Erasmus University Rotterdam
Erasmus School of Economics

Master Thesis

Real-Life Workforce Scheduling: Constraint Modeling

Author: **P.J. de Voogd**
Student number: **409118**
Supervisor: **Prof. Dr. A.P.M. Wagelmans**
Supervisor ORTEC: **L.M. Fijn van Draat, MSc**
Co-reader: **Prof. Dr. D. Huisman**

January 2, 2020

Abstract

This thesis studies the constraints used in practice for solving real-life workforce scheduling problems. It is shown that it is possible to categorize most constraints used in practice into different sets of mathematically equivalent constraints. Furthermore, it is shown that using the obtained categories allows the constraints to be included in a mixed-integer programming formulation. The resulting formulation is solved using a commercial solver. The performance is tested by solving the formulation for a set of benchmark instances provided by Curtois (2014) and a set of real-life instances provided by Ortec. We find that about half of the benchmark instances can be solved to optimality. Furthermore, we show that by using an adjusted two-stage version of the formulation, all of the problem instances can be solved up to a satisfactory level. For three of the instances, we find solutions that are better than any that were known before. Furthermore, for five of the instances we find lower bounds that are better than any that were known before.

Next, we find that the exact algorithm performs better than Ortec's optimizer for most of the real-life instances. The formulation can be solved to optimality for four of the real-life instances, leading to significant improvements compared to Ortec's optimizer for three of them. The remaining three instances can be solved with an optimality gap of at most 0.4 percent. In summary, we find that by using the developed constraint categories in combination with an exact algorithm we can efficiently model and solve benchmark instances as well as real-life instances.

Contents

Chapter 1

Introduction

In this thesis, we investigate the workforce scheduling problem. In this problem, we seek to perform a set of shifts by assigning these to employees. Together, the shifts assigned to an employee should be in line with labour regulations. Additionally, every employee can have certain wishes and preferences regarding a certain assignment of shifts. Not satisfying these preferences will lead to a penalty. The overall problem is to assign as many shifts as possible to the employees, such that each of the individual assignments is feasible and the sum of penalties is minimized.

At Ortec, this problem is currently being solved using the Ortec Workforce Scheduling optimizer. It first applies a genetic algorithm to find a reasonable solution. Then, it uses a combination of ruin-and-recreate procedures and local search to refine this solution. Within this algorithm, all feasibility constraints are put into a so-called black box. After creating a given roster, all these constraints are checked to determine whether the roster is feasible. Then, the score of the resulting roster is determined. Alternatively, one could open this black box, and formulate (some of) its constraints explicitly. In this way, knowledge of the constraints at hand can be incorporated during the creation of a roster, rather than afterwards. Ortec is interested to know whether doing so might allow the constraints to be modeled more efficiently. This can be formalized by the following research question:

> *How can we model real-life workforce scheduling problem instances, such that they can be solved efficiently?*

To investigate this question, a set of constraint categories is used. We find that most of the constraints used in real life can be modeled using one or more of these categories. Furthermore, each of the categories can be modeled linearly.

Next, it is of interest to know whether the resulting model allows for exact solution approaches and whether such a solution approach would be sufficiently effective to solve real-life problem instances. This can be formalized by the second research question:

To what extent can we solve real-life workforce scheduling problems using an exact approach?

To investigate this question, a mixed-integer programming formulation (MIP) is used. This requires that all constraints are taken out of the black box and modeled linearly. To solve the formulation, we use the commercial solver CPLEX, provided by IBM. This solver is considered to be among the state of the art in optimization software.

To test the MIP, we apply it to a set of benchmark instances and a set of real-life instances. The benchmark instances consist of a variety of problems, gradually increasing in size. These will be used to test how our methodology compares to other exact approaches and at which problem size it becomes too time-consuming. At that point, it might be better to apply a heuristic. We compare our solutions to known lower bounds and best solutions. This test is summarized by the following subquestion:

How does our methodology compare to the state of the art in exact solution approaches?

Even if the MIP performs well on the benchmark instances, it might be that it falls short in solving real-life problems. Therefore, a set of problems from Ortec's customers is used, which consists of constraints that are less easy to model. We show how these constraints can be modeled using the constraint categories and then we solve the MIP. These instances will be used to test whether our approach is applicable in real life. We compare our solutions to those obtained from Ortec's Workforce Scheduling algorithm. This test is summarized by the following subquestion:

How does our methodology perform on real-life problem instances?

The outline of the remainder of the report is as follows: In Chapter 2, we discuss the existing literature on this problem. In Chapter 3, we discuss Ortec's Workforce Scheduling optimizer. In Chapter 4, we show how we can categorize different types of constraints, and how such categories can be modeled linearly. Next, we present the MIP in Chapter 5. Then, we discuss the benchmark instances in Chapter 6 and show the results of these instances in Chapter 7. Afterwards, we discuss the real-life instances in Chapter 8 and present the results of these instances in Chapter 9. Finally, we discuss some potential for future research and conclude in Chapter 10.

