TU WIEN Informatics

Exact Methods for the Time Frame Rostering Problem

In the Context of Tram Driver Rostering

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Lukas Frühwirth, BA
Matrikelnummer 51836522

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dr.techn. Nysret Musliu

Wien, 12. August 2024

Lukas Frühwirth

Nysret Musliu

TU WIEN Informatics

Exact Methods for the Time Frame Rostering Problem

In the Context of Tram Driver Rostering

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Lukas Frühwirth, BA
Registration Number 51836522

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dr.techn. Nysret Musliu

Vienna, August 12, 2024

Lukas Frühwirth

Nysret Musliu

Erklärung zur Verfassung der Arbeit

Lukas Frühwirth, BA

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 12. August 2024

Lukas Frühwirth

Danksagung

Ich möchte meiner Familie für ihre Unterstützung in den letzten Jahren meinen tiefsten Dank aussprechen. Besonderer Dank gilt meiner Partnerin, Teresa, für ihre kontinuierliche Ermutigung, für das Korrekturlesen meiner Arbeit und für das Hinterfragen meiner Ideen, was wesentlich zur Qualität dieser Diplomarbeit beigetragen hat. Ich bin auch meiner Mama sehr dankbar, dass sie es mir ermöglichte, mich auf das Schreiben zu konzentrieren, während sie sich um Ilvy kümmerte.

Außerdem möchte ich Uschi und Bernhard herzlich danken, die mir einen Raum zur Verfügung stellten, um an meiner Diplomarbeit zu arbeiten, sowie für ihre Unterstützung in der Betreuung von Ilvy. Zusätzlich danke ich dem öffentlichen Verkehrsunternehmen für die Bereitstellung der Daten und für die Einführung in das Thema.

Ich möchte mich auch bei Lorenz bedanken, mit dem ich über manche Methoden dieser Arbeit diskutieren konnte. Ein besonderer Dank gilt auch meinem Betreuer, der mich über das letzte Jahr hinweg unterstützte und mich ermutigte, diese Arbeit als Artikel bei der PATAT Konferenz 2024 einzureichen.

Ohne all diese Unterstützung wäre es mir nicht möglich gewesen, meine Diplomarbeit zu vollenden.

Acknowledgements

I would like to express my deepest gratitude to my family for their support over the past years. Special thanks go to my partner, Teresa, for her continuous encouragement, for proofreading my work, and for challenging my ideas, which greatly contributed to the quality of this thesis. I am also profoundly grateful to my mother for enabling me to focus on my writing by taking care of Ilvy.

Furthermore, I extend my heartfelt thanks to Uschi and Bernhard for providing me with a room to work on my thesis, as well as for their support in taking care of Ilvy. Additionally, I am thankful to the public transportation company that provided the data and introduced me to the topic at hand.

I would also like to thank Lorenz, with whom I could discuss various methods used in this work. A special thanks also goes to my supervisor, who supported me throughout the past year and encouraged me to submit this work as an article to the PATAT Conference 2024.

Without all this support, I would not have been able to complete my thesis.

Kurzfassung

In dieser Diplomarbeit wird eine formale Definition eines in der Praxis vorkommenden kombinatorischen Optimierungsproblems im Bereich der Personaleinsatzplanung, speziell im Bereich der Dienstplanung für Straßenbahnfahrer:innen, vorgestellt. Aktuelle Ansätze in der Straßenbahn-Turnuserstellung bieten den Fahrer:innen oft nur unzureichende mittelfristige Planungssicherheit. Es ist daher ein Ziel im öffentlichen Verkehrswesen, robustere Dienstpläne zu erstellen, die den Mitarbeiter:innen eine solche mittelfristige Planungssicherheit bieten. Um dieses Problem anzugehen, führe ich das Konzept des Rahmenzeiten-Turnusses formal ein. Anstatt den Turnuspositionen direkt Schichten zuzuweisen, werden den Positionen zunächst Rahmenzeiten zugewiesen. Diese Rahmenzeiten sind Zeitintervalle weit genug, um mehrere Schichten abdecken zu können. Die Schichtzuweisung erfolgt erst wenige Tage vor dem eigentlichen Arbeitstag. Somit verbessern Rahmenzeiten-Turnusse die mittelfristige Planungssicherheit, da Straßenbahnfahrer:innen garantiert wird, nur Schichten innerhalb ihrer zugewiesenen Rahmenzeit zu erhalten.

Das Ziel des Rahmenzeiten-Turnusproblems besteht darin, Rahmenzeiten optimal einem Turnus zuzuweisen, sodass verschiedene Bedingungen erfüllt werden. In dieser Arbeit gebe ich eine allgemeine Problembeschreibung sowie eine formale Definition des Rahmenzeiten-Turnusproblems. Basierend auf dieser Definition wird ein löser-unabhängiges Modell des Problems vorgestellt. Darüber hinaus vergleiche ich zwei Löser, Gurobi und OR-Tools CP-SAT, anhand realer Instanzen und zeige, dass optimale oder nahezu optimale Lösungen in angemessener Zeit gefunden werden können. Zusätzlich überprüfe ich diese Lösungen mittels Monte-Carlo-Simulationen. Dabei werden die Abwesenheiten der Straßenbahnfahrer:innen simuliert und es wird überprüft, ob die anschließende Schichtzuweisung weiterhin möglich ist. Die Ergebnisse dieser Simulationen zeigen, dass das Modell den Anforderungen der allgemeinen Problembeschreibung entspricht.

Abstract

In this diploma thesis, I introduce a formal definition of a real-world combinatorial optimization problem in the field of workforce scheduling, specifically in the area of tram driver rostering. Current tram driver rostering methods struggle with providing the tram drivers with medium-term planning security. Thus, creating more robust rosters that offer such medium-term planning security for employees is a desired goal in the public transportation sector. To tackle this problem, I formally introduce a new approach called time frame rostering. In this approach, instead of directly assigning shifts to roster positions, time frames are first allocated to roster positions. These time frames are intervals wide enough to accommodate a variety of shifts. The shift assignment takes place only a few days before the actual workday. Thus, time frame rosters provide medium-term planning security, as tram drivers are only assigned shifts within their designated time frames.

The goal of the time frame rostering problem is then to optimally assign time frames to a roster such that several constraints are met. In this thesis, I provide a high-level problem description as well as a formal definition of the time frame rostering problem. Based on this definition, a solver-agnostic model of the problem is presented. Furthermore, I compare two state-of-the-art solvers, Gurobi and OR-Tools CP-SAT, on real-world instances and demonstrate that optimal or almost optimal solutions can be found in a reasonable amount of time. Additionally, these solutions are verified by Monte Carlo simulations. In this approach, I simulate the tram drivers' absences and check whether subsequent shift assignment is still possible. The results of these simulations show that the model works as intended and corresponds to the high-level problem description.

Contents

CHAPTER 1

Introduction

Similar to other professions that operate in shifts, such as those in the medical field or industrial manufacturing, the shifts of tram drivers are assigned to a rotating schedule called a roster. Traditionally, this roster was created by assigning shifts to specific roster positions either by hand or by using workforce scheduling algorithms [HHB20]. However, this approach proved inconvenient for tram drivers as well as rostering managers. A roster is typically scheduled weeks or months in advance, hence it undergoes several changes until the final shift assignment due to changes in shift plans, fluctuations in staff headcount, and various other factors. Consequently, the literature covers methods for constructing more robust rosters [GVX22], [WSVB21], such as calculating the optimal amount of reserve shifts [XS12], [IM15], as well as re-rostering methods [Wic19]. These methods, however, might still be inconvenient for tram drivers, as they potentially require a substantial number of reserve shifts or the repeated reallocation of shifts, which deteriorates the medium-term planning security for tram drivers.

As a consequence thereof, the idea emerged to introduce a time frame roster. In this approach, instead of directly assigning shifts to specific roster positions, first, time frames are assigned to roster positions. These time frames are intervals wide enough to accommodate a variety of shifts. The final shift assignment takes place only a few days before the actual workday. At this stage, shifts replace the drivers' time frames where the shifts' intervals fit within the time frames' intervals. Thus, the time frame roster provides medium-term planning security for tram drivers, as drivers will only be assigned shifts within their designated time frames.

However, to the best of my knowledge, the existing literature does not provide a formal definition or a formulated model of the time frame rostering problem that is considered in this thesis. Furthermore, an efficient method for optimally assigning time frames to roster positions while considering specific criteria is currently unavailable. A main criterion is, for example, that even if drivers are absent, it must still be possible to assign shifts to drivers without violating their time frames.

The time frame rostering problem is a real-life combinatorial optimization problem with specific requirements tailored to the operation of tram networks. The design of the time frames requires a negotiation process and agreement between the employer and employees. Therefore, for the purpose of this thesis, the time frames and their specifications (start time, end time, type), which were provided by a public transportation company, are regarded as predetermined.

1.1 Aim of the Thesis

The aim of this thesis is to investigate whether the time frame rostering problem can be solved optimally or near optimally in a time-efficient and practically feasible manner. In the context of tram driver rostering, this implies obtaining a close-to-optimal solution within a few hours, preferably in less than two hours. To achieve this, my objectives are as follows:

- To formally define the time frame rostering problem.
- To formulate a solver-agnostic model of the time frame rostering problem.
- To compare and empirically evaluate two different exact solving methods, integer programming (IP) and constraint programming (CP), using real-world instances.

1.2 Contributions

The main contributions of this thesis are:

- I introduce a real-world combinatorial optimization problem in the context of tram driver rostering.
- I provide a formal problem definition and formulate a solver-independent model of the time frame rostering problem.
- I publish real-world instances based on data provided by a public transportation company.
- I empirically evaluate different exact solving methods using real-world instances and show that these are (almost) optimally solvable within 120 minutes.
- I demonstrate the feasibility of my solutions by simulating staff absences and subsequent shift assignment.

1.3 Structure of the Thesis

This thesis is structured as follows: In the next chapter, I provide some theoretical background to constraint and integer programming. In Chapter 3, I present a high-level problem description as well as an overview of the related work. This is followed by a formal definition of the time frame rostering problem, and a solver-agnostic model formulation in Chapter 4. Subsequently, in Chapter 5, I evaluate the model by solving real-world instances using the MIP solver Gurobi [Gur23] and the CP solver OR-Tools CP-SAT [PD23]. Additionally, several variable and domain-value selection strategies as well as applying redundant constraints and symmetry breaking are evaluated. The solutions are then verified by using Monte Carlo Simulations and by modeling the subsequent shift assignment. Finally, I draw my conclusions in Chapter 6.

CHAPTER 2

Theoretical Background

In this thesis, I will address the time frame rostering problem by modeling it as a constraint optimization problem (COP) in a solver-independent manner. I will then compare the performance of two solvers, each utilizing different exact methods, in solving the time frame rostering problem. The first solver, OR-Tools CP-SAT, combines constraint programming and SAT-solving techniques, while the second solver, Gurobi, employs integer programming algorithms. Accordingly, this chapter will provide the theoretical background to these two approaches.

2.1 Constraint Programming

Constraint programming (CP) is a paradigm for solving constraint satisfaction problems (CSP) which are typically complex combinatorial search problems. A variety of techniques stemming from research in artificial intelligence, theoretical computer science, databases, and operations research have been successfully used to solve such combinatorial problems in fields like scheduling, planning, routing, molecular biology, graph algorithms, computer vision, linguistics, (electrical) engineering, and many more [RVBW06], [Apt03]. Since CSPs are the cornerstone of constraint programming, I first need to define what a CSP is, using the definition from Freuder and Mackworth [RVBW06, p. 16]: A CSP is a triple $\mathcal{P} = \langle X, D, C \rangle$ consisting of:

$X = \langle x_1, ..., x_n \rangle$	n-tuple of variables
$D = \langle D_1, ..., D_n \rangle$ s.t. $x_i \in D_i$	n-tuple of domains
$C = \langle C_1, ..., C_m \rangle$ where $C_j = \langle R_{S_j}, S_j \rangle$	m-tuple of constraints

The relation R_{S_j} is a subset of the k-ary Cartesian product of the domains defined in the scope S_j:

$$R_{S_j} \subseteq D_1 \times ... \times D_k \text{ where } D_i \in S_j,\ |S_j| = k$$

An n-tuple $A = \langle a_1, ..., a_n \rangle$ s.t. $a_i \in D_i$ satisfies a constraint $C_j = \langle R_{S_j}, S_j \rangle$ iff $A_{S_j} \in R_{S_j}$, where A_{S_j} is a restricted k-tuple of A to variables of which their domains are in S_j. A constraint C_j is referred to as unary, binary, or k-ary if S_j consists of one, two or k domains respectively. A solution to the CSP $\mathcal{P}$ is then an n-tuple A that satisfies every constraint $C_j \in C$. CSPs are classified as satisfiable or consistent if a solution exists; otherwise, they are deemed unsatisfiable or inconsistent.

To find a solution to a CSP with finite domain size, one could enumerate all possible n-tuples and check each one to see if all constraints are satisfied. However, because CSPs are typically combinatorial search problems, this simple brute-force approach quickly becomes infeasible. Consequently, a variety of algorithms have been developed to expedite the search process. These algorithms can be categorized into two groups: inference and search techniques.

One of the primary inference methods developed in the 1970s is the use of network consistency algorithms for constraint propagation [Mac77]. These algorithms derive new constraints by identifying implicit constraints and making them explicit, thus narrowing the search space. An essential search technique that guarantees finding a solution is backtracking [GB65]. In backtracking, a solution is constructed by assigning values to variables incrementally. If an assignment violates a constraint, the algorithm backtracks by undoing the last assignment and trying a new value. To further enhance performance, inference and search methods are usually combined. In addition to these techniques, solvers like OR-Tools CP-SAT solver employ SAT-solving methods like Conflict-Driven-Clause-Learning [RVBW06], [MMZ+01].

