TU WIEN Informatics

Why Is There No (Better) Solution?

Explainability Results for the Multi-Objective Rotating Workforce Scheduling Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Logic and Computation

eingereicht von

Esther Mugdan, BSc
Matrikelnummer 12306977

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Professor Dr. Nysret Musliu
Mitwirkung: Dr. Lucas Kletzander

Wien, 20. Februar 2025

Esther Mugdan	Nysret Musliu

TU WIEN Informatics

Why Is There No (Better) Solution?

Explainability Results for the Multi-Objective Rotating Workforce Scheduling Problem

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Logic and Computation

by

Esther Mugdan, BSc
Registration Number 12306977

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Professor Dr. Nysret Musliu
Assistance: Dr. Lucas Kletzander

Vienna, February 20, 2025

Esther Mugdan | Nysret Musliu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Esther Mugdan, BSc

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.

Wien, 20. Februar 2025

Esther Mugdan

Acknowledgements

First, I want to thank my supervisors, Nysret Musliu and Lucas Kletzander, for their support and valuable feedback and the members of the DBAI research group for their helpful inputs. A special thanks to my office mates, Ida Gjergji and Tommaso Mannelli Mazzoli, for taking the time to discuss my many questions.

I would like to express my sincere thanks to my family for supporting me and believing in me and to my father for proof reading this thesis. Lastly, I am deeply grateful to Andreas Mesaros for his unconditional love and support during my studies.

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

Kurzfassung

Scheduling ist ein wichtiger Aspekt in der industriellen Arbeit. Von der Zuweisung von Aufträgen an verschiedene Maschinen bis hin zu Schichtplänen für Mitarbeitern müssen effiziente Pläne erstellt werden, um Anforderungen abzudecken und Arbeitsvorschriften einzuhalten. Für kleine Unternehmen können solche Pläne zwar noch händisch erstellt werden, es ist jedoch schwierig, dabei den Überblick über alle Anforderungen nicht zu verlieren. Daher gibt es verschiedene Methoden, wie beispielsweise Constraint Programming und heuristische Suche, die diesen Prozess automatisieren können. Obwohl sie (optimale) Pläne erzeugen können, die den Problemspezifikationen entsprechen, kann es passieren, dass Interessenvertreter inkompatible Spezifikationen vorgeben oder dass sie mit dem resultierenden Plan nicht zufrieden sind und eine bessere Lösung verlangen. Sollte ein Programm keine Lösung finden, ist es wichtig die Gründe dafür anzugeben. Dadurch werden die Spezifikationen, die zur Unlösbarkeit beitragen, identifiziert und Lockerungen der Anforderungen vorgeschlagen, durch die das Problem lösbar wird. Auch wenn ein passender Plan für ein Problem gefunden wird, ist nicht garantiert, dass er den Vorstellungen der Interessenvertreter entspricht. Daher ist es wünschenswert, mehrere Pläne zu produzieren, die Vorteile für unterschiedliche Aspekte bieten. Dadurch ist es möglich, verschiedene Lösungen zu vergleichen und zu analysieren, welche Kompromisse gemacht werden müssen, um eine Lösung in einem bestimmten Aspekt zu verbessern.
In dieser Arbeit untersuchen wir das Rotating Workforce Scheduling Problem, für das wir zwei Ziele verfolgen: (1.) Die Entwicklung eines Frameworks, das Erklärungen für Instanzen mit inkompatiblen Spezifikationen generiert, und (2.) die Nutzung von heuristischen Methoden, um viele verschiedene Lösungen zu generieren und zu analysieren. Dadurch erhalten wir Einblicke in Synergien und nötige Kompromisse zwischen Spezifikationen. Wir implementieren beide Frameworks und evaluieren diese auf Benchmarks des Rotating Workforce Scheduling Problems. Wir zeigen wie Minimal Correction Sets verwendet werden können um Erklärungen für unlösbare Problem-Instanzen und Konfigurationen zu erzeugen. Zu diesem Zweck führen wir eine Fallstudie und Experimente durch die zeigen, dass Erklärungen effizient generiert werden können. Ausserdem untersuchen wir grosse Mengen an Lösungen, die wir mit unserem zweiten Ansatz generieren. Wir zeigen, dass diese Lösungsmengen effizient generiert und wie diese dargestellt werden können. Mittels der Lösungen können wir die Beziehungen zwischen Anforderungen unterschiedlicher Instanzen analysieren. Zuletzt evaluieren wir verschiedene Modifikationen des base-line Ansatzes und vergleichen diese mit dem aktuellen State-of-the-art.

