking at every spot of the roster. If there is a duty then either the day before, or the day after has to be no rest day. This can be done with the following SAT clauses when the duty is not on Monday or Sunday:

$$\bigwedge_{s\in S}\bigwedge_{d\in(1,5)}\bigwedge_{i\in D_d}(\neg x_{is}\vee\neg z_{s(d-1)}\vee\neg z_{s(d+1)}) \tag{8.22}$$

When the duty is on Monday or Sunday, careful administration is needed, because then one of the rest days is part of a different sequence. This means that jumping from the first to the last week and vice versa also will come into play again. These clauses are only False when all three things happen, a duty on day d in sequence s and a rest day the day before and the day after.

Early and Late/Night duties must be divided by at least one rest day

The third constraint consists of two parts. First it must forbid that an early and late or night duty appear on consecutive days. Again, we will only show the constraint when both days are in the same week. When this is not the case, one must watch out with the indices.

$$\bigwedge_{d\in(0,5)}\bigwedge_{\substack{i\in D_d^E\\ j\in D_{d+1}^L\cup D_{d+1}^N}}\bigwedge_{s\in S}(\neg x_{is}\vee\neg x_{js}) \tag{8.23}$$

$$\bigwedge_{d\in(0,5)}\bigwedge_{\substack{i\in D_{d+1}^E\\ j\in D_d^L\cup D_d^N}}\bigwedge_{s\in S}(\neg x_{is}\vee\neg x_{js}) \tag{8.24}$$

Then it must enforce that if an early duty and a late or night duty appear two days apart, then a rest day is in between, otherwise this day could get nothing assigned to it, which means there would be a reserve shift or WTV day.

$$\bigwedge_{d\in(0,4)}\bigwedge_{\substack{i\in D_d^E\\ j\in D_{d+2}^L\cup D_{d+2}^N}}\bigwedge_{s\in S}(\neg x_{is}\vee\neg x_{js}\vee z_{s(d+1)}) \tag{8.25}$$

$$\bigwedge_{d\in(0,4)}\bigwedge_{\substack{i\in D_{d+2}^E\\ j\in D_d^L\cup D_d^N}}\bigwedge_{s\in S}(\neg x_{is}\vee\neg x_{js}\vee z_{s(d+1)}) \tag{8.26}$$

To satisfy these clauses if there is an early duty on day d and a late or night duty two days later or vice versa, a rest day must be assigned to the day in between. The same constraint should be added for the case that there are two days in between the two duties of different type, to enforce at least one of the days in between to be a rest day. The other could then be reserve or WTV. We can also add it for the cases that three or four days are in between, but this probably goes fine automatically, as there is a limited amount of reserve and WTV days. Another option would be not adding it and in the extremely rare case that it would happen that three or four reserve shifts are placed in between demand another solution.

Avoid single rest days

If you want to forbid one single rest day this can be done by saying that if there is a rest day then the day before, or the day after must be a rest day too. The constraint would be of the form:

$$\bigwedge_{d\in(1,5)} \bigwedge_{s\in S} (\neg z_{sd} \vee z_{s(d-1)} \vee z_{s(d+1)}) \tag{8.27}$$

And with a small adaptation the constraint could also be added when the initial rest day was on Monday or Sunday.

If we only want to forbid certain patterns containing one single rest day this can be done the following way (ignoring switches of sequence):

$$\bigwedge_{d\in(0,4)} \bigwedge_{\substack{i\in D_d^{T_1} \\ j\in D_{d+2}^{T_2}}} \bigwedge_{s\in S} (\neg x_{is} \vee \neg x_{js} \vee \neg z_{s(d+1)}) \tag{8.28}$$

If we want to forbid Late - Rest - Duty we choose T_1 to be Late and T_2 to be the union of Early, Late and Night.

These are general preferences that could be build in the model using the same techniques as used before. However, it might happen that some crew base has different preferences. In that case it is also possible to adjust the model to adhere to these preferences.

8.5.1 Post-Processing Reserves

Let us assume that in the end a solution has been found. Then it most probably will be possible to make a lot of changes within this solution that do not influence the penalty. So although the solution was optimal with respect to attractiveness given the fairness level, this most likely is not the only solution that has this property. Most MAX-SAT solvers have the option to output the top k solutions, which are the k solutions with lowest penalty for some given k. If these are solutions with equal penalty the solver will find them extremely fast. So which one of these solutions should we pick? Picking the first one the solver gives is a valid option, as this solution will be fair and attractive, but we might be able to do better without much effort.

Remember we assigned reserve shifts to all days on which no duty and no rest day was assigned. Later we must choose which of these days become real reserves, and which become WTV. When picking the final solution we could look at the potential of the reserve shifts.

When an employee is called up on a day he is reserve this can only be for those hours on which he is allowed to work according to the CLA. So if he for example had a duty the day before he must have had twelve hours rest before he can start his reserve shift. This means that not every reserve shift is equally valuable. It seems logical to measure the value of a reserve shift as the time that the employee is available, but there is a better way. When an employee is called up, he must take

over a shift of another employee that dropped out. Therefore we will measure the value of a reserve shift as the number of shifts that would fit into this reserve shift.

Still, the solution with the highest value might not be the preferred one. The most important thing is that every duty is covered by at least one reserve shift, such that if the employee that was rostered to perform this duty drops out he can be replaced.

If the Basic Schedule is made accurately there will be very few reserve shifts, as the reserve shifts are only used to fill up the last week. However, reserves are needed because employees will drop out regularly. Therefore an option is to include these reserve shifts in the rosters of the employees. Basic Schedules are made manually and planning more reserve shifts makes it easier to create the Basic Schedule because they have very few restrictions. Furthermore, because of the Red Weekend rule it seems best if roster groups have a size that is a multiple of three.

So when the MAX-SAT solver picks the k best solutions, we pick the one for which the reserve shifts have the highest potential. First we select on the number of duties that is covered by at least one reserve. If this is equal, we select on the total number of shifts that can be covered by all reserve shifts, meaning that duties that are covered n times count as n.

Chapter 9

Solving Method

The solving method consists of several steps, which include the algorithms introduced in the previous sections. Then the entire solving procedure is shown in Algorithm 2.

The algorithm starts with initiating thresholds. In the MAX-SAT problem (Attractiveness) violation of a soft constraint leads to a penalty. A global threshold represents the amount of penalty that is considered acceptable. Furthermore a GroupViolation is used to set a limit on the amount of violation that is considered acceptable for one single roster group. After the initialization the algorithm enters a while loop that only stops when the total violation is below the global threshold and all groups have a violation below the group threshold, or if no solution can be found anymore.

The loop starts with running the Fairness subalgorithm. In this phase a fair division of duties among the roster groups is made. Then, a for-loop is entered that for all groups takes the outcome of the fairness phase and determines the most attractive solution for this particular division by solving the MAX-SAT instance specified in Section 8.2.

If the problem is satisfiable, the algorithm returns a roster for this group and a number that equals the penalty corresponding to violating soft constraints. If it is unsatisfiable, this violation value will be infinity, or in practical usage, any number that is higher than the threshold will achieve the desired result. In this case forbidding this division can be done by adding a constraint to the fairness phase that explicitly forbids to assign this division of duties again to this group. If the problem was satisfiable, but the penalty was too high the same can be done. Additionally heuristics can be used to make the feedback stronger.

Depending on the choice of thresholds it might happen that all groups are satisfiable, and have an acceptable violation, but still the global violation is too high. In that case at least one change has to be required, because otherwise the algorithm would enter an eternal loop that keeps giving the same solution every iteration. This constraint becomes redundant when a group was unsatisfiable, or had a too high weight, because the constraints added in that case would be stronger.

The algorithm finishes if the global violation is smaller than the threshold, and Stop is equal to

zero. To achieve the last, all groups must be satisfiable and have less penalty than the group threshold. Another way the algorithm finishes is when no feasible division can be found anymore in the fairness phase. In that case, the best found solution so far is returned, but this will be a solution for which the thresholds are not respected. This could happen when the fairness criteria are put too strict, the thresholds are chosen too low, or when a heuristic is chosen that forbids divisions too aggressively.

Finding the optimal solution

When the Threshold is set to zero, the GroupThreshold to infinity and no heuristics would be used, the algorithm will simply enumerate solutions that satisfy the chosen fairness requirements, until it finds a solution that does not have any penalty or until no fair division exists anymore. Therefore, in that case the algorithm will always find the optimal solution with respect to attractiveness considering the given fairness limitations. Remember that in the fairness phase there was a choice to either optimize fairness, or just to set bounds. When optimization is chosen and a solution without penalty would be found, this solution is optimal with respect to attractiveness, and no division could be found that achieves the same attractiveness and is more fair. When only the bounds are used and a solution without penalty is found, it might happen that another solution exists that also does not have a penalty, but would be better with respect to fairness.

Heuristic approach

When other thresholds or heuristics are used, the guarantee of eventually finding the optimal solution will be gone but the algorithm will speed up as there will be a lot of divisions that will not be investigated anymore.

Attractiveness without the Basic Schedule

When the choice is made to not use the Basic Schedule in the MAX-SAT algorithm, some changes are made to the total algorithm. First, instead of Attractiveness, the AttractivenessWithoutBasic-Schedule subalgorithm (Section 6.2) is used. Furthermore the heuristics cannot be used anymore. The high weighted days and high weight cuts heuristics directly make use of the Basic Schedule and therefore are not applicable anymore. The high weighted groups and high weighted pairs heuristics could potentially be used, but the risk of forbidding good combinations of duties becomes a lot bigger, as there is way more freedom in the places were duties could end up within a roster group. Therefore, using these heuristics would probably lead to excluding good solutions very quickly. The adjusted algorithm is shown in Algorithm 3.

