Exact Methods for the Time Frame Rostering Problem in the Context of Tram Driver Scheduling

Lukas Frühwirth and Nysret Musliu

Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Favoritenstraße 9, 1040 Vienna, Austria

Abstract. Creating more robust rosters that offer medium-term planning security for employees is a desired goal in the public transportation sector. To tackle this problem, we introduce a new approach in the context of tram driver scheduling 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 paper, we formally define the time frame rostering problem and provide a solver-independent model of the problem. Furthermore, we compare two state-of-the-art solvers on real-world instances and demonstrate that optimal or almost optimal solutions can be found in a reasonable amount of time. Additionally, we verify these solutions by simulating absences and subsequent shift assignment.

Keywords: Tram Driver Scheduling, Crew Rostering, Public Transport Scheduling,

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 [8]. 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 [6], [16], such as calculating the optimal amount of reserve shifts [18], [9], as well as re-rostering methods [17]. 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 our knowledge, the existing literature does not provide a formal definition or a formulated model of the time frame rostering problem that we consider in this paper. 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 paper, the time frames and their specifications (start time, end time, type) are regarded as predetermined.

The main contributions of this paper are:

- We provide a formal problem definition and formulate a solver-independent model of the time frame rostering problem.
- We publish real-world instances based on data provided by a public transportation company.
- We empirically evaluate different exact solving methods using these real-world instances and show that these are optimally solvable within 120 minutes.
- We demonstrate the feasibility of our solutions by simulating staff absences and subsequent shift assignment.

This paper is part of a master's thesis [5] that is currently under submission and is expected to be published by September 2024.

The article is structured as follows: In the next section, we provide an overview of the related work, followed by a high-level problem description in section 3. In section 4, we precisely define the time frame rostering problem, and formulate a solver-independent model. Subsequently, in section 5, we evaluate the model by solving real-world instances using the linear solver Gurobi [7] and the constraint solver OR-Tools [14]. Finally, we draw our conclusions in section 6.

2 Related Work

Tram driver scheduling can be considered a subset of crew scheduling, where each crew consists of just a single member, the tram driver. The time frame rostering problem is then a subset of the challenges encountered in crew planning or, generally, in workforce scheduling. To the best of our knowledge, the time frame rostering problem defined in this paper has not been addressed in the existing literature.

Crew planning for (public) transportation typically comprises two primary stages: the first stage involves crew scheduling, also known as shift design or shift/duty scheduling, where shifts are designed based on a predetermined timetable. The second stage, referred to as crew rostering, entails assigning these shifts to typically rotating or cyclical rosters [8].

These two stages are also integral components of many workforce scheduling approaches in general [11]. While the two stages are often treated as separate optimization problems [8], integrated approaches in crew planning with a single objective function exist [1], [12]. On a more general level, solving the general employee scheduling problem, as demonstrated by Kletzander and Musliu [10], is also done by integrating several stages.

Extensive literature explores how crew scheduling [8], crew rostering [8], and workforce scheduling problems in general [2,4,15], can be addressed using mathematical programming, constraint programming, answer-set programming, heuristics, metaheuristics, and combinations thereof. In shift scheduling, mathematical programming is the most popular approach [15]. Mathematical programming typically uses a set covering formulation, introduced by Dantzig in 1954 [3]. Our model of the time frame rostering problem is also based on such a formulation.

As mentioned in the introduction, a crew schedule, and particularly a crew roster, undergoes several re-rosterings due to factors such as timetable changes, construction sites, fluctuations in headcount, absences, and more. Thus, the literature also covers methods to deal with uncertainty by constructing more robust rosters [6], [16], such as introducing reserve duties [9], determining the optimal amount of reserve shifts [18], as well as re-rostering methods [17]. However, the methodology of introducing time frames and initially assigning these frames to rotating rosters based on the crew schedule (shifts), followed by assigning shifts within these time frames, has not been proposed to date. This time frame roster is more robust to changes than a typical shift roster, as the shift assignment occurs at a later stage where many factors leading to re-rostering are already known and accounted for in the assignment process.

