MASTER THESIS
OPERATIONS RESEARCH AND QUANTITATIVE LOGISTICS

A Column Generation Approach for the Integrated Crew Rostering Problem with Fairness and Attractiveness Requirements

Rick S. H. WILLEMSEN (429967)
Supervisor: Prof.dr. Dennis Huisman
Second assessor: Dr. Wilco van den Heuvel
August 27, 2020

In this thesis, we consider the Cyclic Crew Rostering Problem (CCRP), in which cyclic rosters for groups of employees are constructed. The quality of these rosters is evaluated on basis of fairness and attractiveness. A column generation approach is proposed to solve the resulting model. Suitable columns are found by solving a Shortest Path Problem with Resource Constraints (SPPRC). In order to speed up the SPPRC we introduce forward and backward completion bounds, which actively prune sets of columns. Next, we apply the proposed solution method on real-life instances from Netherlands Railways. We show that the completion bounds reduce overall computation time, such that attractive rosters containing up to 300 duties can be constructed. Furthermore, slightly increasing fairness or attractiveness requirements does not influence the performance of the solution approach.

Contents

Chapter 1

Introduction

The creation of rosters in public transport is a complex decision problem, where the preferences of the employees are balanced against cost efficiency. This problem is also familiar for Netherlands Railways (Nederlandse Spoorwegen in Dutch, abbreviated as NS), which is the major public transport operator in the Netherlands. More than 1.3 million passengers travel with NS on a daily basis. To facilitate this, NS employs a workforce of more than 20,000 people to support and operate their services, including 3,000 train drivers.

In case the services of NS are suspended, for example due to strikes, this has a large societal impact. These nationwide strikes took place numerous times in the past, until the conflict ended with the introduction of the Sharing-Sweet-and-Sour rules (Abbink et al., 2005), which state that (un)popular work is equally distributed over groups of employees. These rules, which were agreed upon by both NS and the labour unions, are aimed at increasing the quality of work. NS incorporated these rules in their crew planning process using Operations Research (OR) techniques. However, these new rules cannot prevent all strikes, see for example the 24 hours strike of 2019, which disrupted the train network in the Netherlands[1]. Although OR techniques cannot solve every conflict, they do play an important role in increasing employee satisfaction, which indirectly influences the probability of having a strike as well.

The rostering of employees is an important part of the crew planning process. In the literature, the assignment of duties to employees is known as the Crew Rostering Problem (CRP). Public transport companies, including NS, often consider the Cyclic CRP (CCRP), in which each crew member performs a different part of the roster in a sequential order. However, some stages of constructing the rosters are still conducted manually NS. This is a time consuming

[1] https://www.ad.nl/binnenland/treinverkeer-en-ov-grote-steden-plat-op-28-mei~a8530fd4/

task, not only due to the large amount of crew members and duties, but also because of the large amount of rules (from the labour unions) and preferences that should be taken into account. Furthermore, the Sharing-Sweet-and-Sour rules are also incorporated into the rostering process, by making all rosters equally attractive for all crew members within a crew base. A first step in making an integrated approach, while simultaneously accounting for the fairness and attractiveness of the rosters, was made by Breugem (2020). Starting with (manually created) roster templates containing duty and rest day patterns, a method that assigns the duties to the actual rosters is developed. The quality of the rosters is assessed by considering a set of duty attributes.

The CCRP is often solved sequentially, see for example Sodhi and Norris (2004), Lezaun et al. (2006) and Hartog et al. (2009). Only recently, there is a trend to integrate all steps in the crew rostering approach (Xie and Suhl, 2015; Breugem, 2020). The concepts of attractiveness and fairness of the rosters are well known in the literature. However, Breugem (2020) is the first to explore the trade-off between the two. This shows that the integrated crew rostering process, while taking into account attractiveness and fairness, is still considered a non-trivial problem and solving it requires state-of-the-art OR techniques.

In this thesis, we take the final step to integrate the crew rostering process for a single crew base at NS. This approach uses only two inputs: the available duties to be covered and the employees, who are divided over several roster groups. The model is solved in a single step, such that no manual intervention is required to obtain the crew rosters. The quality of the obtained rosters are evaluated on similar duty characteristics as in Breugem (2020).

The model is applied on a real-life instance from the NS crew base in Amersfoort. Employing this solution approach, we show that it is possible to solve the integrated crew rostering problem in a reasonable amount of time. Furthermore, using a sensitivity analysis we illustrate how attractiveness and fairness can be increased in the obtained rosters.

The remainder of this thesis is organised as follows. Firstly, Chapter 2 explains the current crew planning process at NS in more detail. This is followed by Chapter 3, which reviews the current literature regarding the CCRP. Chapters 4 and 5 introduce the mathematical model and present the implementation details, respectively. Afterwards, Chapter 6 compares several strategies to solve the CCRP on instances from NS. After fixing the best performing strategy, a sensitivity analysis is performed in Chapter 7. Finally, conclusions are drawn and recommendations for future research are given in Chapters 8 and 9.

Chapter 2

Problem description

In this chapter, we provide a short description of NS and the different levels of decision problems that it faces. Afterwards, we delve into the crew planning process and introduce the relevant concepts. Next, the properties and constraints regarding the crew roster are introduced. Finally, we give an overview of the current crew rostering process at NS and we put the contribution of this thesis into perspective.

2.1 Railway planning problems at NS

In railway planning, four phases can be distinguished based on their corresponding time horizon: strategic planning, tactical planning, operational planning and operational control (Abbink, 2014). Strategic planning is used for long term decision making that influence the goals and objectives over a longer period. Examples include the investment decisions for the location and size of physical facilities, which are planned years in advance. Tactical planning involves the efficient allocation of the available resources, where the time span ranges from a few months up to a year. This typically concerns problems such as the construction of a general timetable or crew schedule. These generic timetables are then finalised in the operational planning stage, resulting in day-to-day schedules. Lastly, operational control adjusts the timetables in real-time to address any unanticipated disturbances.

All these planning phases have different characteristics, such as planning horizon, objective and level of detail. Therefore, each phase requires its own specialised solution approach. For example, strategic planning requires accurate forecasts over a long period of time. On the other hand, tactical and operational planning need to allocate the resources as efficiently as possible. These problems are often encountered in the OR literature. Lastly, operational control requires

fast decision making in order to decrease delay when a disturbance happens. At NS the situation is no different and various approaches are used to solve the above mentioned problems, ranging from mathematical programming techniques to simulation studies. We refer to Huisman et al. (2005) and Kroon et al. (2009) for a more thorough discussion.

In this thesis, we only consider the tactical planning for the train drivers and conductors, which are collectively referred to as *crew members*. The planning of the crew is divided into two phases: crew scheduling and crew rostering. In the former phase the Crew Scheduling Problem (CSP) is solved. This involves the construction of *duties*, which are defined as blocks of work that can be performed by a crew member. Figure 2.1 gives an example of a duty. Each duty starts and ends at the same station. Furthermore, each duty should contain a sufficient meal break time (denoted by the star) and is not allowed to exceed 9.5 hours. The second phase, which is crew rostering, assigns these duties to the crew. At NS, these duties are constructed in a centralised process. Afterwards, the duties are allocated to the crew bases, which subsequently create their own rosters in a decentralised manner.

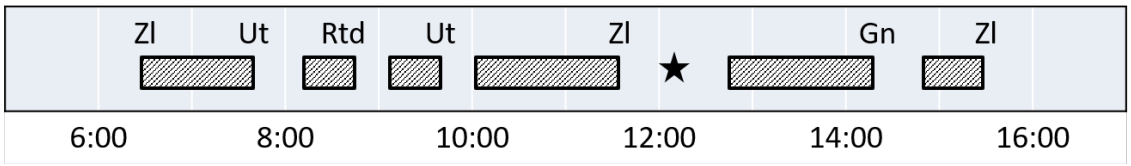

Figure 2.1: Example of a possible duty. For each block the departure and/or arrival station is given. This duty visits the cities of Zwolle (Zl), Utrecht (Ut), Rotterdam (Rtd) and Groningen (Gn).

In crew scheduling, it is important to create duties that do not violate the collective labour agreement, while minimising the operational costs. Constraints ensure for example that the duty length is not violated or that the meal break is sufficiently long. Since the duties are linked to a certain departure and arrival station, this implies that the duties are linked to a specific crew base. To improve the overall quality, the Sharing-Sweet-and-Sour rules are incorporated in the centralised scheduling process (Abbink et al., 2005). These rules divide the popular and unpopular work over the crew bases as fairly as possible. The scheduling process is solved using a column generation approach. The crew scheduling problem is solved as a generic annual plan and changes are only made six times per year. This means that the crew schedules for a specific day are only adjusted slightly. In this way, the crew members know what is expected from them months in advance.

Three criteria can be used to evaluate the quality of the end result of the crew planning phase: efficiency, fairness and attractiveness (Breugem, 2020). *Efficiency* is defined as the min-

imum number of employees that is necessary. *Fairness* is enforced through the Sharing-Sweet-and-Sour rules, which make sure that the popular and unpopular work are equally distributed over the crew bases. These first two criteria are mostly relevant for the crew scheduling phase. However, fairness can also be enforced on a crew base level by ensuring that the work is fairly divided over the roster groups within a crew base. Finally, *attractiveness* is based on the final roster, for example rest times and variation of the roster play an important part here.

2.2 Crew rostering

In crew rostering, the constructed duties have to be combined into sequences, satisfying several constraints determined by the labour agreement. These constraints relate to the number of working hours, days off or rest times. Furthermore, NS uses *cyclic rosters*, meaning that each week, all employees work on a different roster within a so-called *roster group*. A possible cyclic roster is visualised in Figure 2.2. The switching between rosters occurs in a fixed order. For example, in week 1 the first employee works on row 1 and the second employee works on row 2. In week 2 the first employee works on roster 2 and the second employee works on row 3. This means that the number of rows in a roster is equal to the number of employees in that roster group. In this way, all employees from the same group cycle through the roster, such that at the end of the cycle all employees have performed all duties. This means, that if an employee completed the roster in week 5, then next week it will continue working on the roster in week 1. The *rows* correspond to single working weeks and the *columns* represent the day of the week. Each *cell* can be interpreted as a single working day for an employee. This working day can have several types: regular duty, day off and the so-called WTV, CO and RES days.

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	R	L	L	N	N	R	E
2	E	R	L	L	E	R	R
3	E	R	E	E	N	N	N
4	R	R	E	L	R	R	R
5	L	E	R	R	L	L	L

Figure 2.2: Example of a basic schedule for a group with five members.

On a regular duty day there are three types of possible duties: early (E), late (L) or night (N). The WTV day is technically a day off, except that the crew member can opt to sell this day off and work instead. A CO day is also a day off, which can be earned by a crew member in case their working hours on certain time slots (e.g. during the weekend or early/night shifts) exceed a certain threshold. The CO day becomes a working day in case the crew member does not work enough hours. Finally, a RES day is defined as a day on which the crew member should be available, in case the work of a fellow crew member has to be taken over (e.g. due to illness or holiday).

Furthermore, the rosters have to satisfy an extensive list of rules, based on the *collective labour agreement*. Also, the attractiveness of the rosters is taken into account by using *roster preferences* obtained from the employees. Lastly, the (un)popular duties are equally spread over the roster groups by making use of so-called duty attributes.

2.2.1 Collective labour agreement

Below are the most important rules and regulations from the collective labour agreement (CLA) from NS.

1. The rest time after each duty should be at least 12 hours. After a night duty, which ends later than 2:00 a.m., the rest time should be at least 14 hours. Minimum rest time after three or more consecutive night duties is 46 hours.

2. There should be two rest days per week on average. A day off is a period of at least 30 hours. In general, the length of a rest day is 6 hours plus the number of rest days times 24 hours.

3. There should be a rest period of at least 36 hours in each period of 7x24 hours. Or there should be a rest period of at least 72 hours in each period of 14x24 hours. These rules are enforced using a rolling horizon for all periods of 7x24 (or 14x24) hours.

4. There is a maximum of 7 days with consecutive duties. WTV and RO do not count as a duty.

5. The average working time per week is at most 40 hours.

6. There is a maximum of 36 night duties per 16 weeks.

7. At least once every three weeks there is a so-called *Red Weekend*. This rest period lasts at least 60 hours and starts before Saturday at 12:00 a.m. and ends after Monday 4:00 a.m.. This is also implemented using a rolling horizon.

2.2.2 Roster preferences

Rosters that take into account the following properties are often preferred by the crew members.

1. Train drivers need to drive on each line that they are allowed to drive on. If they do not use a certain line for a period of time they lose their license to drive there. A similar rule holds for the type of trains, the train driver has to use certain train types at least once, otherwise they lose knowledge on how to control that specific train type.
2. Make a free weekend as long as possible.
3. If possible, WTV-days are spread over different days in the week and are only scheduled once every two weeks.
4. Two or more consecutive days off are preferred.
5. A series of exactly three duties after each other with the same type (E, L or N) is preferred. If a series of the same type is present, then the starting times are preferably only increasing or decreasing (e.g. 5:00 - 5:30 - 6:00 a.m. or 6:00 - 5:30 - 5:00 a.m.).
6. A single day off should be made as long as possible by ending with an early duty and starting with a late duty. This includes the patterns E - R - L and E - R - N.
7. When changing from a late duty to an early duty, try to place two days off in between (e.g. L - R - R - E or N - R - R - E).
8. A forward rotating roster is preferred (e.g. E - L - N).
9. Maximum of 5 consecutive days with duties.
10. A single duty is not preferred, which correspond to R - duty - R.
11. Short cyclic rosters are preferred, which means no more than three similar types of duties after each other.
12. In every row of the roster there should be at least one day off.

2.2.3 Duty attributes

Fairness can be measured using *duty attributes*. Each duty consists of multiple trips, leading to certain characteristics. Some duties are preferred over others, for example short trips are often less desirable compared to longer trips, which cover large parts of the Netherlands. Also, the duty length and the amount of work on double decker trains are seen as duty attributes. In this thesis, we use the following six duty attributes.

1. Average duty length per day. The length of a duty is calculated as the difference between start and end time of a duty.

2. Average workload per week. The workload is defined as the sum of all the duty lengths within the same week.

3. Percentage of type-A work. Crew members prefer work of type-A, which consists for example of trips on Intercity trains, which only stop at large stations. Therefore, these trips are often longer than average.

4. Percentage of aggression work. Some trips have a higher chance of passenger aggression.

5. Percentage of work on double decker trains. These trips are undesirable since they require the crew member to climb up and down the stairs.