Algorithm 2: Rostering

```
Initiate Threshold;
Initiate GroupThreshold;
BestSolution ← "No solution found";
Violaton ← Threshold + 1;
Stop ← 0;
while Violation > Threshold OR Stop > 0 do
    Violation ← 0;
    Stop ← 0;
    Run Fairness;
    if Fairness returns Infeasible then
        return BestSolution;
    else
        for i in NumberOfRosterGroups do
            Run Attractiveness(i);
            if Attractiveness returns UNSAT then
                Add constraint (8.18) to Fairness that forbids this division for group i;
                GroupViolation ← Threshold + 1;
                Stop ← Stop + 1;
            else if GroupViolation > GroupThreshold then
                Add constraint (8.18) to Fairness that forbids this division for group i;
                Run FeedbackHeuristics;
                Stop ← Stop + 1;
            Violation ← Violation + GroupViolation;
        if Violation > Threshold then
            Add a constraint to Fairness to require at least one change in the divisions;
        if Violation is smallest so far then
            Update BestSolution;
return BestSolution;
```

Algorithm 3: Rostering without Basic Schedule

```
Initiate Threshold;
Initiate GroupThreshold;
BestSolution ← "No solution found";
Violaton ← Threshold + 1;
Stop ← 0;
while Violation > Threshold OR Stop > 0 do
    Violation ← 0;
    Stop ← 0;
    Run Fairness;
    if Fairness returns Infeasible then
        return BestSolution;
    else
        for i in NumberOfRosterGroups do
            Run AttractivenessWithoutBasicSchedule(i);
            if AttractivenessWithoutBasicSchedule returns UNSAT then
                Add constraint (8.18) to Fairness that forbids this division for group i;
                GroupViolation ← Threshold + 1;
                Stop ← Stop + 1;
            else if GroupViolation > GroupThreshold then
                Forbid this division for group i;
                Stop ← Stop + 1;
            Violation ← Violation + GroupViolation;
        if Violation > Threshold then
            Require at least one change in the divisions;
        if Violation is smallest so far then
            Update BestSolution;
return BestSolution;
```

Chapter 10

Comparison with the Mixed Integer Program

We already found out that a full satisfiability approach did not work for the Rostering Problem. Breugem [6] investigated a mixed integer approach, which did work on relatively small instances. On a larger instance the model needed too much time to find a good solution. However, Breugem did not use the exact same attractiveness constraints as we did. It could be that a mixed integer programming approach does work when attractiveness is defined as in our model. In this section we will investigate multiple ways of describing the problem as a mixed integer program. In particular, we will propose two ways to add attractiveness and prove that both describe the same problem. In the results section we will investigate the performance of both methods. Then we will compare the results with the results of the Hybrid Model to see which approach performs better in which settings. This will help us to decide whether the Hybrid Model is a valid approach to solve the Rostering Problem.

10.1 Mixed Integer Model

The feasibility constraints of the mixed integer model are created from the SAT clauses described in section 6 about the SAT model. We create the feasibility constraints by transforming the SAT clauses into cardinality constraints as described in Section 8.2.2. The fairness constraints are exactly the same as in the fairness part of the Hybrid Model of Section 8.1. Finally, attractiveness constraints must be added. For each attractiveness constraint a variable is introduced that models the violation of the constraint. These variables all have a lower bound of zero. The objective is minimizing the weighted sum of all violation variables. The weight can be different for each constraint.

10.1.1 Multiple Ways to Model Attractiveness

Each attractiveness constraint can be added in multiple different ways. We will use two approaches and investigate whether one works better than the other. The first approach uses cardinality constraints. The second approach is similar to the cuts in Section 8.3. Breugem [6] used the pricing problem of the branch and price algorithm to deal with most of the attractiveness constraints. However, constraints that covered more than one week, for example constraints about a Sunday and Monday were added explicitly to the master problem. This was done in a very similar way to the cardinality method.

10.1.2 Cardinality Constraint Approach

The first way to add attractiveness constraints is via cardinality constraints. When in the Basic Schedule a violation of an attractiveness constraint is possible, then for every pair of duties that would violate this constraint we add the following constraint.

$$x_{is} + x_{js} \leq 1 + v_{ij}^c \tag{10.1}$$

The variable v_{ij}^c will appear in the objective function, which will be a minimization. When both duties are assigned to the same sequence, v_{ij}^c must be equal to 1, else it will be 0 as it is minimized in the objective. The superscript c is used to indicate each different attractiveness constraint.

When the attractiveness constraint is about two days in different weeks it slightly changes:

$$x_{is} + x_{j(s+1)} \leq 1 + v_{ij}^c \tag{10.2}$$

$$x_{is} + x_{j(s+1-|S_r|)} \leq 1 + v_{ij}^c \tag{10.3}$$

In the objective function weights can be used to indicate the magnitude of violation. We choose these weights equal to the number of minutes that the attractiveness constraint is violated by. For example, if two duties have only twelve hours and thirty minutes in between them and the attractiveness constraint asks for thirteen hours, then v_{ij}^c appears with weight 30 in the objective.

10.1.3 Cut Approach

The second way to add attractiveness constraints is to use the v-variables to directly encode the violation. Contrary to the cardinality constraints, that were the same for each constraint, these cuts are specific for each attractiveness constraint. The general form of the cuts is:

$$\pm \sum_{i \in D_{d_1}} x_{is_1} \cdot q_i \pm \sum_{j \in D_{d_2}} x_{js_2} \cdot q_j \odot T \pm v_{sd}^c \tag{10.4}$$

Here q_i and q_j are parameters corresponding to duty i and duty j, such as the start or end time. $\odot \in \{\leq, \geq\}$. T is the threshold to determine whether there is a violation and v^c_{sd} determines the violation of the constraint. Next we will describe exactly what the cuts are we will be using.

Early duties must have increasing starting times

The first attractiveness constraint models that early duties have increasing starting times to avoid penalty. A constraint is added for each place in the Basic Schedule at which two early duties appear directly after each other in sequence s:

$$\sum_{i \in D_d} x_{is} \cdot start(i) - \sum_{j \in D_{d+1}} x_{js} \cdot start(j) \leq v^c_{sd} \tag{10.5}$$

By the feasibility constraints exactly one of the x_{is} is equal to 1, and exactly one of the x_{js} is equal to 1. If duty j starts later than duty i, then v^c_{jd} will be 0, else it will be equal to the number of hours that the second job starts later than the first. If we give all variables weight 60 in the objective this is exactly the penalty we wanted to assign. When the duties are on Sunday and Monday and it is not the last sequence of a roster group the constraint changes to:

$$\sum_{i \in D_6} x_{is} \cdot start(i) - \sum_{j \in D_0} x_{j(s+1)} \cdot start(j) \leq v^c_{sd} \tag{10.6}$$

When it is the last week of the roster group it becomes:

$$\sum_{i \in D_6} x_{is} \cdot start(i) - \sum_{j \in D_0} x_{j(s+1-|S_r|)} \cdot start(j) \leq v^c_{sd} \tag{10.7}$$

Late duties must have decreasing end times

This constraint is very similar to the last one, but now we use the end times instead of the start times. Furthermore, the signs are reversed as we now want decreasing times instead of increasing. The constraints will look as follows:

$$\sum_{i \in D_{d+1}} x_{is} \cdot end(i) - \sum_{j \in D_d} x_{js} \cdot end(j) \leq v^c_{sd} \tag{10.8}$$

$$\sum_{i \in D_0} x_{i(s+1)} \cdot end(i) - \sum_{j \in D_6} x_{js} \cdot end(j) \leq v^c_{sd} \tag{10.9}$$

$$\sum_{i \in D_0} x_{i(s+1-|S_r|)} \cdot end(i) - \sum_{j \in D_6} x_{js} \cdot end(j) \leq v^c_{sd} \tag{10.10}$$

At least 13 hours between duties

The next attractiveness constraint requires the time in between duties to be at least thirteen hours to avoid penalty. The constraint is slightly different than the previous ones as now the penalty not only depends on the start and end times, but also on the number of hours we require to be in between. In the example we choose thirteen hours, but another number could be chosen, as long as it is at least twelve. When a number smaller than twelve hours is chosen the constraint becomes redundant as twelve hours are required by the CLA.

$$\sum_{i \in D_d} x_{is} \cdot end(i) - \sum_{j \in D_{d+1}} x_{js} \cdot start(j) \geq 13 - v^c_{sd} \tag{10.11}$$

$$\sum_{i \in D_6} x_{is} \cdot end(i) - \sum_{j \in D_0} x_{j(s+1)} \cdot start(j) \geq 13 - v^c_{sd} \tag{10.12}$$

$$\sum_{i \in D_6} x_{is} \cdot end(i) - \sum_{j \in D_0} x_{j(s+1-|S_r|)} \cdot start(j) \geq 13 - v^c_{sd} \tag{10.13}$$

When the rest time is less than 13 hours v^c_{sd} will be equal to the time that the rest is shorter. A penalty higher than 60 is impossible as a pair that would get such a penalty is forbidden by the feasibility constraints. When the rest time is more than 13 hours v^c_{sd} will be equal to 0 as it is minimized.

One rest day must be at least 34 hours long

For the last attractiveness constraint we need to distinguish more cases. If the rest day is on Monday or Sunday the indices change. The constraint is added when in the Basic Schedule one single rest day appears and the day before and after have a regular duty.

$$\sum_{i \in D_d} x_{is} \cdot end(i) - \sum_{j \in D_{d+1}} x_{js} \cdot start(j) \geq 34 - v^c_{sd} \tag{10.14}$$

$$\sum_{i \in D_6} x_{is} \cdot end(i) - \sum_{j \in D_1} x_{j(s+1)} \cdot start(j) \geq 34 - v^c_{sd} \tag{10.15}$$

$$\sum_{i \in D_6} x_{is} \cdot end(i) - \sum_{j \in D_1} x_{j(s+1-|S_r|)} \cdot start(j) \geq 34 - v^c_{sd} \tag{10.16}$$

$$\sum_{i \in D_5} x_{is} \cdot end(i) - \sum_{j \in D_0} x_{j(s+1)} \cdot start(j) \geq 34 - v^c_{sd} \tag{10.17}$$

$$\sum_{i \in D_5} x_{is} \cdot end(i) - \sum_{j \in D_0} x_{j(s+1-|S_r|)} \cdot start(j) \geq 34 - v^c_{sd} \tag{10.18}$$

10.2 The Model

Now we have all the constraints that we need to describe the Rostering Problem as a mixed integer program. The MIP model looks as follows:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{c} w_c v_c & \\
\text{subject to} \quad & \sum_{s \in S} x_{is} = 1 & \forall i \in D \\
& \sum_{i \in D_d^T} x_{is} = 1 & \forall d \in [0,6], T = B_{sd} \\
& x_{is} \in \{0,1\} & \forall i \in D, \forall s \in S \\
& v_c \geq 0 & \forall c \\
& 0 \leq \epsilon \leq M & \\
& 0 \leq \delta_a \leq M_a & \forall a \\
& \text{Feasibility Constraints(8.2.2)} & \forall k \in K \\
& \text{Fairness of workload (8.3)} & \forall r \in R \\
& \text{Fairness of other attributes (8.4)} & \forall r \in R, \forall a \in A \\
& \text{Attractiveness constraints(10.1.1)} &
\end{aligned}
$$

10.3 Both Methods Describe the Same Problem

We proposed two ways to formulate attractiveness in the Rostering Problem. Now we will prove that both indeed describe the exact same problem.