Chapter 2

Literature Review

This chapter discusses a review of literature on the subject of workforce scheduling. Firstly, we present a selection of review papers that can be consulted for more extensive literature reviews. Secondly, we discuss exact solution approaches to the workforce scheduling problem. Thirdly, we look at some heuristic approaches. Next, we discuss constraint categorization. Then, we consider benchmark instances to the problem. Finally, we explain how this thesis fits within the literature, and we identify the gap in research we wish to fill with this thesis.

2.1 Review papers

As the goal of this thesis is not to give an exhaustive literature review, we refer the reader to some papers that did have this goal in mind. Early reviews of research in the field of workforce scheduling were performed by Aggarwal (1982) and Tien & Kamiyama (1982). An extensive literature review was performed by Ernst et al. (2004a; 2004b) referencing over 700 papers. Further literature reviews have been written by Cheang et al. (2003), Burke et al. (2004), and Van den Bergh et al. (2013). A more specialized literature review was written in the field of airline crew scheduling by Gopalakrishnan & Johnson (2005).

2.2 Exact Solution Approaches

Workforce scheduling belongs to a group of optimization problems known as NP-hard problems (Brucker et al., 2011). It is generally assumed that there exist no polynomial-time solution approaches to solve these types of problems. This implies that as the problem size becomes large, the time it takes to solve these problems to optimality becomes exponentially large. Therefore, there are two main approaches to solving these types of problems: heuristics and exact algorithms. The methodology applied in this thesis is part of the second group. In general, exact algorithms perform best on relatively small problem instances, whereas heuristics perform best on large problem

instances. Therefore, it is of interest to know what the maximum problem size is that can be solved within a reasonable time.

To solve the problem exactly, a common approach is to use column generation (Desrochers & Soumis, 1989; Bard & Purnomo, 2005; He & Qu, 2012), often combined with branch and bound, which is known as branch and price (Beliën & Demeulemeester, 2008; Maenhout & Vanhoucke, 2010). In these approaches, the problem is described by a set-covering formulation, which is then solved with column generation. Another way is by formulating the problem as an explicit integer linear programming formulation (Caprara et al., 2003), and solving this using branch and bound or branch and cut. The solution approach used in this thesis is similar to this. Additionally, Lagrangian relaxations can be applied (Bard & Purnomo, 2007).

2.3 Heuristics

As an alternative to solving the problem exactly, one might like to apply a heuristic. An advantage of heuristics is that they generally scale better for large problem instances. With exact approaches, it often occurs that for large scale instances, the algorithm becomes very slow, and no good solutions are being found within a reasonable time. Heuristics try to overcome this problem by foregoing the search for optimal solutions. In general, they are fast, problem-specific approaches that attempt to find a very good solution within a small amount of time. Although the approach presented in this thesis does attempt to find an optimal solution, knowledge of good heuristics is still important, because for some problem instances the exact approach may not be effective.

Within the field of workforce scheduling, a wide range of heuristics is being applied. Some efficient algorithms include variable neighbourhood search (Burke et al., 2008; Remde et al., 2007), tabu search (Burke et al., 1998), and a memetic approach (Burke et al., 2001). The heuristic used by Ortec, which serves as a comparison in this thesis, combines a genetic algorithm (similar to the memetic approach) with a ruin-and-recreate algorithm (similar to variable neighbourhood search) and local search.

2.4 Categorizing Constraints

At the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018), some work has been performed on unifying nurse rostering models (Kingston et al., 2018). Part of this was an attempt to categorize the constraints that are used in practice into mathematically equivalent groups. We have taken this idea and extended it to include more advanced constraints. Chapter 4 discusses this in more detail.

2.5 Benchmark Instances

To test the efficiency of our methodology, we apply it to a set of benchmark instances provided by Curtois (2014). These instances contain a set of problems with varying characteristics, ranging from relatively small problems with about ten to twenty employees covering shifts over two to four weeks to large ones with up to 150 employees covering a year of shifts. All instances can be found at `http://www.schedulingbenchmarks.org/`. Over there, a rosterviewer program can also be downloaded that can be used to check whether a given schedule is feasible and what its total penalty is. This has been used to verify the results in this thesis.

These instances are useful because they provide insight into the performance of our methodology in comparison to state-of-the-art methods. However, they are more stylized than problems that will be found in practice. As a consequence, real-life problem instances tend to be more difficult to model and/or solve. Therefore, previous research on real-life workforce scheduling typically involves using heuristics, a limited set of shift-types, a limited set of constraints, or other simplifications. With this thesis, we aim to investigate an area of research where none of these simplifications are incorporated while still aiming at an optimal solution. The next chapters describe how we intend to do that.

Chapter 3

Ortec Workforce Scheduling

At the moment, workforce problems are being solved by human planners aided by Ortec's Workforce Scheduling optimizer (Post & Veltman, 2004). This optimizer has been in development for about 20 years and is being used by dozens of Ortec's customers. It is considered to be highly efficient and among the state of the art in heuristic solution approaches to workforce scheduling problems. This optimizer starts by generating an initial solution using a genetic algorithm. Next, this initial solution is improved using local search. Whenever a local optimum is found, ruin-and-recreate is used to escape from this solution. The next sections discuss these steps in more detail.