6. Repetition Within Duty (RWD) values. These are calculated as the total number of routes divided by the total number of distinct routes within a duty. Low RWD values are preferred, since this means more variation for the crew member.

2.2.4 Formal description

Having defined the necessary definitions, it is now time to formalise the Cyclic Crew Rostering Problem (CCRP). In the CCRP, rosters are created for each crew base, which adhere to the cycling property. This CCRP is solved for each crew basis separately. Each crew basis contains multiple roster groups, where the members of each roster group have similar characteristics, such as full-time or part-time employees.

2.3 Current crew rostering process at NS

In each timetable year, three steps have to be performed for a single crew base. Within a crew base, all (members of a) roster groups are known in advance. In the first step, the duties

obtained from the CSP are divided over the roster groups in a fair way. In the second step, the representative makes the *basic roster* which has to satisfy the CLA rules. This basic roster or schedule contains patterns consisting of duties and days off. Since the CSP is solved six times per year, the basic roster should be flexible enough to incorporate those changes as well. In the third step, the basic roster is used as a template to assign the actual duties. These three steps are explained in more detail below.

2.3.1 Assignment of duties to roster group

The basic roster is made at the beginning of a new timetable year and is then used throughout the year. The rosters are made per crew base (for example Utrecht). The duties obtained from the crew scheduling phase and the crew members of the given crew base are the inputs for the creation of a basic roster. The duties are allocated to the roster groups in that crew base. This should be done such that each group has a similar amount of (un)popular work. The division of popular work within a group is trivial, since all crew members cycle through the same roster within their group. These basic rosters are made from scratch manually, even for the largest crew base in Utrecht (Hartog et al., 2009). The duties are divided by means of an auction, where every group has a representative. Only in case no feasible roster can be made for a roster group, the duties may be switched between groups.

2.3.2 Construction of the basic roster

After the allocation, each representative attempts to construct a basic roster for its own group manually, while striving to satisfy all previously mentioned requirements. These basics schedules define on each day the type of work assigned to a crew member or whether a day off is scheduled. As illustrated in Section 2.2, this leads to a matrix in which each cell indicates on which day the crew member performs which duty type. As soon as a feasible roster is found, the representative is finished. Otherwise, it is still possible to exchange duties between groups. Afterwards, the basic rosters are not changed on a daily basis. In case a change in the crew schedule is made, the basic roster should be able to handle these changes. If the CSP cannot be solved, the roster might be changed on a specific day.

2.3.3 Obtaining the final roster

Since the basic roster remains unchanged throughout the year, whereas the CSP is solved roughly six times a year, this implies that the rosters have to be updated as well. As input

for this last phase, NS uses the basic roster from each roster group and a set of (newly) generated duties. Afterwards, the generated duties are assigned to the available spots in the basic roster. This currently happens manually, but can of course also be done in a more automatic process. Hartog et al. (2009) show that NS crew workers actually prefer the rosters generated by an algorithm over the manually constructed rosters. Furthermore, Breugem (2020) improves the allocation of duties to crew bases and also analyses the trade-off between fairness and attractiveness of the rosters. In this thesis, the creation of the basic schedules is integrated with the assignment of the duties to the time slots given by the basic schedule.

2.4 Contribution

Breugem (2020) integrates the first and third step, such that the duties are fairly allocated over the roster groups, while obtaining satisfactory rosters. This thesis makes the final step for an integrated crew rostering process. All three steps are combined, such that the creation of the basic roster happens simultaneously with the assignment of duties to the actual roster. By performing all steps simultaneously, we prevent that suboptimal decisions are made in each individual step. Therefore, it is possible to improve the overall solution quality (Xie and Suhl, 2015).

Chapter 3

Literature review

The literature on personnel rostering problems is quite extensive and these models are applied to a broad range of industries, including call centers, airlines, health care and public transport. Careful planning and accounting for preferences of crew members can lead to significant improvements in productivity and satisfaction. We refer the interested reader to Ernst et al. (2004) for a review on commonly used methods and models for a variety of applications. Cheang et al. (2003) and Burke et al. (2004) present comprehensive literature reviews on nurse rostering, while Kohl and Karisch (2006) focus on rostering in the airline industry. In public transport the CCRP is often used, whereas acylic rosters are more common in other industries. The focus of this literature review is the CCRP. In particular, we show that a sequential approach is still quite common and we elaborate on how the concepts of attractiveness and fairness are incorporated into the models.

Since the CCRP is a difficult problem, solving it commonly requires the use of heuristic approaches. Ernst et al. (1998) consider the rostering of crew members for freight trains in Australia. In some cases, it is necessary to transport crew members to a different depot outside of a shift, this is referred to as *paxing*. The train company has to provide additional road transport, since the freight trains do not have capacity for passengers. Their model minimises the operational costs while penalising paxing. Furthermore, they also take into account that workload is balanced between crew at the same depot and between depots. They solve this problem using simulated annealing.

Caprara et al. (1997) propose several formulations for crew rostering. They develop both a multi-commodity network flow model as well as a set partitioning model. In their model, they only take attractiveness into account. For example, each week can include at most one

of the following characteristics: two duties with external rest, one long duty or two overnight duties. Furthermore, after each week there has to be a sufficient rest period. They evaluate their method on instances from the Italian railways.

In earlier work, the scheduling and rostering process are often solved simultaneously, due to the smaller scale of the available instances. For example, Ernst et al. (2001) solve the integrated scheduling and rostering for the Australian railways using both cyclic and acylic rosters. In general, they minimise the operating costs while penalising unwanted duties, based on the preferences of the train drivers. For example, it is preferred that all night shifts or all day shifts are put together. They solve the resulting model by relying on the fact that full enumeration over all possible duties is possible, due to the sparseness of the rail network in Australia. For more dense networks (e.g. the Dutch rail network) this approach would not be feasible.

Freling et al. (2004) also solve the integrated scheduling and rostering problem for several applications, such as the railway and airline industry. The rostering phase is modelled as a set covering formulation. In their model, they minimise the workload and penalise duties that have undesired characteristics, such as layovers, sequences of heavy duties and standby duties. Furthermore, the workload is balanced over all crew members by enforcing a threshold. They develop a branch-and-price algorithm to solve the resulting model, which worked well on the instance form the airline industry. As the running time for the railway application was too high, a heuristic approach was introduced and the decomposed problem was solved sequentially.

It is not uncommon that large instances of the CCRP are solved by splitting the whole process in sequential phases. Sodhi and Norris (2004) were one of the first to apply this approach in practice. They split the rostering problem into two main stages. In the first stage the rest day pattern is made. Then, in the second stage the duties are assigned to the rosters. The first stage is further split into several steps, some of which are performed manually, whereas others are dealt with using solvers or heuristic approaches. Three types of duties are considered: early, late and night duties. They take attractiveness into account by using a soft constraint that penalises the use of mixed weeks, i.e. weeks in which there are both early and late duties. Furthermore, they maximise the number of consecutive days off (excluding weekends) and (regular or long) weekends. The method is evaluated on instances from London Underground.

Lezaun et al. (2006) split the rostering process into four different parts. First of all, the reserve days (which are grouped in weeks) are assigned to the drivers. Afterwards, the duty type (early, late and night duties) patterns are constructed per week. In the third phase, these

patterns are combined such that they fit in a rotating roster for a group of drivers. Finally, the reserve weeks from the first phase are combined with the rotating rosters from the previous step, in order to obtain an assignment in which all drivers have an equal amount of early, late and night duties and similar amount of weekends off. They applied this model to the metro network in Bilbao. The obtained rosters were accepted by both the firm's representatives and the employees.

Hartog et al. (2009) split the rostering problem into two parts (similar to Sodhi and Norris (2004)). They first create patterns in which the type of duty (e.g. early, late or night) is determined. In the second phase they assign the actual duties to the feasible places in the roster. Both models are formulated as an assignment problem, where undesirable duty patterns are penalised in the objective function. They show that rosters can be obtained much faster using this process than the manually generated rosters. Furthermore, both the train drivers and conductors preferred the rosters obtained from the models over the manually generated rosters.

Mesquita et al. (2013) integrate the vehicle and crew scheduling problem with the crew rostering problem, using pre-defined day off patterns. In the objective, they consider operational costs as well as balancing measures. These measures include the amount of short and long trips. They solve this using a heuristic, which is based on Benders decomposition. They show the benefits of this approach using real-world data from bus companies in Portugal.

Nishi et al. (2014) propose a decomposition algorithm for crew rostering, in which their main objective is to distribute the workload as fairly as possible. The workload is distributed over the crew members by minimising the maximum average working time. In the master problem of the decomposition, a set of duties is assigned to a set of rosters. Afterwards, a set of subproblems is solved, where feasible cyclic sequences are constructed that conform to the working regulations. The performance of their decomposition approach is shown on instances from the Japanese railway company.

Recently, it has become common to include more extensive fairness concepts in the CCRP, which do not solely include the workload distribution. Borndörfer et al. (2015) incorporate attractiveness by penalising inappropriate sequences of duties in their rostering process. An example of such a duty sequence is a backward rotation, in which the duty on the next day starts earlier compared to the current duty. The preferences of employees are also taken into account using hard constraints. Furthermore, they also spread the unwanted duties (night or weekend duties) over all employees, to improve fairness. They present both a network flow problem and

a set partitioning problem. They solve their model using a heuristic approach similar to the Lin-Kernighan heuristic. Their method is evaluated on both a cyclic crew scheduling problem and an acyclic toll enforcement problem.

Integrated crew rostering is receiving more attention in the literature. Xie and Suhl (2015) are one of the first to computationally compare the sequential and integrated approach. They solve the crew rostering problem for both cyclic and non-cyclic rosters. They propose a multi-commodity network flow problem, containing three different objectives: operating costs, fairness and preferences of the crew. Fairness is measured by the distribution of workload, number of days off and unpopular activities. Their proposed integrated approach is compared to a sequential approach (based on Sodhi and Norris (2004)). They show on instances from German bus companies that the integrated approach obtains a better solution quality.

Borndörfer et al. (2017) solve the integrated scheduling and rostering problem for drivers in public transport using Benders decomposition, where the scheduling and rostering problem are solved in the master- and subproblem, respectively. In their model, they penalise rosters that are not preferred by the drivers. For example, isolated duties or a weekend with a single day off are both penalised. An isolated duty is a duty between two free days. In the rostering phase, they make use of so-called *duty templates*, which aggregate similar types of duties based on their key characteristics. In their approach, they only use the starting time of a duty. However, they also suggest that other characteristics such as duty type, home depot or duty duration can be used to group similar duties together. This facilitates a more general aggregation compared to Hartog et al. (2009), who only use three duty types. This aggregation leads to a reduction in the number of linking constraints, giving rise to a smaller problem. A weaker linking between the original duties is expected to reduce the quality of the rosters. However, Borndörfer et al. (2017) show on real-world data from a public transport company that the use of these duty templates actually improved the obtained roster at almost no extra cost for the scheduling phase.

The trade-off between fairness and attractiveness is analysed by Breugem (2020). Similarly to Hartog et al. (2009), duty templates with three possibilities (early, late and night) are used as input. First of all, he shows using an exact branch-and-price-and-cut approach that fairness should not be the sole evaluation criteria, since this comes at the cost of attractiveness. Also, the decrease in attractiveness due to a fixed fairness level is not distributed evenly over all the roster groups. These conclusions highlight the fact that both measures should be carefully balanced. Furthermore, a family of formulations for the CCRP are analysed, in order to obtain

insights into how a strong formulation can be made. These models are evaluated on data from NS. The rules and regulations that were used in Breugem (2020) are of a similar structure as the ones discussed in the previous chapter. They include the rest time, rest days, workload and Red Weekends (corresponding to items 1, 2, 5 and 7 respectively from the CLA). The models are solved using column generation in a branch-and-price algorithm. The results demonstrate that choosing a certain formulation heavily influences both the solution quality and running time. Based on these conclusions and recommendations a suitable model is chosen for this thesis. This mathematical model is presented in detail in the next chapter.

Chapter 4

Methodology

In this chapter, we introduce a mathematical model to solve the CCRP. First, the mathematical notation is explained, which is followed by an introduction of the formulation for the CCRP. Afterwards, the constraints that have to be implemented are explained in more detail.

4.1 Mathematical model

We use the formulation for the CCRP as proposed by Breugem (2020). The set D denotes the set of duties and T denotes the set of cells in a roster. The set of roster sequences is denoted by S. A roster sequence $s \in S$ consists of a mapping between cells $t \in T$ and duties $d \in D$, which we denote by (t, d). We divide the problem into a set of clusters K. The constraints that are *implicitly* defined in a cluster $k \in K$ is denoted by $Q_k \subseteq Q$. This means that all constraints contained in the set $Q_K = \cup_{k \in K} Q_k$ are modelled implicitly using roster sequences. All other constraints in the set $Q \setminus Q_K$ still have to be modelled *explicitly*. Each roster sequence belongs to a cluster set S_k from cluster $k \in K$. Lastly, Q is a set containing all modelled roster constraints, where each roster constraint q is modelled using a set of linear constraints $p \in P_q$.

Furthermore, the following parameters are introduced. The parameter h^k_{ds} shows whether roster sequence $s \in S_k$ contains duty d or not. The parameter c^k_s penalises the use of a roster sequence $s \in S_k$ belonging to cluster $k \in K$. Lastly, f^p_{td} denotes whether a linear constraint $p \in P_q$ is assigned to a combination (t, d) and the threshold value for violating a constraint is denoted by b_p. The violation of the constraint has to belong to the interval $\delta_q = [0, u_q]$, using a penalisation of c_q.

Finally, we define the binary decision variable x^k_s when roster sequence $s \in S_k$ is assigned to cluster $k \in K$. The violation is modelled using the variable $\delta_q \in \Delta_q$, which denotes how

much the roster constraint $q \in Q \setminus Q_K$ is violated. The resulting formulation of the CCRP is shown below.

$$\min \sum_{k \in K} \sum_{s \in S_k} c_s^k x_s^k + \sum_{q \in Q \setminus Q_k} c_q \delta_q \tag{4.1}$$

$$\text{s.t.} \sum_{s \in S_k} x_s^k = 1 \qquad \forall k \in K \tag{4.2}$$

$$\sum_{k \in K} \sum_{s \in S_k} h_{ds}^k x_s^k = 1 \qquad \forall d \in D \tag{4.3}$$

$$\sum_{k \in K} \sum_{s \in S_k} \sum_{(t,d) \in s} f_{td}^p x_s^k \leq b_p + \delta_q \qquad \forall q \in Q \setminus Q_K, p \in P_q \tag{4.4}$$

$$x_s^k \in \mathbb{B} \qquad \forall k \in K, s \in S_k \tag{4.5}$$

$$\delta_q \in \Delta_q \qquad \forall q \in Q \setminus Q_K \tag{4.6}$$

The objective minimises the sum of the roster sequence costs and the penalties associated with the explicitly modelled roster constraints. Constraints (4.2) and (4.3) model the correct assignment of duties, by ensuring that each cluster is assigned to exactly one roster sequence and each duty is used exactly once. Constraints (4.4) model the explicit constraints, which can take the form of soft or hard constraints. Lastly, Constraints (4.5) and (4.6) restrict the decision variables to their domains.