Lemma 3 *The Rostering Problem where attractiveness is modeled using the cardinality method is equivalent to the Rostering Problem where attractiveness is modeled using the cuts method.*

Proof. We take the same assignments for all x_{is} variables in the cardinality method and the cuts method. If the solution is feasible for the cuts method then it is also feasible for the cardinality method as the only difference are the attractiveness constraints. The attractiveness constraints can always be satisfied by setting violation variables to one. Vice versa, if the solution is feasible for the cardinality method, it is also feasible for the cuts method. For every attractiveness constraint at most one violation variable can be one. A feasible solution for the cuts method can be found by making violation variables non-zero for each attractiveness constraint for which a violation variable in the cardinality method was forced to one.

Now we must show that the objective value of both problems is the same. Every violation variable that is forced to one in the cardinality method corresponds with violating an attractiveness constraint. But then in the cuts method there is a constraint that models this attractiveness constraint. As the assignment of x-variables was the same in both models this means that the violation variable of this cut takes on the value corresponding to the number of hours the constraint is violated by. In the objective, this is multiplied by 60. By definition this was also the weight that was given to the violation variable in the objective function of the cardinality method.

In the cuts method exactly one x-variable is equal to one in both sums of each cut. The others are all zero. Now if the value of the violation variable is greater than zero it means that the pair of x-variables that is set to one does violate the attractiveness constraint. By construction the penalty is equal to the number of minutes the constraint is violated by. But in that case in the cardinality method a constraint for this pair was added and the violation variable will be forced to one. This means that in the objective the number of minutes this constraint is violated by is added.

So the violation of an attractiveness constraint appears in the objective function in the cardinality method if and only if it appears in the objective function of the cuts method. Hence, for every assignment of x-variables the solution for both methods is the same.

□

10.4 Differences Between Both Methods

The biggest difference between both methods is the number of constraints needed. The cardinality constraint method needs a constraint at every spot in the roster for every pair that could violate the constraint at that spot. On the other hand, the cuts method only needs one constraint at each spot the constraint can be violated. This clearly are much less constraints. However, each cardinality constraint only has two variables regarding duties and one regarding violation. The cuts method has the violation variable and two sums of all duty variables on that day multiplied with a coefficient.

The question will be whether it is better to use a lot of constraints each involving a small number of variables, or a smaller number of constraints that each have a big number of variables in it. The advantage of the last is that also a lot less variables will appear in the objective.

Chapter 11

Results

The model was implemented in Python using Gurobi 9.0.2 as MIP solver. For the MAX-SAT part the RC2 solver was used, which is part of the Pysat package. It uses Glucose 3.0 as its underlying SAT solver. All experiments are performed using a a 2.4 GHz Intel Core i7-4700MQ processor.

11.1 Test instance

The instance used to test the model is an instance of crew base Amersfoort. It contains a Basic Schedule consisting of seven roster groups and a list of 310 duties with their type, starting time, end time, duty length and the attributes Repetition Within Duty, percentage A-trains, percentage aggression trains and percentage double deckers. The data contains more information on the duties, but this was not used by the model.

The 310 duties consist of 141 early duties, 154 late duties and 15 night duties. The Basic Schedule is 83 sequences long, divided among seven roster groups. The first roster group consists of eight older employees. They have on average only four duties every week instead of four and a half. Furthermore, they have a Red Weekend every two weeks instead of every three. The second roster group has fifteen members and only has early duties. The other five roster groups have twelve members. Group three, four and seven have all types of duties, while group five and six only have late and night duties. In total, there are 49 reserve days in the Basic Schedule.

We will use this Basic Schedule and list of duties to investigate the methods proposed earlier in this thesis.

11.2 Hybrid Model

11.2.1 Using the Basic Schedule

First we will look at the performance of the Hybrid Model that uses the Basic Schedule. We will run the model using different settings and with different feedback heuristics. We want to know if the algorithm performs well in the different parameter settings and if the solutions that the model gives are of high quality. This will help us decide whether the hybrid approach is a suitable way to solve the Rostering Problem.

The first thing we will vary is the amount of feasibility constraints we add to the fairness part. Adding feasibility constraints to the fairness part cannot forbid the optimal solution, as the optimal solution obviously is feasible. It does however reduce the flexibility. We will investigate whether adding more feasibility constraints influences the quality of the solutions found.

The second we can select is the objective function. Either we minimize the weighted sum of ϵ and δ_a or we only put bounds on ϵ and δ_a and use a meaningless objective. The first will take more time, but probably gives fairer solutions. The question is what the effect on attractiveness will be and if the extra time needed does not slow the model down too much. For the objective we take $\delta_a = \delta$ for all a and we give weight 50 to ϵ as fairness of working time is more important than fairness of the other criteria.

The last factor we will look at is which feedback heuristics to use. These heuristics (Section 8.3) can be used to eliminate assignments that are unlikely to appear in a good solution. Some heuristics forbid divisions aggressively, which carries the risk of forbidding also good solutions. Other heuristics are more conservative. We will investigate which (combination of) heuristics work best on the test instance.

The attractiveness constraints that are added are:

- At least thirteen hours between duties
- One single rest day is at least 34 hours
- Early duties must have increasing starting times
- Late duties must have decreasing end times

The penalty for violating the attractiveness constraints is equal to the number of minutes the constraint is violated by.

Feasibility constraints

The first observation is that feasibility constraints are needed in the fairness part because otherwise many runs result in at least one group being unsatisfiable. When a group is UNSAT, the only possible feedback is to not choose that same division again, but if feasibility constraints were added, this would have been known already. Furthermore, the heuristics cannot be used when the subproblem is UNSAT. In the end, a solution must be feasible. Therefore adding feasibility constraints cannot

throw away any feasible solution. It turns out that adding the feasibility constraints to the fairness part does not slow down the fairness part a lot. Another reason to not add feasibility constraints is that there might be less flexibility in the MAX-SAT part, but there was no indication that this was a problem. The algorithm without feasibility constraints did not produce better solutions, or equal solutions quicker.

Optimizing fairness

The second result is that optimizing fairness takes a very long time. After an hour the solver still is nowhere close to finding the optimal solution, as the gap has only improved from 95% for the initial solution at 18 seconds to 89%. Although there are only a couple of variables in the objective function, these variables depend on a huge amount of other variables. On the other side finding a feasible solution within the bounds is done much quicker, even if the bounds are very tight. The time it takes is too long to consider optimizing fairness as an option. Therefore we will only consider the alternative to use a meaningless objective and bounds to limit the violation of perfect fairness.

Feedback heuristics

Now we will compare the feedback heuristics proposed in Section 8.3. Four fairness levels will be compared to see if this influences the performance of the approach. The four levels that will be investigated are:

1. $\epsilon = 0.005, \delta = 0.1$
2. $\epsilon = 0.002, \delta = 0.05$
3. $\epsilon = 0.001, \delta = 0.05$
4. $\epsilon = 0.0005, \delta = 0.025$

We take ϵ much smaller than δ since the margin that is considered acceptable is very small. With an ϵ of 0.005 the difference between the minimum and maximum average workload per week is at most 24 minutes. In practice this might even be too much but it is interesting to see if the model performs differently when the fairness is loose compared to when the fairness is very tight. Choosing such a tight ϵ does mean we cannot ask the same for the other attributes, as no feasible solution exists anymore. Furthermore, fairness of the other attributes is less important.

For each fairness level we run the model without heuristics. Then we try each heuristic separately. Finally we combine the high weighted cuts heuristic with the others.

We will use a time limit of 900 seconds. As global threshold we will use 0. This will probably be unachievable, so the algorithm will run for fifteen minutes, or until no feasible solution can be found anymore by the fairness subalgorithm. As group violation we use 300. As weight threshold we choose 100, meaning that the heuristics are activated when two duties cause a penalty of 100 or more.

Furthermore, we will also try all methods with a time limit of 3600 seconds to see what the difference in quality is between the solutions in the first fifteen minutes and the solutions found later on in the

process. This gives an idea whether it is useful to run the Hybrid Model for a longer time, or if it suffices to run it only a short time. The results of the first three settings (1, 2, 3) are summarized in table 11.1. This table contains for each heuristic the average of all parameter settings. The results for setting 4 are presented separately. The full results can be found in Appendix B.

	Average Best Solution 900s	Average Best Solution 3600s
No Heuristics	1954	1927
High Weighted Pairs	2061	1975
High Weighted Groups	1996	1913
High Weighted Days	2069	1918
High Weighted Cuts	1659	1586
HWP + HWC	1697	1547
HWG + HWC	1672	1666
HWD + HWC	1987	1452

Table 11.1: Summarized Results

If we look at the heuristics separately we see that the high weighted cuts method is the only one that is able to find a much better solution than the solution without the heuristics. This is the case both after fifteen minutes and after an hour. In the full results (Appendix B) we can also see that this heuristic is the only one that outperforms the solution without heuristics for every single parameter setting. Furthermore, for every fairness level the cuts method found the solution with the lowest penalty. Therefore we can conclude that this heuristic does perform best and it is useful to use it, since it clearly improves the quality of the solution.

Another observation about the cuts method is that it finds its best solution late. Furthermore, when doing the runs we observed that the solutions later on in the solving process were on average better than those in the beginning. The cuts method was able to prevent the model from finding very bad solutions later on in the time interval, where the other methods kept giving very bad solutions in some iterations. This is another indication that the cuts method does improve the solutions found.

The other heuristics create solutions that have a very similar quality as the solution without heuristics. Furthermore, they do outperform the solution for some settings, while they have a worse best solution for other settings. This means that these heuristics have little impact on the solution. There also does not seem to be a clear difference in performance for the different fairness levels.

The question remains why the cuts method does perform so much better than the other heuristics. One reason is that it accomplishes much more, as it forbids many assignments at the same time, since every combination that is strictly worse than the one we had will be forbidden. The other heuristics only forbid a very small amount of assignments as they only say something about the specific duties that did cause the penalty. The fact the cuts heuristic forbids many more divisions is also illustrated by the fact that it reached infeasibility for nearly all runs within the hour time limit. This means the model has forbidden so many divisions that no feasible division exist anymore. The other heuristics all ran for the full 3600 seconds in every setting.

We can combine the cuts heuristic with the other ones, but doing this does not improve the quality

of the solution. However, it also does not make it clearly worse. If we look at the results in more detail we see that adding a heuristic to the cuts heuristic did improve the solution four times, and made it worse five times. This again is a sign that the other heuristic have little impact.