3.1 Genetic Algorithm

To generate an initial solution that is of reasonable quality, the optimizer starts with a genetic algorithm (Holland, 1992). Within this algorithm, the individuals correspond to complete schedules. The initial population is generated by splitting the employees into two groups. Then, as many shifts as possible are assigned to the first group and the remainder is assigned to the second (if possible). Then, one of six offspring generating methods is applied. These unassign some shifts and reassign them in other positions to generate a new individual from one or two individuals in the population. This is repeated until the population has doubled in size. The probabilities of selecting one of these methods depend on their past performance. Next, a greedy selection is applied, where the best half of the population is saved and the other half is discarded. This process is stopped after there is no improvement for a given number of iterations, or until a fixed amount of time has passed. Then, the best individual is used as an initial solution for the next method, local search.

3.2 Local Search

To find a local optimum, two operations are performed. The first operation is to take a given shift and assign it in the first feasible position that is better than its current position. The second operation is to take two shifts and swap their positions in the schedule. The same is done with

series of 2 and 3 shifts. These operations are performed for all possible combinations, and they are repeated until neither of them can improve the schedule in any way. When this happens, the schedule is said to be locally optimal. An important managerial consequence of this local search is that it cannot occur that after running the optimizer for two hours, a human planner can move a shift a bit, or swap two shifts and find an improvement. Furthermore, it implies that no improvements can be found using these two simple operations. Therefore, we get stuck in this local optimum. Therefore, the optimizer tries to escape it by using ruin-and-recreate.

3.3 Ruin-and-Recreate

In ruin-and-recreate, a part of the solution is ruined, after which it is recreated (Schrimpf et al., 2000). This constitutes a large step from the current solution to another region of the solution space. To do so, up to five shifts are removed from the current solution and inserted again. Afterwards, local search is applied to improve this new solution to local optimality and the process is repeated.

This algorithm has been implemented by Ortec and the customer databases have been formatted to suit the input. Therefore, the results regarding the performance of the algorithm are readily available. They will be compared to the performance of our exact algorithm.

Chapter 4

Categorizing Constraints

In this chapter, we analyze the constraints that focus on a single employee. In the algorithms of Ortec, about 150 such constraints are being used, many of which allow multiple different parameter settings and exceptions. All of these are necessary to model different customer requirements and preferences. They allow the customers to understand the constraints by formulating them with words, rather than by using mathematics. This allows the users to solve their workforce scheduling problems without a thorough understanding of the underlying mathematics. However, when it comes to solving the actual problem, it does not matter what the underlying real-world meaning of a constraint is. All that matters is what effect it has on the feasible region and the objective of the problem. As such, some constraints differ from one another in their wording and real-life meaning but are mathematically equivalent. In this chapter, we aim to identify such constraints and categorize them. As such, our solution algorithms do not need to be equipped to handle all of these constraints separately, but can do with modeling a smaller number of constraint categories. Additionally, this categorization allows for multiple problem instances (for example, those used by different companies) to be modeled using the same formulation.

To perform this categorization, we have looked at six of Ortec's customers. Together, the different departments of these six customers used 87 of the constraints that are offered by Ortec. Together, the customers should offer a fair representation of the types of constraints that are being used by all of Ortec's customers. For the 87 constraints that are being used, we have analyzed which parameter settings the different customers were using. Then, the settings that are not being used by any of the customers can be ignored. So, if a certain constraint allows a given exception, but none of the customers makes use of it, the exception is ignored. Doing so allows us to model the constraints that are used in practice, rather than the more difficult, irrelevant versions.

The resulting set of constraints has been analyzed, and this resulted in a set of categories that can be used to model these constraints. To determine whether or not a certain constraint can be modeled using a given category, all of the categories have been modeled using a (set of) linear equation(s). Then, a constraint is said to be of a given category if the structure of the constraint allows it to be modeled using these equations. Some categories can actually be viewed as a subset of

another category. As such, some constraints might be considered to be of any of these categories. In that case, the constraint is counted as the most strict of these categories, to keep the categorization as specific as possible. The resulting categories are presented in Table 4.1. It displays a description of each category, together with the number of constraints that can be modeled using this category (out of 87), some examples of the constraints that fall in this category, and mathematical expressions that can be used to linearly model the constraints of this category. To model the mathematical expressions, we make use of variables x_s that are equal to 1 if shift s is assigned to the employee, and 0 otherwise. Furthermore, a shift is considered active if it is assigned to this employee, and a set of shifts is considered active if it contains at least one active shift.

Table 4.1: Constraint Categories

Description, examples, and expression

Unwanted shifts: 27 constraints.

Some shifts s should not be assigned to this employee.

Examples: The employee does not have the relevant skills to perform this shift.

The employee prefers not to work at this shift's location.

$$x_s = 0$$

Unwanted shiftpairs: 16 constraints.

Some pairs of shifts s and f should not both be assigned to this employee.

Examples: There should be at least 10 hours of rest between two shifts.

Consecutive shifts should (not) have the same type.

$$x_s + x_f \leq 1$$

Limited shifts: 10 constraints.

The number of active shifts within a given set S is between given limits l and u.

Examples: On each day, only one shift can be assigned to an employee.