Abstract

Scheduling is a highly relevant aspect of industrial work. From assigning jobs to machines in a time-saving manner to creating shift plans for employees in different fields of work; schedules have to be created to ensure efficiency, cover requirements and adhere to working regulations. While schedules for smaller-scale companies can still be designed manually, keeping track of all constraints and preferences is often a notoriously difficult job. Therefore, different methods that automatise this process, such as Constraint Programming and Heuristic Search, can be employed. However, while they are able to produce (optimal) plans in terms of the problem specification, stakeholders might demand incompatible specifications or not be content with a resulting plan, wanting a "better solution". If no solution can be found, it is important to explain which specifications contribute to infeasibility and how the problem can be relaxed to provide a solution. If a feasible plan can be provided, it is not necessarily accepted by all the stakeholders. Therefore, it can be advantageous to provide multiple solutions with qualities in different aspects. This allows us to compare different solutions and to analyse which compromises have to be made in order to further improve a solution in a certain aspect.
In this thesis we study the Rotating Workforce Scheduling Problem, for which our aim is two-fold: (1.) to develop a framework that generates explanations for instances with incompatible constraints and (2.) to use heuristic methods to provide and analyse multiple different solutions at once. This allows us to gain insights into conflicting constraints, synergies and necessary trade-offs between constraints. We implement frameworks for both aims and perform computational studies of their performances on RWS benchmarks. We show how Minimal Correction Sets can be used to provide explanations for infeasible instances and problem configurations. To do so, we perform a case study as well as experiments on (real-life) instances that reveal that explanations can be efficiently generated. Additionally, we study large sets of solutions generated by our second approach. We show that these sets of solutions can be generated efficiently and how they can be displayed. Using our resulting solutions we analyse the relationships between constraints for different instances of the problem. Lastly, we also compare and contrast modifications of the base-line approach and compare them to the current state of the art.

Contents

CHAPTER 1

Introduction

In many industries, shift schedules are used to regulate the working times of employees. When creating such shift plans, one usually has to take into consideration many constraints and preferences from different parties, resulting in a highly constrained problem.

In workforce scheduling, adhering to workplace regulations and reducing costs are often employers' primary interests. However, the resulting schedules provide the structure for a significant part of the employee's lives. Many different versions of such problems have been addressed over the years (Ernst et al. 2004; Van den Bergh et al. 2013; De Bruecker et al. 2015). While traditionally the focus has been on reducing cost, a growing body of work on employee well-being (Dall'Ora et al. 2016) shows the importance of incorporating these aspects. It is known that effects like fatigue due to bad workplace conditions impact not only economic costs of companies (Rosekind et al. 2010) but also lead to severe consequences in both psychological and physiological health of individuals (Lock, Bonetti, and Campbell 2018; Gärtner et al. 2018), including a higher risk for certain diseases and disorders (Moreno et al. 2019), and reduced social contacts with family and friends (Arlinghaus et al. 2019). Therefore, practical guidelines (Health and Safety Executive 2006) and metrics have been proposed to evaluate risk-related characteristics (Folkard and Lombardi 2006; Folkard, Robertson, and Spencer 2007). Petrovic, Parkin, and Wrigley 2021 presented case studies on personnel planning focusing on the impact of including well-being aspects in the objective function for heuristic optimisation approaches. Shifts are created by minimising a weighted sum of violations of undesired characteristics, like working weekends, long night shift stints, and others.

Scheduling problems with many constraints and optimisation goals are often solved using different methods or solvers. Many of these approaches have successfully been implemented in practice. However, two prominent problems can arise when solving the problem using classical approaches such as Constraint Programming or (Meta-)Heuristic Methods.

1. The method is unable to find a solution. There are two reasons why this can occur. Either the solver did not find a solution within the specified limits, or a solution does not exist, as the specified instance is over-constrained. In any case, when a stakeholder wants a solution to their problem instance, it is generally not enough to tell them that "there is no solution". Instead, it is important to explain why no solution could be found and how the problem can be resolved.

2. The approach yields a solution that matches the stakeholder's specifications. In this case, we expect the stakeholders to accept the solution. However, there is no guarantee that implicit preferences that are obvious to the stakeholder but not to the algorithm are met. Therefore, a stakeholder might ask for "a better solution". Hence, it can be helpful to provide multiple solution options and to explain trade-offs that have to be made when a solution is improved in a specific aspect.

Our aim in this thesis is to develop methods that can address these two problems for the Rotating Workforce Scheduling Problem (RWS). We implement and test our approaches considering six soft constraints that aim to improve schedules for employee well-being. We formalise the concrete goals of this thesis, which are twofold, as follows.

1. Develop a framework which can be used to solve and explain the Rotating Workforce Scheduling Problem under different configurations. For this purpose, we use Constraint Programming. The goal is to obtain a solution to a problem instance and its configuration. If the specified input is unsatisfiable, explanations should be provided which allow the user to understand the cause of infeasibility and resolve it. Using this framework, we can not only tell a stakeholder that there is no solution but also explain *why* this is the case and *how* it can be resolved.

2. Study the Rotating Workforce Scheduling Problem in a true multi-objective setting. We want to use and adapt Pareto Simulated Annealing to provide sets of non-dominated solutions. Using this, we want to study relationships between constraints and derive synergies and conflicts. This allows us to offer multiple solutions of equal quality to stakeholders instead of one solution that might not fit their implicit preferences. Additionally, we can predict what trade-offs must be made if a stakeholder wants to improve a solution in a specific dimension.

The main contributions of this thesis are the implementation of both approaches for the Rotating Workforce Scheduling Problem (RWS). Additionally, we perform case studies and computational evaluations on RWS benchmarks in order to analyse the performance and impact of our methods.