3 Problem Description

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

Conventional Approach Typically, tram driver scheduling 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. Each driver has either two specific consecutive days off or follows a rotational day off schedule, resulting in eight different day off types. Tram drivers are 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 1, 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 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 1. 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.

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	r	r	r	r
o_2	s	off	off	s	s	s	s
o_2	s	...	...	s	s	s	s
o_3	r	r	off	off	s	s	s
o_3	s	s	..	...	r	r	r
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	r	r	r	r	...	...
o_7	off	off	s	s	s	s	s
o_7	s	off	off	r	r	r	r

Table 1: 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

Table 2: Exemplary Time Frame Roster

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. Our 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, we initially allocate time frames to roster positions (see Table 2).

In the time frame roster above, each time frame is indicated by a number between 1 and 15 since we consider 15 different time frames. 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 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 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 (i.e., drivers) 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. We provide a formal definition of the problem and its constraints in the next section.

4 Formal Problem Definition and Model

To formally define the time frame rostering problem, we first need to specify the given data. Secondly, we explain what constitutes a time frame roster, and the role it plays in tram driver scheduling. Thirdly, we will outline how to create a day off schedule. Finally, we 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}, ranging from 0 to 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 different day off types $o_0, ..., o_7$. Types 0 to 6 have two fixed consecutive days off, while type 7 follows a 16-week-long rotating day off schedule, which is provided by a public transport company.
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.

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.

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 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

We 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}:

```
1 min_demand(p_abs, p_suc, lb):
2   If(lb = 0)
3     minDemand = 0
4   Else
5     minDemand = lb
6     While(binom.cdf(minDemand-lb, minDemand, p_abs) <= p_suc
          )
7       minDemand = minDemand + 1
8   Return minDemand
```

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

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

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 we 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, we use the function call: *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 we 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, we first create a day off schedule to determine the size of the roster and the placement of the days off. We 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}:

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

The demand dem_i is then used to determine the day off schedule, for a detailed description, please refer to Appendix 6. Based on the day off schedule, we can proceed to construct and encode the empty roster R containing only the days off. Given this roster R, the goal now is to assign time frames to the empty positions 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.

4.5 Hard Constraints

Given an assigned 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, we 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, we 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, we would need to simulate the shift assignment to time frames which itself is a NP-hard problem [11]. Hence, we break down the abstract constraint into hard constraints $h_1 - h_3$ and soft constraint s_1. Additionally, we 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 5.2).

Minimum Coverage First, we 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:

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

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

Minimum Frame Set The hard constraint h_1 is not sufficient by itself to ensure that shifts fit into the available time frames. We need to guarantee that there are enough frames to accommodate the shifts, taken into account the probability of absences p_{abs}. A shift can lie in the interval of several time frames, hence we obtain a set of time frames

FS_s covering a shift s. We determine the minimum size required for each of these sets of time frames, similar to how we determine the necessary count of drivers and days off:

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

We also compute frame sets FS_j for artificial shifts of length 4 (assumed minimum shift length) to get every possible frame set. Subsequently, we 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}:

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

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, increasing 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. Hence, there are special minimum frame set constraints for split shifts, for details please refer to Appendix 6.

Maximum Frame Set In addition to the hard constraints h_1 and h_2, we define maximum frame sets. Instead of counting shifts with the same frame set, we 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 from frame set FS_j:

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

We 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, we adjust the maximum to match the number of shifts $|S_i|$ for weekday i. In case the maximum is lower than the minimum, we decrease the minimum such that it equals the maximum.

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

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}:

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

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:

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

Our model includes additional hard constraints, which ensure that weekdays with similar shifts also have similar time frames, and that the solution contains a certain number of night and relief frames. These constraints are defined in Appendix 6.

Additionally, our problem includes several soft constraints. These constraints are defined in Appendix 6.