4.2 Implementation

In this thesis, the clusters in K all contain 7 days, starting on Monday and ending on Sunday, such that the clusters are non-overlapping. These clusters correspond to the weeks (rows) of the rosters and are also referred to as *roster sequences*. This is in line with Breugem (2020), who shows theoretically how the cluster size impacts the solution method and also demonstrates using computational results that for instances from NS the formulation based on weekly clusters outperforms other cluster sizes. Since we are solving an extension of this problem, it is expected that these weekly clusters also work well for our problem. An example of three possible roster sequences is visualised in Figure 4.1.

It might happen that the items from the CLA and the roster preferences are too complicated to be implemented efficiently. In that case, the items are simplified. Below we state for which items a simplification is needed.

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
S1	R	126	125	54	56	R	R
S2	12	R	14	29	R	R	R
S3	127	128	R	R	45	40	41

Figure 4.1: Example of three possible roster sequences, containing duties and rest days.

For item 3 from the CLA, there is the issue that the rule does not work when the period of 7x24 (or 14x24) starts or ends during a rest day longer than 36 (or 72) hours. A similar problem arises when looking at the Red Weekends (item 7). If the period starts or ends with a sufficiently long rest day, then the requirement can be ignored and we assume that this roster still adheres to the rules laid down by NS.

For item 3 from the CLA we make two additional assumption: we only enforce periods of 7x24 hours and the start of the period is always at midnight. For example, the rule is enforced on the period of Monday 12:00 a.m. till Sunday 23:59 p.m., but also for the period Tuesday 12:00 a.m. till Monday 23:59 p.m. The first assumption leads to less flexibility, because enforcing the rule on periods of 14x24 hours becomes redundant. Losing this option might come at the cost of a reduction in solution quality. The second assumption reduces the complexity of the problem, since we only have to consider periods starting at a specific hour.

In the CLA, item 6 states that there is a maximum of 36 night duties per 16 weeks. In case a group has less than 16 rows (employees), we recalculate the maximum number of night duties, such that we do not violate the CLA in case the group performs the roster for 16 weeks.

Finally, we simplify the implementation of the Red Weekend (item 7 from the CLA) by assuming that there are no early duties that start before Monday 4:00 a.m. (which is indeed the case for instances from NS). Using this assumption, we change the definition of a Red Weekend to a period of at least 60 hours which starts before Friday 4:00 p.m. and ends before Monday 4:00 a.m.

Chapter 5

Solution approach

The set of possible roster sequences is quite large, which also leads to a large number of binary decision variables x_s^k. Due to the large number of variables, a column generation approach is used to solve the previously discussed model. In the column generation approach, we iterate between the so-called master problem and pricing problem. In the *master problem*, the linear programming (LP) relaxation is solved with a subset of the decision variables. In the *pricing problem*, we try to find beneficial roster sequences, which are subsequently added to the master problem. We stop when no more beneficial roster sequences can be found, implying that the LP relaxation is solved to optimality. This is also referred to as solving the *root node*. To improve the computation time of the column generation approach, it is often useful to initialise the model with a feasible solution.

A graphical illustration of the solution approach is given in Figure 5.1. As mentioned before, we use the CLA, roster preferences and duty attributes as inputs. Before solving the root node, we can opt to create a start solution. After having obtained the optimal LP solution, we have to find a feasible roster. In this thesis, we consider a heuristic approach, where we first find an integer solution and afterwards improve this integer solution by means of a local search algorithm. In this chapter, we describe the aforementioned steps in more detail.

5.1 Start solution

The column generation can start with an empty roster, meaning that no duties are assigned. As a result, the first few iterations generally concentrate on covering the duties. Only after most duties are covered, the pricing problem will try to improve the LP solution. To reduce the total computation time we can make use of multiple start solutions, such that the majority of the

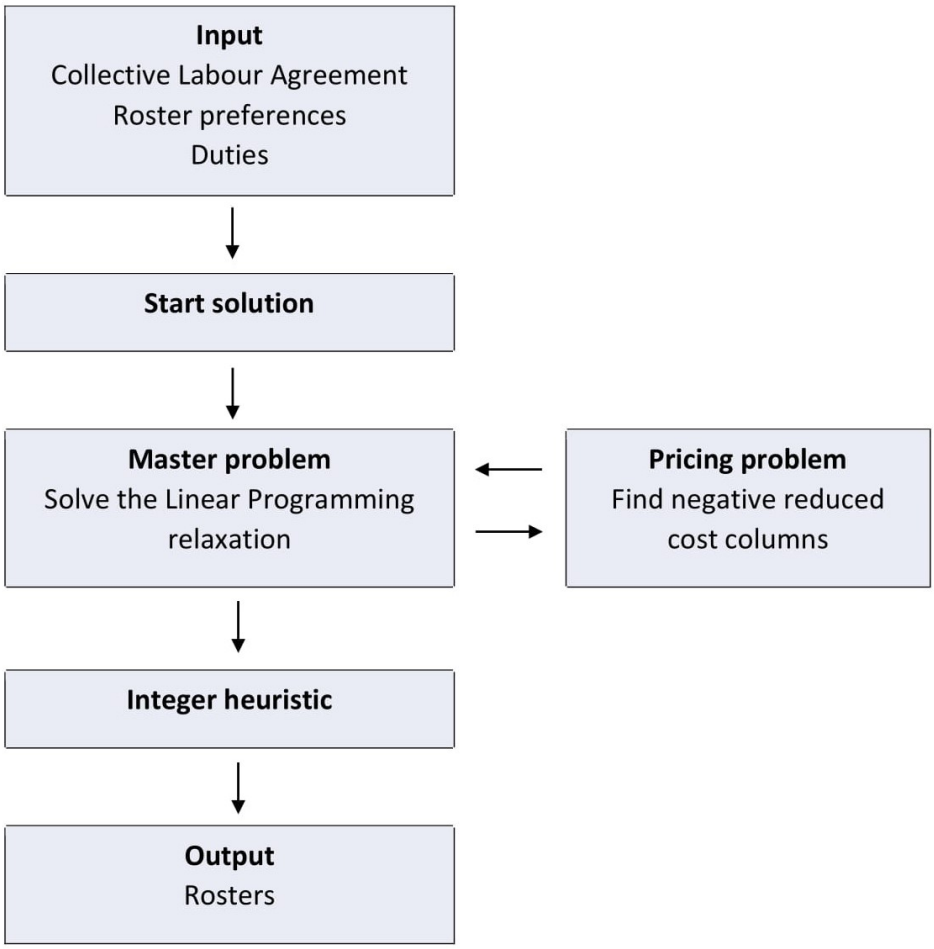

Figure 5.1: Overview of the necessary inputs, outputs and steps taken in the solution method.

duties are already assigned to a specific week. This procedure should be fast and, if possible, have a good objective value. In this thesis we consider two methods: a Randomised Insertion Method (RIM) and a Greedy Randomised Adaptive Search Procedure (GRASP).

5.1.1 RIM

In this basic algorithm, we go through all empty cells in the roster one by one and assign the first available duty. We continue doing this until all duties are covered or if no more duty can be inserted into the roster. Adding multiple start solutions only makes sense if the algorithm returns a different start solution every time. For this reason, we implement two randomisation steps. First of all, the list containing all duties is shuffled. Secondly, the order of going through the empty places in the roster is randomised.

The RIM is an algorithm that does not take the objective into account. So we expect that solutions can be quickly obtained, which comes at the costs of the solution quality. Therefore, we also consider a GRASP, that does take the objective into account.

5.1.2 GRASP

The GRASP consists of two parts. The first part is a greedy assignment part in which we create an initial start solution. Afterwards, we try to improve this start solution using a local search procedure.

In the first part, we first try to assign each available duty to a specific day in a specific week. This leads to a set of best candidate duties for this day based on the objective value. We keep a list of candidate duties and from this set we choose one duty at random. We continue this until no more duties can be assigned in a feasible way. This results in our initial start solution.

In the second part of the GRASP we try to improve the previously obtained solution. We do this by using a so-called *two-opt procedure*, also known as a pairwise comparison. We create a set of all possible swaps between two duties on a similar day but a different week (e.g. both duties are scheduled on a Monday). First we check if such a swap is allowed and does not violate any CLA rules. In case the duties can be swapped, we calculate the objective value. After considering all pairs of duties in the set, we swap the pair that led to the best improvement in objective value. We continue until a time limit or iteration limit is reached or if no more improvements can be made.

When performing the above steps multiple times, we obtain several start solutions. All these start solutions can be added to the master problem.

5.2 Master problem

The master problem is defined using Equations (4.1) - (4.6), where we relax the integrality Constraints on the x_s^k variables. That is, we replace Constraints (4.5) by $x_s^k \geq 0,$ for all $k \in K$ and $s \in S_k$. The penalties in the objective function, c_s^k and c_q are often in the range of 0.001 up to 0.1. Except for the penalty of not scheduling a duty, which is set to 10,000. In this way, the column generation approach primarily focuses on scheduling all duties.

After each iteration of the master problem, we solve the pricing problem in order to obtain roster sequences with negative *reduced cost* (RC). The reduced cost γ_s^k of a roster sequence $s \in S_k$ belonging to a cluster $k \in K$ is defined as follows. Let μ_k, ϕ_d and θ_{qp} correspond to the dual multipliers of Constraints (4.2), (4.3) and (4.4), respectively. Then the reduced cost γ_s^k

can be calculated as

$$\gamma_s^k = c_s^k - \mu_k - \sum_{d \in D} h_{ds}^k \phi_d - \sum_{q \in Q \setminus Q_K} \sum_{p \in P_q} \sum_{(t,d) \in s} f_{td}^p \theta_{qp}. \tag{5.1}$$

We stop solving the master problem when the pricing problem has proven that no more sequences with negative reduced cost exist, which also implies that we obtained the optimal LP solution.

5.3 Pricing problem

In the pricing problem we want to find clusters with negative reduced cost as defined in Equation (5.1), while ensuring that the implicit constraints are satisfied. In this thesis, at most two negative reduced costs columns per cluster are added after solving each pricing problem. The pricing problem is modelled for each cluster $k \in K$ as a Shortest Path Problem with Resource Constraints (SPPRC). For each cluster we construct a directed graph, where each node corresponds to a duty. Duties that are connected by an arc can potentially proceed each other in the same roster sequence.

An example of such as graph is given in Figure 5.2. Starting from the source node s, we keep adding duties until the sink node t is reached. We define a *partial roster sequence* as a roster sequence which did not reach the sink node yet. For example in Figure 5.2 we refer to $[s, 126, 125]$, or simply $[126, 125]$, as a partial sequence.

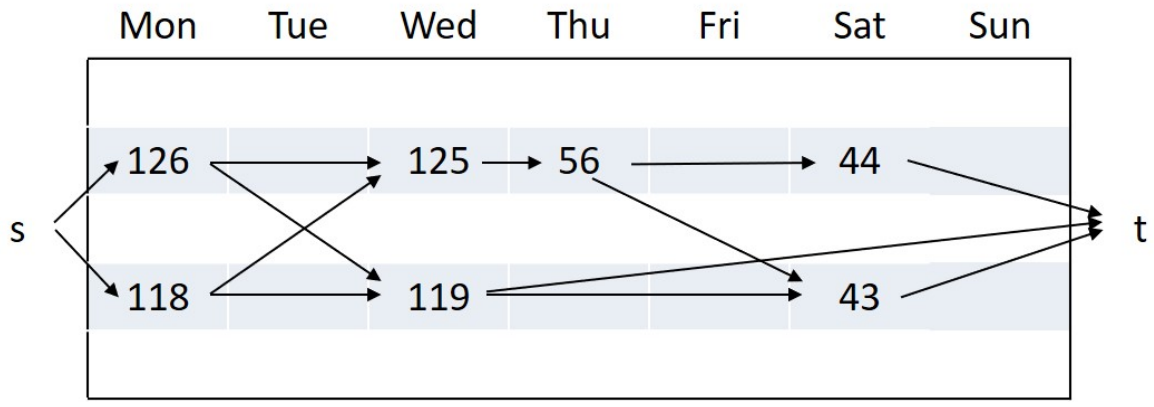

Figure 5.2: Example of a directed graph for the pricing problem.

The reduced cost of a roster sequence can be split into two parts. The first part of the cost comes from the constraints modelled implicitly. It often happens that these *implicit costs* cannot be associated with a single arc, but are the result of a specific series of duties. These costs can

be incorporated using *resource constraints*. Each roster constraint p is related with a resource consumption f_{td}^p and a consumption limit b_p. Violation of the consumption limit is bounded by $\delta_q \in \Delta_q$. The second part corresponds to the dual multipliers of the explicit constraints. These *explicit costs* can be allocated to each arc of the directed graph.

The SPPRC is quite common in the literature of vehicle routing, crew scheduling and crew rostering. The following two types of approaches are often used to solve it to optimality: a *breadth first* labelling algorithm using dynamic programming or a *depth first* algorithm based on completion bounds (Dumitrescu and Boland, 2003). The breadth first strategy is often preferred when two labels can easily be compared, such that one of them can be pruned. On the other hand, the depth first approach is often used to quickly obtain feasible solutions (Lozano et al., 2016). In this thesis, we select the depth first approach, since there is no clear preference for which partial sequence will be the best. Depth first approaches to solve the SPPRC have been successfully applied before, see for example Grötschel et al. (2003) or Breugem (2020).

The pseudocode for a depth first SPPRC is outlined in Algorithm 1. In the initialisation, the partial sequence is created containing only the source node. Furthermore, the resources are updated and checked for any violation. The algorithm is depth first because of the recursive call to the method FINDROSTERSEQUENCE. In each step, it tries to add a duty if the resources are sufficient. The method either terminates when the sink is reached or if the current sequence is infeasible. The methods referring to completion bounds are explained in the next sections.