We test our Constraint Programming implementation using different solvers. We compare the individual results and find that the choice of the solvers for finding explanations in the form of Minimal Correction Sets (MCSs) is crucial. However, using the best solver

combinations, we were able to obtain positive results. We show that explanations in the form of MCSs can efficiently be found for unsatisfiable RWS instances under the satisfaction version of the problem. Additionally, we present how different preferences on constraints can be considered when providing explanations in the form of MCSs. We also consider feasible RWS instances with optimisation goals. Here, we test the feasibility of threshold combinations on the optimisation goals. We find that we can efficiently identify conflicting goals, especially for smaller instances. For larger instances, we do not always obtain optimal explanations; however, for most instances, we can get some explanation.

While an analysis of optimisation goals using MCSs does not yield precise results on their relationships, we can obtain them with our Pareto Simulated Annealing (PSA) framework. We provide a deep analysis of the trade-offs and synergies between optimisation goals on instances of different sizes. Additionally, we compare adaptions of the original PSA algorithm that improve our implementation for the RWS problem. Lastly, we compare our results with a state-of-the-art approach and show where the strengths of our algorithm lie.

This thesis is structured as follows. We give an overview of related work in Chapter 2. Next, we given the Problem Definition for the Rotating Workforce Scheduling Problem (RWS) in Chapter 3 followed by the description of the corresponding constraint model in Chapter 4. Our main contributions are explainability results in Chapter 5 and results for our multi-objective approach in Chapter 6. We conclude the thesis with a summary of our work as well as ideas for future work in Chapter 7.

CHAPTER 2

Related Work

We cover related work on the Rotating Workforce Scheduling as well as on the topics of explanability for scheduling and Pareto Simulated Annealing. We discuss general ideas, algorithms as well as implementations and studies that are relevant to the work we present. Since we cover related work in multiple fields, we present it in different sections.

2.1 Rotating Workforce Scheduling

The Rotating Workforce Scheduling (RWS) problem can be classified as a single-activity tour scheduling problem with non-overlapping shifts and rotation constraints (Baker 1976). Over the years, several approaches have been used to solve the problem, including use in commercial software for almost 25 years (Musliu, Gärtner, and Slany 2002). Initially, only the satisfaction version of the problem was studied, where only hard constraints need to be fulfilled. A complete method based on constraint programming (CP) was introduced by Musliu, Schutt, and Stuckey 2018 and further extended by Kletzander, Musliu, Gärtner, et al. 2019, in particular by introducing several optimisation goals used in practice, turning the satisfaction problem into an optimisation problem. The CP method was used in an instance space analysis by Kletzander, Musliu, and Smith-Miles 2021, leading to the creation of new instances to better cover the transition between feasible and infeasible instances. The current state-of-the-art approach is a branch&cut (b&c) framework by Becker, Schiffer, and Walther 2022, which reduces the runtime substantially compared to previous methods for the satisfaction version and a limited set of optimisation goals.

2.2 Explainability

Explainability for infeasible problems is an essential field of study, which is gaining prominence in many areas. Most notable are SAT-solving, Linear or Integer Programming,

and Constraint Programming. In general, two primary forms of Explainability are of interest, namely (a) finding a maximal part of the problem that is satisfiable, i.e. the MAX-SAT problem, or as in our case, the problem of finding Maximal Satisfiable Subsets (MSS). When dealing with MSSs, we often also provide the complement of an MSS, namely the Minimal Correction Set, which consists of the constraints that need to be removed in order to obtain an MSS. On the other hand, (b) we are often interested in the cause of infeasibility, i.e. finding the minimal unsatisfiable core or, in our case, Minimal Unsatisfiable Subset (MUS). For many fields, algorithms exist that return a single MSS or MUS. These often build on the general idea of growing (shrinking) mechanisms that start with a satisfiable (unsatisfiable) subset of the problem and test whether the addition (removal) of constraints still yields a satisfiable (unsatisfiable) subset. Doing these tests iteratively and adding (removing) the corresponding constraints eventually yields an MSS (MUS). However, few algorithms can efficiently enumerate all or at least more than one MSS/MUS.

2.2.1 Proposed Algorithms

there are tailored approaches for Linear Programming (Van Loon 1981; Gleeson and Ryan 1990) and Numerical Constraint Satisfaction Problems (Martínez Gasca et al. 2007), but they can not easily be generalised. An early general approach was CAMUS (Liffiton and Sakallah 2005; Liffiton and Sakallah 2008; Liffiton and Sakallah 2009), which makes use of the hitting set relation between MUSs and MCSs. In the first phase, all MCSs are computed, followed by the second phase where all minimal hitting sets of the set of MCSs are computed and returned, which correspond precisely to the MUSs. While this approach shows promising results in some cases, the number of MCSs can be exponential in the size of a problem instance, making the first phase of the algorithm intractable. Therefore, while CAMUS can enumerate all MUSs efficiently for some instances, it fails to report a single MUS on others. To circumvent this problem and report MUSs incrementally, the Dualize and Advance Algorithm (DAA, Bailey and Stuckey 2005) was developed. This algorithm also uses the hitting set relation between the constraint sets. However, the goal is to produce the first MUS faster and to use an incremental approach. To this end, MSSs are grown, and their complement (MCS) is added to the set of MCSs found. In the next step, the minimal hitting sets of the MCSs found so far are computed. If a hitting set is unsatisfiable, it is a MUS and given as output. Otherwise, the hitting set is the new seed from which the next MSS will be grown. For enumerating all MUSs, DAA performs considerably worse than CAMUS. Additionally, DAA often struggles with providing the first MUS since a lot of memory is needed in order to compute the intermediate hitting sets.