CSPs and their constraints are defined in a general way, which enables the modeling of many real-world problems that require heterogeneous constraints. For instance, constraints can be arithmetic expressions (like $<$, $=$, ...), logical expressions (such as all variables must be different) or extensional, explicitly enumerating satisfying relations. Combinatorial optimization problems can also be addressed by transforming a CSP into a constraint optimization problem (COP). This is achieved by adding an objective function that is either minimized or maximized. COPs introduce additional solving methods, such as branch and bound algorithms [MJSS16].

The generality of CSPs and COPs presents a dual challenge: while it may be more difficult to apply specialized algorithms, this same generality allows for the modeling of problems that cannot be easily addressed by other methods, such as mixed integer programming (MIP), in a straightforward manner.

A typical example for a CSP is the n-queens problem [GB65]. The goal is to place n queens on an $n \times n$ chess board such that they cannot attack each other. Here is a formulation of the n-queens problem as a CSP:

$$Q = \langle q_1, ..., q_n \rangle \qquad n\text{-tuple of queens}$$
$$D = \langle D_1, ..., D_n \rangle \text{ s.t. } D_i = \{1, ..., n\} \qquad \text{only one queen per column possible}$$

There are several ways to formulate this problem. In the formulation above, I concentrate solely on the queens' positions in the rows, while fixing their columns. Specifically, q_1 will be in column 1, q_2 in column 2, and so forth. I do not fix the rows, thus, the first constraint ensures that all variables are assigned different values (i.e., rows): $C_1 := alldifferent(Q)$. However, since queens can also attack diagonally, we need another constraint: $C_2 := |q_i - q_j| \neq |i - j|$.

The all-different constraint, which states that all variables must be pairwise different, is an excellent example of a global constraint. A global constraint is a high-level constraint with a non-fixed number of associated variables and offers two main advantages: it allows for the modeling of complex properties and ensures that these properties can be efficiently solved using specialized constraint propagation algorithms [RVBW06].

2.2 Integer Programming

Integer programming (IP) belongs to the field of mathematical programming and deals with discrete optimization problems. Integer programming problems can also be seen as a subset of COPs mentioned in the previous section. Consequently, the areas of application are similar to those of constraint programming, such as planning, scheduling, and rostering. Often, when referring to an integer programming problem, actually an integer linear programming (ILP) problem is meant. I will define an ILP as a minimization problem. My definition is based on one provided by Nemhauser and Wolsey [NW99, p. 4]:

$$\begin{aligned} \min\ & cx \\ \text{s.t. } & Ax \geq b \\ & x \in \mathbb{Z}_0^+ \end{aligned}$$

where the unknown variable $x = (x_1, ..., x_n)$ is a non-negative integral n-dimensional vector, c is a given n-dimensional vector, A is a given $m \times n$ matrix, and b is a given m-dimensional vector. The feasible region S is then defined by the set $S = \{Ax \geq b,\ x \in \mathbb{Z}_0^+\}$ and $x \in S$ is a feasible solution. The function $f(x) = cx$ is referred to as the objective function. A solution $x_{opt} \in S$ is optimal iff $cx_{opt} \leq cx\ \forall x \in S$ holds.

The fundamental techniques to solve ILPs date back to the 1940s to 1960s. Dantzig, who introduced the simplex algorithm – the first efficient and still most important algorithm to solve linear programs – in 1947 [Dan51], also was one of the first to develop techniques for ILPs in 1957 [Dan57]. One year later, Gomory [Gom58] introduced the Gomory

cuts, a special cutting-plane method for integer programming. The simplex algorithm is used to solve a relaxation (where x no longer has to be a pure integer vector) of the ILP. Based on the simplex tableau, a cutting plane (i.e., constraint) is added that cuts off fractional solutions while not affecting the feasible region S which contains the integer solutions. Another important method to solve ILPs is the branch-and-bound technique presented by Land and Doig in 1960 [LD60]. Again, the LP relaxation of the ILP is solved. If this solution contains non-integer variables, one of those is chosen and branched upon, meaning the search tree is split into two subtrees: in one, the variable is rounded down to the next integer, and in the other, rounded up. For both subtrees, the LP relaxation is solved, and the obtained objective value is used as a lower bound. If the best-known solution containing only integers is below this lower bound, the subtree is pruned. Padberg and Rinaldi combined these techniques and presented the first branch-and-cut algorithm [PR91].

The methods presented above are still the foundation for state-of-the-art ILP solvers like Gurobi. State-of-the-art solvers additionally use advanced and improved implementations of these techniques, together with preprocessing, heuristics, concurrent solving, and parallel computing. A modeling disadvantage of ILPs is that the constraints have to be linear. Solvers like Gurobi 11 address this issue by allowing a variety of nonlinear constraints, thus enabling the modeling of nonlinear problems. This is achieved by translating nonlinear constraints (such as MIN, MAX, AND, OR constraints) into linear ones (usually by introducing auxiliary variables) and by approximating function constraints (like $y = a^x$) through piecewise-linear approximation [Gur23].

The most common method for solving large ILPs in scheduling and routing is the column generation method, introduced by Gilmore and Gomory in 1961 [GG61]. Large ILPs often have too many variables to consider, hence the basic idea is to decompose the problem into smaller, more manageable subproblems. The master problem is a reduced form of the ILP, initially considering only a subset of variables (columns). This master problem is solved, and new variables are iteratively added until the objective value can no longer be improved. The task of deciding which variable to add next is called the subproblem [Lüb10].

CHAPTER 3

Problem Description and Related Work

3.1 Problem Description

This section provides a high-level problem description using a small sample roster. First, I outline a conventional rostering process that resembles a real-world implementation in a public transportation company. Second, I detail how the time frame rostering methodology diverges from this traditional approach, highlighting the differences and innovations this thesis introduces.

Conventional Approach

Typically, tram driver rostering begins with a set of shifts containing all shifts for a week. Based on the size of this set, the demand for drivers and their days off is calculated. A roster containing only the days off is created, as shown in Table 3.1. This roster is called the day off schedule. Each driver has either two specific consecutive days off or follows a rotational day off schedule. This results in eight different day off types, where types 0 to 6 have two specific consecutive days off, while type 7 follows a rotational day off schedule. The construction of the day off schedule determines the sizes of these day off types, i.e., the number of rows in the roster. Tram drivers are then assigned to different roster weeks within their day off type and rotate through the roster weeks of their own type in ascending order. Upon reaching the last week of their day off type they continue with the first week of their type.

After the creation of the day off schedule, the shifts are allocated. In the exemplary shift roster shown in Table 3.2, each shift s represents a unique shift from the set that contains all shifts for a week. Each shift includes specific details about the work location (tram line), work hours (start time, breaks, end time), and whether it is a split shift (with a

day off type	Mo	Tu	We	Th	Fr	Sa	Su
o_0	off						off
o_0	...						...
o_1	off	off					
o_1	...	...					
o_2		off	off				
o_2		...	...				
o_3			off	off			
o_3			..	...			
o_4				off	off		
o_4				...	...		
o_5					off	off	
o_5					...	...	
o_6						off	off
o_6						...	...
o_7	off	off					
o_7		off	off				
o_7			...	...			

Table 3.1: Exemplary Day Off Schedule

day off type	Mo	Tu	We	Th	Fr	Sa	Su
o_0	off	s	s	s	s	s	off
o_0	...	s	s	s	s	s	...
o_1	off	off	s	s	s	s	s
o_1	...	...	r_e	r_e	r_e	r_e	r_e
o_2	s	off	off	s	s	s	s
o_2	s	...	...	s	s	s	s
o_3	r_l	r_l	off	off	s	s	s
o_3	s	s	..	...	r_l	r_l	r_l
o_4	s	s	s	off	off	s	s
o_4	s	s	s	...	...	s	s
o_5	s	s	s	s	off	off	s
o_5	s	s	s	s	...	...	s
o_6	s	s	s	s	s	off	off
o_6	r_e	r_e	r_e	r_e	r_e	...	...
o_7	off	off	s	s	s	s	s
o_7	s	off	off	r_l	r_l	r_l	r_l
o_7	r_l	r_l	...	...	s	s	s

Table 3.2: Exemplary Shift Roster

day off type	Mo	Tu	We	Th	Fr	Sa	Su
o_0	off	8	7	6	5	5	off
o_0	...	13	15	4	2	1	...
o_1	off	off	8	7	6	5	11
o_1	...	...	12	14	3	2	1
o_2	2	off	off	8	8	7	6
o_2	5	...	...	11	15	4	3
o_3	1	1	off	off	10	8	7
o_3	5	15	..	...	4	3	2
o_4	3	2	1	off	off	9	8
o_4	7	6	11	...	...	13	11
o_5	14	2	2	1	off	off	8
o_5	7	5	11	15	...	...	4
o_6	12	14	3	2	1	off	off
o_6	8	6	6	5	15	...	...
o_7	off	off	13	12	3	2	2
o_7	1	off	off	4	3	3	2
o_7	2	1	...	...	9	10	7

Table 3.3: Exemplary Time Frame Roster

several-hours-long break) or not. Since the size of the day off schedule is determined by accounting for absences, among other things, there are considerably more roster positions than shifts. These empty roster positions are filled with reserve shifts, denoted as either r_e or r_l in Table 3.2. Reserve shifts provide drivers with rough time windows, typically distinguishing only between an early time window r_e (only shifts starting before noon are allowed) and a late time window r_l (only shifts starting after noon are allowed). Drivers with reserve shifts receive notice of their actual shift or if they are on stand-by only a few days before their scheduled workday.

New Approach – Time Frame Rostering

The issue arising from the traditional approach is that drivers with reserve shifts do not know in advance when they will have to work. Moreover, absences of tram drivers and changes in the shift plans might require a reallocation of shifts. The proposed method to prevent reserve shifts and reallocation is to introduce a time frame roster. In a time frame roster, rather than directly assigning shifts, time frames are initially allocated to roster positions.

In the time frame roster outlined in Table 3.3, each time frame is indicated by a number. Time frames are essentially intervals with predefined start and end times. The shift assignment is postponed until a few days before the actual workday, by which time many absences are already known to the roster managers. Shifts are then assigned to time frames so that they fit within the intervals of the frames and are of the correct type. Some

time frames allow for, or even require, split shifts, while others cannot accommodate split shifts.

The objective of the time frame rostering problem is to optimally assign time frames to roster positions so that the shift assignment at a later stage remains possible. This requirement can be broken down into three major constraints:

Firstly, tram drivers may be absent with a certain probability, resulting in two scenarios. On the one hand, at the time of shift assignment, there might be more time frames than shifts. In this case, it must be possible to assign all shifts to time frames; hence, it cannot be the case that shifts remain unassigned because they do not fit within the time frames. On the other hand, if there are fewer time frames than shifts, then all time frames must be assigned a shift, and it cannot be the case that there are time frames left in which none of the remaining shifts fits. The remaining shifts are then covered by drivers working overtime.

Secondly, to adhere to rest period regulations, two consecutive time frames cannot appear in the list of forbidden sequences.

Thirdly, there are restrictions on split shifts during a workweek. A formal definition of the problem and its constraints is provided in the next Chapter 4.

3.2 Related Work

Tram driver rostering is a subset of driver rostering within public transportation and it is very similar to bus driver and subway driver rostering. Driver rostering itself is a subset of crew planning in public transport, where each crew consists of just a single member, in our case, the tram driver. Thus, the time frame rostering problem is a specific challenge encountered within crew planning and, more broadly, within workforce scheduling. It is closely related to both the driver rostering and crew rostering problem. To the best of my knowledge, the time frame rostering problem defined in this thesis has not been addressed in the existing literature.

Crew planning for public transportation typically comprises two primary stages. The first stage is crew scheduling, also known as shift design, shift scheduling or duty scheduling, where shifts are designed based on a predetermined timetable. In this thesis, I use the term "shift" instead of "duty". A shift has a start time, end time, and type, with each tram driver working one shift per workday. The second stage, referred to as crew rostering, involves assigning these shifts to typically rotating or cyclical rosters [HHB20]. These rosters are often created anonymously, meaning shifts are assigned to anonymous roster positions rather than to specific drivers [BSSW17]. The time frame rosters considered in this thesis are also created anonymously.

It is important to note, that the two stages, (crew) scheduling and (crew) rostering, are also integral components of many workforce scheduling approaches in general [Lau96]. Hence, there exists extensive literature on how crew scheduling [HHB20], crew rostering

[HHB20], and workforce scheduling problems in general [BCBL04, EJKS04, VBD+13], can be addressed using mathematical programming, constraint programming, answer-set programming, heuristics, metaheuristics, and combinations thereof. Due to their complexity, the two stages are often treated as separate optimization problems. Nevertheless, Ernst et al. [EJK+01] introduced an integrated optimization model for train crew management, and Lin et al. [LT19], [LJC20], as wells as Borndörfer et al. [BSSW17], showed that integrated approaches in crew planning with a single objective function can yield better solutions and increase driver satisfaction. On a more general level, solving the general employee scheduling problem, as demonstrated by Kletzander and Musliu [KM20], is also done by integrating several stages.

These integrated approaches are relevant for the time frame rostering problem, as it consists of constraints and uses modeling techniques from both stages. Time frame rostering can be seen as an intermediate step between scheduling and rostering. In crew or shift scheduling, mathematical programming is the most popular approach [VBD+13], typically using a set covering formulation introduced by Dantzig in 1954 [Dan54]. The model of the time frame rostering problem proposed in this thesis is also based on such a formulation.