We see that the average best solution after 3600 seconds is better than after 900 seconds, which is obvious as the best solution found cannot become worse. However, the difference is not very large for most settings. Therefore, the Hybrid Model could still be useful when we would only have a short amount of time.

Extremely tight fairness bounds

Setting 4 contains solutions that are extremely fair. In table 11.2 the results are summarized.

	Best Solution 900s	Best Solution 3600s
No Heuristics	3125	2366
High Weighted Pairs	2951	1833
High Weighted Groups	2856	2856
High Weighted Days	2702	2702
High Weighted Cuts	2480	2092
HWP + HWC	2270	1731
HWG + HWC	2009	2009
HWD + HWC	3174	2771

Table 11.2: Extreme Fairness

We clearly see that the penalties are much higher, which is not strange as the solution space has shrinked considerably. In this setting the fairness part takes much more time than for the other settings. With the 900 seconds time limit the solver is only able to find four or five solutions. Therefore the quality of the solutions is also impacted a lot by chance. For this fairness setting it is less clear that the cuts heuristic is best. A possible explanation is that the impact of the other heuristics increases relative to the cuts heuristic, because there are a lot less feasible solutions.

For setting 1, 2 and 3 the cuts heuristic and combinations with it reached infeasibility within the first hour. For the extreme fair setting this did not happen. We did run these settings until it reached infeasibility and present the results in table 11.3.

	Best Solution	Time	Time to infeasibility
High Weighted Cuts	1500	21910.0	22943.2
HWP + HWC	1731	2867.4	3490.3
HWG + HWC	1731	3685.0	4658.1
HWD + HWC	1545	22471.9	25945.1

Table 11.3: $\epsilon = 0.0005, \delta = 0.025$, Time limit $= \infty$

We see that this clearly improves the solution, as the quality now is similar to the other fairness

settings. Furthermore, we see that the time to infeasibility is much shorter for the combinations with the high weighted pairs and high weighted groups heuristics. This shows that indeed the impact of these heuristics increases.

More demanding attractiveness

We change the attractiveness constraint about the rest time between two consecutive duties from thirteen to sixteen hours to see if the model is capable of finding good solutions when more demanding attractiveness constraints are added. The threshold for groups is changed to 1000 as 300 is not realistic anymore.

When doing this we see a clear drop in performance, as the MAX-SAT part takes much more time. Most groups still solve very fast, but some groups take a lot of time. The reason is that now many violations will happen and it is difficult for the solver to determine which ones. The penalties also become very high. When no heuristics were used only five solutions were found within the time limit of an hour. The best was 7463.

When we use the cuts heuristic we must adapt the threshold, because if we keep it at 100 the model becomes infeasible after only three iterations. The reason is that now the rest time constraint also generates cuts. Before, this was not the case, since the constraint could be violated with at most 60. This means that there could be cuts for multiple different constraints at the same spot in the roster, which leads quickly to infeasibility. When we increase the threshold to 200, the model still reaches infeasibility after only 5 iterations, but the best solution found is 6693, which is clearly better than without the heuristic.

Conclusions

From these observations, choosing only the high weighted cuts heuristic seems to be the best choice, as this is the only one that consistently outperforms the solution without heuristics. This heuristic can be combined with other heuristics, but this does not improve the solution consistently.

When extremely fair solutions are required the model can still find solutions. However, it takes more time per iteration. Therefore the improvements made later on in the time window are much more significant than for the less fair settings. There, the improvements made after fifteen minutes were very small in most cases.

When we have very demanding attractiveness constraints, the MAX-SAT problem takes much longer to solve. Also the cuts heuristic becomes too demanding, which leads to infeasibility within only a few iterations. The model still finds solutions within a short amount of time, but the ability of the model to improve the solutions has become significantly worse.

11.2.2 Without the Basic Schedule

We also proposed a model in which the Basic Schedule was not used anymore in the attractiveness part. This way, the duties can be arranged freely by the MAX-SAT solver, as long as the solution is feasible with respect to the CLA. The hope is we can reduce the attractiveness penalty, because we are not limited to solutions that fit the Basic Schedule anymore. A downside is that we now need

many more constraints to describe the problem as we must add the CLA rules that were implicitly met by the Basic Schedule, but also attractiveness constraints about patterns of different types of duties that were not needed previously.

In the first run we only add feasibility constraints. This means we remove the constraint that the Basic Schedule must be followed and add the constraints specified in Section 6.2. We keep the attractiveness constraints we already had about early and late duties, single rest days and rest periods. For all the experiments we will use a fairness level of $\epsilon = 0.002$ and $\delta = 0.05$.

No additional attractiveness

In this case a solution with 0 penalty is found in 52.1 seconds. However, if we take a look at the solution we see that such a solution would never be accepted. A lot of very long rest periods occur. In total seven times a period of four or more consecutive rest days appears. This means that also a lot of long working periods must occur, while shorter periods are preferred. Furthermore, the pattern rest - duty - rest appears fourteen times. Finally, a lot of changes between types happen without a rest day in between, especially a transition from early to late. This is because there is nothing that stops the model from doing this, as there is almost always at least thirteen hours between the end of an early duty and the start of a late duty. Also, transitions from late to early often happen via a reserve shift, which is in two ways undesirable. For the employees this is bad because they have no real chance to adapt their rhythm. For NS this is bad because the time window the employee could serve as a reserve is very small.

No rest - duty - rest and rest day between transitioning

Now we will add the attractiveness constraints that try to forbid the rest - duty - rest pattern and those that request a rest day when there is a transition from early to late or night or vice versa. As penalty we use 15. Now, roster group 0 does not find a solution anymore within 900 seconds. The problem is the attractiveness constraint that tries to avoid rest - duty - rest. Since group 0 consists of older employees, they have one additional rest day every week, which makes it more difficult to avoid single duties. If we remove this constraint only for this group the model finds a solution with penalty 58 in 94.1 seconds. The solution clearly improved, but still it would not be accepted. There are still too many long working periods of five, six or seven days. Forbidding a single duty did not solve this issue.

No rest - duty - rest, rest day between transitioning and rest every six days

Therefore, we now add a constraint that asks for at least one rest day every six days. Again a solution with penalty 58 is found. This took the model 81.2 seconds. By adding this attractiveness constraint we did solve the problem that many long working periods occurred, but we created another problem. Now the solution has 83 single rest days. This is not desirable, especially if there is a late or night duty before the rest day, or an early duty directly after it.

No rest - duty - rest, rest day between transitioning, rest every six days and single rest days

Now we will add a constraint that penalizes early duties after a single rest day and late or night duties before a single rest day. This time no solution is found within 900 seconds. Roster group 0, 3 and 4 are not able to find a solution fast. We already concluded that this last constraint contradicts the others in a way, because single rest days are very useful to create a short cyclic roster.

No rest - duty - rest, rest day between transitioning, rest every six days and at most three consecutive rest days

Another way of trying to avoid long rest periods would be penalizing solutions that have more than three consecutive rest days instead of requiring a rest day every six days. Again, roster group 0 is the bottle neck. When this group is not considered the model finds a solution with penalty 58 in 84.2 seconds. The reason roster group 0 is difficult is most likely because there is only one reserve shift. The reserve shifts create flexibility as there are hardly any constraints regarding them. Therefore, they can be used to fill up the last spots.

The solution for the other groups does not contain rest periods of more than three days anymore. However, it did not prevent the model from scheduling six or seven days without a rest day multiple times. Therefore, ideally we would want both of the constraints, but this makes the problem even harder. In the next experiment we will look if adding both is possible if we remove the attractiveness constraints that require early duties to have increasing starting times and late duties to have decreasing end times.

Remove attractiveness about consecutive early and consecutive late duties

Now we do get a solution very fast. In only 50.0 seconds a solution with penalty zero is found. We even get a solution without penalty in 50.4 seconds if we add an attractiveness constraint to ask for at least one rest day every five days instead of 6.

Reserves

In section 8.5.1 we mentioned that there are most probably multiple rosters that have the same penalty, as a lot of changes do not influence the attractiveness penalties. Therefore we suggested letting the solver output multiple solutions and pick the one for which the reserves could be filled in such that all duties are covered by at least one reserve shift. It turned out that for this dataset nearly all solutions had reserve shifts such that all duties are covered. However, if there is a Basic Schedule that includes less reserve shifts it could still be useful to use this approach to decide which roster will be the final solution.

Conclusion

We have seen that the model without the Basic Schedule is able to find solutions when only a limited amount of attractiveness is considered. However, when all attractiveness constraints we would like to add are added the solver is not able to find a solution within the time limit for multiple groups. When it is not possible anymore to find a solution with no or very low penalty the solver seems to have difficulties trying to find out which violations to take. This is not strange as there are a very large amount of options, since we do not have a Basic Schedule to restrict the number of feasible

assignments.

Suggestions for improvement

When not all attractiveness constraints are added the model produces solutions with very low penalty. A suggestion to improve would be using the model without the original attractiveness constraints to make a Basic Schedule. The fairness part of the model could be adapted such that is does not need a Basic Schedule, but only divides the duties fairly among the groups. We can use the model without the original attractiveness constraints to create a solution. From this solution we can make a Basic Schedule by choosing for each day the type of duty that was assigned to that day in the solution. Then the original Hybrid Model or a MIP could be used on this new Basic Schedule. It would be interesting to see if this would improve the overall quality of the solutions.

Another option to solve the problem that no solution is found within the time limit could be using a different solver. The RC2 MAX-SAT solver is UNSAT based, meaning it starts with the goal of violating no soft-clauses and every iteration it allows itself to violate more soft clauses. This way, every iteration we get either UNSAT or an optimal solution. A SAT based MAX-SAT solver finds a solution, then calculates the penalty and in the next iteration it tries to improve the solution. The advantage is that there already are solutions before the optimum is reached. At this point in time, the best MAX-SAT solvers are all UNSAT based, but for this purpose a SAT based solver might be more suitable because we could stop the solver and at least have a solution. It is difficult to say what would be the quality of that solution.

11.3 MIP Model

11.3.1 Performance of the Different Methods

In this section we look at the performance of the MIP model. We provided two ways of adding attractiveness to the model and will try both methods on different fairness levels. As base case we will use $\epsilon = 0.002$ and $\delta = 0.05$. We will look what happens when either one of them is decreased, and when both are chosen very tight.

Furthermore, we will make changes in attractiveness constraints to make them more demanding. In one experiment we will increase the number of hours between two consecutive duties to sixteen hours. In another we increase the number of hours that one single rest day should last from 34 to 40 hours. The goal of these experiments is to see whether one method scales better than the other with more demanding attractiveness constraints.