As mentioned before, *completion bounds* are often used in combination with a depth first approach. Given a partial sequence, this completion bound shows the maximum reduced cost that can still be attained from this point. In case the completion bound is worse than the best reduced cost found until now, it is not fruitful to continue the search using this partial sequence, which is therefore pruned. In this way, we try to detect early on which partial sequences are worthwhile to consider. A common way to compute these bounds is using Lagrangian relaxation (Dumitrescu and Boland, 2003). However, in this thesis the completion bounds are computed using a set of rules based on domain knowledge. Breugem (2020) successfully applied this approach when solving the CCRP. Furthermore, we incorporate these bounds in two different settings: forward and backward completion bounds.

Algorithm 1 Shortest Path Problem with Resource Constraints

Input: directed graph
Output: optimal path (roster sequence)

```
1: allSequences ← {}
2: lowerboundRC ← ∞
3: sequence ← sourceNode
4: update resources
5: FINDROSTERSEQUENCE(sequence)

6: procedure FINDROSTERSEQUENCE
7:     boundOutput ← {}
8:     for duty in dutyList do
9:         sequence ← duty
10:        update resources
11:        if duty = sinkNode then
12:            allSequences ← sequence
13:            update lowerboundRC
14:            boundOutput ←BACKWARDCOMPLETIONBOUNDCALCULATE
15:        else
16:            if INFEASIBLE or FORWARDCOMPLETIONBOUND then
17:                boundOutput ←BACKWARDCOMPLETIONBOUNDCALCULATE
18:            else
19:                boundOutput* ←FINDROSTERSEQUENCE(sequence)
20:                dutyList ← BACKWARDCOMPLETIONBOUNDPRUNE(boundOutput*)
21:            end if
22:        end if
23:    end for
24: end procedure
```

5.3.1 Forward completion bounds

Forward completion bounds are based on the look-ahead principle. Given a partial sequence, we want to estimate how much further the reduced cost can decrease. For example, in Figure 5.3 we have a partial sequence A: $[126, 125]$, which ends on Wednesday. Using domain knowledge we can come up with a number of (best case) scenarios to calculate the lowest possible reduced cost for this particular partial sequence.

These bounds can be recomputed every time a duty is added to the partial sequence, which happens on line 16 in Algorithm 1. This bound consists of the current RC, a minimum explicit cost (based on the dual multipliers) and a minimum implicit cost. This calculation is shown on line 1 in Algorithm 2. If the lowest possible RC is still higher than the lower bound found until now, then it is clear that this partial sequence can be safely pruned.

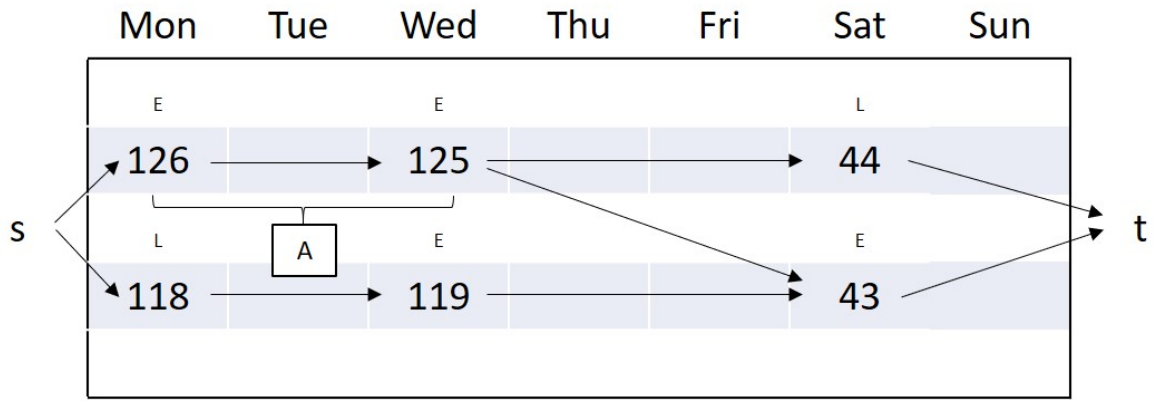

Figure 5.3: Forward completion bound example. Given the partial sequence A, we can calculate a lower bound on the remaining reduced cost to the sink.

The domain knowledge to compute these lower bounds can for example be incorporated as follows. Suppose we have a partial sequence from Monday until Saturday and a duty is scheduled on Saturday, then we know that this sequence cannot contain a Red Weekend. Therefore we can ignore the potential explicit cost of having a Red Weekend. A similar thought process is possible for all other explicit and implicit costs, leading to a final lower bound for the partial sequence.

Algorithm 2 Forward completion bounds

Input: $sequence$
Output: $true$ if $sequence$ should be pruned

1: $lowestPossibleRC \leftarrow currentRC + minExplicitCost + minImplicitCost$
2: **if** $lowestPossibleRC > lowerboundRC$ **then**
3: **return** $true$
4: **end if**

In our implementation, both the minimum explicit and implicit cost are are calculated based on a set of rules. Furthermore, we differentiate between calculating the entire bound in every step (after adding a single duty) or only calculating the bound once. In case we only calculate the bound once, we have to compute a bound that holds for any partial sequence, leading to weaker bounds. So there exists a trade-off between spending more time on calculating a strong bound for each individual sequence and spending less time for a weaker bound. In order to quantify this trade-off, we naturally consider the following four forward strategies.

- FS1: Calculate the explicit and implicit cost in every step.
- FS2: Calculate the explicit cost once and calculate the implicit cost in every step.

- FS3: Calculate the explicit cost in every step and calculate the implicit cost once.
- FS4: Calculate the explicit and implicit cost once.

5.3.2 Backward completion bounds

Backward completion bounds are quite similar to its forward counterpart, except that we now reverse the thought process. Given that we arrive at the sink with sequence A, we can compute the exact RC. This information could help us to prune similar sequences that have not reached the sink yet. This idea is similar to the so-called pulse algorithm proposed by Lozano et al. (2016), to solve the elementary SPPRC. In their method they either prune on basis of infeasibility, bounds or roll-back. The last pruning method decides whether it is still worthwhile to continue the last choice made (e.g. which subtree to explore). A *subtree* is defined as all available duties that can still be added (in a fixed order) to a partial sequence. The difference with the method proposed in this thesis is that we actually do continue to explore the chosen subtree, but we use the information found to prune similar subtrees that are still unexplored.

The intuition behind these backward completion bounds is given in Figure 5.4. If we have already reached the sink using sequence A, then we know what the RC are for the sequence $[126, 125, 44]$ (and $[126, 125, 43]$). We also know that partial sequence B is going to be extended to the sequence $[126, 119, 44]$ (and $[126, 119, 43]$). Clearly, the sequences obtained from A are quite similar to the sequences from B, meaning that their RC costs should not differ too much. Using this information we can compute what the potential RC of partial sequence B is and prune B if necessary.

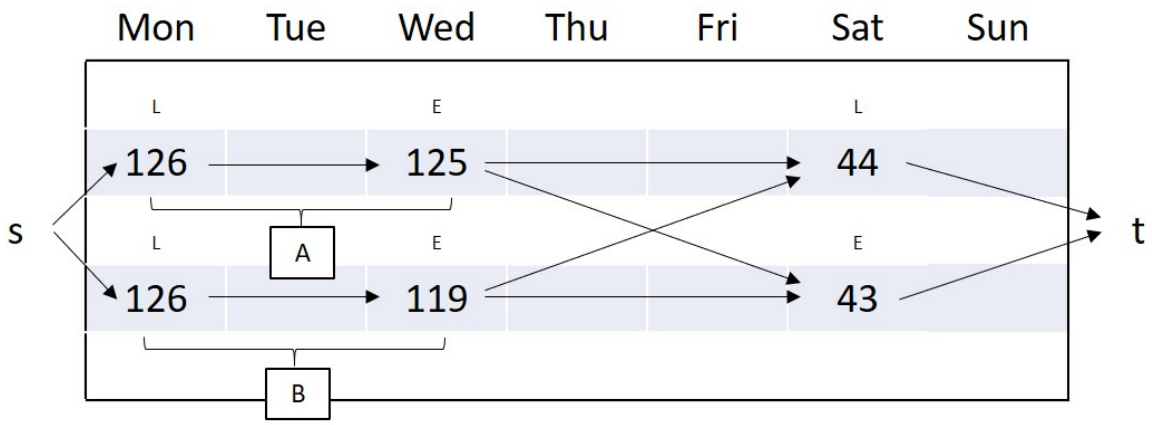

Figure 5.4: Backward completion bound example. Given that we reach the sink from partial sequence A, we can compute a bound for partial sequence B.

To be specific, given RC_A, the reduced cost of sequence A, and the currently lowest reduced cost RC^*, we first calculate the difference between the two: $\tau = RC_A - RC^*$. Here, τ denotes how much the RC of sequence A needs to decrease in order to find a sequence corresponding to the lowest RC. Next, we define I_A as the set of sequences that are almost identical to sequence A. We say that a sequence $i \in I_A$ if they are exactly the same except for a single duty, however, this duty must be of the same type (early, late or night) as in sequence A. When considering all sequences $i \in I_A$, we define $\kappa = \max_{i \in I_A}\{RC_A - RC_i\}$. Therefore, κ can be interpreted as the maximum difference between the reduced cost of sequence A and the reduced cost of similar sequences. Therefore, we can use the comparison between the values of τ and κ to prune similar sequences. That is, if $\tau < \kappa$ then we do not prune the sequences $i \in I_A$. Note that $\tau < \kappa$ implies that $RC^* > \min_{i \in I_A}\{RC_i\}$. The calculation is performed whenever a sequence reached the sink or is pruned, which happens on lines 14 and 17 in Algorithm 1 when Algorithm 3 is called.

Algorithm 3 Backward completion bounds Calculate

Input: sequence A
Output: $true$ if a sequence similar to A can improve the current $lowerboundRC$

1: $\kappa \leftarrow \max_{i \in I_A}\{RC_A - RC_i\}$
2: $\tau \leftarrow RC_A - RC^*$
3: **if** $\tau < \kappa$ **then**
4: **return** $true$
5: **end if**

Note that we do not state that we can prune the sequences, the reason for this is that we do not actually store all the possible sequences in memory. We can only prune on basis of a duty. In order to prune a specific duty, we need to have more information. Suppose now that we have already reached the sink with sequences $A_1, ..., A_n$, such that the sets of $I_1, ..., I_n$ are known and their corresponding values for $\tau_1, ..., \tau_n$ and $\kappa_1, ..., \kappa_n$ have been computed as well. Now, assume we have a partial sequence B. We can find which similarity sets I contain the sequence B and what the corresponding values of τ and κ are. Thus, if $\tau_j > \kappa_j$ for all sets that satisfy $B \in I_j$, then $RC_B > RC^*$. This implies that partial sequence B can never be lower than the optimal reduced cost found until now. In other words, the partial sequence can be safely pruned.

The pruning of a partial sequence B happens on line 20 of Algorithm 1. At this point in the algorithm we have obtained the relevant information on the sequences $A_1, ..., A_m$ and their

similarity sets $I_1, ..., I_m$. The relevant information can be summarised by the pairs (τ_j, κ_j) : $B \in I_j, j = 1, ..., m$, where $m \leq n$. Since we are using a depth first, the pruning only works locally (e.g. for the subtree starting from partial sequence B), therefore we use m instead of all n similarity sets. This implies that we do not need to consider the similar sequence A_j, where $j > m$, if the dissimilarity happens before the current duty.

Consider again the example in Figure 5.4. For partial sequence B: $[126, 119]$ we do not need to have information on sequences that differ before the current duty. For example, the sequence $[124, 119, 43]$ is not relevant for pruning partial sequence B, since the duties 126 and 119 are already fixed. Therefore, we only consider a subset of m sequences, instead of all similar sequences n. This step is shown in Algorithm 4, where we consider all m sequences that are similar to the current sequence B. We iterate over all m options, and if for least one option we might improve the lower bound we stop. If none of the m sequences leads to an improvement we can safely prune the similar duties.

Algorithm 4 Backward completion bounds Prune

Input: $boundOutput$ obtained from BACKWARDCOMPLETIONBOUNDCALCULATE
Output: remove similar duties from the $dutyList$

```
for j = 1, ..., m do
    if boundOutput(j)=true then
        return
    end if
end for
prune all duties similar to current duty
```

Feasibility condition

Of course, the above procedure only works if the subtree belonging to sequence A is exactly the same as the subtree of sequence B. If on the contrary, the subtree of B contains sequences not considered by A, then the results stated above no longer hold. This situation is visualised in Figure 5.5, where partial sequence A does not always reach the sink, such that we do not have enough information available to prune partial sequence B.

In our case, we only have to consider items 1 and 2 from the CLA for our implicit feasibility check. Item 1 can be split into two parts. The first part ensures that the break after each duty is long enough (12 or 14 hours). For instances from NS this always results in a similar subtree for A and B (e.g. by changing a duty of a similar type the break is still sufficient). The second

part states that the minimum rest time after three or more consecutive night duties should be 46 hours. This clearly complicates matters, since the duty that is different between A and B might be a night duty that may or may not lead to a sufficient break. A similar situation might occur for item 2, which states what the length of a series of rest days should be. For these infeasible sequences, we are still interested in computing a lower bound on the RC, such that we might be able to prune partial sequence B.

Suppose that we have the (feasible) partial sequence A, for which we can compute a lower bound. Afterwards we add an additional duty leading to partial sequence C. In case C is infeasible, then the easiest way to obtain a valid lower bound is by reusing the previous lower bound from A. Graphically it is easy to see that if a valid lower bound was found for A, this must always be a valid lower bound for C as well (see Figure 5.6).

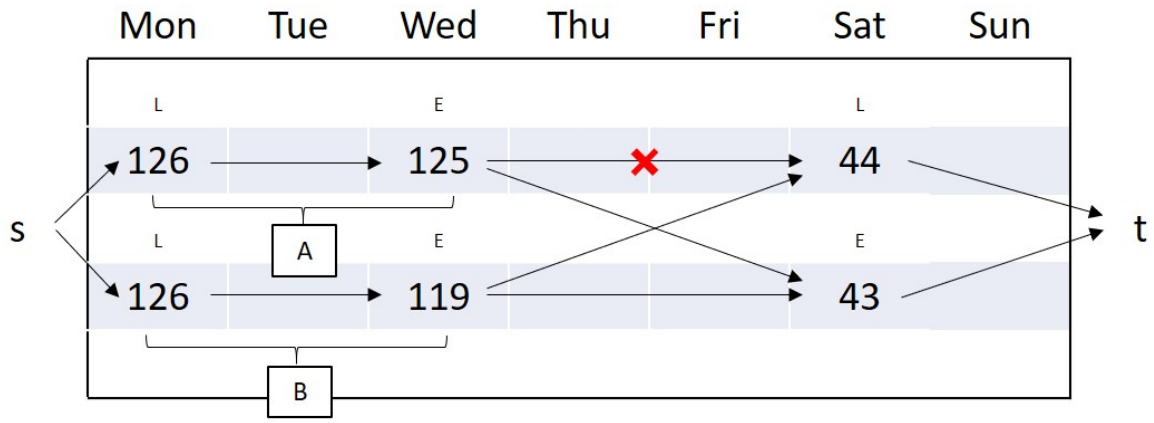

Figure 5.5: We can only compare partial sequences A and B if their subtrees are similar. In this specific example we are not allowed to prune sequence B, because duty 44 cannot be scheduled after duty 125.