4.6 Decision Variable

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

4.7 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:

$$\min\ w_1s_1 + w_2s_2 + w_3s_3 + w_4s_4 + w_5s_5 + w_6s_6 \qquad w_1, w_2, w_3, w_4, w_5, w_6 \in \mathbb{N}^+$$
$$\text{s.t. } h_1 - h_8 \text{ hold}$$

4.8 Solver-independent Model

Based on the problem definition provided in this section, we create a solver-independent model of the time frame rostering problem by using MiniZinc [13]. A MiniZinc model allows us to use both linear and constraint solvers. Our model[1] includes all hard and soft constraints as well as the objective function outlined in the problem definition. To speed up the solving process, we also make use of redundant constraints and symmetry breaking.

Redundant Constraints We include redundant constraints for the forbidden sequence constraint. This is done by exploiting the time frames numbering. For some workweek types and time frames we can state, for example, that the following time frame must be smaller than or equal to the current one without checking the list of forbidden sequences.

Symmetry Breaking We can pre-assign time frames $f \in FS_j$ if there is only a single frame in the set ($|FS_j| = 1$). In such cases, there is no choice of time frames, and we know, due to the minimum frame set constraint, that a certain number of these time frames have to be in the roster. We pre-assign these time frames so that they are uniformly distributed across the roster and assigned to positions where they do not influence the overall solution quality.

5 Evaluation

We evaluate our proposed model 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. We also make this data available to the scientific community[2]. Instance properties, time frame properties and parameter settings can be found in Appendix 6. The evaluation process involves several steps: Firstly, we solve the instances using two different state-of-the-art solvers and compare the results. Secondly, we validate the solutions using a simulation model. Finally, we discuss the obtained results in detail.

5.1 Results

To evaluate our solver-independent model, we solve the instances using two conceptually different state-of-the-art solvers: linear solver Gurobi 11.0.0 [7] and constraint solver OR-Tools 9.8.3296 [14].

The test setup looks as follows: For each instance, we set the time limit to 120 minutes. 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.

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

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

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

Table 3 shows the objective values, optimality gaps and runtimes for both solvers:

			Gurobi		OR-Tools		Gurobi	OR-Tools
Inst.	|S|	|R|	Obj. Value ov	Gap g	Obj. Value ov	Gap g	Runtime (mm:ss)	Runtime (mm:ss)
1	1,380	364	1,868,046	optimal	5,127,240	87.31%	16:15	TL
2	1,159	308	4,808,321	optimal	31,525,200	91.11%	10:04	TL
3	1,282	338	3,487,611	optimal	7,832,640	67.93%	39:49	TL
4	1,456	382	20,100,971	optimal	190,637,000	91.10%	21:35	TL
5	2,539	652	4,212,492	0.040%	113,196,000	99.39%	TL	TL
6	2,738	702	6,784,287	0.594%	303,801,000	99.83%	TL	TL
7	1,344	354	2,041,618	0.015%	4,138,020	74.92%	TL	TL
8	1,134	302	905,588	optimal	1,438,480	68.78%	17:29	TL
9	1,224	324	780,783	optimal	3,436,120	90.28%	29:46	TL
10	1,375	362	1,070,283	optimal	2,389,440	85.27%	42:38	TL
11	2,478	636	2,970,358	0.004%	10,427,500	94.47%	TL	TL
12	2,599	666	2,062,171	0.040%	160,549,000	99.80%	TL	TL