By doing these experiments we hope to see if the performance is most influenced by the fairness level, or by the severity of the attractiveness constraints. Furthermore, we try to find out whether using a MIP is a suitable option for solving the Rostering Problem as we formulated it.

We will illustrate the differences between the methods using different parameter settings. We will use a time limit of 3600 seconds. We look when the model finds its first solution. Furthermore we will look at the penalty of the final solution, which is either the optimal solution or the solution found at the time limit. We keep track of the time on which the final solution has been found to see if the model still made improvements towards the end of the time. Finally, we state the best bound the solver found at the time limit. The gap between the final solution and the best bound gives an indication on how close to optimality the solver came. If the solver found the optimal solution, the time it took is in brackets after the best bound.

Base case

As a base case we use the parameter setting $\epsilon = 0.002$ and $\delta = 0.05$.

	First Solution	Final solution	Time final solution	Best bound
Cardinality	118 seconds	906	2125 seconds	136.3.0
Cuts	52 seconds	899	153 seconds	899 (595 seconds)

Table 11.4: $\epsilon = 0.002$ and $\delta = 0.05$

In table 11.4 we clearly see that the cuts perform better for this setting. It finds a solution faster and even finds the optimal solution very quickly. It took the solver 153 seconds to find it, but only after 595 seconds it found out the solution was optimal. The cardinality method does not find the optimal solution within an hour. It finishes with a penalty of 906 and a best bound of 136.3. This shows that the cardinality method is not able to find tight lower bounds within a reasonable time.

Tighter fairness bounds for attributes

Now we will look what happens when more fairness is asked for the attributes other than workload.

We will keep ϵ at 0.002 but change δ to 0.025. This means less assignments are allowed. The results are shown in table 11.5 below.

	First Solution	Final solution	Time final solution	Best bound
Cardinality	2508 seconds	1022	3275 seconds	757.8
Cuts	869 seconds	899	3252 seconds	899 (3252 seconds)

Table 11.5: $\epsilon = 0.002$ and $\delta = 0.025$

Now the cuts method finds the optimal solution in 3252 seconds. The cardinality method does find a solution, but it takes much longer and the quality of the solution is worse.

An interesting fact is that the objective of the optimal solution is exactly the same than when ϵ was equal to 0.05. However, the solution is different. This time the model needs much more time to find it. This is due to the fact that the solution space is smaller. The optimal solution in the first experiment was not feasible anymore as it was not between the fairness bounds. Apparently, it is possible to find another solution with the same penalty that is between the tighter fairness bounds. It takes much longer to find it because it is more difficult to find an incumbent when the solution space is smaller. This means that a smaller part of the branch and bound tree can be cut off, and therefore many more nodes must be explored. This can be seen in the statistics of the solver. When the bounds where tightened the solver explored 241097 nodes. When the looser bounds were used only 10554 nodes were explored. This factor is almost exactly the factor of the difference in time and therefore explains the slower solving time.

Tighter fairness bound for the workload

Now we set δ back to 0.05 and change ϵ to 0.001 to make the fairness constraint for the average workload per week tighter.

	First Solution	Final solution	Time final solution	Best bound
Cardinality	134 seconds	916	3505 seconds	131.0
Cuts	39 seconds	899	206 seconds	899 (1102 seconds)

Table 11.6: $\epsilon = 0.001$ and $\delta = 0.05$

In table 11.6 above we see that the cuts method is able to find the optimal solution. Interesting to see is that an initial solution was found quicker than in the base case. The objective of the optimal solution is 899, which again is the same value but a different solution. This is unexpected, as we would expect a worse solution when the constraints are tighter. The explanation is that there are a lot of places where duties could be interchanged without affecting the attractiveness constraints. Making changes in these duties could affect the fairness, because the attributes of the duties and the length of the duties could be different. In this setting it took the solver 206 seconds to find the optimal solution, which is longer than in the base case. In the end it took the solver 1102 seconds to conclude that the solution was optimal. This is also longer than in the base case. The fact that we found an initial solution faster was probably due to luck, as the other statistics were slower than for the base case.

The cardinality method needed a little bit more time to find a first solution compared to the base case, but the difference is small. It ended with a solution of 916 and a best bound of 131.0. Both are worse than for the base case, which is expected as the solution space became smaller.

The differences are a lot smaller than when we changed δ, which is probably due to the fact that δ restricts four attributes, while ϵ only restricts one.

Extremely tight fairness bounds

In the next runs we use extremely strict fairness bounds, namely an ϵ of 0.0005 and a δ of 0.025. In this case both methods do not find a solution. We increase the time limit to 9000 seconds to see if we can eventually find a solution, but again both methods do not find a single solution. Therefore, we can conclude that using only a mixed integer program does not work anymore as the fairness requirements are very tight. The hybrid method however did find a solution for this setting as we showed in the previous section. Therefore using the Hybrid Model would be a valid option for this setting.

More demanding attractiveness for rest times

Next we will look what happens when an attractiveness constraint becomes more demanding. We will set ϵ back to 0.002 and δ to 0.05, so we can compare with the base case. We will change the rest time attractiveness constraint (1). Instead of thirteen, we will ask sixteen hours. This means that the cardinality method will get much more constraints, as there are now a lot more pairs at each spot that would violate the constraint. The cuts method will have the same number of constraints as before, but there will be a lot more possibilities for violation variables to become non-zero. We already saw that the Hybrid Model had a significant drop in performance when making this change. The question is if this is also the case for the MIP. As time limit we again use 3600 seconds. We present the results in table 11.7.

	First Solution	Final solution	Time final solution	Best bound
Cardinality	2326 seconds	9478	3551 seconds	1542.9
Cuts	124 seconds	3479	2762 seconds	3374.0

Table 11.7: $\epsilon = 0.002$ and $\delta = 0.05$

We see that this change causes a very big difference in the time to find the first solution, as the cuts method did find a solution in 124 seconds, while the cardinality method needed as many as 2326 seconds. Furthermore the solution that is found by the cuts method approaches the optimal solution within the time limit as the gap is only 3.0%. The cardinality method does find a solution eventually, but this is a very bad one. Furthermore, the solver has difficulty improving the solution. After 3600 seconds it found a solution with penalty 9478, which is not close to the optimal solution at all. Also the lower bound was again a lot less tight than for the cuts methods. Even the initial solution found by the cuts method after 124 seconds was with a penalty of 4237 way better than the final solution after an hour for the cardinality method.

We clearly see that the cuts method scales better in attractiveness. This was expected as making more demanding attractiveness requirements does not change the number of constraints and

variables in the cuts method, but it does add a lot of contraints and variables in the cardinality method. Therefore the cuts method only needs a bit more than twice as much time as before to find a feasible solution, while the cardinality method needs more than twenty times as many seconds.

The cardinality method has the same issues as the Hybrid Model, as it has difficulties finding and improving the solution. For the cuts method the drop in performance is a lot smaller. This means that the MIP with the cuts method is the best option when we want to add very demanding attractiveness constraints.

More demanding attractiveness for single rest days

To verify the conclusion that the cuts method scales better with more demanding attractiveness we will now make another attractiveness constraint more demanding. Instead of 34 we will ask for at least 40 hours for one single rest day. The time between two duties on consecutive days will be set back to thirteen hours. ϵ and δ will stay at 0.002 and 0.05 respectively.

	First Solution	Final solution	Time final solution	Best bound
Cardinality	204 seconds	1176	1353 seconds	841.0
Cuts	95 seconds	1148	117 seconds	1144.4

Table 11.8: $\epsilon = 0.002$ and $\delta = 0.05$

In table 11.8 above we can see that the cuts method finds a solution in 95 seconds, which is an increase of 54 percent compared to the base case. After 117 seconds it finds a solution with a penalty of 1148. After an hour, this is still the best solution and the gap has been reduced to 0.3%. This means that within two minutes the model was able to find a solution that was either optimal or extremely close to it.

The cardinality method finds a solution in 204 seconds, which is an increase of 57 percent compared to when we asked for only 34 hours. This increase is slightly more than the cuts method had. It does not reach the optimal solution within an hour and has difficulties improving the solution. Its final solution was found after 1353 seconds, which means that in the final 2247 seconds no incumbent has been found. In that time the best bound has increased from 454.1 to 841.0. The differences are less extreme than for the other attractiveness constraint. This is logical as this change is much less demanding. The constraint is only added for pairs that are before and after a single rest day. The other constraint was about all places where two consecutive duties appear in the Basic Schedule, which clearly happens way more times.

Conclusions

In general we can say that the cuts method drastically outperforms the cardinality method. It finds a solution faster in every single setting. Furthermore, the solution at the time limit is better each time. The cuts method even finds the optimal solution in some cases. The main reason the cuts method performs better is that the number of attractiveness constraints is much lower. This also means that the number of extra variables needed to model attractiveness is much lower. These extra constraints and variables make it more difficult for the cardinality method to find an integer solution, which causes the model to slow down considerably.

11.3.2 Using a Start Solution

In the previous section saw that for extremely tight fairness criteria the MIP-model was not able to find any solution in 9000 seconds, while the Hybrid Model found multiple solutions within fifteen minutes. We can use the solution the hybrid method finds as a start solution in the MIP-model. This way, it starts with an incumbent and a lot of the branch and bound tree can be pruned. Now the question is whether the MIP is able to improve the solution.

We look at the setting where $\epsilon = 0.0005$ and $\delta = 0.025$. As a start solution we used the solution with penalty 2480 generated with the cut heuristic in the Hybrid Model. It takes 123 seconds to improve the solution for the first time. The incumbent becomes 2457. The solver is able to slowly improve the solution. After 9000 seconds the solver was able to find a solution with penalty 1035, while without the start solution no solution had been found at all during 9000 seconds. The lower bound is 876.4. The solution found is a lot better than the solutions found by the Hybrid Model, but without the solution of the Hybrid Model we could not have found it.

When we give the solver the first solution found by the Hybrid Model, which has a penalty of 3174, it takes the model 295 seconds to find a solution which was better than the 2480 found by the Hybrid Model itself. However, it took the Hybrid Model 641 seconds to find the solution with penalty 2480, while the initial solution was found after only 70 seconds. This means that in this case, using the initial solution could be better. However, for other settings a significant improvement of the initial solution might be found quicker.