To obtain solutions for real-world instances of the crew scheduling and crew rostering problem within a reasonable amount of time, the most common approach is the column generation method (see Section 2.2). Lin et al. [LJC20] demonstrated that their branch-and-price-and-cut algorithm, which is essentially a combination of column generation and branch-and-bound techniques, effectively solves real-world instances of the integrated bus driver scheduling and rostering problem. Yunes et al. [YMDS05] developed a hybrid column generation technique that utilizes both mathematical and constraint programming to solve urban transit crew planning problems. Since I introduce a formal definition to a problem for which no existing approaches and techniques have been tested, I will provide a solver-agnostic model that allows us to compare integer and constraint programming methods.

As mentioned in the introduction, a crew schedule, and particularly a crew roster, undergoes several re-rosterings due to a variety of factors. These factors include changes in timetables, construction sites, fluctuations in headcount, staff absences, and more. A conventional roster, in which shifts are directly assigned to roster positions, is highly susceptible to such changes, and they often require reassigning a subset of the shifts or even creating an entirely new roster. Thus, the literature offers methods to deal with the inherent uncertainty by constructing more robust rosters [GVX22], [WSVB21]. In this thesis, I define the robustness of a roster as the degree to which the roster is insensitive to disturbances [IEE90, p. 64], [VJLS94], [GVX22]. Methods to managing disturbances in the rostering process can be divided into two categories: anticipatory and reactive approaches. Anticipatory approaches aim to create more robust roster by introducing, for example, reserve shifts [XS12]. In contrast, reactive approaches focus on advanced re-rostering methods that create new rosters by making, for example, as few changes as possible to existing rosters [Wic19]. Xie et al. [XS12] present a stochastic model for

the rota (i.e., rotating workforce) scheduling problem in public bus transport, which is a subset of the crew rostering problem. Traditionally, the number of reserve shifts per weekday was determined either as a fixed number or as a percentage of the daily shifts. However, this simple approach did not account for fluctuating absence rates on different weekdays, leading to suboptimal costs for the company and reduced driver satisfaction. Xie et al. addresses this issue by considering weekday-specific absence rates and by introducing optional reserve shifts in addition to the regular reserve shifts. As a result, the model can determine the optimal number of reserve shifts based on the specific absence rates. Ingels and Maenhout [IM15] propose a different but still anticipatory method. First, they construct a roster that contains work shifts, reserve shifts, and days off. In this roster the reserve shifts are scheduled using a predefined strategy. They then simulate uncertainties on a daily basis and, based on these simulations, convert work shifts to reserve shifts or vice versa, reassign shifts, or cancel shifts as needed. Finally, they evaluate the newly crafted roster by comparing it to the initial roster constructed in the first step. Ingels and Maenhout also acknowledge the trade-off between anticipatory and reactive approaches. Rosters with a large number of reserve shifts are highly robust, but may result in many unnecessary shifts, and drivers might not know their actual working times until shortly before their shifts. Conversely, rosters with fewer reserve shifts make sure that most drivers know their working times well in advance, but there might not be enough personnel to cover all shifts, and even minor disturbances can lead to last-minute changes of the drivers' schedules, which have a detrimental impact on the drivers' personal lives [WSVB21]. This trade-off is also addressed by Wickert et al. [WSVB21], who introduce two metrics to estimate the robustness of a roster against unexpected staff absences and the expected re-rostering costs. Using these metrics, they demonstrate the level of robustness that leads to the best overall solution and discuss how this level is highly dependent on the specific rostering problem.

I propose a methodology for constructing more robust rosters, which has recently started to be used in practice but has not yet been formally defined or discussed in the scientific literature. The method also differs substantially from thoses mentioned in the previous paragraph. Time frame rostering acts as an intermediate stage between shift scheduling and rostering and thus, is an anticipatory method. This approach introduces time frames, which are initially assigned to rotating rosters based on the crew schedule (shifts). Subsequently, shifts are allocated within these time frames. To the best of my knowledge, this method has not been proposed before. The time frame roster is more robust to changes than a typical shift roster because the shift assignment occurs at a later stage, where many factors leading to re-rostering are already known and can be accounted for in the assignment process.

CHAPTER 4

Formal Definition and Solver-Independent Model Formulation

To formally define the time frame rostering problem, I will first specify the given data. Secondly, I will explain what constitutes a time frame roster, and the role it plays in tram driver rostering. Thirdly, I will outline how the day off schedule is created. Finally, I will define the hard constraints, soft constraints, decision variables, and the objective function.

4.1 Provided Data

The provided data comprises four groups: shift-related, time frame-related, day off-related and constraint-related. This section specifies each of them.

1. Shift data:
   - There is a set of n shifts denoted by $S = \{s_1, ..., s_n\}$. Each shift has a start and end time denoted as an interval $[a_{s_i}, b_{s_i}],\ i \in \{1, ..., n\}$.
   - The weekday of a shift is represented by $w_{s_i},\ i \in \{0, ..., 6\}$.
   - S_i represents the set of shifts for weekday i:
     $s \in S_i \Leftrightarrow w_s = i,\ i \in \{0, ..., 6\}$.
   - Each shift has an assigned shift type t_{s_i}, with the value of 1 if the shift is a split shift and 0 otherwise.

2. Time frame data:

   - There is a set of m time frames indicated by $F = \{1, ..., m\}$, each with a start and end time forming an interval $[c_i, d_i],\ i \in \{1, ..., m\}$.
   - Each time frame has an assigned type t_i, with 0 allowing only shifts of type 0, 1 allowing only shifts of type 1, and 2 allowing shifts of any type.

3. Day off data:

   - There are 8 day-off types, o_0 to o_7. Types o_0 to o_6 have two fixed consecutive days off, while type o_7 follows a 16-week rotating schedule, which is commonly used in practice. This schedule includes 12 weeks rotating through day-off pairs (e.g., So/Mo, Mo/Tu, ..., Fr/Sa), plus four additional weeks with weekends off, due to fewer shifts on weekends.

4. Constraint data:

   - A list seq_{fb} of forbidden time frame sequences.
   - A list seq_{ud} of undesirable time frame sequences and their penalties.
   - A list $forb_{wt}$ of forbidden times frames for each workweek type wt. This workweek type is used to encode, on one hand, the proximity of a workday to the next day off, and on the other hand, whether a roster position is designated for early or late shifts.

4.2 Definition of a Time Frame Roster

A time frame roster R possesses the following properties:

- The size of the roster is determined by the sum of the sizes of each day off type. Thus, a roster R consists of $|R| = \sum_{i=0}^{7} |o_i|$ roster weeks.
- Each roster week $r_{o_i^j}$ has an assigned day off type o_i at week j and consists of 7 days.
- A roster position $r_{o_i^j,k}$ is then defined as a specific weekday k within the roster week $r_{o_i^j}$.
- Each roster position, excluding those designated as days off, will be assigned a time frame.
- Each roster position $r_{o_i^j,k}$ features a workweek type wt, ranging from -1 to 30.
- R_i represents a list of time frames assigned to roster R for weekday i.

4.2.1 Operating Principle of the Roster

Each driver is associated with a specific day off type o_i and a unique roster week $r_{o_i^j}$. Tram drivers rotate through the roster weeks of their own day off type in ascending order. Upon reaching the last week of their day off type $o_i^{|o_i|}$, they continue with the first week of their day off type o_j^1. However, there might be more roster weeks than drivers, since the number of weeks assigned to each day off type has to be even. This is the case because drivers alternate between "early" and "late" weeks. During the early week, their shifts typically start before 10am, while in the late week, their shifts begin after 10am. Due to this alternating schedule, an even number of roster weeks is necessary for each day off type.

Time frame rosters are anonymous, i.e., the specific assignment of drivers to roster weeks is not given, putting the focus solely on constructing the roster itself. To comprehend the rotational principle of the roster, it is crucial to define what constitutes two consecutive positions in the roster.

4.2.2 Definition of Consecutive Roster Positions

Given the roster positions

$$r_{o_i^k,m},\ r_{o_i^l,n} \in R$$

then $r_{o_i^k,m}$ is immediately followed by $r_{o_i^l,n}$ iff one of the following statement holds:

- Two consecutive positions (i.e., days) in the same roster week: $k = l,\ n = m + 1$.
- Sunday in one week (but not the last week of a day off type) followed by Monday in the next week: $l = k + 1,\ m = 6,\ n = 0$.
- Sunday in the last week of a day off type followed by Monday in the first week of the same day off type: $k = |o_i|,\ l = 1,\ m = 6,\ n = 0$.

4.3 Algorithm for Calculating the Minimum Number of Drivers Needed

I propose the *min_demand* algorithm to calculate the required number of drivers for a certain number of shifts given the drivers' absence probability. The algorithm starts with a lower bound (e.g., number of shifts) and increases the demand until the lower bound is covered with a probability of p_{suc}, i.e., until the binomial cumulative distribution function returns a probability greater than p_{suc}.

This is the binomial cumulative distribution function used in the *min_demand* algorithm:

$$binom.cdf(k, n, p) = P(X \leq k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

Algorithm 4.1: $min\_demand(p_{abs}, p_{suc}, lb)$

```
if lb = 0 then
    minDemand ← 0;
else
    minDemand ← lb;
    while binom.cdf(minDemand − lb, minDemand, p_abs) ≤ p_suc do
        minDemand ← minDemand + 1;
    end
end
return minDemand
```

Given a number of n trials, the probability p of a trial being successful and a number of k successes, the binomial cumulative distribution function returns the probability that there are k or fewer successes.

The *min_demand* algorithm will be used in several constraints, so I want to clarify its meaning by providing an example. Assume that drivers are absent with a probability of 10% ($p_{abs} = 0.1$). Let's also assume that we have 10 shifts and aim to cover 9 of them ($cov_{fac} = 0.9,\ lb = 9$) with a probability of 99% ($p_{suc} = 0.99$). The question is then: How many drivers do we need to ensure that at least 9 drivers show up to work 99% of the time?

To determine this number, I use the *min_demand* algorithm ($min\_demand(0.1, 0.99, 9)$). In the first iteration, the function *binom.cdf*(0,9,0.1) is called and returns the likelihood that from 9 drivers, 0 or fewer are absent given the absence probability of 10%. This value is 0.38742, which is smaller than the required 0.99, so the while loop continues. The loop stops with the call *binom.cdf*(4,13,0.1), where the binomial cumulative distribution function returns 0.99354. This means that in 99.354% of the days, no more than 4 out of 13 drivers are absent. Thus, we determine that the required number of drivers to cover the shifts is 13.

One might ask: Aren't there 10 shifts to be covered, not 9? That is correct, but I aim to cover only a certain percentage of shifts, as the remaining ones can be handled by drivers working overtime if necessary. If all shifts were covered with a probability of 99%, there would quite often be too many drivers on stand-by. The coverage factor provides the ability to regulate the extent to which you want to have more drivers working overtime (lower coverage, risk of being understaffed) or more drivers on stand-by (higher coverage, risk of being overstaffed).

4.4 Creating a Day Off Schedule

Before assigning time frames to roster positions, first, a a day off schedule is created to determine the size of the roster and the placement of the days off. I assume that

drivers may be absent with a binomially distributed probability p_{abs}. Based on real-world observations, the demand for drivers dem_i for each weekday i is determined by calculating the minimum even number of drivers such that the number of drivers showing up is at least $(cov_{fac} \cdot 100)\%$ of the number of shifts $|S_i|$ with a probability of p_{suc}:

$$dem_i = \left\lceil \frac{min\_demand(p_{abs},\ p_{suc},\ \lceil |S_i| \cdot cov_{fac} \rceil)}{2} \right\rceil \cdot 2$$

Every driver works 5 days per week, but the sum $dem = \sum_{i=0}^{6} dem_i$ is not necessarily a multiple of 5. Hence, I need to slightly adjust the demand:

$$frac = \frac{\sum_{i=0}^{6} d_i}{5} - \left\lfloor \frac{\sum_{i=0}^{6} d_i}{5} \right\rfloor$$

If $frac = 0.2$ then $dem_5 \mathrel{+}= 2,\ dem_6 \mathrel{+}= 2,$
else if $frac = 0.4$ then $dem_6 \mathrel{-}= 2,$
else if $frac = 0.8$ then $dem_6 \mathrel{+}= 2,$
else if $frac = 0.8$ then $dem_5 \mathrel{-}= 2,\ dem_6 \mathrel{-}= 2$

I decided to use only Saturdays and Sundays for the adjustment as they have separate shift sets making sure that days with the same number of shifts have the same required number of drivers. The number of roster positions of roster R is determined by summing up the demands and dividing by 5:

$$|R| = \frac{\sum_{i=0}^{6} dem_i}{5}$$

The day off demand do_i for weekday i is then the size of the roster minus the driver demand dem_i:

$$do_i = |R| - dem_i, \quad i \in \{0, ..., 6\}$$

The days off have to be two consecutive days. Furthermore, the size of a day off type must be even and I limit the size of type o_i to $|o_i| \leq 2|o_{i+1}|$ and $2|o_i| \geq |o_{i+1}|$ for $i \in \{0, ..., 5\}$. This is done to ensure a rather even distribution of day off types. In order to find the optimal day off schedule, I formulate the following model in MiniZinc[1] [NSB+07]:

Variables:

$$|o_i| \text{ ... size of day off type } i \in \{0, ..., 6\}$$

$$x_i = \begin{cases} |o_i| + |o_{i+1}|, & \text{if } i < 6 \\ |o_0| + |o_6|, & \text{if } i = 6 \end{cases}$$

[1] https://github.com/lukasfruehwirth/time_frame_rostering/blob/main/day_off_type_size.mzn

Objective function:

$$\begin{aligned}
\min \quad & \sum_{i=0}^{6} |do_i - x_i| \\
\text{s.t.} \quad & \sum_{i=0}^{6} |o_i| = |R| \\
& |o_i| \mod 2 = 0 && i \in \{0, ..., 6\} \\
& |o_i| \leq 2|o_{i+1}| && i \in \{0, ..., 5\} \\
& 2|o_i| \geq |o_{i+1}| && i \in \{0, ..., 5\}
\end{aligned}$$

The decision variable is the size of each day off type, denoted as $|o_i|$. Since drivers have two consecutive days off, I introduce an auxiliary variable x_i, which represents the number of roster positions for weekday i designated as days off. This is achieved by summing up the sizes of two consecutive day off types. For example, day off types o_0 and o_1 include a day off on Monday, thus $x_0 = |o_0| + |o_1|$. The objective is to find the sizes of day off types that minimize the absolute difference between the pre-calculated demand for days off do_i and the days off represented by the variable x_i subject to the constraints specified in the model above.