Within a week, at most 6 shifts can be assigned to an employee.

$$l \leq \sum_{s \in S} x_s \leq u$$

Weighted limited shifts: 5 constraints.

The weighted sum of active shifts within a given set S is between given limits l and u.

Note: the weights w_s depend on characteristics of the relevant shifts, for example working time.

Examples: On each day, an employee can spend at most 12 hours performing an on-call shift.

Within a week, an employee should work at least 20 hours and at most 40 hours.

$$l \leq \sum_{s \in S} w_s x_s \leq u$$

Limited sets: 4 constraints.

The number of active sets S within a given set of sets Z is between given limits l and u.

Examples: An employee can only work during at most 2 out of 4 of weekends.

The number of different shift types should be limited.

$$\begin{aligned}
x_S &\geq x_s, && \forall S \in Z,\ s \in S, \\
x_S &\leq \sum_{s \in S} x_s, && \forall S \in Z, \\
l &\leq \sum_{S \in Z} x_S \leq u, \\
x_S &\in \{0, 1\}, && \forall S \in Z
\end{aligned}$$

Continued on the next page.

Table 4.1 - Continued from the previous page

Limited consecutive sets: 5 constraints.
The number of consecutive active sets is between given limits l and u.
Examples: The number of shifts in one series should be between 3 and 6.
An employee should not be assigned stand-alone shifts.
Let the sets be denoted by $Z = \{S_1,\ S_2, \ldots, S_n\}$.
The lower bound is modeled by forbidding any sequence that has fewer than this number of consecutive active sets.

$$x_{S_e} + (m - \sum_{d=e+1}^{e+m} x_{S_d}) + x_{S_{e+m+1}} > 0,\ m = 1, \ldots, l-1,\ e = 1, \ldots, n-m-1^1$$

The upper bound is modeled by forbidding any sequence that has more than this number of consecutive active sets.

$$\sum_{d=e}^{e+u} x_{S_d} \leq u,\ e = 1, \ldots, n-u$$

This table shows descriptions of the different constraint categories, examples of these categories, how they can be modeled, and how many of the 87 constraints are of this category.

Note that the mathematical expressions in Table 4.1 are modeling the constraints as if they were hard constraints. That is, adding such equations to a model would mean that it is not allowed that they are violated. We refer to such constraints as requirements. Alternatively, there can be constraints that are not strictly required but yield a penalty if they are violated. Such constraints are referred to as criteria.

There are two types of criteria. The first type gives a penalty equal to the weight of the constraint whenever it is violated. The second multiplies the weight of this constraint with the total violation to obtain the penalty. Additionally, this violation may be squared. In practice, the weight of the constraint is set by the customer using it. This indication how important it is to satisfy the constraint.

Criteria can be modeled with the use of an auxiliary variable. This variable will indicate whether or not the constraint is satisfied. For example, let us suppose that a penalty of C is obtained if more than l shifts are assigned to this employee in a single week. Then, we need to add the following set of constraints to the model:

[1] This constraint forbids sequences of length m. That is, sequences such as $0, 1, 1, 0$, for which m is 2. This is done by requiring that if all x variables in the summation are equal to 1, one of the x variables directly before or after it must also be 1. So, for $m = 2$ the sequence $0, 1, 1, 0$ would yield a violation, but $1, 1, 1, 0$ or $0, 1, 1, 1$ would not. Note that even though $0, 1, 0, 0$ would not give a violation for $m = 2$, it will already have been forbidden by $m = 1$.

$$Ky \geq \sum_{s \in S_w} x_s - l, \ \forall w \tag{4.1}$$

Where K is some arbitrarily large number, for example, the maximum number of shifts on one day, y is either 0 or 1, and S_w is the set of shifts in week w. Then, the term Cy is added to the objective function.

Alternatively, if a penalty of C is obtained for every shift beyond the l-th that is assigned to the same employee in the same week, we need to add the following set of constraints to the model:

$$y \geq \sum_{s \in S_w} x_s - l, \ \forall w \tag{4.2}$$

With y an integer between 0 and K. Then we need to add the term Cy to the objective, and square y if necessary. Note that all the expressions discussed so far focus on a single employee. If they should be added to a general model, however, they should be added once for every employee to which they apply. The next chapter further elaborates on this point. Using the above equations, any limited shifts constraint can be modeled as a criterion. For other categories, a similar approach can be used. This is done by taking the difference between the left hand side and right side of the equations, and stating that Ky or y should be larger than this difference similar to the expressions above.

Using these categories, all of the constraints used in the benchmark instances and used by the Ortec customers have been analyzed. They have been placed into one of these categories where possible. In the end, all of the benchmark constraints and 67 out of 87 of the Ortec constraints could be modeled using one of the categories. The result of this is shown in Appendix A for the benchmark instances and two of the Ortec customers.

The main reason most of the remaining twenty Ortec constraints could not be fitted directly into one of these categories is that they allowed for one or more exceptions or were using an if-then structure. For example, one of the constraints states that if a given week contains consignation shifts, this week can have at most a given number of hours of work. Put differently, a given week must contain no consignation shifts, or this week can have at most a given number of hours of work.