Table 3: Results – Comparison of Gurobi and OR-Tools

To improve the performance of both solvers, we tried to circumvent modeling hard constraints as soft constraints with high penalties (see soft constraint s_4). Instead of relying on $below(h_{2_d})$ and $below(h_{2_e})$, we pre-determine the maximum number of positions in roster R for weekday i that can accommodate frames of type $t_f \in \{1, 2\}$, according to the specified workweek types wt for each roster position. If the minimum demand for h_{2_d} and h_{2_e}, as formalized in Section 4.5, surpasses this maximum, we adjust the minimum to match the identified maximum. This approach allows us to define h_{2_d} and h_{2_e} as hard constraints in our model, while ensuring the model's satisfiability. The soft constraint s_4 is thus reduced to $s_4 = below(h_{2_f}) \cdot pen_{hard\_con}$. Nonetheless, this modification to the MiniZinc model did not really affect Gurobi's performance and only slightly improved OR-Tools' performance, which remained significantly below that of Gurobi. Consequently, we did not include the results in this paper. However, results for the modified model can be found in the master's thesis [5].

5.2 Simulation

To check whether our abstract constraint formulated in section 4.5 does indeed hold, we simulate absences. This is done by discarding some frames according to p_{abs}:

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

Where $rand$ is for each frame in R_i a newly sampled, random real number between 0 and 1.

Using this reduced list of frames, we simulate the shift assignment for weekday i by deploying a simulation model[3] modeled in MiniZinc. For a detailed model description, please refer to Appendix 6.

Since the simulation (shift assignment) itself is NP-hard [11] and rather time-consuming, we 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, we simulate 100 shift assignments. 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 4 presents the simulation results for all instances and solvers:

			Gurobi	OR-Tools
Inst.	|S|	|R|	Sim. Result sim_{suc}	Sim. Result sim_{suc}
1	1,380	364	99.14%	97.48%
2	1,159	308	98.43%	86.86%
3	1,282	338	98.57%	98.00%
4	1,456	382	96.14%	26.86%
5	2,539	652	99.29%	62.57%
6	2,738	702	99.71%	23.86%
7	1,344	354	99.00%	98.57%
8	1,134	302	98.71%	97.86%
9	1,224	324	99.14%	96.86%
10	1,375	362	99.00%	97.43%
11	2,478	636	99.71%	98.71%
12	2,599	666	98.71%	27.43%

Table 4: Simulation Results

5.3 Discussion

Table 3 clearly demonstrates that our model performs significantly better with the solver Gurobi compared to the OR-Tools solver. Gurobi consistently achieves superior results, quickly finding optimal solutions for 7 out of 12 instances and coming very close to optimality for the remaining ones. Notably, even for the larger instances 5, 6, 11, and 12, Gurobi produces solutions with a gap to the best known lower bound smaller than 0.6%. 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 67.93%. Thus, all solutions

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

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. However, these additional solutions provide only minor improvements over the first solution found. It appears that OR-Tools struggles with our model's objective function, which consists of several independent soft constraints.

Table 4 shows that for 11 out of 12 solutions generated by Gurobi, we achieve a success rate exceeding ${p_{suc}}^2 \cdot 100 = 98.01\%$. 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 4. If we take a closer look at the solution of this instance, we can see that the softened hard constraints h_{2_d} and h_{2_f} have to be violated in order to not violate hard constraint h_5 (forbidden frames), resulting in an objective value significantly surpassing 1,000,000, as Table 3 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. Since we have softened some hard constraints, it becomes 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 4 shows that for 10 out of 12 solutions generated by OR-Tools, the success rate is below ${p_{suc}}^2 \cdot 100 = 98.01\%$.

6 Conclusion

In this study, we 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 scheduling. We translate the abstract problem description into a formal problem definition and provide a solver-independent model using MiniZinc. This translation involves processing the given data to derive lower and upper bounds for some of the hard constraints of the model. Additionally, we 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 linear solver Gurobi is able to solve 7 out of 12 real-world instances to optimality in less than an hour. For the remaining instances, Gurobi generates solutions very close to the optimum within two hours. Furthermore, the simulation shows that solutions based on our 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 we allowed certain hard constraints to be violated to satisfy others. This is a trade-off that practitioners should consider when using our model. In summary, it can be stated that we have successfully developed a model for the time frame rostering problem, that can be used to 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. Moreover, further investi-

gation 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.