After finding the optimal size for each day off type o_0 to o_6, day off type o_7 is introduced, which has a 16-week-long rotating day off schedule. The structure of this schedule is provided by a public transportation company. Based on real-world observations, the size of day off type o_7 is calculated by using a fixed share of 30%:

$$|o_7| = \left\lfloor \frac{|R| \cdot 0.3}{16} \right\rfloor \cdot 16$$

In the model above, the constraint is set to $\sum_{i=0}^{6} |o_i| = |R|$. However, since I am now introducing another day off type, I reduce the sizes of day off types o_0 to o_6 accordingly such that $\sum_{i=0}^{7} |o_i| = |R|$ holds and such that the number of roster positions for weekday i designated as days off remains unaffected:

$$|o_i| = \begin{cases} |o_i| - \frac{|o_7|}{8}, & \text{if } i < 6 \\ |o_i| - \frac{|o_7|}{4}, & \text{if } i = 6 \end{cases}$$

Based on the day off schedule, we can proceed to construct and encode the empty roster R. As an example, Table 4.1 presents the roster for the instance 4 containing only the days off (encoded as -1 and 0) and the workweek type encoding (20 to 30). The first column indicates the total roster week, the second column specifies the day off type, and the third column denotes the roster week within that day off type. Columns 4 to 10 represent the weekdays. Given this roster R, the goal now is to assign time frames to the empty positions (encoded with numbers 20 to 30) in R such that the hard constraints are not violated and the objective function value is minimized. The subsequent three sections define the hard and soft constrains, as well as the objective function.

$\|R\|$	o	$\|o\|$	R_0	R_1	R_2	R_3	R_4	R_5	R_6
1	0	1	0	30	29	28	27	26	-1
2	0	2	0	25	24	23	22	21	-1
3	0	3	0	30	29	28	27	26	-1
4	0	4	0	25	24	23	22	21	-1
5	0	5	0	30	29	28	27	26	-1
6	0	6	0	25	24	23	22	21	-1
7	0	7	0	30	29	28	27	26	-1
8	0	8	0	25	24	23	22	21	-1
9	0	9	0	30	29	28	27	26	-1
10	0	10	0	25	24	23	22	21	-1
11	0	11	0	30	29	28	27	26	-1
12	0	12	0	25	24	23	22	21	-1
13	0	13	0	30	29	28	27	26	-1
14	0	14	0	25	24	23	22	21	-1
15	0	15	0	30	29	28	27	26	-1
16	0	16	0	25	24	23	22	21	-1
17	0	17	0	30	29	28	27	26	-1
18	0	18	0	25	24	23	22	21	-1
19	0	19	0	30	29	28	27	26	-1
20	0	20	0	25	24	23	22	21	-1
21	1	1	-1	0	30	29	28	27	26
22	1	2	-1	0	25	24	23	22	21
23	1	3	-1	0	30	29	28	27	26
24	1	4	-1	0	25	24	23	22	21
25	1	5	-1	0	30	29	28	27	26
26	1	6	-1	0	25	24	23	22	21
27	1	7	-1	0	30	29	28	27	26
28	1	8	-1	0	25	24	23	22	21
29	1	9	-1	0	30	29	28	27	26
30	1	10	-1	0	25	24	23	22	21
31	1	11	-1	0	30	29	28	27	26
32	1	12	-1	0	25	24	23	22	21
33	1	13	-1	0	30	29	28	27	26
34	1	14	-1	0	25	24	23	22	21
35	1	15	-1	0	30	29	28	27	26
36	1	16	-1	0	25	24	23	22	21
37	1	17	-1	0	30	29	28	27	26
38	1	18	-1	0	25	24	23	22	21
39	1	19	-1	0	30	29	28	27	26
40	1	20	-1	0	25	24	23	22	21
41	2	1	21	-1	0	30	29	28	27
...	2	...	...	...	...	...	...	...	...
60	2	20	26	-1	0	25	24	23	22
61	3	1	22	21	-1	0	30	29	28
...	3	...	...	...	...	...	...	...	...
80	3	20	27	26	-1	0	25	24	23
81	4	1	23	22	21	-1	0	30	29
...	4	...	...	...	...	...	...	...	...
100	4	20	28	27	26	-1	0	25	24
101	5	1	24	23	22	21	-1	0	30
...	5	...	...	...	...	...	...	...	...
120	5	20	29	28	27	26	-1	0	25
121	6	1	25	24	23	22	21	-1	0
...	6	...	...	...	...	...	...	...	...
222	6	102	30	29	28	27	26	-1	0
223	7	1	0	30	29	28	27	26	-1
224	7	2	0	25	24	23	22	21	-1
225	7	3	-1	0	30	29	28	27	26
226	7	4	-1	0	25	24	23	22	21
227	7	5	20	-1	0	30	29	28	27
228	7	6	26	-1	0	25	24	23	22
229	7	7	21	20	-1	0	30	29	28
230	7	8	27	26	-1	0	25	24	23
231	7	9	22	21	20	-1	0	30	29
232	7	10	28	27	26	-1	0	25	24
233	7	11	23	22	21	20	-1	0	30
234	7	12	29	28	27	26	-1	0	25
235	7	13	24	23	22	21	20	-1	0
236	7	14	30	29	28	27	26	-1	0
237	7	15	25	24	23	22	21	-1	0
238	7	16	30	29	28	27	26	-1	0
239	7	17	0	30	29	28	27	26	-1
240	7	18	0	25	24	23	22	21	-1
241	7	19	-1	0	30	29	28	27	26
242	7	20	-1	0	25	24	23	22	21
243	7	21	20	-1	0	30	29	28	27
244	7	22	26	-1	0	25	24	23	22
245	7	23	21	20	-1	0	30	29	28
246	7	24	27	26	-1	0	25	24	23
247	7	25	22	21	20	-1	0	30	29
248	7	26	28	27	26	-1	0	25	24
249	7	27	23	22	21	20	-1	0	30
250	7	28	29	28	27	26	-1	0	25
251	7	29	24	23	22	21	20	-1	0
252	7	30	30	29	28	27	26	-1	0
253	7	31	25	24	23	22	21	-1	0
...	7	...	...	...	...	...	...	...	...
302	7	80	30	29	28	27	26	-1	0

Table 4.1: Roster for Instance 4 containing only the Days Off and the Workweek Type Encoding

4.5 Hard Constraints

Given an assigned time frame roster R and the probability p_{abs} that drivers are absent, it must be possible (with a probability of success of at least ${p_{suc}}^2$) to assign every present driver a shift such that the shift fits within the interval provided by the assigned time frame and is of the correct type. There are two independent occasions to violate this abstract constraint: First, I only ensure with a probability of $P(A) = p_{suc}$ that there remain enough time frames f of type $t_f > 0$ to accommodate all split shifts and secondly, I guarantee with probability of $P(B) = p_{suc}$ that there remain enough time frames f of type $t_f \neq 1$ to accommodate the regular shifts, hence the overall probability of success is $P(A \cap B) = {p_{suc}}^2$. To check whether this abstract constraint holds, I would need to simulate the shift assignment to time frames which itself is a NP-hard problem [Lau96]. Hence, I break down the abstract constraint into hard constraints $h_1 - h_3$ and soft constraint s_1. Additionally, I verify for each solution (i.e., for each time frame roster) by simulating absences and subsequent shift assignment whether constraints $h_1 - h_3$ and s_1 were successful in ensuring that the aforementioned constraint holds (for details see Section 5.4).

4.5.1 Minimum Coverage

First, I define a coverage: Each weekday is split into time intervals $[\tau_j, \tau_{j+1})$ of length $\tau_{j+1} - \tau_j = 0.5$ (30 min), starting with $\tau_1 = 3$ (i.e., 3am on the current day) and ending with $\tau_{55} = 30$ (i.e., 6am on the next day). The shift coverage of a time interval $[\tau_j, \tau_{j+1})$ for weekday i is defined as the number of shifts $s \in S_i$ that start before τ_{j+1}, end after τ_j and are either split shifts or not:

$$min\_cov_{i,j} = \sum_{s \in S_i,\ t_s=0,\ a_s<\tau_{j+1},\ b_s>\tau_j} 1$$

$$min\_cov\_split_{i,j} = \sum_{s \in S_i,\ t_s=1,\ a_s<\tau_{j+1},\ b_s>\tau_j} 1$$

The frame coverage is defined analogously:

$$fr\_cov_{i,j} = \sum_{f \in R_i,\ t_f=0,\ c_f<\tau_{j+1},\ d_f>\tau_j} 1$$

$$fr\_cov\_split_{i,j} = \sum_{f \in R_i,\ t_f>0,\ c_f<\tau_{j+1},\ d_f>\tau_j} 1$$

The frame coverage must be greater or equal the shift coverage:

$$h_{1_a}: \ fr\_cov_{i,j} \geq min\_cov_{i,j} \qquad i \in \{0, ..., 6\},\ j \in \{1, ..., 54\}$$

$$h_{1_b}: \ fr\_cov\_split_{i,j} \geq min\_cov\_split_{i,j} \qquad i \in \{0, ..., 6\},\ j \in \{1, ..., 54\}$$

4.5.2 Minimum Frame Set

The hard constraint h_1 is not sufficient by itself to ensure that shifts fit into the available time frames. I need to guarantee that there are enough frames to accommodate the shifts, despite drivers may be absent with a probability of p_{abs}. A shift can lie in the interval of several time frames, hence I obtain a set of time frames FS_s covering a shift s. I determine the minimum size required for each of these sets of time frames, similar to how I determine the necessary count of drivers and days off:

$$FS_s = \{f \mid f \in F,\ t_f \neq 1,\ s \in S,\ [a_s, b_s] \in [c_f, d_f]\}$$
$$FS_{split_s} = \{f \mid f \in F,\ t_f = 1,\ s \in S,\ [a_s, b_s] \in [c_f, d_f]\}$$

I also compute frame sets FS_j for artificial shifts of length 4 (assumed minimum shift length) to get every possible frame set, resulting in 30 different frame sets. Subsequently, I count the shifts having the same frame set FS_j for weekday i:

$$count_{FS_{i,j}} = |\{s \mid s \in S_i,\ FS_j = FS_s\}|$$
$$count_{FS_{split_{i,j}}} = |\{s \mid s \in S_i,\ FS_{split_j} = FS_{split_s}\}|$$

The size of a frame set FS_j for weekday i is defined as the sum of the count of frame f appearing in roster column R_i over all frames $f \in FS_j$:

$$|FS_{i,j}| = \sum_{f \in FS_j} \sum_{f \in R_i} 1 \qquad |FS_{split_{i,j}}| = \sum_{f \in FS_{split_j}} \sum_{f \in R_i} 1$$

The minimum size of a frame set FS_j for weekday i is then the number of frames $|FS_{i,j}|$ necessary to cover at least $(cov_{fac} \cdot 100)\%$ of shifts possessing the frame set FS_j or subsets thereof with a probability of p_{suc}:

$$min_{FS_{i,j}} = min\_demand(p_{abs},\ p_{suc},\ \sum_{FS_k \subseteq FS_j} \left\lceil cov_{fac} \cdot count_{FS_{i,k}} \right\rceil)$$

The size of a frame set must be greater or equal the minimum size for this set:

$$h_{2_a}:\ |FS_{i,j}| \geq min_{FS_{i,j}} \qquad i \in \{0, ..., 6\},\ j \in \{1, ..., 30\}$$

Split shifts are treated differently since they contain a long break, which increases the overall shift duration. This makes it more important for drivers to known if their associated time frames allow or even require a split shift. Therefore, the number of time frames in the roster that require split shifts to be assigned equals 80% of the total split shifts (see h_{2_b}, h_{2_c}). The utilization of 80% aims to accommodate small changes in the shift plan without making the rosters unsatisfiable. The minimum size of a split frame set FS_{split_j} is defined similarly to a regular frame set, except that it must cover 100% of split shifts (see h_{2_d}, h_{2_e}).