For this example, only one of the two conditions has to hold. This allows the use of an or-constraint. It can be seen that the condition 'a given week must contain no consignation shifts' can be modeled using a limited shifts constraint and the condition 'this week can have at most a given number of hours of work' can be modeled using a weighted limited shifts constraint. Now, let A be 1 if the limited shift condition holds, and 0 otherwise. Next, let B be 1 if the weighted limited shifts condition holds, and 0 otherwise. Then, the constraint can be modeled as $A + B \geq 1$.

This allows another fifteen constraints to be modeled using two of the standard categories and an or-constraint. What's more, the remaining five constraints could also be modeled using a

combination of categories and or-constraints. However, this would require a lot of modeling, whereas handling these constraints separately without the categories would be less cumbersome. Therefore, we chose to model these constraints separately. One of these constraints is being described in more detail in Section 8.2.1.

Altogether, we believe this indicates that these categories allow for a wide variety of real-life problem instances to be modeled in a standardized fashion. In the next section, we show how this standardization allows for the use of an exact solution approach to solve such problem instances.

Chapter 5

Mixed-Integer Programming Problem

In this chapter, we describe an exact approach to solve the workforce scheduling problem. For each employee, it is decided which shifts to assign to him or her. In the previous chapter, we have shown how a wide variety of constraints can be modeled using variables x_s, that are 1 if shift s is assigned to the employee, and 0 otherwise. In this chapter, we add an index i to these variables to distinguish between employees.

Besides employee-specific constraints, the vertical requirements of shifts need to be modeled. If shift s is assigned to fewer employees than is demanded, a penalty is obtained. This is equal to C_s times the difference between the demand of shift s and the number of times it is assigned.

In this thesis, we assume that at most one shift can be assigned to an employee on each day. As such, if we add an artificial day-off shift to all the shift variables of a single day and employee, this sum should always be equal to one. These day-off variables can then be used to simplify some of the constraints. For example, constraints relating to the length of a series or the number of weekends with work can be simplified using these variables.

Next, we present a mixed-integer programming formulation (MIP) of the problem. In this chapter, we omit all the constraints used to model the categories explained in the previous chapter. To apply the MIP, all of these constraints need to be added as well. This can be done by applying the expressions presented in Table 4.1 to the variables x_{is}.

In formulating the problem, we use the following notation:

Sets:

Let C be the set of criteria.

Let I be the set of employees.

Let I_c be the set of employees to which criterion c applies

Let S be the set of shifts.

Let S^+ be the set of shifts including artificial day-off shifts.

Let S_d^+ be the set of shifts on day d including an artificial day-off shift..

Parameters:

Let C_c be the cost of violating criterion c by 1 unit.

Let D_s be the demand of shift s.

Variables:

Let x_{is} be 1 if shift s is assigned to employee i, and 0 otherwise.

Let y_{ci} be the auxiliary variable that is equal to the violation of criterion c by employee i.

Let z_s be the effect on the objective of the vertical requirements of shift s.

Then, we can formulate the problem as follows:

$$\min \quad \sum_{s \in S} C_s z_s + \sum_{c \in C} \sum_{i \in I_c} C_c y_{ci} \tag{5.1}$$

$$\text{s.t.} \quad \sum_{s \in S_d^+} x_{is} = 1, \qquad i \in I, d \in D, \tag{5.2}$$

$$z_s \geq D_s - \sum_{i \in I} x_{is}, \qquad s \in S, \tag{5.3}$$

$$x_{is} \in \{0, 1\}, \qquad i \in I, s \in S^+, \tag{5.4}$$

$$y_{ci} \in \{0, 1\}, \qquad c \in C, i \in I_c, \tag{5.5}$$

$$z_s \geq 0, \qquad s \in S \tag{5.6}$$

Here, the objective is to minimize the sum of the costs obtained for not assigning shifts to any employee and the costs of violating the criteria. Constraints (5.2) ensure that every employee is assigned exactly one shift each day, including day-off shifts. Constraints (5.3) make sure that the auxiliary variable z_s is at least equal to how much shift s is being underscheduled. Finally, domain requirements are ensured by Constraints (5.4) to (5.6).

In practice, it might not always be true that at most one shift can be assigned to an employee on one day. If this does not hold, Constraints (5.2) need to be removed from the model and possibly replaced by some additional unwanted shiftpair constraints to avoid overlapping shifts. Such constraints may also be necessary if the problem instance contains night shifts, such that two shifts on consecutive days could have some overlapping period. In this thesis, we will show for all problem instances we consider, that it is impossible to assign multiple shifts starting on the same day to the same employee. In what follows, we will always check this assumption before applying the MIP.

Chapter 6

Benchmark Instances

This chapter analyzes the benchmark instances in more detail. First, we describe the different types of constraints that are used. Then, we present some descriptive statistics for each of the different instances.

The benchmark instances consist of a set of 24 problems that vary in size from very small to very large. These are summarized in Table 6.1. In all instances, a set of shift types is provided, each of which is of a given length. Some of these types are not allowed to be planned consecutively. Then, for all the days in the relevant planning horizon, these shift types need to be planned a given number of times. If a given shift type is assigned too many times, or too few times, a penalty is incurred. Therefore, we add one shift for each of these shifttype-day pairs for which there is demand.