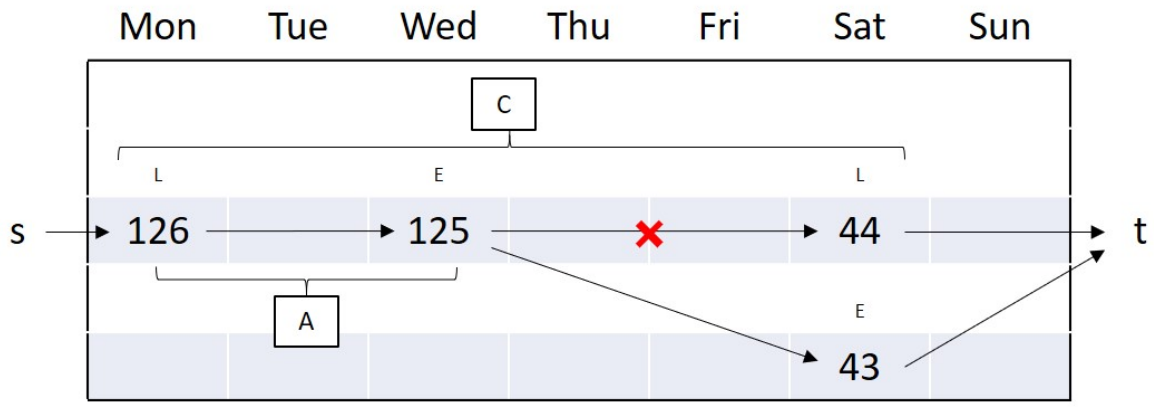

Figure 5.6: If partial sequence C is infeasible, we can still use the valid lower bound of partial sequence A, which has been calculated before.

5.3.3 Pricing Problem Heuristic

The algorithm using backward completion bounds can easily be changed into a heuristic procedure. Instead of calculating the threshold κ, we set the value ourselves. By choosing a lower value, more duties are pruned. However, this also means that we can no longer prove that the root node is solved to optimality.

5.4 Obtaining integer solutions

Solving the column generation as described above only leads to an LP solution of the root node. In order to obtain an actual roster, it is necessary to convert the LP solution into an integer solution. There are several methods to perform this conversion. We first consider two heuristics in which parts of the roster are fixed in an iterative process. In case there are still duties unassigned, we apply a local search heuristic.

It can happen that the found integer solution has a worse objective value compared to one of the start solutions. In that case, we use the best start solution instead.

5.4.1 Integer heuristic

The following two heuristics are used to create an integer solution from the LP solution.

- IH1: Fix the column with the largest value, then solve the master problem again and repeat.
- IH2: Fix the column with the largest value, then solve the master and pricing problems again until no more columns with reduced cost are found and repeat. This is also known as *dive-and-fix* in the literature.

These heuristics can be improved by fixing more than one column if applicable. The number of columns should not be too small, leading to a large number of iterations. On the other hand, fixing a large number of columns at the same time might lead to poor integer solutions.

Furthermore, when choosing which column to fix, we might want to prioritise columns with more duties and sequences containing duties on Friday, Saturday and Sunday. If the heuristic contains these extra weights we refer to it with a star (e.g. IH2*). We suspect that these weights makes it easier to find a feasible solution, since scheduling Red Weekends is often a bottleneck.

5.4.2 Local search

Finally, it might happen that after applying any of the above named strategies not all duties are covered. In that case, the heuristic is followed up by one of the following local search methods.

- FH1: Assign each duty to the first feasible place in a roster sequence.
- FH2: First apply FH1, then use a destroy and repair procedure.

The first method places the unassigned duty into the roster as fast as possible, without taking into account the objective value. We prioritise the assignment of duties to weeks that contain the most days off. This is to prevent an unbalanced roster in which certain weeks are totally full and other weeks are almost empty.

The first method might not be able to find a solution in which all duties are covered, since most of the roster is fixed. This problem can be circumvented by the second method, which uses a destroy method. We first remove a few roster sequences and unassign the corresponding duties. Afterwards, we repair the solution by applying a dive-and-fix procedure (similar to IH2). If the number of unassigned duties is less than before, we retain the obtained solution, otherwise we reverse the changes. We continue this procedure until all duties are covered or a stopping condition is met.

Note that neither the heuristic nor the additional local search can guarantee a solution in which all duties are covered.

Chapter 6

Computational experiments

In this chapter, we decide which strategies and parameters to choose for the solution approach by means of computational experiments. The order in which each part is discussed follows a similar structure as the previous chapter (see Figure 5.1). That is, we first determine whether start solutions should be added to the master problem. Afterwards, we investigate how the pricing problem can be improved. Finally, several heuristics to find integer solutions are compared.

The master problem is solved using the commercial solver CPLEX 12.1. All experiments are performed using a 1.8 GHz Intel Core i7 processor.

6.1 Description of the instances

In total we consider 12 different instances of train conductors from NS, which are summarised in Table 6.1. We distinguish between two types of instances: small and large.

In this chapter, we mostly focus on the small instances (1 - 6) when evaluating different strategies in the solution approach. These instances are suitable, because they are easier to solve in terms of computation time, since a single roster group is present with at most 12 employees. Furthermore, these instances are representative of the larger ones when looking at the distribution of the duty types. There are balanced instances (1, 2, 3 and 6) that contain a similar amount of early and late duties and the other instances (4 and 5) only contain a single duty type.

The large instances (7 - 12) are more difficult to solve. These instances contain more employees and sometimes also more roster groups compared to the small instances. After having fixed the strategies and parameters, most of these instances are used in the upcoming sensitivity analysis in Chapter 7. The distribution of duty type also differs per instance. Instance 8 contains

primarily early duties, whereas instance 9 contains mostly late duties. These are likely more difficult to solve, because there is less flexibility. On the other hand, the backward completion bounds may be more effective, since more duties can be pruned at the same time. Instances 10 and 11 are more balanced in terms of duty types. The largest instance (12) corresponds to the crew base in Amersfoort.

Table 6.1: The number of roster groups, employees and duties per instance.

Type	ID	Groups	Employees	Duties			
				Early	Late	Night	Total
Small	1	1	8	17	14	0	31
	2	1	12	23	20	2	45
	3	1	12	22	23	2	47
	4	1	12	0	40	5	45
	5	1	12	0	40	4	44
	6	1	12	24	17	2	43
Large	7	1	15	55	0	0	55
	8	3	39	102	37	4	143
	9	3	32	17	94	9	120
	10	3	39	77	63	7	147
	11	3	32	64	51	4	119
	12	7	83	141	154	15	310

6.2 Start solution

As mentioned earlier, we consider two different strategies to obtain a set of start solutions: the RIM and the GRASP. First, we evaluate each method individually compared to a baseline without any start solution. Afterwards, we compare which of the two methods is more suitable.

6.2.1 RIM

Table 6.2 shows the results when using the RIM for the small instances. The first row corresponds to the baseline where no start solution is used. The subsequent rows denote the number of start solutions that are made and all of them are added to the master problem.

The table illustrates that the average computation time and the number of iterations of the master problem already decrease by a factor of four after adding only 10 start solutions. This means it is beneficial to include several start solutions when initialising the column generation approach. Furthermore, the column containing the start solution time shows that obtaining these start solutions is not difficult, 50 start solutions can be found on average within 2 seconds.

We observe a decrease in computation time when just a few start solutions are added. This

can also be concluded when looking at Table 9.1 in the Appendix, where the individual computation times are shown. For example, instance 5 takes almost 90 minutes to solve when no start solution is present, while it only takes around 10 minutes after adding 10 start solutions. Furthermore, for instance 5 having 10 start solutions is better compared to 20, 30 or 40 start solutions. Given that a decreasing relation is found between the computation time and the number of start solutions for most other instances, it is probable that the solver accidentally found a few good roster sequences for instance 5 when only 10 start solutions were present. We also see that instance 4 and 5 always have the longest computation time compared to the other instances. This is as expected, since these two instances only contain late duties.

Table 9.2 in the Appendix reports that the number of iterations of the master problem often decrease on the individual level when more start solutions are added. Consequently, it can be concluded that adding more start solutions is beneficial, since it decrease the total computation time as well as the number of iterations. Accordingly, 50 start solutions is deemed the most suitable for the RIM.

Table 6.2: Average results over small instances when adding start solutions using the RIM. Root node time includes time spend on solving the master problem and pricing problem.

Start solutions	Total time (s)	Start solution time (s)	Root node time (s)	Columns	Master iter.
0	2186.63	-	2186.63	3702.00	161.33
10	388.16	0.35	387.81	924.83	41.50
20	477.75	0.74	477.00	810.67	36.33
30	474.17	1.18	472.98	776.00	34.83
40	472.30	1.66	470.64	693.33	31.17
50	416.36	2.07	414.30	651.17	29.50

6.2.2 GRASP

In the first step of the GRASP, we choose a random duty out of a candidate list containing 5 duties. The second step, in which the best possible swap is determined, is repeated at most 30 times. In general, we observe that less than 30 iterations are needed to find a local optimum.

Similar to the RIM, Table 6.3 shows that also for the GRASP is it beneficial to add start solutions, as this leads to an overall decrease in the time to solve the root node as well as a decrease in the number of iterations.

Based on the total computation time, it seems most beneficial to have 20 start solutions. This is confirmed when examining the individual computation times in Table 9.3 in the Appendix.

Based on the total computation time, we indeed see that for most instances 10 or 20 start solutions is the preferred choice.

However, when considering the number of iterations in the root node we find in Table 6.3 that 50 start solutions results in the least amount of master iterations. This can also be seen on the individual level in Table 9.4 in the Appendix. Most instances use the least amount of iterations when 40 or 50 start solutions are added.

The reason why the total computation time is increasing, while the number of iterations is decreasing is explained by Table 6.3. Adding more start solutions actually leads to less computation time in the root node. However, finding the start solutions itself is quite costly. For 50 start solutions around 30% of the total computation time is spent on finding start solutions.

The best setting for the GRASP seems to be either 20 or 50 start solutions, depending on the total computation time or number of iterations respectively.

Table 6.3: Average results over small instances when adding start solutions using the GRASP. Root node time includes time spend on solving the master problem and pricing problem.

Start solutions	Total time (s)	Start solution time (s)	Root node time (s)	Columns	Master iter.
0	2186.63	-	2186.63	3702.00	161.33
10	316.55	21.70	294.85	657.33	29.67
20	304.29	49.29	255.00	538.83	24.50
30	366.89	83.33	283.56	505.33	22.83
40	385.77	104.76	281.01	453.00	20.67
50	395.08	128.30	266.78	443.33	20.33

6.2.3 Comparison GIM and GRASP

When comparing the total computation time when using the GIM and GRASP we see that the GRASP often performs better, given the same amount of start solutions. However, when 50 start solutions are used, the difference between the two methods is on average only 20 seconds. This is because the RIM only needs on average 2 seconds to find all 50 start solutions, whereas the GRASP requires more than 2 minutes. This difference is then balanced out in the time to solve the root node.

The GRASP is able to improve the running time of the root node, because the start solutions found have much better objective values. Table 6.4 shows the average best found start solution for both the RIM and the GRASP. A large difference in objective values is expected, since the RIM does not take the objective into account when creating the start solutions.

Even though for small instances the time spent on calculating good start solutions outweighs the costs, we expect for larger instances that performing the GRASP might cost too much time compared to the reduction in the root node time. That is, the GRASP might not scale well for instances with multiple roster groups. Furthermore, we see that using a simple RIM we can already obtain a large decrease in the total computation time. Therefore, we continue using the RIM with 50 start solutions.

Table 6.4: Comparison of the RIM and GRASP on time spend creating all start solutions and the objective of the best start solution. Average computation time and best start solution for the small instances.

Start solutions	RIM		GRASP	
	Start solution time (s)	Best start solution	Start solution time (s)	Best start solution
10	0.35	5022.29	21.70	1687.49
20	0.74	1688.95	49.29	20.74
30	1.18	1688.86	83.33	20.71
40	1.66	1688.84	104.76	20.71
50	2.07	1688.71	128.30	20.71

6.3 Pricing problem

To solve the pricing problem to optimality, we consider both forward and backward completion bounds. Additionally, we show the performance of the pricing problem heuristic.

6.3.1 Forward completion bounds

Table 6.5 shows the results of the proposed forward strategies compared to a benchmark which does not make use of any bounds. Note that only the time to solve the root node is reported, because the start solution strategy is fixed throughout the comparison. The pricing problem is always solved to optimality, irrespective of the chosen strategy. As a result, the number of columns and iterations stays the same.

Interestingly, three out of the four strategies perform worse in terms of computation time compared to the benchmark. These three strategies are actually all able to prune some (partial) sequences before reaching the sink. Therefore, the number of times the sink is reached is decreased. However, calculating these forward completion bounds itself is also time consuming. In these three cases it seems that in the time it takes to prune a subtree, we could already explore that subtree instead.

Only strategy FS2, which calculates the explicit cost once and the implicit cost in every step, is able to outperform the benchmark. The difference in computation time is more than 150 seconds on average. This means that even though we often calculate the implicit cost, this does not slow down the pricing problem. Therefore, it is surprising that FS4, which calculates both the explicit cost and implicit cost once, is performing worse than FS2. We might expect the average time to solve the root node for FS4 to be similar as well, since even less calculations are needed. As this is not the case, we expect that the bounds calculated by FS2 are stronger than the ones obtained by FS4. Having more accurate bounds lead to more pruning or earlier pruning in the subtree. Therefore, we continue with strategy FS2 to compute the forward completion bounds

Table 6.5: Comparison of several forward completion bound strategies. Average results on small instances.