Since it is often not feasible to enumerate all MUSs and as CAMUS and DAA are often unable to return at least a single MUS, Liffiton and Malik 2013 focus on efficiently enumerating *some* MUSs. The authors propose the MARCO algorithm, which should provide the first MUS in roughly the same time t as the best algorithm designed for computing a single MUS. In addition, any following MUS should be returned as quickly as

possible (roughly after a period of t). MARCO stands for Mapping Regions of Constraint Sets, as the algorithm builds on a boolean formula *map* which simulates a mapping function $f(\mathcal{P}(C)) \mapsto \{0, 1\}$ which maps each subset of the constraints to 1 iff the subset has not yet been evaluated in terms of satisfiability. The algorithm starts with an empty *map* formula. In each iteration, a model of *map* is calculated, from which the set of constraints that are true in the model are extracted. If this set of constraints is satisfiable, it is grown to an MSS, which is returned. The *map* formula is combined with a clause over all constraints not in the MSS. (Intuitively: all subsets of the MSS are satisfiable hence to obtain a new MSS, at least one constraint not in the current MSS must be present) If this set of constraints is not satisfiable, it is shrunk to an MUS, which is returned. The *map* formula is combined with a clause over all negated constraints in the MUS. (Intuitively: all supersets of the MUS are unsatisfiable; hence, to obtain a new MUS, at least one constraint in the current MUS must not be present) The MARCO algorithm uses two solvers: one solver for growing and shrinking (i.e. repeatedly testing the satisfiability of constraint sets) and a second solver (*map*-solver) to generate models of the *map* formula. For the second solver, incremental solving techniques are advantageous. A computational study (Liffiton and Malik 2013) showed that compared to CAMUS and DAA, MARCO can provide at least one MUS/MSS in a timely manner. CAMUS can often find more MUSs on an instance. In comparison, MARCO can find MUSs on twice as many instances as CAMUS. It provides the first MUS considerably faster, making MARCO the best algorithm for finding (some) MUSs in a reasonable time.

2.2.2 Application to Scheduling Problems

To our knowledge, the use of Minimal Unsatisfiability Sets (MUS) as the main form of explanation for a concrete scheduling problem has not been studied yet. However, a general approach based on Abstract Argumentation has been proposed for the explanation of scheduling problems (Čyras, Letsios, et al. 2019). This work was extended by a concrete implementation called *Schedule Explainer* including a graphical user interface (GUI) which allows for interactive explanations (Čyras, Lee, and Letsios 2021). The system is designed for makespan scheduling and gives explanations for infeasibility, efficiency of a schedule, and for unmet fixed decisions which are provided by the user.

Agrawal, Yelamanchili, and Chien 2020 developed *Crosscheck*, an explainable scheduling tool tested for the Mars 2020 Rover Mission. Their work again includes a GUI where the schedule can be visualised. The problem studied consists of a set of activities, each with their own constraints that need to be scheduled. For activities which failed to be scheduled explanations are given by the tool. The explanations given indicate which constraints of the specific activity and which global constraints need to be relaxed in order to schedule the activity.

2.3 Constraint Programming

Our focus for using Explainability lies in the field of Constraint Programming (CP). It allows us to directly model the problem at hand as a set of constraints over decision variables. Therefore, in our case, a MUS, MCS or MSS is a subset of all constraints. Since many problems can be modelled and solved using Constraint Programming many different solvers are available. However, most of them adhere to their own modelling language, making comparisons between solvers cumbersome. In order to circumvent this problem, Nethercote et al. 2007 introduced MiniZinc as a standard CP modelling language that allows the integration of multiple different solvers. Leo and Tack 2017 introduced the FindMUS solver for MiniZinc which builds on the MARCO algorithm and provides MUSs, making the generation of explanations and debugging available in MiniZinc.

While MiniZinc is prominent for modelling Constraint Programs, we decided to use the CPMpy (Guns 2019) Python library to model our problem. In comparison to MiniZinc, the constraints can be directly modelled in Python, and the model as well as results can be directly processed further. Additionally, CPMpy allows the use of a range of different solvers and provides tools to directly generate explanations within the framework. In their workshop Bleukx, Guns, and Tsouros 2024 show how different forms of explanations can be provided for the Nurse Rostering Problem using CPMpy. The library provides an implementation of MARCO which can be used with the desired solvers. It allows for the generation of a single MUS/MCS/MSS and even supports finding an optimal MUS/MCS/MSS according to preferences given for the constraints.