There are frames that can accommodate regular as well as split shifts, hence I also calculate the minimum demand for all early week time frames including the once designated for split shifts (see h_{2_f}). Furthermore, time frames of type greater 0 are only allowed if the set of shifts contains split shifts (see h_{2_g}).

$$
\begin{aligned}
h_{2_b}:\ & |FS_{split_{i,j}}| = |\{s \mid s \in S_i,\ t_s = 1\}| \cdot 0.8 \\
& i \in \{0,...,6\},\ FS_{split_j} = \{f \mid f \in F,\ t_f = 1\} \\
h_{2_c}:\ & |FS_{split_{i,j}}| \geq count_{FS_{split_{i,j}}} \cdot 0.8 \\
& i \in \{0,...,6\},\ |FS_{split_j}| = 1 \\
h_{2_d}:\ & |FS_{split_{i,j}}| \geq min\_demand(p_{abs},\ p_{suc},\ |\{s \mid s \in S_i,\ t_s = 1\}|) \\
& i \in \{0,...,6\},\ FS_{split_j} = \{f \mid f \in F,\ t_f > 0\} \\
h_{2_e}:\ & |FS_{split_{i,j}}| \geq min\_demand(p_{abs},\ p_{suc},\ count_{FS_{split_{i,j}}}) \\
& i \in \{0,...,6\},\ FS_{split_j} = \{\{12,14\},\{13,15\}\} \\
h_{2_f}:\ & |FS_{i,j}| + |FS_{split_{i,h}}| \geq \\
& min\_demand(p_{abs},\ p_{suc},\ |\{s \mid s \in S_i,\ t_s = 1\}| + \sum_{FS_k \subseteq FS_j} \left\lceil cov_{fac} \cdot count_{FS_{i,k}} \right\rceil) \\
& i \in \{0,...,6\},\ FS_j = \{1,2,3,4,11,14,15\} \\
h_{2_g}:\ & |\{s \mid s \in S_i,\ t_s = 1\}| = 0 \implies |\{f \mid f \in R_i,\ t_f > 0\}| = 0 \\
& i \in \{0,...,6\}
\end{aligned}
$$

For some instances, enforcing hard constraint h_{2_d} leads to unsatisfiability. The reason is that time frames of type $t_f > 0$ can only be assigned to the first and second workday of the early week (workweek types 24 and 25) and to the last workday of the late week (workweek type 26). There are two possibilities to deal with this issue. One is to just replace the hard constraint by a soft constraint like it is done for hard constraint h_{2_f} (see Section 4.6.3). The other possibility (not possible for h_{2_f}) is to calculate the maximum of $|FS_{split_{i,j}}|$ in constraint h_{2_d}. This can be done by simply counting the number of roster positions of weekday i associated with workweek types 24 to 26. I denote this maximum as max_{split_i}. If now

$$min\_demand(p_{abs},\ p_{suc},\ |\{s \mid s \in S_i,\ t_s = 1\}|) > max_{split_i}$$

I replace constraint h_{2_d} for weekday i with

$$h_{2_d}:\ |FS_{split_{i,j}}| \geq max_{split_i} \quad i \in \{0,...,6\},\ FS_{split_j} = \{f \mid f \in F,\ t_f > 0\}$$

In Section 5.3.3, I compare the above mentioned possibilities.

4.5.3 Maximum Frame Set

In addition to the hard constraints h_1 and h_2, I define maximum frame sets. Instead of counting shifts with the same frame set, I count how many shifts can be covered using any frame f in a frame set FS_j. This gives us the maximum number of shifts coverable by a frame of frame set FS_j:

$$max\_count_{FS_{i,j}} = |\{s \mid s \in S_i,\ \exists f \in FS_j :\ [a_s, b_s] \in [c_f, d_f]\}|$$

I once again employ the *min_demand* algorithm but this time in reverse. The maximum size of a frame set FS_j for weekday i is determined by the maximum number of frames $|FS_{i,j}|$ such that there is only a probability of $1 - p_{suc}$ for there to be more frames than shifts to cover:

$$max_{FS_{i,j}} = min\_demand(p_{abs},\ 1 - p_{suc},\ max\_count_{FS_{i,j}} + 1) - 1$$

The size of a frame set must be smaller or equal the maximum size for this set. However, this is restricted to frame sets of size less than 4, as larger sets render rosters infeasible. For frame sets of size greater than 3, I adjust the maximum to match the number of shifts $|S_i|$ for weekday i. In case the maximum is lower than the minimum, I decrease the minimum such that it equals the maximum:

$$h_{3_a} :\ |FS_{i,j}| \leq max_{FS_{i,j}} \qquad i \in \{0, ..., 6\},\ |FS_j| < 4$$
$$h_{3_b} :\ |FS_{i,j}| \leq |S_i| \qquad i \in \{0, ..., 6\},\ |FS_j| \geq 4$$

4.5.4 Forbidden Sequences

Some sequences of time frames are forbidden, primarily due to rest time regulations. A consecutive time frame assignment (f_1, f_2) in roster R cannot appear in the list of forbidden sequences seq_{fb}:

$$h_4 :\ \forall (f_1, f_2) \in R :\ (f_1, f_2) \notin seq_{fb} \qquad f_1, f_2 \in F$$

4.5.5 Forbidden Frames

Drivers follow an alternating scheme where in one week they are only assigned early shifts, and in the subsequent week, only late shifts. Split shifts can only be assigned during the first two days of the early week or the last day of the late week. A workweek type is utilized to encode this information for each roster positions. A time frame assignment f to a roster position of workweek type wt cannot appear in the list of forbidden time frames for this workweek type:

$$h_5 :\ \forall f \in R :\ f \notin forb_{wt} \qquad f \in F,\ f = r_{o_i^j, k},\ wt = wt_{r_{o_i^j, k}}$$

Algorithm 4.2: *max_deviation*$(min\_cov, dev\_allowed)$

```
for i ∈ {0, ..., 5} do
    x ← Σ_{j=0}^{29} min_cov_{i,j};
    y ← Σ_{j=0}^{29} min_cov_{i+1,j};
    if x < y then
        dev ← y/x;
    else
        dev ← x/y;
    end
    maxDev[i] ← dev_allowed + ⌈(dev − 1) · 500⌉
end
return maxDev
```

4.5.6 Maximum Frame Deviation

Weekdays with similar shifts should also have similar time frames. To ensure this, I define a maximum deviation in the time frame counts between two weekdays. The *max_deviation* algorithm (see Algorithm 4.2) uses the minimum coverage definition (see h_1). To measure the similarity of shifts between two days, I sum the shift coverage of these days and divide the greater coverage by the smaller one. The shift coverage between two days is identical if *dev* equals 1. I allow a slight variation in the time frame counts between days with identical shifts, using the parameter *dev_allowed*. For example, if the shift coverage of one day is 5% greater than that of the next day, the maximum deviation is calculated as $maxDev = dev\_allowed + 25$.

For weekdays 0 to 5 the absolute difference in the count of frame f between two consecutive days $(i, i+1)$ must be smaller or equal $maxDev[i]$ (see h_{6_a}). For weekdays 0 to 4 the absolute difference in the count of frame f between any of these days must be smaller or equal $maxDev[i]$ (see h_{6_b-d}).

$$h_{6_a}: \left|\sum_{f\in R_i} 1 - \sum_{f\in R_{i+1}} 1\right| \leq maxDev[i] \qquad i \in \{0,...,5\},\ f \in F$$

$$h_{6_b}: \left|\sum_{f\in R_i} 1 - \sum_{f\in R_{i+2}} 1\right| \leq maxDev[i] \qquad i \in \{0,1,2\},\ f \in F$$

$$h_{6_c}: \left|\sum_{f\in R_i} 1 - \sum_{f\in R_{i+3}} 1\right| \leq maxDev[i] \qquad i \in \{0,1\},\ f \in F$$

$$h_{6_d}: \left|\sum_{f\in R_i} 1 - \sum_{f\in R_{i+4}} 1\right| \leq maxDev[i] \qquad i = 0,\ f \in F$$

4.5.7 Night Frames

Time frames with an end time after 2am are called night frames and are only allowed if there are also shifts ending after 2am. Furthermore, for frame sets consisting of only night frames, the maximum frame set size is set to the minimum frame set size:

$$h_{7_a}: \ |\{s \mid s \in S_i, \ b_s > 26\}| = 0 \implies |\{f \mid f \in R_i, \ d_f > 26\}| = 0$$
$$h_{7_b}: \ max_{FS_{i,j}} := min_{FS_{i,j}} \quad i \in \{0, ..., 6\}, \ FS_j = \{f \mid f \in F, \ d_f > 26\}$$

4.5.8 Relief Frames

Two of the time frames $FS_r = \{11, 15\}$ are considered to be relief frames, since they can accommodate early and late shifts. These time frames are necessary to balance potential imbalances between early time frames $FS_e = \{1, 2, 3, 4, 11, 14, 15\}$ and late time frames $FS_l = \{5, 6, 7, 8, 9, 11, 15\}$ and thus, appear in both sets $FS_r = FS_e \cap FS_l$. I determine the minimum size of frame set FS_r by calculating the maximum difference between early and late frames given that drivers are absent with probability p_{abs}:

$$lb_i = \arg\min_x(\sqrt{1 - p_{suc}} \leq binom.cdf(x, \ min(|FS_{i,e}|, |FS_{i,l}|), \ 1 - p_{abs}) - 1$$
$$ub_i = \arg\min_x(1 - \sqrt{1 - p_{suc}} \leq binom.cdf(x, \ max(|FS_{i,e}|, |FS_{i,l}|), \ 1 - p_{abs})$$
$$h_8: \ min_{FS_{i,r}} = ub_i - lb_i \quad i \in \{0, ..., 6\}$$

4.6 Soft Constraints

4.6.1 Deviation from Desired Coverage

In addition to the hard constraints $h_1 - h_3$, I introduce a soft constraint that penalizes the deviation from the desired coverage. The aim is to satisfy the abstract constraint mentioned at the beginning of Section 4.5. I calculate the desired coverage by again using the *min_demand* algorithm:

$$cov\_des_{i,j} = min\_demand(p_{abs}, \ p_{suc}, \ min\_cov_{i,j})$$
$$cov\_split\_des_{i,j} = min\_demand(p_{abs}, \ p_{suc}, \ min\_cov\_split_{i,j})$$

It is more important to avoid undercoverage than overcoverage, therefore, to calculate the deviation penalty, I square the difference between frame coverage and desired coverage if the frame coverage is greater, and take the difference to the power of 4 otherwise. The difference between split coverage and desired split coverage is simply squared:

$$cov\_pen_{i,j} = \begin{cases} (cov\_des_{i,j} - fr\_cov_{i,j})^4 & \text{if } cov\_des_{i,j} \leq fr\_cov_{i,j} \\ (cov\_des_{i,j} - fr\_cov_{i,j})^2 & \text{if } cov\_des_{i,j} > fr\_cov_{i,j} \end{cases}$$
$$cov\_split\_pen_{i,j} = (cov\_split\_des_{i,j} - fr\_cov\_split_{i,j})^2$$
$$s_1 = \sum_{i=0}^{6} \sum_{j=1}^{55} cov\_pen_{i,j} + cov\_split\_pen_{i,j}$$

4.6.2 Undesirable Sequences

Some sequences of time frames are undesirable, primarily due to rest time regulations. A consecutive time frame assignment (f_1, f_2) in roster R appearing in the list of undesirable sequences seq_{ud} incurs a penalty $p(f_1, f_2)$ (penalty function p is given):

$$s_2 = \sum_{(f_1,f_2)\in R,\ (f_1,f_2)\in seq_{ud}} p(f_1, f_2)$$

4.6.3 Below Minimum Frame

The hard constraint h_{2_f} sometimes results in unsatisfiability. To prevent this outcome, I transform this hard constraint into a soft constraint and introduce a high penalty if the frame set count falls below the minimum. The function $below(h)$ returns the extent to which a hard constraint h is undershot:

$$s_4 = below(h_{2_f}) \cdot pen_{hard\_con}$$

4.6.4 Frame Deviation

Weekdays with similar shifts should also have similar time frames. This is in particular the case for weekdays 0 to 4, since they usually have very similar sets of shifts. I penalize the frame deviation between these weekdays:

$$s_3 = \sum_{i=1}^{4} \sum_{f=1}^{15} |\sum_{f\in R_i} 1 - \sum_{f\in R_{i+1}} 1|$$

4.6.5 Same Frame Next Day

To provide drivers with some variety in the sequence of time frames, I introduce a penalty for consecutive assignments of the same time frame (f, f) in roster R:

$$s_5 = \sum_{(f,f)\in R,\ f\in F} 1$$

4.6.6 Same Frames Next Week

To achieve a more even distribution of time frames within a day off type, I introduce a penalty for assigning the same time frames to two consecutive early or late weeks. The penalty s_6 is similar to s_5, as simply all occurrences are counted.

4.7 Decision Variables

The roster positions represent the decision variables. For each roster position $r_{o_i^j,k}$ in roster R that is not assigned a day off, a time frame $r_{o_i^j,k} := f \in F$ is assigned.

4.8 Objective Function

The objective of this problem is to assign time frames to roster positions such that the costs associated with soft constraints s_1 to s_6 are minimized while ensuring that hard constraints h_1 to h_8 hold. To balance the costs, each soft constraint is multiplied by an adjustable weight before aggregation, resulting in the following function:

$$\begin{aligned} \min \quad & w_1 s_1 + w_2 s_2 + w_3 s_3 + w_4 s_4 + w_5 s_5 + w_6 s_6 \qquad w_1, w_2, w_3, w_4, w_5, w_6 \in \mathbb{N}^+ \\ \text{s.t.} \quad & h_1 - h_8 \text{ hold} \end{aligned}$$

4.9 Solver-independent Model

Based on the problem definition provided in this section, I create a solver-independent model of the time frame rostering problem by using MiniZinc [NSB+07]. A MiniZinc model allows one to use both linear and constraint solvers. The model[2] includes all hard and soft constraints as well as the objective function outlined in the problem definition. Where possible, I use global constraints and provide MiniZinc with lower and upper bounds to variables and functions.