Strategy	Root node time (s)	Columns	Master iter.
-	414.30	651.17	29.50
FS1	804.32	651.17	29.50
FS2	259.41	651.17	29.50
FS3	525.66	651.17	29.50
FS4	585.16	651.17	29.50

6.3.2 Backward completion bounds

Table 6.6 compares the benchmark containing only forward completion bounds with a strategy where both forward and backward completion bounds are used. Both options use strategy FS2 to compute the forward completion bounds. The total computation time to solve the root node drops by more than 50 seconds on average. Interestingly, the number of columns and iterations needed decreases slightly when using backward completion bounds. This can be attributed to the fact that the backward strategy prunes all similar duties that cannot improve the current lower bound. Therefore, this method might also prune sequences that do have a negative reduced cost which is higher than the current lower bound. These pruned sequences are only added in the next iteration, in case they still have a negative reduced cost.

6.3.3 Pricing problem heuristic

The previously discussed backward strategy still solves the pricing problems to optimality and therefore also solves the LP relaxation to optimality. However, we can also choose a fixed

Table 6.6: Comparison of a strategy with and without backward completion bounds. Average results on small instances.

Strategy	Root node time (s)	Columns	Master iter.
-	259.41	651.17	29.50
Backward	201.84	648.83	29.33

threshold instead of calculating it exactly. This leads to a pricing problem heuristic for which the results are shown in Table 6.7.

The table shows that for decreasing values of κ, the time to solve the root node decreases as well, which drops from an average of 93 seconds to an average of 2 seconds. This is as expected, since a lower κ implies that we are more likely to prune a sequence, such that less sequences have to be explored. When looking at the relation between κ and the lower bound or the number of iterations there does not seem to be a clear pattern. This is not unusual, since the presented results are obtained from a heuristic, so a higher value of κ does not necessarily imply a better lower bound.

The lowest computation time for solving the root node is achieved when setting κ to its minimum value of 0, corresponding to an average of 2 seconds. The difference between the found lower bound and the actual lower bound is 0.09%. This computation time is also much lower compared to the time from the backward strategy shown in Table 6.6. The backward strategy takes on average a factor 100 more time, although it does find the optimal solution of the LP relaxation.

Based on the fact that $\kappa = 0$ results in the lowest computation time and the found lower bound is quite close to the actual lower bound for these instances we continue with $\kappa = 0$.

Table 6.7: Average results over small instances when using parameter κ in the pricing problem heuristic. For each parameter value the percentage difference between the found lower bound and the actual lower bound of 20.18 is reported.

κ	Lower bound*	Difference (%)	Root node time (s)	Columns	Master iter.
0.10	20.18	0.00	92.97	685.67	31.17
0.09	20.18	0.00	93.27	681.67	31.33
0.08	20.18	0.00	96.50	712.33	32.67
0.07	20.19	0.04	74.09	700.00	31.50
0.06	20.18	0.00	72.13	757.17	35.00
0.05	20.18	0.00	62.81	748.17	34.33
0.04	20.18	0.00	47.45	694.67	31.50
0.03	20.18	0.00	39.15	765.17	34.50
0.02	20.18	0.00	29.17	822.33	37.67
0.01	20.18	0.00	22.81	937.67	42.83
0.00	20.20	0.09	2.03	480.67	25.00

$*$ The lower bound of the LP solution found using a pricing problem heuristic.

6.4 Obtaining integer solutions

In this section, we introduce several methods to find integer solutions, using as basis the previously obtained solution of the LP relaxation. When comparing several heuristics to obtain an integer solution, we always start with the same LP solution. Therefore, the time to solve the root node is not included in the upcoming comparisons.

6.4.1 Integer heuristic

For both strategy IH1 and IH2, we fix at most 5 roster sequences in each iteration. Additionally, IH2* denotes the strategy where more weights are given to sequences containing a busy weekend. Table 6.8 shows that IH1 takes on average the least amount of time, while IH2 and IH2* take around 3 to 4 seconds. Furthermore, the table shows that all three strategies find the same integer solution for all instances. This is caused by the fact that none of the integer solutions is able to outperform the best start solution.

Table 6.9 compares the solutions found by the integer heuristic with the best start solution. For these instances none of the integer heuristics are able to find a roster where all duties are covered. This can be seen from the objective, which is higher than 10,000 (corresponding to a penalty of rejecting a duty). The best start solution always finds a roster containing all duties, except for instance 2.

When comparing the solutions found by the integer heuristics, we see that IH1 performs

Table 6.8: Average results over small instances using different integer heuristics. The best integer objective is reported, as well as the number of rejected duties. The final column denotes the percentage of cases that the integer heuristic found a better integer solution compared to the available start solutions.

Strategy	Objective	Lower bound	Gap (%)	Integer time (s)	Rejected duties	Better than start solution (%)
IH1	1688.71	20.18	7813.08	0.25	0.17	0.00
IH2	1688.71	20.18	7813.08	3.10	0.17	0.00
IH2*	1688.71	20.18	7813.08	4.11	0.17	0.00

worse compared to IH2 and IH2*. This is as expected, since IH2 and IH2* make use of a dive-and-fix procedure. After having fixed a set of roster sequences, this information is used to generate new roster sequences that might fit better in the current roster. Even though IH2 and IH2* have a longer computation time, we continue with these two heuristics since they obtain a better integer solution. Both methods are extended using a local search in the next section, afterwards the best performing method is chosen.

Table 6.9: Integer objectives obtained from the integer heuristic compared to the best start solution.

ID	Start solution	IH1	IH2	IH2*
1	15.38	50015.92	10014.62	10014.87
2	10023.67	90023.92	50023.26	60023.48
3	23.07	130024.63	20022.04	40022.56
4	23.27	110024.03	30022.88	30022.86
5	23.60	60024.01	10022.64	20022.73
6	23.30	120024.87	40022.71	40023.40
Average	1688.71	93356.23	26688.02	33354.98

6.4.2 Local search

Based on the previous results we select the two best performing integer heuristics. Both these methods can be extended by two different types of local search methods: FH1 and FH2. This leads us to the following four possibilities which are shown in Table 6.10. The reported integer times include the time spent on both the integer heuristic and the local search parts. For FH2 we destroy at least 10% of the roster sequences in each iteration and repair this by applying a dive-and-fix in which a single roster sequence is added each time.

The table shows that without the local search it takes 3 to 4 seconds on average to perform the integer heuristic. With local search the running time is at most 20 seconds on average. The strategies using FH1 instead of FH2 have the lowest computation time as expected, since the destroy and repair method is time consuming. Furthermore, we see that the local search does

improve the solutions. For all strategies, we see that on average in more than 50% of the time we outperform the best start solution. This is a substantial improvement with respect to the 0% we found earlier when no local search is used. Strategy IH2-FH1 finds the lowest objective on average. This is also the only strategy that finds for all instances a roster in which all duties are covered. This suggests that IH2-FH2 is the preferred strategy.

Table 6.10: Average results over small instances using different integer heuristics and local searches. The best integer objective is reported, as well as the number of rejected duties. The final column denotes the percentage of cases that the integer heuristic found a better integer solution compared to the available start solutions.

Strategy	Objective	Lower bound	Gap (%)	Integer time (s)	Rejected duties	Better than start solution (%)
IH2-FH1	1688.34	20.18	7811.04	3.27	0.17	50,00
IH2-FH2	21.59	20.18	6.86	17.87	0.00	50,00
IH2*-FH1	1688.38	20.18	7811.26	4.36	0.17	50,00
IH2*-FH2	1688.25	20.18	7810.66	13.82	0.17	66,67

Table 6.11, which shows the objective values on an individual level, tells a different story. Out of the four strategies IH2*-FH attains the lowest objective value on average. This is more important, since it shows that this method is more adept at transforming rosters that do not cover all duties into feasible rosters. For example, the method IH2-FH2, which was the best based on the previous table, still has two instances in which a single duty is not covered and one instance in which four duties are not covered after applying a local search. However, IH2*-FH2 only has three instances for which a single duty is not covered. Being able to cover more duties is more important, when we do not have a start solution in which all duties are covered. Therefore, strategy IH2*-FH2 is used to find integer solutions in the upcoming sections.

Table 6.11: Integer objectives obtained from the integer heuristic with local search compared to the best start solution.

ID	Start solution	IH2-FH1	IH2-FH2	IH2*-FH1	IH2*-FH2
1	15.38	14.71	14.71	14.79	14.79
2	10023.67	10023.07	22.55	10023.27	10023.27
3	23.07	10021.96	40022.83	22.07	22.07
4	23.27	10022.65	10022.53	10022.79	10022.63
5	23.60	10022.64	10022.67	10022.65	10022.55
6	23.30	22.32	22.32	10023.22	22.53
Average	1688.71	6687.89	10021.27	6688.13	5021.31

6.5 Performance of all instances

In the previous sections, the most suitable parameters and method are fixed based on a set of small instances. We initialise the column generation approach with 50 start solution obtained using a RIM. Afterwards, the pricing problem is solved heuristically, while also using forward and backward completion bounds. Finally, the LP solution is transformed into an integer solution by using a dive-and-fix procedure followed-up with a local search. In this section, the performance of the other (mostly larger) instances using these preferred settings are shown. The goal of this section is twofold. Firstly, the most time consuming parts of the solution approach are highlighted. Secondly, the instances 8 - 11 are used as a benchmark for the upcoming sensitivity analysis.

Tables 9.5 and 9.6 in the Appendix provide an overview of the performance of the small instances 1 - 6 based on the preferred settings. In case a roster in which all duties are covered is found (corresponding to objectives lower than 10,000), the actual gap ranges from 5 up to 9%, this shows that the objective of the integer solution is quite close to the LP solution. The latter table shows the time spent on all parts of the solution approach. The fastest time to solve the root node is attained by instances 4 and 5, which were previously the instances with the longest computation time. This is because instances 4 and 5 only contain late duties and in the backward completion bounds all similar duty types are pruned. In case the integer heuristic finds a roster in which all duties are covered, no time is spent on the local search. In case a local search is necessary, we see that it often accounts for roughly 50% of the total computation time. It is also interesting to see that almost no time is spent on solving the master problem. We expect similar results for the larger instances 7 - 12.

Table 6.12 shows that for larger instances the number of columns and iterations of the master problem increases greatly. Small instances use on average 3,000 columns and are often solved within 30 iterations, whereas the largest instance needs more than 145,000 columns and almost 80 iterations. Even though the instance is larger, our solution approach is able to find feasible rosters for all shown instances. The reported gaps are not that large, ranging from 3 up to 15%. Note that the actual gaps can be larger, since the lower bound is found using a pricing problem heuristic.

Additionally, in 4 out of 6 cases the integer heuristic is able to find a better solution relative to the given start solutions, which is also the case for the smaller instances. Table 6.13 shows

that in case both the integer heuristic and the start solution find a solution in which all duties are covered (corresponding to objectives lower than 10,000), the integer heuristic always reports a lower objective value. This shows that the integer heuristic also works well on larger instances.

Table 6.12: Results obtained using the preferred settings. Lower bound and gap are based on the pricing problem heuristic. The best found objective and the number of rejected duties is reported. The last column indicates whether the solution from the integer heuristic is better compared to the start solution.

ID	Objective	Lower bound*	Gap (%)	Columns	Master iter.	Rejected duties	Better than start solution
7	27.80	26.81	3.69	4028	17	0	Yes
8	76.33	68.79	10.97	47125	44	0	No
9	59.68	57.26	4.23	11319	36	0	Yes
10	72.35	68.44	5.72	33032	51	0	Yes
11	64.22	56.23	14.21	22639	42	0	No
12	154.05	146.22	5.35	145251	78	0	Yes

$*$ The lower bound of the LP solution found using a pricing problem heuristic.

Table 6.13: Integer objectives obtained from the integer heuristic the preferred settings compared to the best start solution.

ID	Start solution	Integer heuristic
7	28.95	27.80
8	76.33	10073.66
9	20064.45	59.68
10	77.45	72.35
11	64.22	10059.40
12	10164.65	154.05

Table 6.14 shows how much time is spent on each part of the solution approach. For larger instances, the time to create a start solution increases quite fast, even though we merely use a simple insertion method. For small instances with only one roster group, the start solutions are generally found within 2 seconds. For the instances with three roster groups, the time to find all the start solutions increases to 10 seconds on average. However, when considering the large instance with seven roster groups it almost takes a minute to find all 50 start solutions.

Unsurprisingly, the time to solve the root node, which consists of the master problem and pricing problem time, is low compared to the total time. The low computation time of the master problem is due to the formulation of the problem. The computation time of the pricing problems can be attributed to the pricing heuristic, which prunes similar sequences as early as possible. For the largest instance this leads to a computation time of roughly 60 seconds to solve the root node.

For most methods, a large portion of the time is dedicated to the integer heuristic and the local search. This makes sense, since finding a solution in which all duties are assigned is not trivial. The integer heuristic takes a lot of time, as only 5 roster sequences are fixed at the same time. The local search takes less time than expected. In three cases we are lucky that no local search was necessary at all. In the other three cases, a roster in which all duties are present was found after applying the local search for five minutes.

Table 6.14: The total computation time followed by the time spend on the individual parts of the solution approach when using the preferred settings.

ID	Total time (s)	Start time (s)	Master time (s)	Pricing time (s)	Integer time (s)	Local search time (s)
7	18.42	2.65	0.01	0.60	15.12	-
8	889.14	17.32	0.33	26.80	565.86	278.83
9	170.34	10.99	0.11	24.81	134.38	-
10	467.80	11.02	0.21	48.86	407.53	-
11	387.06	4.44	0.06	10.80	167.83	203.94
12	2917.03	58.47	1.30	61.78	2585.92	209.56

Chapter 7

Sensitivity analysis

In the previous chapter, the most suitable parameters and methods are derived based on a set of small instances. Using these preferred settings as a baseline, we now consider how the quality of the rosters changes when the inputs vary. To be specific, we investigate the implications of adjusting the CLA regulations or the weights in the roster preferences. These sensitivity analyses are only performed on the large instances with three roster groups. If the instances are too small (e.g. consisting of a single roster group), then fairness cannot be assessed properly. The largest instance (12) is not used due to the long computation time.

7.1 Maximum average workload per week

The average workload per week is incorporated as a duty attribute. Meaning that the workload within a roster group should be equally spread over the weeks. However, it also means that between groups the average workload should be similar. The penalty might not always lead to desirable results, such that the workload is not fairly divided over the groups. Therefore, we also consider a hard limit on what the maximum average workload can be per roster group. In the benchmark setting the maximum was 32 hours per week, in this section, we also consider 31 and 30.5[1] hours as maximum working hours per week.