2.4 Pareto Simulated Annealing

While other multi-objective methods such as NSGA-II (Deb et al. 2002) exist, we here focus only on Pareto Simulated Annealing. The Pareto Simulated Annealing Algorithm was originally proposed by Czyżak and Jaszkiewicz 1997. It extends the application of Simulated Annealing (SA, Kirkpatrick, Gelatt, and Vecchi 1983) to the multi-objective setting. SA is based on observations in statistical mechanics. An initial solution is modified iteratively, where modifications are accepted according to the metropolis criterion, which depends on a decreasing temperature such that with time, fewer worsening changes are accepted. For PSA, multiple initial solutions are run with Simulated Annealing, exchanging information on properties of the other solutions such that a large part of the search space is covered. In contrast to SA, Pareto Simulated Annealing yields a set of non-dominated solutions instead of a single solution. This is especially useful when multiple optimisation goals are of interest, which are traditionally combined in a weighted sum to obtain a single objective.

2.4.1 Implementation for Scheduling Problems

Pareto Simulated Annealing and variants of the algorithm has been used to solve many multi-objective scheduling problems, e.g. in work by Suresh and Mohanasundaram 2006; Duh and Brown 2007; Hamm, Beißert, and König 2009; Jarosław, Czesław, and Dominik 2013. Additionally, different variants of algorithms for multi-objective Simulated Annealing were discussed in a review by Amine 2019. We will discuss two specific applications of PSA to scheduling problems in detail. These are of interest as they propose modifications to PSA that we make use of. Drexl and Nikulin 2008 apply Pareto Simulated Annealing to the Airport Gate Assignment Problem, and they are the first to consider a multi-objective variant of the problem. The authors propose that PSA can be enhanced by restarting bad solutions, a concept that we apply in our implementation. In this case, solutions which are stuck in a part of the search space for a specific time are replaced by a fresh solution. Mischek and Musliu 2024 use Pareto Simulated Annealing in a decision support system for the Test Laboratory Scheduling Problem. For their implementation of PSA the authors enforce that no objective may be disregarded during the optimisation of the many constraints at hand. This is done by enforcing a minimum weight for each constraint in the weight vector guiding the search of the solution.

CHAPTER 3

Problem Definition

In the Rotating Workforce Scheduling (RWS) Problem, pre-defined shift types are considered. These have to be assigned to employees according to several constraints so as to form a complete schedule over multiple weeks. The assignment of shift types to certain employees and days must adhere to the required demand. Additional restrictions are given in the form of minimal and maximal lengths for sequences of shifts and limitations to consecutive shift assignments. In many applications, a rotating schedule, where each employee rotates through the same sequence of shifts with different offsets, is a preferred way of scheduling. We distinguish between hard and soft constraints, where hard constraints must be met and soft constraints are optional objectives that optimise the solution. We provide a formal definition of the problem and introduce new soft constraints that focus on employee well-being.

3.1 Formal Definition

For the general problem definition of RWS we build on definitions and notation that have been introduced previously (Musliu, Gärtner, and Slany 2002; Musliu, Schutt, and Stuckey 2018). An instance of the Rotating Workforce Scheduling problem is given by the following formulation.

n	Number of employees.
d	Length of the schedule. The total length of the planning period is $n \cdot d$, as each employee rotates through all n rows. We set $d = 7$, corresponding to the number of days in a week.
$\mathbf{A}$	Set of work shifts (activities), enumerated from 1 to m. We consider instances with 2 or 3 shifts where shift 3 is a night shift. A day off is denoted by a special activity O with numerical value 0, $\mathbf{A}^+ = \mathbf{A} \cup \{O\}$.

T	Temporal requirements matrix, describing the demand. An $m \times d$-matrix where each element $T_{i,j}$ corresponds to the number of employees that need to be assigned shift $i \in \mathbf{A}$ at day j.
ℓ_w, u_w	Minimal and maximal length of blocks of consecutive work shifts.
ℓ_a, u_a	Minimal and maximal lengths of blocks of consecutive assignments of shift a for each $a \in \mathbf{A}^+$.
$\mathbf{F}_2, \mathbf{F}_3$	Sequences of shifts of length 2 and 3 that are forbidden in the schedule (e.g. N D, a night shift followed by a day shift). This is typically required due to legal or safety concerns.

The task is to construct a cyclic schedule S, represented as an $n \times d$-matrix, where each $S_{i,j} \in \mathbf{A}^+$ denotes the shift or day off that employee i is assigned during day j in the first period of the cycle.

Example 3.1. Assume we have $n = 4$ employees and plan for a length of $d = 7$ days. A resulting schedule must have length $4 \cdot 7$. The set of shifts $\mathbf{A}$ consists of a day (D), afternoon (A) and night (N) shift. Days off are denoted by $-$. The demand T is specified such that one employee needs to be assigned each each shift for all days except Sundays where no afternoon shift (A) is required. We want employees to work $\ell_w = 5$ to $u_w = 7$ days in a row. Additionally, a block of work free days should be of length $\ell_- = 2$ to $u_- = 3$. Consecutive assignments of day and afternoon shifts should span over $\ell_D = \ell_A = 2$ to $u_D = u_A = 5$ days. Night shift spans on the other hand are limited to $\ell_N = 2$ to $u_N = 3$ days. We forbid day shifts after an afternoon shifts as well as day or afternoon shifts after night shifts ($\mathbf{F}_2, \mathbf{F}_3$). A sample schedule for this problem can be found in Table 3.1. This schedule is in use for a month (4 weeks). In the first week employee 1 covers the shifts from row 1, employee two the shifts from row 2 and so forth. In the following week all employees rotate to the next row, i.e. employee 1 covers the shifts from row 2, etc. and employee 4 covers the shifts from row 1. This rotation is performed until all employees have rotated through the whole schedule (4 weeks).