[2] https://github.com/lukasfruehwirth/time_frame_rostering/blob/main/frame_rostering_problem.mzn

CHAPTER 5

Experimental Evaluation

The proposed model is evaluated using a diverse set of twelve real-world instances. These twelve distinct sets of shifts are extracted from real-world shift plans, that have been provided by a public transportation company. I also make this data available to the scientific community[1]. The evaluation process involves several steps: Firstly, I present the experimental setup, instance properties as well as parameter settings. Secondly, I solve the instances using two different state-of-the-art solvers and compare the results. Thirdly, I validate the solutions using a simulation model. Finally, I discuss the obtained results in detail.

5.1 Experimental Setup

To evaluate the solver-independent model, I solve the instances using two conceptually different state-of-the-art solvers: mixed-integer programming solver Gurobi 11.0.0 [Gur23] and constraint solver OR-Tools CP-SAT 9.8.3296 [PD23]. Additionally, I tested the lazy clause generation solver Chuffed 0.13.1 [CDLBS10]. Despite extending the time limit to 10 hours, Chuffed was unable to solve any of the instances. Consequently, I excluded the Chuffed solver from the experiments.

The test setup looks as follows: The solvers are allowed to use up to 20 threads. The experiments are run on an Intel i5-13500 2.5GHz processor with 20 logical units and with 32GB of RAM available. Calculations for the day off schedule as well as for the hard constraints $(h_1, h_2, h_3, h_6, h_7, h_8)$ and the soft constraints (s_1, s_4) are done in Python 3.12 [VRD09]. The MiniZinc model and solvers are then called within Python using the MiniZinc Python 0.9.0 interface [mzp].

[1] https://github.com/lukasfruehwirth/time_frame_rostering

5.2 Instance Properties, Time Frames and Parameter Settings

Table 5.1 illustrates the main properties of the twelve instances, where the column $|Sp_i|$ stands for the number of split shifts in weekday i:

Inst.	$\|R\|$	$\|S\|$	$\|S_0\|$ - $\|S_3\|$	$\|S_4\|$	$\|S_5\|$	$\|S_6\|$	$\|Sp_0\|$ - $\|Sp_4\|$	$\|Sp_5\|$	$\|Sp_6\|$
1	364	1,380	223	223	142	123	29	4	0
2	354	1,344	217	219	132	125	22	0	0
3	308	1,159	185	185	123	111	27	6	0
4	302	1,134	183	183	110	109	21	3	0
5	338	1,282	204	204	139	123	23	9	0
6	324	1,224	194	194	134	120	15	0	0
7	382	1,456	233	233	153	138	37	7	0
8	362	1,375	218	218	148	137	23	0	0
9	652	2,539	408	408	265	234	56	10	0
10	636	2,478	400	402	242	234	43	3	0
11	702	2,738	437	437	292	261	60	16	0
12	666	2,599	412	412	282	257	38	0	0

Table 5.1: Instance Properties

Table 5.2 lists all time frames including their interval length and type:

Time Frame	Interval Length (hh:mm)	Type
1	10:30	0
2	11:00	0
3	12:30	0
4	12:30	0
5	11:00	0
6	12:00	0
7	12:30	0
8	11:00	0
9	7:30	0
10	15:00	0
11	16:00	0
12	16:30	1
13	17:00	1
14	12:30 / 16:30	2
15	16:00 / 17:00	2

Table 5.2: Time Frame Properties

Time frames 14 and 15 can accommodate regular and split shifts and are, therefore, a combination of other time frames. This is the reason why there are two different interval lengths given for these time frames in Table 5.2. If time frame 14 is assigned a regular shift, then its properties are equal to those of time frame 4. If time frame 14 is assigned a split shift, then its properties are equal to those of time frame 12. Similarly, if time frame 15 is assigned a regular shift, then its properties are equal to those of time frame 11. Conversely, if time frame 15 is assigned a split shift, then its properties are equal to those of time frame 13.

For all instances, I use the following parameter settings:

Parameter	Setting
cov_{fac}	0.90
p_{abs}	0.25
p_{suc}	0.99
$dev\_allowed$	5
pen_{hard_con}	1,000,000
w_1	1
w_2	1
w_3	10,000
w_4	1
w_5	100
w_6	100

Table 5.3: Parameter Settings

5.3 Experimental Results

I define the gap g between the objective value ov of a solution and the best known lower bound lb_{best} as:

$$g = (\frac{|lb_{best} - ov|}{ov}) \cdot 100$$

In the following section, I present and compare the results of various search strategies for variable selection and domain-value choice, the impact of redundant constraints and symmetry breaking, and the performance of the two solvers Gurobi and OR-Tools.

5.3.1 Variable Selection and Domain-Value Choice

I tested the following variable selection and domain-value choice strategies for OR-Tools CP-SAT solver with a time limit of one hour, without redundant constraints or symmetry breaking. Gurobi was not included in the test setup, as it does not allow the specification of these strategies:

Variable selection strategies[2]:

- *first_fail*: Select variables with the smallest domain size.
- *input_order*: Select variables based on the input order.
- *smallest*: Select variables with the smallest domain-value.
- *largest*: Select variables with the largest domain-value.

Domain-value choice strategies[3]:

- *indomain_min*: Choose smallest domain-value.
- *indomain_max*: Choose largest domain-value.
- *indomain_split*: Split the domain and choose the lower half.
- *indomain_reverse_split*: Split the domain and choose the upper half.

Table 5.4 presents the results for all combinations of search strategies using the OR-Tools CP-SAT solver. OR-Tools was able to find solutions to all instances for 8 out of the 16 search strategies. The variable selection strategy *input_order* in combination with the domain-value choice strategy *indomain_max* performed the best. However, when examining the optimality gap, the variation seems quite small.

[2] https://www.minizinc.org/doc-2.8.4/en/lib-stdlib-annotations.html#search-annotations

[3] https://www.minizinc.org/doc-2.8.4/en/lib-stdlib-annotations.html#value-choice-annotations

Variable Selection	Domain-Value Choice	|Inst.|	Avg. Obj. Value $\overline{ov}$	Avg. Gap $\overline{g}$
input_order	indomain_max	12	58,800,330	96.69
largest	indomain_reverse_split	12	79,484,915	97.17
largest	indomain_min	12	74,455,278	97.31
smallest	indomain_split	12	74,513,008	97.72
largest	indomain_max	12	61,422,667	97.85
input_order	indomain_min	12	67,817,118	98.07
smallest	indomain_min	12	72,699,967	98.26
smallest	indomain_max	12	84,509,642	98.38
largest	indomain_split	11	56,521,735	96.91
input_order	indomain_reverse_split	11	61,671,054	97.32
smallest	indomain_reverse_split	11	64,759,774	97.59
input_order	indomain_split	11	55,669,136	97.62
first_fail	indomain_min	11	62,169,073	97.94
first_fail	indomain_reverse_split	10	63,806,620	97.00
first_fail	indomain_split	10	51,712,160	97.37
first_fail	indomain_max	9	64,129,189	97.57

Table 5.4: Results of Different Variable Selection and Domain-Value Choice Strategies for OR-Tools

5.3.2 Redundant Constraints and Symmetry Breaking

In order to improve the performance of the solvers, I experiment with redundant constraints and symmetry breaking.

Redundant Constraints

I include a redundant constraint for the forbidden sequence constraint (h_4) by exploiting the time frames numbering. Specifically, for time frames 1-3 and 7-10, one knows from the list of forbidden sequences that if any of these time frames are assigned to a roster position, the subsequent roster position must be assigned a time frame with a number smaller than or equal to the current one. This eliminates the need to check the list of forbidden sequences for these specific cases and allows for more explicit modeling.

The definition of the redundant constraint RC is based on the definition of two consecutive time frame assignment (f_1, f_2) in roster R:

$$RC: \ \forall (f_1, f_2) \in R: \ (0 < f_1 < 4 \vee 6 < f_1 < 11) \implies f_2 \leq f_1 \qquad f_1, f_2 \in F$$

Symmetry Breaking

To reduce the size of the search space, I aim to reduce the number of symmetrical solutions. My approach is to pre-assign time frames $f \in FS_j$ when there is only a single frame in the set ($|FS_j| = 1$). This is the case for the following time frames: 1, 2, 6, 7, 8, 10, and 12. According to hard constraints h_{2a} and h_{2c}, there must be a minimum number of these frames assigned to roster R for each weekday i: $min_{FS_{i,j}}$ and $count_{FS_{split_{i,j}}}$. Thus, I pre-assign these time frames while ensuring that they are uniformly distributed across the roster and assigned to positions where the influence to the overall solution quality is minimal. As a result, this approach is technically not considered symmetry breaking, as the optimal objective values with a pre-assigned time frame roster are slightly worse than those obtained by starting with an empty roster. However, I still use this approach, as the impact is minimal and typically not significant in practice, with an average increase of the objective value of only 1.5% for instances 1 to 7, as shown in Table 5.8.

The following example illustrates the pre-assignment process: Let the number of roster positions of workweek type 21 for weekday i be denoted as $p_{21,i}$, and the minimum number of time frame 1 that must be assigned to weekday i as $min_{1,i}$. To distribute the pre-assignments evenly, I calculate for weekday i: $x_{1,i} = \lfloor p_{21,i}/min_{1,i} \rfloor$. I then pre-assign time frame 1 to every $x^{th}_{1,i}$ roster position of workweek type 21 for weekday i. This method is applied similarly to time frames 2, 6, 7, 8, 10, and 12, and to workweek types 22, 28, 29, 30, 30, and 25 respectively. However, this symmetry breaking method can slightly influence the optimal objective value, as the time frames are pre-assigned to positions of specific workweek types, which might not always be optimal concerning the soft constraints.

Table 5.5 outlines a roster where the above-mentioned symmetry breaking method has been applied to. The pre-assigned time frames are represented by the numbers 1 to 15 (see Table 5.2).

To evaluate whether the redundant constraint (RC) and/or symmetry breaking (SB) enhance the solver's performance, I tested four different scenarios: using neither the redundant constraint nor symmetry breaking (-RC, -SB), using only symmetry breaking (-RC, +SB), using only the redundant constraint (+RC, -SB), and using both (+RC, +SB). I set the time limit to two hours.

$\|R\|$	o	$\|o\|$	R_0	R_1	R_2	R_3	R_4	R_5	R_6
1	0	1	0	8	29	28	27	26	-1
2	0	2	0	12	24	23	2	1	-1
3	0	3	0	8	29	28	27	26	-1
4	0	4	0	25	24	23	22	21	-1
5	0	5	0	8	29	28	27	26	-1
6	0	6	0	25	24	23	22	21	-1
7	0	7	0	8	29	28	27	26	-1
8	0	8	0	12	24	23	2	1	-1
9	0	9	0	8	29	28	27	26	-1
10	0	10	0	25	24	23	22	21	-1
11	0	11	0	8	29	28	27	26	-1
12	0	12	0	25	24	23	22	21	-1
13	0	13	0	8	29	28	27	26	-1
14	0	14	0	12	24	23	2	1	-1
15	0	15	0	8	29	28	27	26	-1
16	0	16	0	25	24	23	22	21	-1
17	0	17	0	30	29	28	27	26	-1
18	0	18	0	25	24	23	22	21	-1
19	0	19	0	30	29	28	27	26	-1
20	0	20	0	12	24	23	2	1	-1
21	1	1	-1	0	8	29	28	27	26
22	1	2	-1	0	12	24	23	2	1
23	1	3	-1	0	8	29	28	27	26
24	1	4	-1	0	25	24	23	22	21
25	1	5	-1	0	8	29	28	27	26
26	1	6	-1	0	25	24	23	22	21
27	1	7	-1	0	8	29	28	27	26
28	1	8	-1	0	12	24	23	2	1
29	1	9	-1	0	8	29	28	27	26
30	1	10	-1	0	25	24	23	22	21
31	1	11	-1	0	8	29	28	27	26
32	1	12	-1	0	25	24	23	22	21
33	1	13	-1	0	8	29	28	27	26
34	1	14	-1	0	12	24	23	2	1
35	1	15	-1	0	8	29	28	27	26
36	1	16	-1	0	25	24	23	22	21
37	1	17	-1	0	30	29	28	27	26
38	1	18	-1	0	25	24	23	22	21
39	1	19	-1	0	30	29	28	27	26
40	1	20	-1	0	12	24	23	2	1
41	2	1	1	-1	0	8	29	28	27
42	2	2	26	-1	0	12	24	23	22
43	2	3	21	-1	0	8	29	28	27
44	2	4	26	-1	0	25	24	23	22
45	2	5	21	-1	0	8	29	28	27
46	2	6	26	-1	0	25	24	23	22
47	2	7	1	-1	0	8	29	28	27
48	2	8	26	-1	0	12	24	23	22
49	2	9	21	-1	0	8	29	28	27
50	2	10	26	-1	0	25	24	23	22
51	2	11	21	-1	0	8	29	28	27
52	2	12	26	-1	0	25	24	23	22
53	2	13	1	-1	0	8	29	28	27
54	2	14	26	-1	0	12	24	23	22
55	2	15	21	-1	0	8	29	28	27
56	2	16	26	-1	0	25	24	23	22
57	2	17	21	-1	0	30	29	28	27
58	2	18	26	-1	0	25	24	23	22
59	2	19	1	-1	0	30	29	28	27
60	2	20	26	-1	0	12	24	23	22
61	3	1	2	1	-1	0	8	29	28
...	3	...	...	...	...	...	...	...	...
80	3	20	27	26	-1	0	12	24	23
81	4	1	23	2	1	-1	0	8	29
...	4	...	...	...	...	...	...	...	...
100	4	20	28	27	26	-1	0	25	24
101	5	1	24	23	2	1	-1	0	8
...	5	...	...	...	...	...	...	...	...
120	5	20	29	28	27	26	-1	0	25
121	6	1	12	24	23	2	1	-1	0
...	6	...	...	...	...	...	...	...	...
222	6	102	30	29	28	27	26	-1	0
223	7	1	0	30	29	28	27	26	-1
...	7	...	...	...	...	...	...	...	...
302	7	80	30	29	28	27	26	-1	0