To incorporate overcoverage in the MIP, we add auxiliary variables z_s^o to the model. These are handled the same way as the variables z_s, but for z_s^o expression (5.3) is adjusted to:

$$z_s^o \geq \sum_{i \in I} x_{is} - D_s, \ \ s \in S, \tag{6.1}$$

Furthermore, the coefficients of z_s^o in the objective should be equal to the cost of one unit of overcoverage. In most benchmark instances this is equal to 1.

Next, there is a set of employees to which the shifts can be assigned. All of these employees have a maximum number of shifts he or she can perform, a maximum and minimum number of minutes he or she can work, a maximum and minimum number of consecutive shifts he or she can work, a maximum number of weekends he or she is allowed to work, and a set of days on which he or she wants to have a day off. All of these requirements have to be satisfied to be feasible.

Finally, there are some personal preferences. These are summarized in two soft constraints. First, there is a set of shifts he or she wants to work. Secondly, there is a set of shifts he or she does not want to work.

To solve these problems, we need to model all of the relevant hard constraints and soft constraints. Table A.1 in Appendix A.1 shows how to use the constraint categories discussed in Chapter

4 to do this. One of the constraints states that at most one shift can be assigned on each day, so we can apply the MIP with Constraints (5.2).

Table 6.1: Descriptive statistics of the benchmark instances.

Instance	Weeks	Employees	Shifts	Shift types
1	2	8	71	1
2	2	14	108	2
3	2	20	154	3
4	4	10	182	2
5	4	16	288	2
6	4	18	299	3
7	4	20	315	3
8	4	30	482	4
9	4	36	410	4
10	4	40	693	5
11	4	50	811	6
12	4	60	1007	10
13	4	120	1737	18
14	6	32	692	4
15	6	45	941	6
16	8	20	671	3
17	8	32	1088	4
18	12	22	1116	3
19	12	40	1857	5
20	26	50	4468	6
21	26	100	8718	8
22	52	50	9633	10
23	52	100	16079	16
24	52	150	22590	32

This table shows descriptive statistics of the different benchmark instances. It shows the number of weeks that have to be scheduled, the number of employees to assign to, the number of shifts to assign, and the number of different shift types for each instance.

Chapter 7

Solving the Benchmark Instances

In this chapter, we present the results that were obtained when solving the MIP for the benchmark instances. After doing so, we identify some difficult problem instances and show an alternative approach that is better able to tackle these instances. All of the results in this thesis have been obtained by implementing the MIP using CPLEX 12.9 and running the resulting program on a 1.90 GHz processor with 16 GB of memory.

7.1 Results: Solving the MIP

Table 7.1 shows the results obtained by solving the MIP for the benchmark instances, with a maximum computation time of one hour. Presented are the lower bound, rounded up to the nearest integer, along with the objective value of the best solution, the computation time in seconds, the optimality gap between the objective value and the lower bound, and the actual gap between the objective value and the optimal solution. For instances 15 and 19 to 24 the optimal solution is not yet known, so the best-known solution is used instead. This value is taken from the results available online. Optimal values are presented in bold.

From Table 7.1 it can be seen that for eleven of the 24 benchmark instances the lower bound and objective value are equal, implying optimality. Additionally, three instances can be solved with an optimality gap of at most two percent. Finally, ten of the instances seem to be too large to solve to optimality. For three of those instances, no solution was found at all. Taking a closer look at the results, it can be seen that the optimality gap increases roughly along with the instances. However, there are some clear outliers. Instances 9, 13 and 15 seem to have been solved a lot worse than the problems of similar size.

Taking a look at the formulation shows that the number of decision variables depends on the values of the descriptive statistics from Table 6.1. Therefore, larger values imply more decision variables, which in turn might lead to a more difficult problem. From Table 6.1 it can be seen that only instance 24 has more employees than instance 13. This may explain why instance 13 is difficult to solve. It also has quite a lot of shifts to plan, compared to the other instances of comparable

Table 7.1: Results Benchmark Instances

Instance	Objective	Computation Time (sec.)	Lower Bound	Optimality Gap (%)	Best Known Solution	Actual Gap (%)
1	**607**	0.4	**607**	0.0	**607**	0.0
2	**828**	1.1	**828**	0.0	**828**	0.0
3	**1001**	1.6	**1001**	0.0	**1001**	0.0
4	**1716**	13	**1716**	0.0	**1716**	0.0
5	**1143**	37	**1143**	0.0	**1143**	0.0
6	**1950**	26	**1950**	0.0	**1950**	0.0
7	**1056**	76	**1056**	0.0	**1056**	0.0
8	1312	3606	1287	1.9	**1300**	0.9
9	**439**	3611	198	121.7	**439**	0.0
10	**4631**	719	**4631**	0.0	**4631**	0.0
11	**3443**	374	**3443**	0.0	**3443**	0.0
12	**4040**	1988	**4040**	0.0	**4040**	0.0
13	3044	3600	**1348**	125.8	**1348**	125.8
14	**1278**	650	**1278**	0.0	**1278**	0.0
15	5320	3603	3812	39.6	3834	38.8
16	3226	3607	3217	0.3	**3225**	0.0
17	5856	3603	5739	2.0	**5746**	1.9
18	4664	3602	4372	6.7	**4459**	4.6
19	6192	3609	3142	97.1	3149	96.6
20	6953	3604	4759	46.1	4943	40.7
21	921567	3601	21122	4263.2	21159	4255.4
22	-	3600	26212	-	33155	-
23	-	3600	16990	-	17428	-
24	-	3600	-	-	48777	-