Empl.	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D	D	N	N	-
2	-	-	A	A	A	A	N
3	N	N	-	-	D	D	D
4	A	A	N	N	-	-	-

Table 3.1: Example of a rotating workforce schedule

3.2 Hard Constraints

In total we consider ten hard constraints that are defined as follows:

h^{overlap}	A day must have at most one shift $a \in \mathbf{A}^+$ assigned.
h^{minon}	A sequence of working days must span over at least ℓ_{w} days in a row.
h^{maxon}	A sequence of working days must span over at most u_{w} days in a row.
h^{minoff}	A sequence of off days must span over at least ℓ_O days in a row.
h^{maxoff}	A sequence of off days must span over at most u_O days in a row.
h^{minshift}	A sequence of working days consisting only of shift a must span over at least ℓ_a days, for each shift $a \in \mathbf{A}$.
h^{maxshift}	A sequence of working days consisting only of shift a must span over at most u_a days, for each shift $a \in \mathbf{A}$.
$h^{\text{forbidden}}$	Any sequences in $\mathbf{F}_2$ must not occur in the schedule S.
$h^{\text{forbidden3}}$	Any sequences in $\mathbf{F}_3$ must not occur in the schedule S.
h^{circular}	Any forbidden sequence must not span over the last and first day in the schedule S.

3.3 Soft Constraints

We also consider optimisation variants of RWS and extend the soft constraints introduced by Kletzander, Musliu, Gärtner, et al. 2019 with further objectives to be minimised. These aim to increase the well-being of employees and to meet applicable best practices of Britain's national regulator for workplace health and safety (HSE) shift work guidelines (Health and Safety Executive 2006). We address the following best practices.

BP1 *Where shifts are long (> 8 hours), for night shifts and for shifts with early morning starts, it may be better to set a limit of 2-3 consecutive shifts.*

BP2 *In general, limit consecutive working days to a maximum of 5-7 days and make sure there is adequate rest time between successive shifts.*

BP3 *Build regular free weekends into the shift schedule.*

Using these guidelines we formulate the soft constraints. We address BP1 by the constraint $s^{\mathrm{N}>3}$. BP2 is implemented by $s^{\ell_{\mathrm{dev}}}$. Lastly, BP3 is addressed by three constraints already introduced by Kletzander, Musliu, Gärtner, et al. 2019 namely $s^{\mathrm{ww}}, s^{\mathrm{d}_{\max}}$ and $s^{\mathrm{d}_{\mathrm{rms}}}$. Additionally we propose s^{Nww} to broaden the definition of free weekends. The resulting six soft constraints to be minimised are defined as follows.

$s^{\mathrm{N}>3}$	Minimise the number of consecutive night shifts exceeding 3.
$s^{\ell_{\mathrm{dev}}}$	Minimise the squared deviation of working sequence lengths to 5.
s^{ww}	Minimise the number of working weekends. A weekend is free if Saturday and Sunday are off. Otherwise, it is a working weekend.
$s^{\mathrm{d}_{\max}}$	Minimise the maximum distance $d_{\max}$ between consecutive free weekends ($d_{\max} = n + 1$ if no weekend is free).
$s^{\mathrm{d}_{\mathrm{rms}}}$	Minimise the root mean of squared distances between weekends $\sqrt{1/n \sum_{i=1}^{n} \hat{d}_i^2}$, where $\hat{d}_i$ is the distance minus 1 to the next free weekend if weekend i is free, and n otherwise.
s^{Nww}	Minimise the number of working weekends, where a weekend is only considered free if there is additionally no night shift on Friday.

3.4 Problem Solutions

A solution x to an RWS instance consists of a cyclic schedule S of dimensions $n \times d$ that assigns shifts from $\mathbf{A}^+$ to each employee and day.

Given an instance I of the RWS problem we can set the (potentially empty) set of soft constraints $\mathcal{S}$ to be considered. This configuration determines which soft constraints are optimised. Additionally, we have the option of using thresholds to set bounds on the desired soft constraints.

A solution x to a problem instance, given its configuration, is evaluated over the set of hard constraints $\mathcal{H}$ and its set of soft constraints $\mathcal{S}$ using the violation vector $v(x)$. With each constraint $c \in \mathcal{H} \cup \mathcal{S}$ we associate a violation $v_c(x)$ that indicates how often c is violated in the solution. Using this, we can define the following.