Acknowledgements The financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

Appendix A - Day Off Schedule

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, we 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 += 2, dem_6 += 2,
else if $frac = 0.4$ then dem_6 −= 2,
else if $frac = 0.8$ then dem_6 += 2,
else if $frac = 0.8$ then dem_5 −= 2, dem_6 −= 2

We determine the number of roster positions of roster R 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 we limit the size of type o_i to $|o_i| \leq 2|o_{i+1}|$ and $2|o_i| \geq |o_{i+1}|$ for $i \in \{0, ..., 5\}$. In order to find the optimal day off schedule, we formulate the following model in MiniZinc[4] [13]:

Variables:

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

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

Objective function:

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

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

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%:

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

Day off types 0-6 are reduced accordingly:

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

Appendix B- Minimum Frame Sets for Split Shifts

The number of time frames in the roster allowing only 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 we 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}).

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

Appendix C - Additional Hard Constraints

Maximum Frame Deviation Weekdays with similar shifts should also have similar time frames. To ensure this, we define a maximum deviation of a time frame count between two weekdays. The following algorithm uses the minimum coverage definition (see h_1):

```
max_deviation(min_cov, dev_allowed):
  For(i in {0,...,5})
    x = ∑_{j=0}^{29} min_cov_{i,j}
    y = ∑_{j=0}^{29} min_cov_{i+1,j}
  If(x < y)
    dev = y / x
  Else
    dev = x / y
  maxDev[i] = dev_allowed + ceil(500(dev − 1))
Return maxDev
```

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 $h_{6_{b-d}}$).

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

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

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

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

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:

$$h_{7_a} : |\{s \mid s \in S_i,\ b_s > 26\}| = 0 \implies |\{f \mid f \in R_i,\ d_f > 26\}| = 0$$

$$h_{7_b} : max_{FS_{i,j}} := min_{FS_{i,j}} \quad i \in \{0, ..., 6\},\ FS_j = \{f \mid f \in F,\ d_f > 26\}$$

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 $FS_r = FS_e \cap FS_l$. We 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}:

$$lb_i = \arg\min_x(\sqrt{1-p_{suc}} \leq binom.cdf(x,\ min(|FS_{i,e}|, |FS_{i,l}|),\ 1-p_{abs}) - 1$$

$$ub_i = \arg\min_x(1-\sqrt{1-p_{suc}} \leq binom.cdf(x,\ max(|FS_{i,e}|, |FS_{i,l}|),\ 1-p_{abs})$$

$$h_8:\ min_{FS_{i,r}} = ub_i - lb_i \quad i \in \{0, ..., 6\}$$

Appendix D - Soft Constraints

Deviation from Desired Coverage In addition to the hard constraints $h_1 - h_3$ we 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. We calculate the desired coverage by again using the *min_demand* algorithm:

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

To calculate the deviation penalty, we 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:

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

$$s_1 = \sum_{i=0}^{6} \sum_{j=1}^{55} cov\_pen_{i,j} + cov\_split\_pen_{i,j}$$

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):

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

Below Minimum Frame The hard constraints $h_{2_d} - h_{2_f}$ sometimes result in unsatisfiability. To prevent this outcome, we transform them into soft constraints and introduce a high penalty if the frame set counts fall below the minima. The function $below(h)$ returns the extent to which a hard constraint h is undershot:

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

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. We penalize the frame deviation between these weekdays:

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

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

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

Same Frames Next Week To achieve a more even distribution of time frames within a day off type, we 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, we simply count all occurrences.