This table shows the results obtained by solving the MIP for the benchmark instances. For each of the 24 instances, the objective value of the final solution is given, along with the computation time in seconds, the lower bound, the optimality gap, the best-known solution, and the actual gap. Optimal values are presented in bold.

length. Interestingly, the optimal objective value of this instance is 1,348. This is only 0.1 above the lower bound that was found. Apparently, determining a tight lower bound is a lot easier than determining a good solution. This shows that for this instance, the MIP can still be useful, as it would be able to prove that any solution with an objective value of 1,348 would be optimal. So, even though it could not determine the optimal solution itself, it might still be used to determine

when another solution approach has found the optimal solution.

For problem instance 15, there is no single aspect of the problem that seems to be especially difficult. However, all of the values in Table 6.1 seem to be relatively large when compared to the other instances of similar length. For example, instances 14 and 16 have fewer employees, fewer shifts to plan, and fewer shift types, and instances 17 and 18 have a lot fewer employees to schedule than instance 15. This is actually the smallest instance for which the optimal objective value is not yet known as the best-known lower bound presented online is 3,823 and the best-known solution is 3,834.

Finally, it seems instance 9 does not have any clear reason for being difficult to solve, but it has one of the largest optimality gaps. However, the solution that was found, was actually optimal, leading to an actual gap of 0.0 percent. Therefore, it seems that for this instance, the difficult part is determining a tight lower bound. It could be that the solution space of this instance is relatively large or that there are a lot of solutions with a very good objective value, that are close to being feasible. Looking at Table 6.1 it can be seen that the number of shifts that have to be planned is relatively low for instance 9. This may indicate that the feasible region is larger than for other instances, as employees might have a smaller workload in this instance. Secondly, it can be seen that the actual optimal value for instance 9 is the lowest of all the instances, which could also be an indication that the feasible region is relatively large, and that there may be some very good solutions, close to being feasible.

The MIP has also been able to find new lower bounds for instances 20 to 23 that were tighter than any lower bounds that were known before. For instance 20, this was increased from 4,743 to 4,759. For instance 21, a lower bound with penalty 21122 was found, the previous best was 20,883. For instance 22, an increase from 24,064 to 26,212 was found. Finally, for instance 23, the previous lower bound was just over 3,000. This has been increased to 16,990. From this, it should be clear that the MIP manages to find very tight lower bounds for the large instances.

Although the benchmark instances only comprise a single problem of each size, and it contains a different set of constraints than the constraints used by Ortec, we may still get a general idea about the size of problems that could be solved efficiently by the MIP. From Tables 6.1 and 7.1 we can see that the problems of up to 4 weeks and 60 employees or up to 6 weeks and 32 employees can be solved up to optimality. Slightly larger problems might also be attempted, but it is quite likely that such instances would take more than an hour to solve to optimality. Furthermore, from instance 9, we can see that the structure of the problem matters as well. So, problems smaller than our suggested bounds may not be solved to optimality either. Altogether, there might be some departments for which the bounds are lower, and some for which the bounds are higher, but we believe they provide a reasonable estimation.

Next, we take a closer look at the benchmark instances that could not be solved well and try to overcome this.

7.2 Solving Large Instances

Now, we believe that the main reason the last few benchmark instances cannot be solved as well as the others, is that they require too many decision variables, such that efficiently going through the branch-and-cut tree becomes very time-consuming. Therefore, we next attempt to adjust our solution approach such that fewer decision variables are needed. This can be done in four ways: reduce the number of weeks during which to schedule, reduce the number of employees to assign to, reduce the number of shifts to assign, or reduce the number of shift types.

Reducing the number of weeks could be done by using a rolling window. In such a solution approach, only a subset of the weeks is considered. Then, the shifts during these weeks are assigned as well as possible. Afterwards, we move the window forward to the next period, possibly using a small overlap window. This process is repeated until all weeks are planned.

The advantage of this approach is that the problem size can be significantly reduced. The disadvantage of this approach is that the individual periods are interdependent, and there are a lot of constraints that are relevant over the entire scheduling period. This means that ensuring that the final roster remains feasible becomes quite a tedious task.

Reducing the number of employees to assign to or reducing the number of shifts to assign seems like two approaches that are best done simultaneously. This can be done by selecting a subset of the employees and a subset of the shifts and assigning only those shifts to the employees. Then, new subsets are selected, taking into account which shifts have been assigned so far.

The final option is to reduce the number of shift types. To do so, we have developed a two-stage solution approach. In the first stage, some of the different shift types are merged and the MIP is solved using the merged shifts. Then, in the second stage, we decide which of the original shifts to assign instead of the merged shifts that were assigned in the first stage. The next sections describe this approach in more detail.