Different settings and start solutions

We can also use this strategy for other settings. For example the $\epsilon = 0.002$ and $\delta = 0.025$ case. Here it took the solver 869 seconds with the cuts method to find a solution. We first let run the Hybrid Model for 200 seconds. We find a solution with penalty 2381 and use this as start solution. Then, the solver is able to find the optimal solution in 2718 seconds, instead of 3252 before. This is a decrease of 534 seconds, while the Hybrid Model runs only for 200 seconds. Also the cardinality method improves when this start solution is given to the solver. Then it is able to find a solution with penalty 989, which is better than 1022 when the start solution was not used. However, interestingly, the lower bound was only 309.0, which is a lot less than when the start solution was not used. Also, after 3400 seconds, the solution found was 1033, which was not better than without the start solution after 3600 seconds, meaning that first running the Hybrid Model for 200 seconds and then the MIP for 3400 seconds with the start solution was worse than running the MIP for 3600 seconds without a start solution.

When we let the Hybrid Model run for 400 seconds we get a solution with penalty 1882, which is a clearly better start solution. The cuts method now only needs 1236 seconds to solve the problem and find the optimum. This means that including the 400 seconds for the hybrid method combining the approaches speeds up the solving process with nearly 50 percent. Of course, this is no guarantee as there are no guarantees with the Hybrid Model, but when the MIP-solver takes very long to find an initial solution it is very likely to be beneficial to use the Hybrid Model to get a decent solution as start.

When we use the cardinality method and the start solution of 1882 the model after 3200 seconds

ends with a solution with penalty 1282. So running the Hybrid Model for 400 seconds and then the MIP model for 3200 seconds is worse than running the MIP for 3600 seconds without a start solution. After 3600 seconds the solution has been improved to 1259. Interestingly, this is also worse than when no start solution was used. Also the lower bound is only 101.5, which is a lot lower than when the start solution was not used. Another interesting observation is that when we gave a better start solution to the MIP with the cardinality method, the final solution became worse. This suggests that a start solution negatively impacts the cardinality method, unless it was not able to find any solution by itself.

Getting a start solution

Another way to get an initial solution would be to first run the MIP without attractiveness constraints and use this as a start solution. However, this is worse than picking the first solution of the Hybrid Model. The Hybrid Model also takes a solution without attractiveness, but then it optimizes attractiveness locally within the groups. We did conclude that having a better starting solution could improve the performance. Therefore this shows that using the Hybrid Model as prepossessing step is useful to speed up difficult instances.

11.4 Finding the Optimal Solution with the Hybrid Model

When the Hybrid Model is ran without heuristics, group threshold infinity and global threshold zero the model should return the optimal solution eventually. For the base case we know that the model has an optimal solution with penalty 899. We will now try to find this solution with the Hybrid Model.

However, after running 72 hours, the model has only found a best solution with penalty 1684, which was found after 49 hours and 23 minutes. This shows that there are an enormous amount of solutions, and that only a small part of the solutions has a very low penalty. This explains why the high weighted days, high weighted pairs and high weighted groups heuristics do not seem to improve the performance, as only forbidding a very small amount of bad solution accomplishes not much. However, if the heuristic at some point accidentally forbids some duties that could give a good solution this might be very costly. The cuts heuristic does remove many more solutions and therefore does have a significant impact.

With the help of the MIP we know that the optimal solution has 899 penalty points. Furthermore, we can extract that the highest violation yields a penalty of 206 for the late duties constraint. The other constraints have at most a violation of 51 minutes. Now we add the cuts heuristic with a threshold value of 206 for the late duties and 51 for the other constraints. As group threshold we take 445 for group 4 and 106 for the other groups. This means that we do not throw away the optimal solution, but we do cut off a very large part of the solution space.

Still the model does not find the optimal solution in 20 hours. This shows that even though we calibrated the heuristics extremely well it still will take the model a very long time to find the best solutions. The best solution found was, with a penalty of 1206, the best solution we did find with the Hybrid Model. The reason it still takes such a long time is that the cuts, and mainly those for the late duties, only cut of solutions for which the violation of constraints was higher than

the highest violation that appeared for this constraint in the optimal solution. Furthermore, the heuristics do not influence the number of violations that occur for a group. Therefore, a lot of the times we see that a group has many more small violations than it has in the optimal solution. We do however see that the quality of the solutions that are found during the solving process is a lot higher than with other thresholds, which is logical as we cut off more bad solutions.

From these findings we can conclude that the model could potentially find the optimal solution or a solution that is very close to the optimum, but either it takes very long or we have to get extremely lucky as there are a huge amount of solutions and only a very small amount of them have a very low penalty.

Chapter 12

Conclusion and Discussion

In this thesis we have seen that the Rostering Problem is an NP-Complete problem. Many attempts have been done in the literature to find a good way of solving the problem, or variants of it. In the case of NS Breugem [6] made a branch and price model that was able to solve the problem for relatively small instances. He proved that there is a trade off between fairness and attractiveness.

In this thesis we investigated the possibilities of using a satisfiability based approach to solve the problem. We found a way to formulate the feasibility constraints as a SAT formula. Then we looked at multiple ways to formulate fairness constraints using SAT clauses. We found out that it was possible to do this using binary decision diagrams, sorters or adders. However, all three methods had clear disadvantages and made the model unsolvable within reasonable time. Therefore this way of adding fairness is not suitable to solve the Rostering Problem.

We proposed a Hybrid Model that uses mixed integer programming to divide the duties between the groups, and a MAX-SAT model to roster the duties as attractive as possible within the roster group they were assigned to. Then the model gives feedback to the mixed integer model to restrict it. We explored multiple ways of giving this feedback.

We concluded that it is best to add all feasibility constraints to the first part of the model. We thought this might lead to less attractive solutions as we could lose flexibility, but this turned out to be no problem. When the feasibility constraints were not added, many times one or more groups became UNSAT in the MAX-SAT part, which leads to lost time. Furthermore, we concluded that we should not optimize fairness, but instead only use the bounds to achieve fairness. Optimizing fairness takes a very long time which causes the model to slow down extremely.

We looked at four different heuristics to improve the performance of the algorithm and found out that the high weighted cuts heuristic did consistently increase the quality of the solution. However, the other three heuristics did not outperform the model without heuristics on a consistent basis. Also combining the cuts heuristic with another heuristic did not turn out to be better than only using the high weighted cuts heuristic. When more demanding attractiveness constraints are added,

the performance of the model drops. It needs more time to find solutions and the potential of the cuts heuristic decreases, because infeasibility is reached in a small amount of iterations. Finding the optimal solution with the Hybrid Model is possible, but very unlikely as the number of possible solutions is enormous. Even when the heuristics are added with nearly perfect parameter settings the Hybrid Model did not find the optimal solution within twenty hours.

When the Basic Schedule is not used in the attractiveness part the full model is too difficult for the MAX-SAT solver, as it happens regularly that groups take an extremely long time to solve. When not all attractiveness constraints are added, the model was able to find solutions with extremely low penalties, but since not all attractiveness was added these solutions are not useful in practice. A suggestion for improvement was done, namely using the model without Basic Schedule to create a Basic Schedule and then use the Hybrid Model or a MIP to then fill in this new Basic Schedule as attractive as possible. Furthermore, it would be interesting to look at the possibility to use a SAT based MAX-SAT solver for this problem, so it will not happen as often that a group does not find any solution.

Additionally, we looked at a mixed integer program that modeled fairness and attractiveness exactly how we defined it. We proposed the cardinality method and the cuts method to add attractiveness and found out that the cuts method worked much better. The bottleneck for the cardinality method is the number of constraints and variables, which is much higher than with the cuts method. The fact that the constraints in the cuts method each contain many variables compared to only three for the cardinality method did not outweigh this difference, as the cuts method was better in every setting. The mixed integer model did work fine, especially when the cuts method was used to model attractiveness. However, when a very high level of fairness was required also the cuts method was not able to find a single solution within hours. The Hybrid Model was able to find solutions for this setting within only a couple of minutes. We could use the solution of the Hybrid Model as a start solution for the MIP-model. We saw that this clearly improves the performance of the MIP-model, especially in the settings where the MIP-model needed a long time to find an initial solution.

At the start of this thesis we made the assumption that we could quantify attractiveness and that it is the same for everyone. In practice this is more complicated. We can ask ourselves if a violation of 100 minutes is really ten times as bad as a violation of ten minutes. Breugem [6] defined attractiveness differently. For example, he chose to use a fixed penalty, but then a violation of one minute counts equal to a violation of 100 minutes. As a result our model will prefer a solution with a lot of small violations, while Breugem's model will prefer a solution with fewer, but possibly very high violations. Furthermore, using fixed penalties would mean that the heuristics as we defined them would not be useful anymore.

In practice everyone has his own attractiveness constraints and perception of the badness of certain violations. It is impossible, both manually and with the help of a model, to adhere to all these preferences and to make a roster that everyone finds attractive. This makes it very difficult to determine what an optimal roster would look like.

Breugem's model had not the same constraints to define attractiveness. As we have seen, the performance of the model can heavily depend on the formulation and the formulation depends on the definition of attractiveness that has been chosen. Therefore, a different set of attractiveness constraints could lead to another performance level of the algorithm.

Our suggestion for someone that wants to make fair and attractive rosters is to first think carefully about the definition of attractiveness and the meaning of violations of attractiveness constraints. According to these definitions a proper model must be chosen.

The goal of this thesis was to investigate whether it is useful to use a satisfiability approach to model the Rostering Problem. We have seen that the fairness requirements make that a full SAT approach does not work, as doing additions in SAT is extremely expensive. However, this does not mean that Boolean Satisfiability cannot be used to solve the Rostering Problem. The hybrid approach that combines MIP and SAT techniques finds solutions very quickly, even when a very high level of fairness is required. However the quality of the solutions with respect to attractiveness is not as good as when a MIP is used, especially when attractiveness is modeled via the cuts formulation in the MIP. When extremely fair solutions are required the MIP does not find a solution within reasonable time. In that case the hybrid approach is a valid alternative as it is able to find good solutions very fast.

Bibliography

[1] Erwin Abbink et al. "Reinventing Crew Scheduling at Netherlands Railways". In: *Interfaces* 35 (Oct. 2005), pp. 393–401. DOI: 10.1287/inte.1050.0158.

[2] Erwin Abbink et al. "Reinventing Crew Scheduling at Netherlands Railways". In: *Interfaces* 35 (Oct. 2005), pp. 393–401. DOI: 10.1287/inte.1050.0158.

[3] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. "A Translation of Pseudo Boolean Constraints to SAT". In: *JSAT* 2 (Mar. 2006), pp. 191–200. DOI: 10.3233/SAT190021.