> **Definition 3.1** (Feasibility)**.** *Let x be a solution of an instance I with hard constraints $\mathcal{H}$. Then x is called feasible if $v_h(x) = 0$ for all $h \in \mathcal{H}$.*

Definition 3.2 (Optimalility)**.** *Let x be a solution of an instance I with soft constraints $\mathcal{S}$. Then x is called optimal if x is feasible and $\sum_{s\in\mathcal{S}} v_s(x)$ is minimal.*

When dealing with true multi-objectivity instead of a summed objective, optimality of a solution needs to be redefined. In this case, we consider Pareto-optimality. A solution x is Pareto-optimal if it is not dominated by any solution x' in the solution space, as defined below.

Definition 3.3 (Dominance)**.** *Let x be a solution of an instance I with soft constraints $\mathcal{S}$. Then x is called non-dominated if x is feasible and there is no (known) solution $x' \neq x$ of I st. $v_c(x') \leq v_c(x)$ for all $c \in \mathcal{H} \cup \mathcal{S}$ and $v_c(x') < v_c(x)$ for at least one $c \in \mathcal{H} \cup \mathcal{S}$.*

When working with thresholds on soft constraints, we additionally define a constant threshold vector $t(x)$ which must be taken into account. Thresholds can be defined for any soft constraint $s \in \mathcal{S}$ in the current configuration. For a solution x of an instance configuration with thresholds, we additionally define the following.

Definition 3.4 (Acceptability)**.** *Let x be a solution of an instance I with soft constraints $\mathcal{S}$ and thresholds t. Then x is called acceptable if x is feasible and $v_s(x) \leq t_s(x)$ for all $s \in \mathcal{S}$ where a threshold is set.*

CHAPTER 4

Constraint Modelling

The Rotating Workforce Scheduling problem (RWS) has been solved using a constraint model in MiniZinc[1] using only hard constraints (Musliu, Schutt, and Stuckey 2018) and some soft constraints (Kletzander, Musliu, Gärtner, et al. 2019). We build on the previous formulations and adapt them using the Python library CPMpy[2] from Guns 2019. We additionally implement further soft constraints that have not yet been considered. In the following, we give a high-level description of how the hard and soft constraints are implemented.

We represented a solution (i.e. a schedule) as a one-dimensional list $\mathtt{S}$ of length $n \cdot d$, where each entry $\mathtt{S}_i$ is an element of $\mathbf{A}^+$ for all $i \in \{1, \ldots, n \cdot d\}$. As the schedule is cyclic and workdays are always allowed following a day off, we can define the first entry $\mathtt{S}_1$ as a working day and the last entry $\mathtt{S}_{n \cdot d}$ as a day off. This allows for easier computation of subsequent constraints. However, we need to define an offset $o \in [0, d)$ that indicates at which position in $\mathtt{S}$ the solution schedule S starts. For the sake of simplicity, we omit offset and modulo calculations for the schedule when describing the constraints.

We distinguish three different types of constraints: definitions, hard bounds and soft bounds. Definitions amount to the way the constraints are defined and implemented. The hard and soft bound constraints are used to provide explanations for infeasible configurations.

For hard constraints we introduce decision variables representing the demand of workers per day and shift and minimal and maximal values for block lengths. These decision variables have restricted domains. For demand and minimal block lengths, the domain is $[0, \ldots, val]$, where val corresponds to the value of the data (i.e. T, $\ell_{\mathrm{w}}, \ell_0, \ell_1$ etc.) given in the input instance. For maximal block lengths, the domain is $[val, \ldots, n \cdot d]$, where val is again the value of the corresponding data input (i.e. u_{w}, u_0, u_1 etc.). To enforce that

[1] https://www.minizinc.org/
[2] https://github.com/CPMpy/cpmpy

the decision variables correspond exactly to the value they represent, we introduce what we call hard bounds. For each variable representing demand or minimal block lengths, we introduce up to 5 separate bounds as shown below.

$$\mathtt{v}_i \geq j \quad \forall j \in \{max(0, i-5), i\} \tag{4.1}$$

where i represents the demand of a day and shift or a minimal block length. Analogously, for maximal block lengths up to 5 separate bounds are introduced.

$$\mathtt{v}_i \leq j \quad \forall j \in \{i, min(i+5, n \cdot d)\} \tag{4.2}$$

where i represents a maximal block length. This allows us to give more precise information on how an infeasible configuration must be altered to regain feasibility, i.e we know by *how much* we need to relax a certain numerical input, not only *that* it needs to be altered.

The soft bounds correspond to the thresholds set in the configuration. For each soft constraint s we again use a decision variable $\mathtt{v}_s$ that represents its value. The soft bounds for constraints with specified thresholds then correspond to the following.

$$\mathtt{v}_s \leq t_s \tag{4.3}$$

where t_s is the threshold value set on soft constraint s.

To ensure that the demand is met by our schedule $\mathtt{S}$ we introduce the following constraints, that force the equality of the demand of each shift and day and the sum of corresponding entries in $\mathtt{S}$. The constraints are given below.

$$\forall i \in \{1, \ldots, d\}\Big(\forall a \in \mathbf{A}(\sum_{j=0}^{n-1} \mathtt{S}[7 * j + i] = T_{a,i})\Big) \tag{4.4}$$

We will now show how the hard and soft constraints are modelled with the corresponding decision variables introduced above.