Table 5.5: Roster for Instance 4 containing Days Off, Workweek Type and pre-assigned Time Frames

Inst.	Objective Value – OR-Tools -RC, -SB	-RC, +SB	+RC, -SB	+RC, +SB
1	61,692,400	61,104,300	**51,265,300**	62,444,800
2	**8,349,260**	48,869,100	11,819,100	44,182,400
3	29,090,300	44,681,700	**26,391,200**	37,934,800
4	**10,368,800**	13,912,300	27,540,800	18,561,000
5	**35,383,800**	66,228,900	50,076,500	51,505,300
6	1,227,730	1,316,770	**1,091,050**	1,659,560
7	69,982,400	**68,267,000**	92,312,000	85,254,400
8	**4,133,890**	11,220,500	35,938,000	8,587,770
9	**74,365,300**	129,392,000	132,111,000	97,588,100
10	48,646,200	92,990,900	**44,830,100**	93,214,300
11	**30,210,800**	140,374,000	187,507,000	185,239,000
12	**53,760,400**	55,130,300	57,340,100	77,332,300
Total	**427,211,280**	733,487,770	718,222,150	763,503,730

Table 5.6: Comparison of Objective Values with or without Redundant Constraints and Symmetry Breaking for OR-Tools

Inst.	Optimality Gap – OR-Tools -RC, -SB	-RC, +SB	+RC, -SB	+RC, +SB
1	99.17	**99.00**	99.54	99.22
2	**94.51**	98.78	97.16	99.23
3	**98.77**	99.06	99.01	98.93
4	**98.54**	98.77	99.51	99.11
5	**92.99**	96.03	94.84	95.11
6	87.63	87.98	**85.53**	91.00
7	81.49	**81.00**	86.02	84.80
8	**95.81**	98.50	99.57	98.15
9	**99.14**	99.54	99.58	99.33
10	**99.04**	99.44	99.09	99.48
11	98.18	**97.69**	99.67	99.72
12	99.38	99.33	**99.32**	99.57
Avg.	**95.39**	96.26	96.57	96.97

Table 5.7: Comparison of Optimality Gaps with or without Redundant Constraints and Symmetry Breaking for OR-Tools

	Objective Value – Gurobi			
Inst.	**-RC, -SB**	**-RC, +SB**	**+RC, -SB**	**+RC, +SB**
1	**912,721**	923,031	913,055	923,031
2	**876,593**	907,902	**876,593**	907,902
3	**674,730**	683,559	**674,730**	683,559
4	**397,838**	408,825	**397,838**	412,106
5	**2,882,328**	2,896,002	**2,882,328**	2,896,004
6	**379,272**	381,779	**379,272**	381,779
7	**13,358,777**	13,361,053		13,361,053
8	680,880	**679,934**		679,958
9		**2,314,417**	2,347,749	**2,314,417**
10				**1,686,856**
11				**5,274,173**
12				**1,293,978**

Table 5.8: Comparison of Objective Values with or without Redundant Constraints and Symmetry Breaking for Gurobi

Table 5.6 presents the objective values for all instances under the four different scenarios using the OR-Tools CP-SAT solver. Table 5.7 shows the respective optimality gaps. The best objective values and gaps are highlighted in bold. For most instances, the scenario without the redundant constraint and without symmetry breaking performed the best. This suggests that the CP-SAT solver does not benefit from either of these two approaches. However, it is worth noting that the solver does not come close to the optimum in any of the four scenarios.

As one can see from Table 5.8, using the proposed symmetry breaking method results in slightly worse objective values. Gurobi achieves the best solutions for instances 1 to 7 when neither the redundant constraint nor symmetry breaking is applied. However, under this setting, Gurobi fails to find any solutions for instances 9 to 12 within the two-hour time limit. Disabling symmetry breaking and adding the redundant constraint does not significantly affect the performance. The solver is then able to find a solution for instance 9, but none for instances 7 and 8. When only symmetry breaking is applied, I am able to also get solutions for instances 9 and 10. By employing both symmetry breaking and the redundant constraint, Gurobi manages to find solutions for all instances, even achieving optimal solutions for 6 of them (see Table 5.9). Here, optimal means that the solution is optimal given the pre-assigned roster. This configuration is also the fastest in finding optimal solutions, as shown in Table 5.10. It seems that symmetry breaking is especially beneficial when solving larger instances.

Table 5.11 compares the average number of solutions generated for the four different scenarios. The CP-SAT solver constructs about ten times as many time frame rosters compared to Gurobi. However, the initial solutions found by Gurobi are usually already better than the best solution computed by OR-Tools CP-SAT solver.

	Optimality Gap – Gurobi			
Inst.	**-RC, -SB**	**-RC, +SB**	**+RC, -SB**	**+RC, +SB**
1	0.0068	0.0245	1.2229	optimal
2	0.1254	0.0472	0.0655	0.0217
3	optimal	optimal	0.0001	optimal
4	optimal	0.0646	0.0050	optimal
5	0.0395	0.0015	0.0089	optimal
6	0.0044	optimal	optimal	optimal
7	0.0084	0.0042		optimal
8	1.1278	0.0751		0.1021
9		0.0536	3.2458	0.0010
10				0.1425
11				0.0142
12				0.3358

Table 5.9: Comparison of Optimality Gaps with or without Redundant Constraints and Symmetry Breaking for Gurobi

	Runtime – Gurobi, Time Limit TL = 2:00 (h:mm)			
Inst.	**-RC, -SB**	**-RC, +SB**	**+RC, -SB**	**+RC, +SB**
1	TL	TL	TL	0:56
2	TL	TL	TL	TL
3	0:48	0:26	TL	0:22
4	1:02	TL	TL	0:24
5	TL	TL	TL	1:43
6	TL	1:27	1:21	1:42
7	TL	TL	TL	1:24
8	TL	TL	TL	TL
9	TL	TL	TL	TL
10	TL	TL	TL	TL
11	TL	TL	TL	TL
12	TL	TL	TL	TL

Table 5.10: Comparison of Runtimes with or without Redundant Constraints and Symmetry Breaking for Gurobi

Inst.	Gurobi Avg. #Sol.	OR-Tools Avg. #Sol.
1	32	285
2	23	295
3	45	407
4	10	381
5	20	322
6	37	334
7	25	298
8	39	271
9	14	67
10	13	59
11	8	50
12	9	55

Table 5.11: Average Number of Solutions generated by Gurobi and OR-Tools

As an example, Table 5.12 shows an optimal time frame roster for instance 4, computed by Gurobi utilizing both the redundant constraint and symmetry breaking.

$\|R\|$	o	$\|o\|$	R_0	R_1	R_2	R_3	R_4	R_5	R_6
1	0	1	0	8	8	8	7	6	-1
2	0	2	0	12	12	3	2	1	-1
3	0	3	0	8	7	6	5	11	-1
4	0	4	0	15	15	3	3	2	-1
5	0	5	0	8	7	6	5	11	-1
6	0	6	0	13	14	3	2	1	-1
7	0	7	0	8	7	6	11	4	-1
8	0	8	0	12	12	3	2	1	-1
9	0	9	0	8	7	6	5	11	-1
10	0	10	0	11	15	3	3	2	-1
11	0	11	0	8	8	8	7	6	-1
12	0	12	0	15	3	2	2	1	-1
13	0	13	0	8	7	6	5	11	-1
14	0	14	0	12	12	3	2	1	-1
15	0	15	0	8	7	6	5	11	-1
16	0	16	0	13	13	3	2	1	-1
17	0	17	0	7	6	5	11	4	-1
18	0	18	0	11	15	3	3	2	-1
19	0	19	0	8	7	6	5	11	-1
20	0	20	0	12	14	3	2	1	-1
21	1	1	-1	0	8	8	8	7	7
22	1	2	-1	0	12	14	3	2	1
23	1	3	-1	0	8	8	8	7	7
24	1	4	-1	0	12	3	2	1	1
25	1	5	-1	0	8	8	8	7	7
26	1	6	-1	0	11	15	3	2	1
27	1	7	-1	0	8	8	8	7	7
28	1	8	-1	0	12	12	3	2	1
29	1	9	-1	0	8	8	8	7	7
30	1	10	-1	0	13	13	3	1	1
31	1	11	-1	0	8	8	8	7	7
32	1	12	-1	0	12	12	3	2	1
33	1	13	-1	0	8	8	8	7	7
34	1	14	-1	0	12	12	3	2	1
35	1	15	-1	0	8	8	8	7	7
36	1	16	-1	0	12	14	3	1	1
37	1	17	-1	0	8	8	8	7	7
38	1	18	-1	0	11	15	3	2	1
39	1	19	-1	0	8	8	8	7	7
40	1	20	-1	0	12	12	3	2	1
41	2	1	1	-1	0	8	7	6	5
42	2	2	11	-1	0	12	14	3	2
43	2	3	1	-1	0	8	8	8	7
44	2	4	6	-1	0	15	11	4	3
45	2	5	2	-1	0	8	7	6	5
46	2	6	15	-1	0	11	15	3	2
47	2	7	1	-1	0	8	8	7	6
48	2	8	11	-1	0	12	12	4	3
49	2	9	2	-1	0	8	7	6	5
50	2	10	15	-1	0	13	3	2	1
51	2	11	1	-1	0	8	8	7	6
52	2	12	11	-1	0	15	11	4	3
53	2	13	1	-1	0	8	7	6	5
54	2	14	15	-1	0	12	14	3	2
55	2	15	1	-1	0	8	8	7	6
56	2	16	11	-1	0	11	15	4	3
57	2	17	2	-1	0	8	7	6	5
58	2	18	11	-1	0	12	12	3	2
59	2	19	1	-1	0	7	6	5	11
60	2	20	15	-1	0	12	12	4	3
61	3	1	2	1	-1	0	8	7	6
...	3	...	...	...	...	...	...	...	...
80	3	20	11	11	-1	0	12	14	3
81	4	1	3	2	1	-1	0	8	7
...	4	...	...	...	...	...	...	...	...
100	4	20	11	11	11	-1	0	11	4
101	5	1	3	2	2	1	-1	0	8
...	5	...	...	...	...	...	...	...	...
120	5	20	6	5	11	15	-1	0	4
121	6	1	12	12	3	2	1	-1	0
...	6	...	...	...	...	...	...	...	...
222	6	102	7	6	6	5	11	-1	0
223	7	1	0	8	7	6	5	11	-1
...	7	...	...	...	...	...	...	...	...
302	7	80	8	7	6	5	11	-1	0

Table 5.12: Time Frame Roster for Instance 4

5.3.3 Further Experiments

Chuffed

As mentioned in the experimental setup, I also tested the lazy clause generation solver Chuffed 0.13.1 [CDLBS10]. However, even with an extended time limit of ten hours, Chuffed was unable to solve any of the instances. Since Chuffed does not support parallel processing, I conducted a fair comparison by also running Gurobi and OR-Tools on a single thread. Despite this limitation, both Gurobi and OR-Tools managed to find solutions for almost all instances within one hour, whereas Chuffed did not find a single solution in ten hours.

Modeling Hard Constraints as Soft Constraints

In addition to soft constraint s_4, I attempted to model hard constraints h_{2_d} and h_{2_e} as soft constraints with high penalties in order to relax the constraints and assist Gurobi in finding initial solutions. However, this approach did not really affect Gurobi's performance and significantly reduced the performance of the OR-Tools CP-SAT solver. Consequently, I discarded the modified formulation and did not include the results in this thesis.

Extended Time Limit

To assess whether Gurobi is able to find optimal solutions for all instances within a reasonable amount of time, I extended the time limit to four hours and re-ran instances 2 and 8 to 12 using both symmetry breaking and the redundant constraint. Gurobi successfully found an optimal solution for instance 9 in just over two hours (2:06) and came very close to the optimum for the remaining instances, with optimality gaps g of less than 0.1.

5.4 Monte Carlo Simulation to Simulate Tram Drivers' Absences

To check whether the abstract constraint formulated in Section 4.5 does indeed hold, I simulate absences by running Monte Carlo simulations. Monte Carlo simulations are stochastic simulations based on random samples from a probability distribution [RK07, p. 82]. In this case, the drivers' absences p_{abs} follow a binomial distribution. This enables us to simulate absences by discarding some frames based on the probability p_{abs} using the following procedure:

$$R_{i_{sim}} = [f \mid f \in R_i,\ r \leq 1 - p_{abs}]$$

where r is for each frame in R_i a newly sampled, random real number between 0 and 1. If r is smaller or equal to $1 - p_{abs}$, the frame is kept. Otherwise, the frame is discarded (i.e., the driver is absent).