7.3 Stage One: Merging Shift Types

To simplify the original problem, we merge some of the shift types. The resulting sets of merged shifts are used when solving the MIP. While doing so, we take the original constraints into account, such that we can always find a feasible solution by replacing all of the merged shifts by one of the original shifts. To elaborate on this point, we first need to decide what happens when two different shifts are merged.

To merge two shift types, the daily demand is aggregated, and the maximum number of shifts of these types that can be assigned to a given employee are added together. Furthermore, two of the merged shift types can only be assigned on two consecutive days if at least one of the original shift types of the first merged shift can be succeeded by one or more of those of the second merged shift.

For now, we assume that only shifts of the same length are being merged. Later on, we discuss what happens when we relax this assumption. Using this assumption, the constraints concerning work time can remain unadjusted.

Next, an employee may have a preference to (not) work a given shift on a given day. If he does want to perform it, we can incorporate the resulting soft constraint in the model, since it is automatically violated if a merged shift is assigned that does not contain this shift. Alternatively, if he does not want to perform the shift, we might not be able to add the resulting soft constraint to the model. Since we do not know which of the merged shifts the employee will eventually perform, we cannot know whether or not he is going to work the unwanted shift. As such, we can only incorporate this constraint if the shift is not merged at all, if the employee is not allowed to perform any of the other shifts in the merged shift, or if all shifts in the merged shift have the same penalty on the same days.

All other constraints do not need to be handled differently when using merged shifts. So, we can apply the MIP with the above adjustments, on the new set of shifts. The resulting solution will not be feasible yet, as it is not using the original shifts. As such, we now need to adjust it to determine a feasible solution to the original problem. The next section describes this in more detail.

7.4 Stage Two: The Shift Distribution Problem

To translate the solution from stage one to a feasible solution, one might set all shift variables x_{is} to 0, except for those shifts included in the merged shifts that have been assigned. However, we decide to include some more flexibility. If on a given day one of the merged shifts is assigned, we allow that any shift can be assigned on this day, rather than just one of the shifts in this merged shift. On all other days, the employee cannot work at all. We refer to the resulting problem as the shift distribution problem. In essence, the shift distribution problem is equivalent to the original problem, except that all day-off variables are fixed in advance. Note that it is quite important that it is allowed to overschedule some of the shifts, as it may not be possible to avoid doing so.

By allowing all shifts to be used, the feasible region of the shift distribution problem is a lot larger than if we would only allow one of the merged shifts to be used. This means the optimal objective value will be lower or equal than if we would not include this flexibility. Therefore, the only disadvantage of the shift distribution problem is that it may be more difficult to solve than a problem in which we would only allow one of the merged shifts to be used. However, from experimentation, we find that the shift distribution problem is quite easy to solve. In general, it takes at most a minute of computation time to solve it to optimality.

One reason the shift distribution problem might be quite easy to solve is that the feasible region is much smaller than that of the original MIP. Furthermore, many constraints become irrelevant and can be omitted from the problem. To be more specific, all four sets of constraints relating to the

length of a series (shift series and days off series) can be omitted since we have already fixed when we will have days off. Next, the constraints imposing a maximum on the number of weekends can also be removed, because we already know when we are going to work or not. Finally, if all shifts have the same length, the constraints imposing limits on the total work time can also be removed, since we already determined how many shifts we are going to work. In this thesis, we only use the shift distribution problem to convert the solution found in stage one to a feasible solution. However, the only required input of the shift distribution problem is a set of working days. That means that any feasible solution can be used as input. Because it seems to be quite easy to solve, it may be worthwhile to apply the shift distribution problem after any solution approach to potentially refine the final solution.

In summary, we merge some of the shifts in stage one and solve the MIP. The result is then used to determine when an employee should work. This is taken as input for the shift distribution problem. The next step is to decide which shifts should be merged. This is explained in the next section.

7.5 Selecting Shifts to Merge

Now, one has to decide which shifts to merge. By merging more shifts, the objective value of the problem will be larger or equal than before, but the problem becomes simpler. As such, it depends on the instance at hand what approach works best. For example, note that by merging no shifts at all, we are solving the original problem. Therefore, for all instances that can already be solved to optimality, it is best not to merge anything at all. However, for the instances that cannot be solved to optimality, it may be worthwhile to merge some of the shifts. To make this decision easier, we suggest using one of the following approaches.

One possibility is to merge all equivalent shifts. In this case, we decide that shifts are equivalent if they have the same length, and can be proceeded by the same set of shifts. We suggest that this should always be the first step in finding more shifts to merge. However, this may not lead to enough merged shifts yet.

Another possibility is to merge all shifts. In this case, the problem becomes a lot simpler as the only decision that needs to be made is what days to work. Therefore, all shift variables can be removed, leading to a large reduction in decision variables. The disadvantage of this approach is that every shift type is treated the same way. This can cause some problems with the total work time. For every employee, the total work time has to be between given limits. If we treat every shift type the same way, we need to assign a certain length to the merged type. In this thesis, we set this length to be the weighted average length of the shifts, with weights depending on the maximum number of times the shift can be assigned. However, when using this approach we could end up with an infeasible roster. For example, a given employee might get just enough days of