[4] Dimitris Bertsimas, Vivek F. Farias, and Nikolaos Trichakis. "The Price of Fairness". In: *Oper. Res.* 59.1 (Jan. 2011), pp. 17–31. ISSN: 0030-364X. DOI: 10.1287/opre.1100.0865. URL: https://doi.org/10.1287/opre.1100.0865.

[5] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*. Vol. 185. IOS press, 2009.

[6] Thomas Breugem. "Crew Planning at Netherlands Railways: Improving Fairness, Attractiveness, and Efficiency". PhD thesis. C, Jan. 2020. URL: http://hdl.handle.net/1765/124016.

[7] Alberto Caprara et al. "Algorithms for Railway Crew Management". In: *Mathematical Programming* 79 (Feb. 2000). DOI: 10.1007/BF02614314.

[8] Martin Davis, George Logemann, and Donald Loveland. "A Machine Program for Theorem-Proving". In: *Commun. ACM* 5.7 (July 1962), pp. 394–397. ISSN: 0001-0782. DOI: 10.1145/368273.368557. URL: https://doi.org/10.1145/368273.368557.

[9] Steven Den Hartog. "On the Complexity of Nurse Scheduling Problems". In: (Apr. 2016).

[10] N. Eén and Niklas Sörensson. "Translating Pseudo-Boolean Constraints into SAT". In: *J. Satisf. Boolean Model. Comput.* 2 (2006), pp. 1–26.

[11] Niklas Eén and Niklas Sörensson. "An Extensible SAT-solver". In: *Theory and Applications of Satisfiability Testing*. Ed. by Enrico Giunchiglia and Armando Tacchella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518. ISBN: 978-3-540-24605-3.

[12] Ehud Kalai and Meir Smorodinsky. "Other Solutions to Nash's Bargaining Problem". In: *Econometrica* 43.3 (May 1975), pp. 513–518. URL: https://ideas.repec.org/a/ecm/emetrp/v43y1975i3p513-18.html.

[13] Niklas Kohl and Stefan Karisch. "Airline Crew Rostering: Problem Types, Modeling, and Optimization". In: *Annals OR* 127 (Mar. 2004), pp. 223–257. DOI: 10.1023/B:ANOR.0000019091.54417.ca.

[14] Simon Martin et al. “Cooperative search for fair nurse rosters”. In: *Expert Systems with Applications* 40.16 (2013), pp. 6674–6683. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2013.06.019. URL: https://www.sciencedirect.com/science/article/pii/S0957417413003990.

[15] John Nash. “The Bargaining Problem”. In: *Econometrica* 18.2 (Apr. 1950), pp. 155–162. URL: https://ideas.repec.org/a/ecm/emetrp/v18y1950i2p155-162.html.

Appendix A

Encoding Linear Constraints

Een and Sorensen [10] describe three ways of encoding linear constraints in SAT clauses:

- Binary Decision Diagrams
- Adders
- Sorters

These three methods are explained in more detail in this appendix.

A.1 Binary Decision Diagrams

A binary decision diagram is a rooted directed acyclic graph with two terminal nodes, one representing the outcome False, the other the outcome True. The remaining nodes are decision nodes that all represent a Boolean variable. All decision nodes have two children. The first is called the high child, the other the low child. The edge to the high child represents a True assignment, while the edge to the low child represents a False assignment. Then a path from the root to one of the terminal nodes represents an assignment and an outcome True or False following this assignment. It might happen that not all variables appear on the path. In that case the value of the constraint is set, regardless of the values of the variables that did not appear on the path. A binary decision diagram is ordered if the variables appear in the same order in any path from root to terminal node. The order in which the variables appear is important for the size of the diagram. A reasonable choice is to order the variables on size of their coefficient, as variables with a high coefficient are more likely to play a factor in whether the constraint is violated or not. This is also the choice that Een and Sorensen made for the Minisat+ solver [10]. An example of a BDD for the constraint $2a + 3b + 4c \geq 5$ is shown in figure A.1.

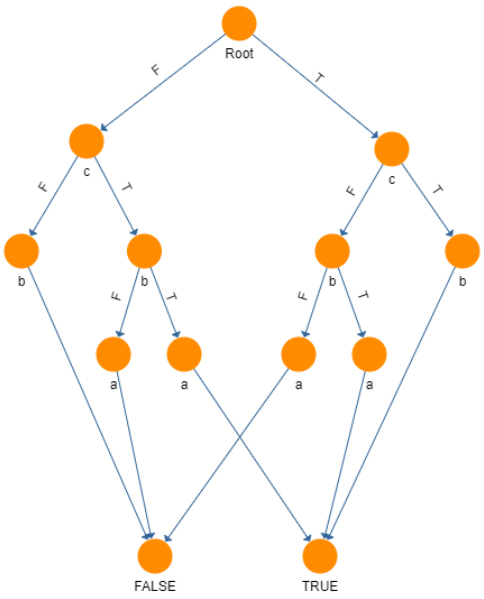


Figure A.1: Binary Decision Diagram representing the constraint $2a + 3b + 4c \geq 5$

After constructing the BDD, it can be represented as a circuit of logical gates. Then the Tseitin transformation is used to translate all the gates of the circuit to clauses, such that there is a one to one correspondence between a True output of the circuit and a satisfiable assignment of the CNF-formula created by the transformation.

If-Then-Else Gates

The BDD translation uses If-Then-Else gates. This gate has three inputs and one output. The inputs are a True-branch, a False-branch and a selector. The output of the gate is True if the selector is True and the True-branch is True, or if the selector is False and the False-branch is True. If the selector is True then the input of the False-branch does not matter. Equivalently, if the selector is False then the input of the True-branch does not matter. This can be summarized in the following implications where s is the selector, t the True-branch, f the False-branch and x the output:

$$(s \wedge t) \vee (\neg s \wedge f) \implies x$$
$$(s \wedge \neg t) \vee (\neg s \wedge \neg f) \implies \neg x$$

Using the laws of De Morgan and distributive property, this can be rewritten as formula in Conjunctive Normal Form:

$$(\neg s \vee \neg t \vee x) \wedge (\neg s \vee t \vee \neg x) \wedge (s \vee \neg f \vee x) \wedge (s \vee f \vee \neg x)$$

To strengthen propagation one could add two redundant clauses, namely:

$$(\neg t \vee \neg f \vee x) \wedge (t \vee f \vee \neg x)$$

These additional clauses are implied by the first four, but help the solver propagating when the selector is not yet known, but the True and False branches are.

Constructing the Binary Decision Diagram

Constructing the BDD happens recursively. We will assume that the constraint is formulated as a larger or equal constraint, where the left hand side contains the variables and the right hand side the bound. If a constraint is not in this format, it can always be rewritten such that it is.

An If-Then-Else gate is constructed with as selector the variable with the highest coefficient that has not yet been assigned. The True-branch will be a new function call corresponding to the variable being True, and as False-branch a function call corresponding to the variable being False. We illustrate this with the help of an example.

We want to transform the constraint $2a + 3b + 4c \geq 5$. In figure A.1 the binary decision diagram corresponding to this constraint is shown.

The variable that will first be looked at is c as it has the largest coefficient. Then the True-branch would be another BDD that corresponds to the constraint that remains when c is True. This is $2a + 3b \geq 1$. The False-branch would be a BDD corresponding to c being False. This BDD represents the constraint $2a + 3b \geq 5$.

This means that if c is True, the ITE gate will give output True if the True-branch is True, which is the case as $2a + 3b \geq 1$. If c is False then the False-branch must be True, which means $2a + 3b$ must be at least 5. These constraints will be generated the same way, which leads to a network of ITE-gates. At some point this will stop as either the remaining coefficients would not be enough to satisfy the constraint, even if they would all be True, or the already determined variables are enough to satisfy the constraint. In those cases the function call will not produce an ITE-gate but just a True or False signal. The resulting circuit is shown in figure A.2

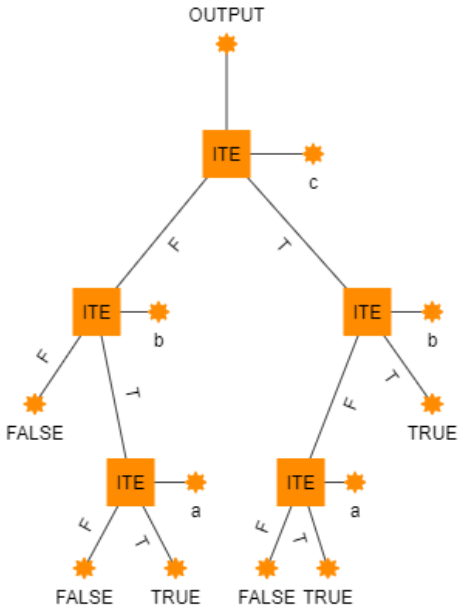


Figure A.2: Circuit representing the constraint $2a + 3b + 4c \geq 5$

This circuit can be translated to clauses by encoding every ITE-gate using the Tseitin transformation. For every line in the circuit that not directly corresponds to a variable in the linear constraint a new variable needs to be introduced. The final output variable represents the outcome of the circuit.

A.2 Adder Networks

An adder network makes use of the representation of the coefficients as a binary number and makes use of k-bits.

Definition 17 *A* k-bit *is a bit that represents the value* k*.*

For example in the binary number 100 the 1 represents the value 2^2, which is four. Therefore the 1 is a 4-bit.

The idea of adder networks is to add up the binary numbers of all activated coefficients, meaning the coefficients for which the corresponding variable is True. Then this value can be compared to the right hand side of the constraint, using a lexicographical comparison on the k-bits. For example if the constraint would be $2a + 3b + 4c \geq 5$, this would be $0a0 + 0bb + c00 \geq 101$ in binary numbers, were a, b and c take on the value of the corresponding variable. If one of them is False the number will be 000 and it does not contribute to the sum, but if the variable is True, the coefficient does

contribute.

An adder network is a circuit consisting of full adder, and half adder gates. A full adder is a gate that as input has three 1-bits and as output the sum of these three 1-bits as a 2 bit binary number. So if all three 1-bits are True the output will be 11, if two of them are True it outputs 10. One True 1-bit results in 01 and if all are False the output is 00. A half adder computes the sum of two 1-bits. Therefore the only possible outputs are 10, 01 and 00. A circuit of multiple adder gates can be used to add up larger numbers. Full adders and half adders can be translated to clauses using the Tseitin transformation.