Using this reduced list of frames, I simulate the shift assignment for weekday i by deploying the following simulation model[4] modeled in MiniZinc:

Variables:

S_i ... list of shifts for weekday i
$R_{i_{sim}}$ ... list of time frames for weekday i
X_i ... list of length $|R_{i_{sim}}|$ with domain 0 to $|S_i|$

Constraints:

$$
\begin{aligned}
c_1 &: \text{ All elements of } X_i \text{ except 0 must be different} \\
c_2 &: \forall s[j] \in S_i : \; t_{s[j]} = 1 \implies j \in X_i \\
c_3 &: \; |R_{i_{sim}}| \leq |S_i| \implies 0 \notin X_i \\
c_4 &: \; |R_{i_{sim}}| > |S_i| \implies \sum_{x \in X_i,\; x=0} 1 = |R_{i_{sim}}| - |S_i| \\
c_5 &: \; \forall j \in [0, ..., |R_{i_{sim}}|] : \; ((X[j] > 0) \implies t_{s[X[j]]} = 0) \implies \\
&\quad t_{R_{i_{sim}}[j]} \neq 1 \; \wedge \; [a_{s[X[j]]}, b_{s[X[j]]}] \in [c_{R_{i_{sim}}[j]}, d_{R_{i_{sim}}[j]}] \\
c_6 &: \; \forall j \in [0, ..., |R_{i_{sim}}|] : \; ((X[j] > 0) \implies t_{s[X[j]]} = 1) \implies \\
&\quad t_{R_{i_{sim}}[j]} > 0 \; \wedge \; [a_{s[X[j]]}, b_{s[X[j]]}] \in [c_{R_{i_{sim}}[j]}, d_{R_{i_{sim}}[j]}]
\end{aligned}
$$

I cannot assign the same shift to different time frames (c_1). Split shifts must be assigned (c_2). If the number of remaining time frames is less or equal the number of shifts, all of these time frames have to be assigned a shift (c_3). If the number of remaining time frames is greater than the number of shifts, then all shifts have to be assigned to time frames (c_4). The number of time frames without a shift assigned is then $|R_{i_{sim}}| - |S_i|$. If a shift of type 0 is assigned to a time frame, then this time frame cannot be of type 1 and the shift's interval must lie within the time frame's interval (c_5). If a shift of type 1 is assigned to a time frame, then this time frame cannot be of type 0 and the shift's interval must lie within the time frame's interval (c_6).

Since the simulation (shift assignment) itself is NP-hard [Lau96] and rather time-consuming, I do not simulate the entire roster week at once but separately for each weekday. For each solution (i.e., time frame roster) and weekday i, 100 shift assignments are simulated. The number of failed shift assignments for weekday i is denoted as sim_{fail_i}. The rate of success is then defined as:

$$sim_{suc} = (1 - \frac{\sum_{i=0}^{6} sim_{fail_i}}{700}) \cdot 100$$

The objective is to achieve a success rate $sim_{suc} > p_{suc}$[2] (see Section 4.5). Table 5.13 presents the simulation results for all instances and scenarios using Gurobi, while Table 5.14 presents the simulation results for all instances and scenarios using OR-Tools.

Simulation Results sim_{suc} – Gurobi				
Inst.	**-RC, -SB**	**-RC, +SB**	**+RC, -SB**	**+RC, +SB**
1	99.43	99.29	99.14	99.14
2	99.57	98.71	99.14	99.29
3	99.29	99.29	99.29	99.00
4	98.43	99.43	98.86	98.86
5	99.29	99.14	98.14	99.86
6	99.14	99.86	99.29	99.43
7	97.14	96.86		96.71
8	99.00	98.86		99.29
9		99.57	99.14	98.57
10				99.29
11				98.71
12				99.43
Avg.	98.91	99.00	98.96	98.96

Table 5.13: Simulation Results for Gurobi

Simulation Results sim_{suc} – OR-Tools				
Inst.	**-RC, -SB**	**-RC, +SB**	**+RC, -SB**	**+RC, +SB**
1	95.14	92.86	97.57	92.57
2	98.86	94.71	99.14	96.57
3	97.57	95.71	97.43	95.86
4	98.86	99.71	98.71	98.57
5	97.29	89.29	94.86	96.57
6	97.86	99.14	98.43	98.57
7	92.29	90.57	83.86	87.29
8	99.00	98.43	95.71	98.71
9	98.86	94.86	95.43	98.57
10	99.29	98.14	99.00	98.86
11	97.86	92.00	68.29	67.43
12	99.29	99.14	98.57	99.00
Avg.	97.68	95.38	93.92	94.05

Table 5.14: Simulation Results for OR-Tools

[4] https://github.com/lukasfruehwirth/time_frame_rostering/blob/main/shift_assignment_simulation.mzn

5.5 Discussion

Tables 5.9 and 5.7 clearly demonstrate that the proposed model performs significantly better with the Gurobi solver compared to the OR-Tools solver, especially when the redundant constraint and symmetry breaking is applied. Gurobi consistently achieves superior results, quickly finding optimal solutions for 6 out of 12 instances and coming very close to optimality for the remaining ones. Notably, even for the larger instances 9 to 12 Gurobi produces solutions with a gap to the best known lower bound smaller than 0.34%. Conversely, the constraint solver OR-Tools fails to solve any instances within the time limit to optimality. The smallest gap reached by OR-Tools is 81%. Thus, all solutions provided by OR-Tools are considerably far from the optimal solution. OR-Tools performs comparably well in finding a satisfiable solution and can compute approximately ten times more solutions than Gurobi within the designated time limit (see Table 5.11). However, these additional solutions provide only minor improvements over the first solution found. It appears that OR-Tools struggles with the model's objective function, which consists of several independent soft constraints. However, Gurobi also faces some challenges, albeit of a different nature. Without symmetry breaking or the redundant constraint, it occasionally fails to find any solution within two hours, particularly for the larger instances, such as instances 9 to 12.

It is crucial to obtain solutions close to the optimum; otherwise, the success rate of the simulation falls below acceptable levels. This observation is supported by examining the simulation results computed for solutions provided by the OR-Tools solver. Table 5.14 shows that for 30 out of 48 solutions generated by OR-Tools, the success rate is below ${p_{suc}}^2 \cdot 100 = 98.01\%$. Conversely, Table 5.13 shows that for 32 out of 36 solutions generated by Gurobi, a success rate exceeding ${p_{suc}}^2 \cdot 100 = 98.01\%$ is achieved. This suggests that hard constraints h_1 through h_3 and soft constraint s_1 effectively model the abstract constraint mentioned at the beginning of Section 4.5. The only instance where the success rate falls below 98.01% is instance 7. If we take a closer look at the solution of this instance, we can see that the lower bound for hard constraint h_{2_d} is decreased and that the softened hard constraint h_{2_f} is violated in order to not violate hard constraint h_5 (forbidden frames), resulting in an objective value significantly surpassing 1,000,000, as Table 5.8 reveals. To improve the success rate for these instances, practitioners may need to consider relaxing some of the hard constraints h_4 to h_8 and, in particular, h_5.

CHAPTER 6

Conclusion

In this thesis, I formally introduce the time frame rostering problem, a novel challenge arising from the desire for enhanced medium-term planning security and, generally, more robust rosters in tram driver rostering. I provide an abstract problem description and translate this description into a formal problem definition. Based on this defintion, a solver-independent model using MiniZinc is created. The modeling involves processing the given data to derive lower and upper bounds for some of the hard constraints of the model. Additionally, I verify whether the abstract constraint mentioned in Section 4.5 is indeed fulfilled by simulating tram drivers' absences and subsequent shift assignments.

The results reveal that the MIP solver Gurobi is able to solve 6 out of 12 real-world instances to optimality in less than two hours. For the remaining instances, when redundant constraints and symmetry breaking are applied, Gurobi generates solutions very close to the optimum within two hours. The proposed model, when used with a state-of-the-art solver like Gurobi, efficiently constructs near-optimal time frame rosters, even for rosters with up to 700 rows. Furthermore, the simulation shows that solutions based on my model are indeed feasible and deployable in practice, as for almost all instances and simulation runs, the assignment of shifts to time frames is successfully completed. Some instances had a slightly lower success rate in the simulation because I allowed certain hard constraints, in particular hard constraint h_{2_d} to be violated to satisfy others. This is a trade-off that practitioners should consider when using the model. In summary, it can be stated that the model proposed in this thesis works as intended, as it successfully models the abstract problem description of the time frame rostering problem. Using this model together with state-of-the-art solvers like Gurobi, one can solve real-world instances to (nearly) optimal levels within a reasonable amount of time.

Future work could test if other constraint solvers outperform OR-Tools and explore approaches to improve the performance of constraint solvers. When solving larger instances, a combination of constraint programming and integer programming could be advantageous. The OR-Tools CP-SAT solver is typically quick at finding an initial

solution but performs poorly when it comes to finding solutions close to the optimum. Hence, the idea is to generate an initial solution by the CP-SAT solver and use this solution as a warm start for Gurobi, preventing Gurobi from getting stuck in finding an initial solution. Future work could also address the time frame properties itself. Further investigation into optimizing time frame rosters may involve penalizing the interval width of the time frames to further enhanced medium-term planning security of tram drivers. Another way to improve driver satisfaction might be to incorporate their preferences into the time frame rostering model.

Overview of Generative AI Tools Used

This diploma thesis was proofread with the assistance of ChatGPT-4o. ChatGPT-4o was used to correct grammar and spelling errors as well as to make minor stylistic adjustments to improve the readability of this thesis. The content and arguments of this thesis were neither generated nor altered by ChatGPT.

List of Tables

List of Algorithms

Bibliography

[Apt03] Krzysztof Apt. *Principles of constraint programming*. Cambridge university press, 2003.

[BCBL04] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *J. Sched.*, 7(6):441–499, 2004.

[BSSW17] Ralf Borndörfer, Christof Schulz, Stephan Seidl, and Steffen Weider. Integration of duty scheduling and rostering to increase driver satisfaction. *Public Transport*, 9:177–191, 2017.

[CDLBS10] Geoffrey Chu, Maria Garcia De La Banda, and Peter J Stuckey. Automatically exploiting subproblem equivalence in constraint programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings 7*, pages 71–86. Springer, 2010.

[Dan51] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951. Originally published in 1947.

[Dan54] George B Dantzig. A comment on edie's "traffic delays at toll booths". *Journal of the Operations Research Society of America*, 2(3):339–341, 1954.

[Dan57] George B Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

[EJK+01] Andreas T Ernst, Houyuan Jiang, Mohan Krishnamoorthy, Helen Nott, and David Sier. An integrated optimization model for train crew management. *Annals of Operations Research*, 108(1):211–224, 2001.

[EJKS04] Andreas T Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1):3–27, 2004.

[GB65] Solomon W Golomb and Leonard D Baumert. Backtrack programming. *Journal of the ACM (JACM)*, 12(4):516–524, 1965.

[GG61] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[Gom58] Ralph E Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[Gur23] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. `https://www.gurobi.com`.

[GVX22] Liping Ge, Stefan Voß, and Lin Xie. Robustness and disturbances in public transport. *Public Transport*, 14(1):191–261, 2022.

[HHB20] Julia Heil, Kirsten Hoffmann, and Udo Buscher. Railway crew scheduling: Models, methods and applications. *European journal of operational research*, 283(2):405–425, 2020.

[IEE90] IEEE. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[IM15] Jonas Ingels and Broos Maenhout. The impact of reserve duties on the robustness of a personnel shift roster: An empirical investigation. *Computers & Operations Research*, 61:153–169, 2015.

[KM20] Lucas Kletzander and Nysret Musliu. Solving the general employee scheduling problem. *Computers & Operations Research*, 113:104794, 2020.

[Lau96] Hoong Chuin Lau. On the complexity of manpower shift scheduling. *Computers & Operations Research*, 23(1):93–102, 1996.

[LD60] AH Land and AG Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[LJC20] Dung-Ying Lin, Chieh-Ju Juan, and Ching-Chih Chang. A branch-and-price-and-cut algorithm for the integrated scheduling and rostering problem of bus drivers. *Journal of Advanced Transportation*, 2020:1–19, 2020.

[LT19] Dung-Ying Lin and Meng-Rung Tsai. Integrated crew scheduling and roster problem for trainmasters of passenger railway transportation. *IEEE Access*, 7:27362–27375, 2019.

[Lüb10] Marco E Lübbecke. Column generation. *Wiley encyclopedia of operations research and management science*, 17:18–19, 2010.

[Mac77] Alan K Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1):99–118, 1977.

[MJSS16] David R Morrison, Sheldon H Jacobson, Jason J Sauppe, and Edward C Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[MMZ+01] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535, 2001.

[mzp] Minizinc python 0.9.0. `https://minizinc-python.readthedocs.io/en/0.9.0/index.html`. Accessed: 2024-05-30.

[NSB+07] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[NW99] George Nemhauser and Laurence Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, second edition, 1999.

[PD23] Laurent Perron and Frédéric Didier. Cp-sat v9.8, 2023. `https://developers.google.com/optimization/cp/cp_solver`.

[PR91] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[RK07] Reuven Y Rubinstein and Dirk P Kroese. Simulation and the monte carlo method (wiley series in probability and statistics), 2007.

[RVBW06] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[VBD+13] Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

[VJLS94] S. DAVID WU V. JORGE LEON and ROBERT H. STORER. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, 1994.

[VRD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[Wic19] Toni Ismael Wickert. *Personnel rostering: models and algorithms for scheduling, rescheduling and ensuring robustness*. Doctoral thesis, Universidade Federl Do Rio Grande Do Sul and KU Leuven, 2019.

[WSVB21] Toni I Wickert, Pieter Smet, and Greet Vanden Berghe. Quantifying and enforcing robustness in staff rostering. *Journal of Scheduling*, 24(3):347–366, 2021.

[XS12] Lin Xie and Leena Suhl. A stochastic model for rota scheduling in public bus transport. In *Proceedings of 2nd Stochastic Modelling Techniques and Data Analysis International Conference*, pages 785–792, 2012.

[YMDS05] Tallys H Yunes, Arnaldo V Moura, and Cid C De Souza. Hybrid column generation approaches for urban transit crew management problems. *Transportation Science*, 39(2):273–288, 2005.