When looking at Table 7.1, we see that the lower bound does not always increase when decreasing the maximum hours per week. This can be explained by the fact that the pricing problem heuristic is not always solved to optimality. Similarly, the integer objective is also in some cases lower when the maximum average hours per week is more restricted. The instances with 31 and 30.5 hours do not seem to be more difficult to solve, since the number of columns

[1] When using 30 hours it is not always possible to find a roster in which all duties are scheduled.

and iterations for the master problem are quite similar as well. It does not seem to be more difficult to obtain rosters where all duties are covered using the integer heuristic with local search, since in 7 out of 8 cases a better solution is found using the integer heuristic. Also, we do not see that adding the restriction leads to longer computation times than before.

Table 7.1: Results obtained when the maximum average working hours is 32, 31 or 30.5 hours per week per roster group. Lower bound and gap are based on the pricing problem heuristic. The best found objective and the number of rejected duties is reported. The last column indicates whether the solution from the integer heuristic is better compared to the start solution.

ID	Max. hour	Objective	Lower bound*	Gap (%)	Total time (s)	Master iter.	Rejected duties	Better than start solution
8	32	76.33	68.79	10.96	889.14	44	0	No
	31	72.42	68.83	5.22	498.24	37	0	Yes
	30.5	76.58	68.85	11.22	544.58	35	0	No
9	32	59.68	57.26	4.23	170.34	36	0	Yes
	31	59.47	57.28	3.83	605.69	35	0	Yes
	30.5	59.51	57.31	3.83	406.66	47	0	Yes
10	32	72.35	68.44	5.71	467.80	51	0	Yes
	31	72.79	68.36	6.47	412.59	47	0	Yes
	30.5	71.63	68.41	4.70	811.29	43	0	Yes
11	32	64.22	56.23	14.21	387.06	42	0	No
	31	59.04	56.22	5.01	175.29	51	0	Yes
	30.5	59.58	56.19	6.03	374.28	42	0	Yes

$*$ The lower bound of the LP solution found using a pricing problem heuristic.

Table 7.2 shows for the three considered maxima the average working hours, how the duties are distributed over the different groups and what the average working hours are per roster group. Changing the maximum does not lead to a change in the distribution of the duties. In most cases, the duty to group size ratio takes a value between 3.4 and 3.9 for all roster groups. However, we do see an effect on the average working hours per week, which is possible since duties differ in length.

When looking at the difference between the minimum and maximum average working hours we observe that under 32 hours the difference is around 2.5 up to 5.5 hours. This means that in the worst case a roster group has to perform 5.5 hours more duties in each week compared to another roster group. This is not desirable, as this difference accumulates over the span of several weeks. For example, if a roster group with size 12 works more than 5.5 hours every week compared to another roster group, this difference accumulates to 66 hours over the whole cyclic roster.

In case the maximum is set to 31 hours, the difference shrinks between 1.5 and 4.5 hours

per week. Finally, considering the case of 30.5 hours, we obtain a difference between 0.5 and 3 hours per week. This is an improvement compared to the benchmark of using 32 hours as maximum. Therefore, we can conclude that adding the average workload per week as duty attribute does not lead to a fair roster in general. Instead, a strict upper limit is needed as well to enforce a fair distribution of the workload.

Table 7.2: Comparison of the number of duties and the average working hours per week per roster group when the maximum average working hours is adjusted.

ID	Group size	32 hours		31 hours		30.5 hours	
		Duties	Working hours per week	Duties	Working hours per week	Duties	Working hours per week
8	15	59	31.82	56	29.96	55	29.26
	12	41	26.31	40	26.39	47	30.48
	12	43	28.44	47	30.69	41	27.47
9	8	31	31.64	30	29.85	31	30.04
	12	43	27.98	47	30.45	46	30.33
	12	46	30.28	43	29.00	43	29.00
10	15	54	29.18	56	29.61	56	29.74
	12	46	29.00	47	31.00	45	29.74
	12	47	31.66	44	29.12	46	30.21
11	8	27	26.94	31	30.81	29	29.41
	12	45	29.29	42	27.25	45	29.55
	12	47	31.54	46	30.99	45	29.62

7.2 Length of a single day off

According to the second part of item 2 from the CLA, the length of a rest day is calculated as 6 hours plus the number of rest days times 24 hours. In this section, we focus on a single rest day, which should therefore be 30 hours or longer. In practice, the crew prefers to have a rest day of 34 hours or more. We consider whether it becomes more difficult to obtain a roster when changing the minimum length of a rest day and how this effects the length of single rest days. For the length of a day off we consider 30, 32 and 34 hours, where the former one is also denoted as the benchmark.

Table 7.3 shows that the lower bounds do not necessarily increase when we reserve more time for a day off. Similar to the previous section, the results are obtained using a pricing problem heuristic, leading to non-exact lower bounds. Also, when considering the number of columns or iterations we do not see any changes. Only 1 out of 8 cases did not find a roster in which all duties are covered. Finally, the number of times that the integer heuristic with

local search finds a better start solution is quite similar to the benchmark. Also the computation times are similar.

Table 7.3: Results obtained when the length of a single day off is 30, 32 or 34 hours. Lower bound and gap are based on the pricing problem heuristic. The best found objective and the number of rejected duties is reported. The last column indicates whether the solution from the integer heuristic is better compared to the start solution.

ID	Hour	Objective	Lower bound*	Gap (%)	Total time (s)	Master iter.	Rejected duties	Better than start solution
	30	76.33	68.79	10.96	889.14	44	0	No
8	32	76.75	68.90	11.41	520.29	39	0	No
	34	77.12	68.82	12.07	557.28	44	0	No
	30	59.68	57.26	4.23	170.34	36	0	Yes
9	32	59.78	57.28	4.37	165.48	36	0	Yes
	34	59.86	57.26	4.54	155.66	32	0	Yes
	30	72.35	68.44	5.71	467.80	51	0	Yes
10	32	72.59	68.40	6.13	252.47	35	0	Yes
	34	10075.65	68.40	14629.72	755.79	41	1	No
	30	64.22	56.23	14.21	387.06	42	0	No
11	32	59.61	56.18	6.10	159.41	33	0	Yes
	34	59.12	56.16	5.27	177.89	44	0	Yes

$*$ The lower bound of the LP solution found using a pricing problem heuristic.

Table 7.4 reports for each roster group the number of times a single day off occurred and what the minimum and average values are of their respective length. In some cases we see a decline in the number of times a single day off occurs (e.g. instance 9 and 11) when the length increases. This is as expected, since it becomes more difficult to schedule a single day off. For instance, it costs less time to schedule two consecutive days off instead of having two separate days off. The former option costs at least 54 hours, while the latter option costs at least 68 hours (if the length of a single day off is at least 34 hours). As a side effect we observe that some roster groups only have two or more consecutive days off.

The table also shows that the minimum and average length of a single day off increases when the minimum required length changes. In the benchmark setting we observe quite some roster groups where a day off is less than 30.5 hours, which are not attractive for the crew members. When the minimum requirement is set to 32 hours, we only observe two roster groups where the minimum is lower than the preferred 34 hours. Finally, when considering the minimum requirement of 34 hours it seems that for most instances it is possible to satisfy the preferences of the crew. The exception is instance 10, for which no roster was found which covered all duties (see Table 7.3).

From the above, we can conclude it is possible to meet the demands from the train crew to a certain extent. For most instances, finding a roster that covers all duties does not become more difficult when changing the definition of a single day off. So, it is recommended to increase the minimum length of a day off at least to 32 hours.

Table 7.4: Comparison of the minimum and average length of a single day off, when the length of a single day off is adjusted.

ID	Group size	30 hours			32 hours			34 hours		
		obs	min	mean	obs	min	mean	obs	min	mean
8	15	8	30.30	38.27	13	33.05	37.97	10	36.13	41.84
	12	8	31.37	38.29	7	37.32	41.10	9	34.20	38.46
	12	8	32.12	39.97	8	35.88	40.52	5	38.37	42.85
9	8	6	33.82	41.42	2	44.83	49.38	3	35.85	38.22
	12	3	36.50	41.01	0	-	-	2	42.92	45.60
	12	2	33.98	38.04	4	35.35	41.36	3	35.07	41.98
10	15	6	31.42	46.00	9	32.85	42.36	6	34.15	41.83
	12	3	30.20	40.43	5	38.38	46.29	8	36.43	41.66
	12	3	43.93	46.08	2	47.28	48.90	7	34.77	38.99
11	8	9	30.20	38.54	0	-	-	0	-	-
	12	6	34.08	37.74	4	42.90	44.47	5	35.50	41.91
	12	16	30.02	39.53	4	38.85	46.17	6	36.38	45.16

7.3 Pattern preferences

A roster can often be broken down into a series of recurring patterns. For instance, patterns such as E - R - L or L - R - R - E are common in most rosters. Crew members at NS often prefer rosters where the days off are scheduled in an attractive way. Therefore, we only consider three patterns that include at least one day off: item 6, 7 and 10 from the roster preferences. We abbreviate these patterns by RP6, RP7 and RP10, respectively. In this section, we investigate whether we can influence the number of times these patterns occur. The first pattern, RP6, corresponds the patterns E - R - L and E - R - N. Similarly, the patterns L - R - R - E and N - R - R - E correspond to RP7. Finally, the pattern R - duty - R belongs to RP10. For each roster preference, we increase their respective parameter weight by a factor of 10. This leads to the three options shown in Table 7.5, which are compared with the benchmark (option 0). Only the absolute value of the parameter is shown in the table. The patterns denoted by RP6 and RP7 are preferred, so the actual parameter is a (negative) reward. The pattern RP10 is not preferred, so this corresponds to a (positive) penalty.

Table 7.5: Overview of the relative parameter weights for the occurrence of each pattern.

Specification	RP6	RP7	RP10
0	0.1	0.1	0.1
1	**1.0**	0.1	0.1
2	0.1	**1.0**	0.1
3	0.1	0.1	**1.0**

Table 7.6 shows that the lower bound and the objective differ when the relative weights are changed. This is quite logical, since we changed the parameters in the objective function. The number of columns and iterations stays roughly similar. Finally, we see that in almost all cases the integer heuristic is able to outperform the start solution. In the final solutions, no duties are rejected. Also the computation times are similar.

Table 7.6: Results obtained when using a different parameter specification. Lower bound and gap are based on the pricing problem heuristic. The best found objective and the number of rejected duties is reported. The last column indicates whether the solution from the integer heuristic is better compared to the start solution.

ID	Spec.	Objective	Lower bound*	Gap (%)	Total time (s)	Master iter.	Rejected duties	Better than start solution
	0	76.33	68.79	10.96	889.14	44	0	No
8	1	170.12	151.60	12.22	450.83	86	0	Yes
	2	92.21	86.70	6.35	216.44	49	0	Yes
	3	127.82	117.60	8.69	350.00	33	0	Yes
	0	59.68	57.26	4.23	170.34	36	0	Yes
9	1	140.09	137.66	1.77	345.66	43	0	Yes
	2	81.34	77.86	4.47	118.91	34	0	Yes
	3	103.45	99.03	4.46	168.96	42	0	Yes
	0	72.35	68.44	5.71	467.80	51	0	Yes
10	1	165.14	147.64	11.86	354.02	65	0	Yes
	2	87.08	79.88	9.02	314.22	66	0	Yes
	3	142.29	116.13	22.53	566.93	53	0	No
	0	64.22	56.23	14.21	387.06	42	0	No
11	1	130.75	121.21	7.87	209.75	61	0	Yes
	2	71.83	65.19	10.18	311.89	55	0	Yes
	3	102.39	95.81	6.87	184.01	45	0	Yes

$*$ The lower bound of the LP solution found using a pricing problem heuristic.

Table 7.7 compares the number of times each pattern occurs with respect to the benchmark for each roster group for the four specifications (the actual count can be found in Table 9.7 in the Appendix). Specification 1 clearly leads to more patterns of the type RP6 for all instances. In most cases, this also leads to a decrease in the number of patterns of type RP7. This is as expected, since the number of rest days is limited. For some instances, we also observe a slight

increase in the pattern RP10.

When comparing specification 2 with the benchmark, we observe in some cases an increase in the number of times RP7 occurs. The increase is lower, even though we also increased the parameter in the objective by the same factor as in specification 1. This can be explained by the fact that patterns of type RP7 are longer, so therefore more difficult to schedule as well. Additionally, two rest days are needed to create a pattern of type RP7, whereas the pattern RP6 only needs a single rest day. In most cases, the number of times RP6 occurs drops. As mentioned before, there is a substitution effect between the patterns RP6 and RP7, since the number of rest days is limited.

In the final specification, we see a large drop in the number of times RP10 occurs. For example, for instance 8 and 11 the number of times RP10 was present dropped by 12 and 19, respectively. Only instance 10 sees an increasing number of RP10 patterns, because for this instance the start solution is used.

Overall, changing the objective function does not lead to large changes in computation times. Furthermore, the preferences of the crew members can be taken into account by adjusting the relative weights, which can lead to more attractive rosters. However, for some patterns it is easier to control the number of times they occur, which depends on the length of the pattern or on the availability of a certain duty type or rest day.

Table 7.7: Increase or decrease in the number of times a certain pattern occurs compared to the benchmark, when the relative parameter weights are adjusted.

ID	Group size	1			2			3		
		RP6	RP7	RP10	RP6	RP7	RP10	RP6	RP7	RP10
8	15	4	1	-1	0	3	-1	1	3	-3
	12	0	0	-6	2	0	-4	0	1	-6
	12	-2	-3	0	-4	1	-2	-2	1	-3
9	8	-3	1	2	-3	1	0	-3	0	-1
	12	3	0	2	-1	1	3	-1	1	-2
	12	3	1	1	1	1	-1	2	0	-1
10	15	2	0	3	-3	2	5	2	0	7
	12	5	-1	5	2	-4	5	5	-3	9
	12	3	-2	2	-1	2	7	1	0	2
11	8	0	1	-7	-2	0	-5	-2	0	-8
	12	4	-2	6	1	-1	-1	-3	-1	-2
	12	4	-2	-1	-4	-2	-7	-2	-2	-9

Chapter 8

Conclusion

In this thesis, we have shown how to model the Cyclic Crew Rostering Problem (CCRP) in order to obtain attractive rosters for groups of employees. The model is mostly based on the findings of Breugem (2020), who analysed different formulations of the CCRP. In this formulation we use both hard and soft constraints, in order to model the rules as stated in the collective labour agreement (CLA) and roster preferences, respectively. Furthermore, we make use of duty attributes in order to account for fairness between different roster groups. More importantly, it is shown that basic rosters, which indicate a pattern containing duties and rest days, are no longer required as input, which was previously the case at NS. This contributes to the existing trend in the literature to solve the CCRP in a single phase.