Appendix E - Instance and Parameter Settings

Table 5 illustrates the main properties of the twelve instances, where the column $|Sp_i|$ stands for the number of split shifts in weekday i:

Inst.	$\|S_0\|$ - $\|S_3\|$	$\|S_4\|$	$\|S_5\|$	$\|S_6\|$	$\|Sp_0\|$ - $\|Sp_4\|$	$\|Sp_5\|$	$\|Sp_6\|$
1	223	223	142	123	29	4	0
2	185	185	123	111	27	6	0
3	204	204	139	123	23	9	0
4	233	233	153	138	37	7	0
5	408	408	265	234	56	10	0
6	437	437	292	261	60	16	0
7	217	219	132	125	22	0	0
8	183	183	110	109	21	3	0
9	194	194	134	120	15	0	0
10	218	218	148	137	23	0	0
11	400	402	242	234	43	3	0
12	412	412	282	257	38	0	0

Table 5: Instance Properties

For all instances, we use the following parameter settings and time frames:

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	10
w_3	10,000
w_4	1
w_5	100
w_6	100

Table 6: Parameter Settings

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 7: 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 7. 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.

Appendix F - Simulation Model

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:

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

We 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).

References

1. Borndörfer, R., Schulz, C., Seidl, S., Weider, S.: Integration of duty scheduling and rostering to increase driver satisfaction. Public Transport **9**, 177–191 (2017)
2. Burke, E.K., Causmaecker, P.D., Berghe, G.V., Landeghem, H.V.: The state of the art of nurse rostering. J. Sched. **7**(6), 441–499 (2004)
3. Dantzig, G.B.: A comment on edie's "traffic delays at toll booths". Journal of the Operations Research Society of America **2**(3), 339–341 (1954)
4. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European journal of operational research **153**(1), 3–27 (2004)
5. Frühwirth, L.: Exact Methods for the Time Frame Rostering Problem in the Context of Tram Driver Rostering. Master's thesis, Technische Universität Wien (2024)

6. Ge, L., Voß, S., Xie, L.: Robustness and disturbances in public transport. Public Transport **14**(1), 191–261 (2022)
7. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), https://www.gurobi.com
8. Heil, J., Hoffmann, K., Buscher, U.: Railway crew scheduling: Models, methods and applications. European journal of operational research **283**(2), 405–425 (2020)
9. Ingels, J., Maenhout, B.: The impact of reserve duties on the robustness of a personnel shift roster: An empirical investigation. Computers & Operations Research **61**, 153–169 (2015). https://doi.org/https://doi.org/10.1016/j.cor.2015.03.010, https://www.sciencedirect.com/science/article/pii/S0305054815000684
10. Kletzander, L., Musliu, N.: Solving the general employee scheduling problem. Computers & Operations Research **113**, 104794 (2020). https://doi.org/https://doi.org/10.1016/j.cor.2019.104794, https://www.sciencedirect.com/science/article/pii/S0305054819302369
11. Lau, H.C.: On the complexity of manpower shift scheduling. Computers & Operations Research **23**(1), 93–102 (1996)
12. Lin, D.Y., Tsai, M.R.: Integrated crew scheduling and roster problem for trainmasters of passenger railway transportation. IEEE Access **7**, 27362–27375 (2019). https://doi.org/10.1109/ACCESS.2019.2900028
13. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: International Conference on Principles and Practice of Constraint Programming. pp. 529–543. Springer (2007)
14. Perron, L., Didier, F.: Cp-sat v9.8 (2023), https://developers.google.com/optimization/cp/cp_solver
15. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**(3), 367–385 (2013). https://doi.org/https://doi.org/10.1016/j.ejor.2012.11.029, https://www.sciencedirect.com/science/article/pii/S0377221712008776
16. Wickert, T.I., Smet, P., Vanden Berghe, G.: Quantifying and enforcing robustness in staff rostering. Journal of Scheduling **24**(3), 347–366 (2021)
17. Wickert, T.I.: Personnel rostering: models and algorithms for scheduling, rescheduling and ensuring robustness. Doctoral thesis, Universidade Federl Do Rio Grande Do Sul and KU Leuven (2019)
18. Xie, L., Suhl, L.: A stochastic model for rota scheduling in public bus transport. In: Proceedings of 2nd Stochastic Modelling Techniques and Data Analysis International Conference. pp. 785–792 (2012)