Adder Gates

The full adder gate has three inputs x, y and z. It has two outputs c and s. c is called the carry, while s is called the sum. The implications that describe the full adder are:

$$(\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge \neg z) \implies c$$

$$\neg((\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z \vee (x \wedge y \wedge \neg z)) \implies \neg c$$

The carry can be described by the following implications:

$$(y \wedge z) \vee (x \wedge z) \vee (x \wedge y) \implies s$$

$$\neg((y \wedge z) \vee (x \wedge z) \vee (x \wedge y)) \implies \neg s$$

A half adder with inputs x and y and outputs c and s relies on the following implications:

$$(\neg x \vee \neg y) \implies c$$

$$\neg(\neg x \vee \neg y) \implies \neg c$$

$$(x \wedge y) \implies s$$

$$\neg(x \wedge y) \implies \neg s$$

Using the Tseitin transformation these implications can be translated to clauses.

Describing a linear constraint using adders

A vector in which each bit represents the same value is called a bucket. A k-bucket is a bucket in which all bits are k-bits. To add the larger numbers we start with the lowest bucket. Pick three bits of the lowest 2^k-bucket and add them using a full-adder. Each addition gives as output one 2^{k+1}-bit and one 2^k-bit. The first is added to the 2^{k+1}-bucket, the other to the 2^k-bucket. Notice that this removes one bit every iteration. If only two bits remain in a bucket, a half-adder must be used. If only one remains, this is stored as the value of the output bit and we continue to the next lowest bucket. This process repeats until the entire addition is finished and we know all output bits. Then we have a binary representation of the sum of all coefficients of True variables. This can be compared to the binary representation of the right hand side of the constraint. Comparing two binary numbers can easily be done lexicographically starting by the highest valued bit.

Example

In figure A.3 the adder network representing the constraint $2a+3b+4c \geq 5$ is presented. Since this is a small example, we only need half adders. First the constraint is rewritten to $0a0+0bb+c00 \geq 101$. As 3 is the only coefficient that could give a 1-bit the 1-bucket output is equal to b. Both 2 and 3 contain a 2-bit. Therefore a and b are added by a half adder. The result is a 2-bit and a 4-bit. The 2-bit is the output for the 2-bucket. The 4-bit is input for another half adder, together with c. The outputs are a 4-bit and an 8-bit which will be the outputs for the 4-bucket and 8-bucket. The constraint is satisfied if either the 8-bit is True or if the 4-bit is True and the 1-bit or 2-bit is True.

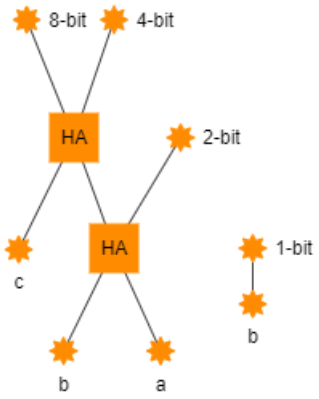


Figure A.3: Adder circuit representing the constraint $2a + 3b + 4c \geq 5$

A.3 Sorters

Where adders use binary additions, sorters make use of unary additions. So basically a sorter counts the number of inputs that are True. If a coefficient is not equal to 1, but let's say 3 than

three input signals are used for this corresponding variable. Then for a constraint that requires the sum of variables to be larger or equal than some value x, the only thing we have to do is to look at the xth entry of the output. If this is True then the constraint is satisfied. However, when the coefficients are larger, this might blow up. Therefore a base B is used. Here B is in mixed radix representation. A base in mixed radix representation means that each base element is a multiple of the previous one, but the factors are not necessarily the same. An example of a base in mixed radix representation is time. A week is seven days, a day is 24 hours. An hour is 60 minutes. Therefore a week is 7 times 24 times 60 minutes.

Coefficients are expressed in elements of the base. An example could be the base $[1, 3, 5]$ and the number 164. Then $164 = 2 \cdot 1 + 4 \cdot (1 \cdot 3) + 10 \cdot (1 \cdot 3 \cdot 5) = [2, 4, 10]$.

All coefficients are expressed in the chosen base and then for each element of the base a separate sorter network is created to add the activated coefficients. Furthermore, carries are added as input for the sorters of subsequent base elements. For example, if the base has an element 1 and an element 3, than a carry from output 3,6,9 and so on will be added as input to the sorter of base element 3. In the end, the outputs of these sorter networks are combined to check whether the constraint is satisfied. As an example we again look at the constraint $2a + 3b + 4c \geq 5$. As base elements we use 1 and 3. This means that:

- $2 = 2 \cdot 1 + 0 \cdot (3 \cdot 1) = [2, 0]$
- $3 = 0 \cdot 1 + 1 \cdot (3 \cdot 1) = [0, 1]$
- $4 = 1 \cdot 1 + 1 \cdot (3 \cdot 1) = [1, 1]$

This means the sorter representing base element 1 will have two input signals of a and one of c. The sorter representing base element 3 will have one input signal corresponding to b, one corresponding to c and one carry input from the sorter representing base element 1. The resulting network is shown in figure A.4. The outputs of the sorter for base element 1 are x_1, x_2 and x_3. For base element 3 we use y_1, y_2 and y_3.

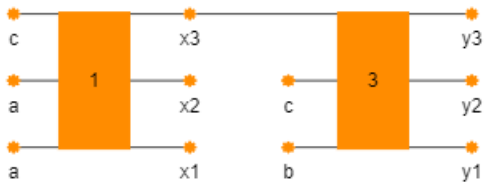


Figure A.4: Sorter circuit representing the constraint $2a + 3b + 4c \geq 5$

Encoding sorters

The sorter itself can be encoded as follows:

$$(\neg x_1 \vee a \vee c)$$

$$(\neg x_2 \vee a)$$

$$\neg x_3 \vee (a \wedge c) = (\neg x_3 \vee a) \wedge (\neg x_3 \vee c)$$

$$(\neg y_1 \vee x_3 \vee b \vee c)$$

$$\neg y_2 \vee ((x_3 \wedge b) \vee (x_3 \wedge c) \vee (b \wedge c)) = (\neg y_2 \vee x_3 \vee b) \wedge (\neg y_2 \vee x_3 \vee c) \wedge (\neg y_2 \vee b \vee c)$$

$$\neg y_3 \vee (x_3 \wedge b \wedge c) = (\neg y_3 \vee x_3) \wedge (\neg y_3 \vee b) \wedge (\neg y_3 \vee c)$$

Now the constraint is satisfied if y_3 is True, if y_2 is True or if y_1 is True and x_2 is True. When y_3 is True, y_2 must also be True. Therefore, we do not need to include y_3 in the formula. This gives the formula:

$$y_2 \vee (y_1 \wedge x_2) = (y_2 \vee y_1) \wedge (y_2 \vee x_2)$$

Appendix B

Results Hybrid Model

The results of all separate runs of the Hybrid Model are shown in the tables in this appendix.

B.1 Time limit 900 seconds

	Best Solution	Time
No Heuristics	2055	126.1
High Weighted Pairs	2095	534.9
High Weighted Groups	1985	461.8
High Weighted Days	2171	814.6
High Weighted Cuts	1659	695.2
HWP + HWC	1995	437.4
HWG + HWC	1448	623.3
HWD + HWC	2283	100.3

Table B.1: $\epsilon = 0.005, \delta = 0.1$

	Best Solution	Time
No Heuristics	1973	92.0
High Weighted Pairs	2017	303.5
High Weighted Groups	2164	730.9
High Weighted Days	2128	887.6
High Weighted Cuts	1707	856.1
HWP + HWC	1506	679.2
HWG + HWC	1813	428.3
HWD + HWC	1513	718.5

Table B.2: $\epsilon = 0.002, \delta = 0.05$

	Best Solution	Time
No Heuristics	1834	387.4
High Weighted Pairs	2072	230.4
High Weighted Groups	1840	507.7
High Weighted Days	1909	633.0
High Weighted Cuts	1611	715.7
HWP + HWC	1590	447.8
HWG + HWC	1755	149.6
HWD + HWC	2165	161.3

Table B.3: $\epsilon = 0.001, \delta = 0.05$

	Best Solution	Time
No Heuristics	3125	566.9
High Weighted Pairs	2951	147.9
High Weighted Groups	2856	180.7
High Weighted Days	2702	422.4
High Weighted Cuts	2480	641.4
HWP + HWC	2270	860.6
HWG + HWC	2009	393.9
HWD + HWC	3174	54.0

Table B.4: $\epsilon = 0.0005, \delta = 0.025$

B.2 Time limit 3600 seconds

	Best Solution	Time	Time to infeasibility
No Heuristics	1994	1184.3	-
High Weighted Pairs	1835	1787.4	-
High Weighted Groups	1978	3483.2	-
High Weighted Days	1857	3408.5	-
High Weighted Cuts	1659	695.2	802.4
HWP + HWC	1544	1328.4	1344.4
HWG + HWC	1448	623.3	765.0
HWD + HWC	1402	1132.7	-

Table B.5: $\epsilon = 0.005, \delta = 0.10$

	Best Solution	Time	Time to infeasibility
No Heuristics	1952	1021.4	-
High Weighted Pairs*	2017	303.5	-
High Weighted Groups	1920	3306.7	-
High Weighted Days	1978	1826.8	-
High Weighted Cuts	1487	2536.6	3537.8
HWP + HWC	1506	679.2	734.6
HWG + HWC	1813	428.3	902.9
HWD + HWC	1513	718.5	925.6

Table B.6: $\epsilon = 0.002, \delta = 0.05$

	Best Solution	Time	Time to infeasibility
No Heuristics	1834	387.4	-
High Weighted Pairs*	2072	230.4	-
High Weighted Groups	1840	595.6	-
High Weighted Days	1909	633.0	-
High Weighted Cuts	1611	678.0	1244.3
HWP + HWC	1590	447.8	854.0
HWG + HWC	1737	1012.9	1127.0
HWD + HWC	1441	2648.3	2817.1

Table B.7: $\epsilon = 0.001, \delta = 0.05$

	Best Solution	Time	Time to infeasibility
No Heuristics	2366	2774.2	-
High Weighted Pairs	1883	1125.7	-
High Weighted Groups	2856	180.7	-
High Weighted Days*	2702	391.0	-
High Weighted Cuts	2092	2318.2	-
HWP + HWC	1731	2867.4	3490.3
HWG + HWC	2009	393.9	-
HWD + HWC	2771	3052.2	-

Table B.8: $\epsilon = 0.0005, \delta = 0.025$

B.3 No time limit

	Best Solution	Time	Time to infeasibility
High Weighted Cuts	1500	21910.0	22943.2
HWP + HWC	1731	2867.4	3490.3
HWG + HWC	1731	3685.0	4658.1
HWD + HWC	1545	22471.9	25945.1

Table B.9: $\epsilon = 0.0005, \delta = 0.025$, Time limit $= \infty$