We solve the problem using a column generation approach, in which only the root node is solved. In order to speed up the column generation approach, we propose two ways to create start solutions. A basic Randomised Insertion Method (RIM) and a more time consuming Greedy Randomised Adaptive Search Procedure (GRASP). The pricing problem is modelled as a Shortest Path Problem with Resource Constraints (SPPRC). In order to deal with larger instances, we propose the use of forward and backward completion bounds, which try to prune as many sequences as possible. These completion bounds calculate whether a sequence can still improve on the current shortest path that is found. Furthermore, we also propose a pricing problem heuristic based on the backward completion bounds. The LP solution found by the column generation approach has to be transformed into an integer solution, for which several strategies are proposed.

The proposed solution approach is applied to real-life instances from NS. Due to the large number of parameters and strategies we first use a set of small instances to fix the best settings.

We demonstrate that initialising the column generation approach with a set of start solutions drastically shortens the computation time. Furthermore, we show that the RIM is preferred over the GRASP, even though the latter provides start solutions with a better objective value. Furthermore, the results show that the pricing problem is improved when forward and backward completion bounds are present. We can also opt for a pricing problem heuristic, which leads to fast computation times, while still providing an LP solution close to optimality. Finally, the comparison of the integer heuristics favours a dive-and-fix procedure, followed by a randomised insertion method and a destroy and repair heuristic. The latter two are only applied if not all duties are covered in the roster by the dive-and-fix procedure.

After fixing all parameters and strategies, a sensitivity analysis is performed. The sensitivity analysis is only applied on instances that contain three roster groups, in order to also evaluate fairness between the roster groups. Firstly, we show that it is necessary to include a strict upper limit when enforcing that the average workload is equally spread over the roster groups. These rosters can be solved using a similar computation time as before, while also having a similar objective value. The second analysis shows that increasing the minimum length of a single day off does not heavily influence the computation time nor the objective value. We show it is possible that all single days off have a length of at least 32 hours. When increasing this to 34 hours, we are not always able to find a roster that covers all duties. In the final analysis, the weights given to certain patterns are changed. It is possible to change the distribution of the number of times each pattern occurs. However, for longer patterns these changes are smaller compared to the shorter patterns, because longer patterns are more difficult to schedule and require more of a specific duty type or rest day.

To conclude, this thesis shows that NS no longer requires basic rosters to solve the CCRP. This means that without manual intervention it is still possible to obtain satisfactory rosters for their employees. Furthermore, in the sensitivity analysis it is shown that the roster can be improved in terms of fairness and attractiveness. NS can use these results in upcoming CLA negotiations.

Chapter 9

Discussion

In this chapter, we first discuss how the restrictions that are made on the assumptions influence the model. Afterwards, we show where potential improvements can be made in the solution approach and recommendations for future research are given.

9.1 Limitations

As mentioned before we simplify item 3 from the CLA, by only using periods of 7x24 hours (with a break of at least 36 hours) and only considering periods that start at midnight. In case we would also allow periods of 14x24 hours (with a break of at least 72 hours), we obtain more freedom when creating rosters. This means it is easier to find a feasible roster. It is difficult to assess whether the rosters obtained in this way are more attractive as well, since the breaks might not be divided equally over the weeks.

In this thesis, the Red Weekends (item 7 from the CLA) are implemented using explicit constraints. A Red Weekend is defined as a free weekend where the last duty ends before Friday 4:00 p.m. This is possible since we assume that duties on Monday start no earlier than Monday 4:00 a.m., such that the rest period is always at least 60 hours. This is also a restrictive assumption, since according to the CLA a weekend of at least 60 hours where the last duty ends on Saturday 12:00 a.m. also counts as a Red Weekend. The results show that our definition of a Red Weekend is often the bottleneck in order to create a feasible roster in which all duties are covered. A potential solution to this problem is to change the definition of a week to a length of 8 days. For example, a week starts on Monday and also ends on a Monday. In that case having a Red Weekend is a property belonging to the week. The master problem has to be slightly adjusted, to force subsequent clusters to have a similar duty or rest day when they

overlap. This formulation would allows more options for a Red Weekend, such that it is easier to find a feasible roster. However, as far as we know there are no results on the performance of overlapping clusters.

9.2 Recommendations

The performance of the backward completion bounds are entangled with the definition of the duty types. Currently, we solely consider early, late and night duties. This means we can only ban all early duties, if we are sure that none of them improves the current shortest path. In case we consider more specific types of duties, each duty type group contains duties that are more similar to each other. This could lead to tighter bounds, such that we can faster prune a group of duties. However, when having too many groups we only prune a few duties at the same time. This might also slow down the entire pricing problem.

For finding an integer solution we make use of a heuristic approach. This can be improved in several ways. For example, the destroy and repair method in the local search takes a lot of computation time, yet it is still unable to cover all the duties. The destroy and repair heuristic might be improved by targeting specific parts of the roster (e.g. a single roster group). Instead of using an integer heuristic, it is also possible to use a branch-and-price framework. This is guaranteed to find the optimal integer solution. This can be used for a fairer comparison in the sensitivity analysis.

Currently, the solution approach is only applied on the crew base in Amersfoort. It is interesting how the performance changes when larger crew bases, such as the one in Utrecht are considered. Also, no instances containing train drivers are considered. When creating rosters for train drivers, we have to consider which line or which train type they are allowed to operate on. The model presented in this thesis can be extended to incorporate these kind of restrictions. It is interesting to see how the performance of the solution approach differs when creating a roster for train conductors or drivers.

At NS, some roster groups have the restriction that they only perform a single duty type (e.g. early duties). This can also be readily incorporated in the solution approach. It is expected that adding such a restriction simplifies solving the instances as well.

Furthermore, more duty attributes may be added to increase the fairness between roster groups. For example, Red Weekends and night duties are not explicitly divided over the different roster groups, leading to an unfair distribution.

Finally, it might be interesting to make the size of the roster groups, which is a now an input, a decision variable as well. A larger roster group has as a benefit that more variation is present for the employee. On the other hand, a shorter roster might have a more structured rhythm in terms of free weekends. This might lead to more attractive rosters.

References

Abbink, E. (2014). *Crew Management in Passenger Rail Transport*. PhD thesis, Erasmus Research Institute of Management (ERIM).

Abbink, E., Fischetti, M., Kroon, L., Timmer, G., and Vromans, M. (2005). Reinventing crew scheduling at Netherlands railways. *Interfaces*, 35(5):393–401.

Borndörfer, R., Reuther, M., Schlechte, T., Schulz, C., Swarat, E., and Weider, S. (2015). Duty rostering in public transport - facing preferences, fairness and fatigue. *CASPT*.

Borndörfer, R., Schulz, C., Seidl, S., and Weider, S. (2017). Integration of duty scheduling and rostering to increase driver satisfaction. *Public Transport*, 9:177–191.

Breugem, T. (2020). *Crew Planning at Netherlands Railways: Improving Fairness, Attractiveness and Efficiency*. PhD thesis, Erasmus Research Institute of Management (ERIM).

Burke, E. K., Causmaecker, P. D., Vanden Berghe, G., and Van Landeghem, H. (2004). The state of the art of nurse rostering. *Journal of Scheduling*, 7:441–499.

Caprara, A., Fischetti, M., Toth, P., Vigo, D., and Guida, P. L. (1997). Algorithms for railway crew management. *Mathematical Programming*, 79:125–141.

Cheang, B., Li, H., Lim, A., and Rodrigues, B. (2003). Nurse rostering problems—a bibliographic survey. *European Journal of Operational Research*, 151(3):447–460.

Dumitrescu, I. and Boland, N. (2003). Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3):135–153.

Ernst, A. T., Jiang, H., Krushnamoorthy, M., Nott, H., and Sier, D. (2001). An integrated optimization model for train crew management. *Annals of Operations Research*, 108:211–224.

Ernst, A. T., Jiang, H., Krushnamoorthy, M., and Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27.

Ernst, A. T., Krishnamoorthy, M., and Dowling, D. (1998). Train crew rostering using simulated annealing. in: Proceedings of ICOTA98, Perth.

Freling, R., Lentink, R. M., and Wagelmans, A. P. M. (2004). A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm. *Annals of Operations Research*, 127:203–222.

Grötschel, M., Borndörfer, R., and Löbel, A. (2003). Duty scheduling in public transit. In Jäger, W. and Krebs, H. J., editors, *Mathematics — Key Technology for the Future*, pages 653–674. Springer, Berlin, Heidelberg.

Hartog, A., Huisman, D., Abbink, E. J. W., and Kroon, L. G. (2009). Decision support for crew rostering at NS. *Public Transport*, 1:121–133.

Huisman, D., Kroon, L. G., Lentink, R. M., and Vromans, M. J. C. M. (2005). Operations research in passenger railway transportation. *Statistica Neerlandica*, 59(4):467–497.

Kohl, N. and Karisch, S. E. (2006). Airline crew rostering: Problem types, modeling, and optimization. *Annals of Operations Research*, 127:223–257.

Kroon, L., Huisman, D., Abbink, E., Fioole, P.-J., Fischetti, M., Maróti, G., Schrijver, A., Steenbeek, A., and Ybema, R. (2009). The new Dutch timetable: The OR revolution. *Interfaces*, 39(1):6–17.

Lezaun, M., Pérez, G., and Sáinz de la Maza, E. (2006). Crew rostering problem in a public transport company. *Journal of the Operational Research Society*, 57(10):1173–1179.

Lozano, L., Duque, D., and Medaglia, A. L. (2016). An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, 50(1):348–357.

Mesquita, M., Moz, M., Paias, A., and Pato, M. (2013). A decomposition approach for the integrated vehicle-crew-roster problem with days-off pattern. *European Journal of Operational Research*, 229(2):318–331.

Nishi, T., Sugiyama, T., and Inuiguchi, M. (2014). Two-level decomposition algorithm for crew rostering problems with fair working condition. *European Journal of Operational Research*, 237:465–473.

Sodhi, M. S. and Norris, S. (2004). A flexible, fast, and optimal modeling approach applied to crew rostering at London underground. *Annals of Operations Research*, 127:259–281.

Xie, L. and Suhl, L. (2015). Cyclic and non-cyclic crew rostering problems in public bus transit. *OR Spectrum*, 37:99–136.

Appendix

Table 9.1: Total computation time (in seconds) when adding start solutions using the RIM for small instances.

ID	Start solutions					
	0	10	20	30	40	50
1	180.53	63.70	53.81	56.61	50.03	47.91
2	1685.74	429.30	366.55	325.81	342.33	247.98
3	1864.26	375.20	326.12	332.22	307.63	277.18
4	3089.51	662.84	518.88	503.40	910.85	1173.94
5	5335.77	673.85	1472.83	1417.71	1106.88	660.60
6	963.98	124.05	128.29	209.26	116.10	90.57
Average	2186.63	388.16	477.75	474.17	472.30	416.36

Table 9.2: Number of master iterations when solving the root node when adding start solutions using the RIM for small instances.

ID	Start solutions					
	0	10	20	30	40	50
1	109	36	29	28	24	23
2	178	46	40	33	33	32
3	173	47	41	42	38	35
4	172	42	33	32	31	32
5	178	39	36	33	31	27
6	158	39	39	41	30	28
Average	161.33	41.50	36.33	34.83	31.17	29.50

Table 9.3: Total computation time (in seconds) when adding start solutions using the GRASP for small instances.

ID	Start solutions					
	0	10	20	30	40	50
1	180.53	32.63	58.91	61.72	67.66	66.04
2	1685.74	331.20	129.73	136.47	142.65	167.91
3	1864.26	266.27	246.47	272.78	255.11	238.65
4	3089.51	524.13	549.90	550.63	492.02	489.78
5	5335.77	505.28	584.44	856.50	1038.83	1067.34
6	963.98	239.79	256.33	323.22	318.33	340.74
Average	2186.63	316.55	304.29	366.89	385.77	395.08

Table 9.4: Number of master iterations when solving the root node when adding start solutions using the GRASP for small instances.

ID	Start solutions					
	0	10	20	30	40	50
1	109	22	18	15	12	12
2	178	33	26	23	21	22
3	173	32	27	25	23	22
4	172	33	26	24	24	24
5	178	28	28	24	24	22
6	158	30	22	26	20	20
Average	161.33	29.67	24.50	22.83	20.67	20.33

Table 9.5: Results obtained using the preferred settings. The best found objective and the number of rejected duties is reported. The last column indicates whether the solution from the integer heuristic is better compared to the start solution.

ID	Objective	Lower bound	Gap (%)	Columns	Master iter.	Rejected duties	Better than start solution
1	14.79	13.97	5.88	734	29	0	Yes
2	10023.27	21.36	46831.62	3978	34	1	Yes
3	22.07	20.97	5.21	2843	25	0	Yes
4	23.27	21.74	7.04	3932	17	0	No
5	23.60	21.82	8.15	3034	22	0	No
6	22.53	21.24	6.05	2060	23	0	Yes

Table 9.6: The total computation time followed by the time spend on the individual parts of the solution approach when using the preferred settings.

ID	Total time (s)	Start time (s)	Master time (s)	Pricing time (s)	Integer time (s)	Local search time (s)
1	4.51	1.79	0.00	1.68	1.04	-
2	26.07	3.02	0.00	2.94	7.48	12.63
3	10.66	1.30	0.00	2.18	7.17	-
4	35.77	2.37	0.00	0.79	4.39	28.22
5	22.70	2.97	0.00	0.74	1.69	17.30
6	5.88	0.95	0.00	1.93	2.87	-

Table 9.7: Comparison of the number of times a certain pattern occurs, when the relative parameter weights are adjusted.

ID	Group size	0			1			2			3		
		RP6	RP7	RP10	RP6	RP7	RP10	RP6	RP7	RP10	RP6	RP7	RP10
8	15	3	2	5	7	3	4	3	5	4	4	5	2
	12	3	2	7	3	2	1	5	2	3	3	3	1
	12	7	3	5	5	0	5	3	4	3	5	4	2
9	8	3	0	1	0	1	3	0	1	1	0	0	0
	12	1	0	2	4	0	4	0	1	5	0	1	0
	12	0	0	2	3	1	3	1	1	1	2	0	1
10	15	4	1	2	6	1	5	1	3	7	6	1	9
	12	1	4	2	6	3	7	3	0	7	6	1	11
	12	3	2	2	6	0	4	2	4	9	4	2	4
11	8	4	0	8	4	1	1	2	0	3	2	0	0
	12	3	3	4	7	1	10	4	2	3	0	2	2
	12	6	3	10	10	1	9	2	1	3	4	1	1