4.1 Hard Constraints

For each of the ten hard constraints, we describe how we enforce its correctness and calculation.

The h^{overlap} constraint does not need to be modelled explicitly, as we represent the schedule using the list $\mathtt{S}$, where each entry is assigned a value from $\mathbf{A}^+$.

We model h^{minon} by enforcing for each first day i of a working block, that the following $\ell_{\text{w}} - 1$ days are working days as well. This amounts to the following constraint.

$$\forall i \in \{1, \ldots, n \cdot d\}\Big((\mathtt{S}[i] \in \mathbf{A} \wedge \mathtt{S}[i-1] \notin \mathbf{A}) \rightarrow \forall j \in \{0, \ldots, \ell_{\text{w}} - 1\}(j > \mathtt{v}_{\ell_{\text{w}}} \vee \mathtt{S}[i+j] \in \mathbf{A})\Big) \tag{4.5}$$

In a similar manner the h^{maxon} constraint can be modelled. We ensure for each first day i of a working block that at least one of the days between $i + \ell_{\text{w}}$ and $i + u_{\text{w}} - 1$ is not a working day.

$$\forall i \in \{1, \ldots, n \cdot d\}\Big((\mathtt{S}[i] \in \mathbf{A} \wedge \mathtt{S}[i-1] \notin \mathbf{A}) \rightarrow \exists j \in \{max(0, \ell_{\text{w}} - 5), \ldots, min(u_{\text{w}}, n \cdot d)\}(j \leq \mathtt{v}_{u_{\text{w}}} \wedge \mathtt{S}[i+j] \notin \mathbf{A})\Big) \quad (4.6)$$

The formulations for h^{minoff} and h^{maxoff} follow in the same manner as above.

$$\forall i \in \{1, \ldots, n \cdot d\}\Big((\mathtt{S}[i] = O \wedge \mathtt{S}[i-1] \neq O) \rightarrow \forall j \in \{0, \ldots, \ell_O - 1\}(j > \mathtt{v}_{\ell_O} \vee \mathtt{S}[i+j] = O)\Big) \quad (4.7)$$

$$\forall i \in \{1, \ldots, n \cdot d\}\Big((\mathtt{S}[i] = O \wedge \mathtt{S}[i-1] \neq O) \rightarrow \exists j \in \{max(0, \ell_O - 5), \ldots, min(u_O, n \cdot d)\}(j \leq \mathtt{v}_{u_O} \wedge \mathtt{S}[i+j] \neq O)\Big) \quad (4.8)$$

Following the same schema we can define h^{minshift} and h^{maxshift}.

$$\forall a \in \mathbf{A}\Big(\forall i \in \{1, \ldots, n \cdot d\}\Big((\mathtt{S}[i] = a \wedge \mathtt{S}[i-1] \neq a) \rightarrow \forall j \in \{0, \ldots, \ell_a - 1\}(j > \mathtt{v}_{\ell_a} \vee \mathtt{S}[i+j] = a)\Big)\Big) \quad (4.9)$$

$$\forall a \in \mathbf{A}\Big(\forall i \in \{1, \ldots, n \cdot d\}\Big((\mathtt{S}[i] = a \wedge \mathtt{S}[i-1] \neq a) \rightarrow \exists j \in \{max(0, \ell_a - 5), \ldots, min(u_a, n \cdot d)\}(j \leq \mathtt{v}_{u_a} \wedge \mathtt{S}[i+j] \neq a)\Big)\Big) \quad (4.10)$$

For $h^{\text{forbidden}}$ we internally model forbidden sequences in a way that allows us to access the set of all shifts that may not follow a shift s.

> **Example 4.1.** Let $\mathbf{F}_2 = \{\langle 2, 1\rangle \langle 3, 1\rangle \langle 3, 1\rangle\}$ then $\mathbf{F}_2[1] = \{\}$, $\mathbf{F}_2[2] = \{1\}$ and $\mathbf{F}_2[3] = \{1, 2\}$.

In general, no shift is forbidden after a day off. This allows us to model the $h^{\text{forbidden}}$ constraint as follows.

$$\forall a \in \mathbf{A}\Big(\forall i \in \{1, \ldots, n \cdot d\}\Big(\mathtt{S}[i] \neq a \vee \forall a' \in \mathbf{F}_2[a](\mathtt{S}[i+1] \neq a')\Big)\Big) \quad (4.11)$$

Note that in the implementation we only consider shifts a where $\mathbf{F}_2[a]$ is not empty.

Lastly, we model $h^{\text{forbidden3}}$ for which we consider each triple $f = (f_0, f_1, f_2)$ in $\mathbf{F}_3$ as follows.

$$\forall f \in \mathbf{F}_3 \Big(\forall i \in \{1, \dots, n \cdot d\} \big(\exists j \in \{0,1,2\} (\mathtt{S}[i+j] \neq f_j) \big) \Big) \tag{4.12}$$

4.2 Soft Constraints

We will now describe the implementation of the soft constraints. The soft constraints are not constraints in the classical sense but rather count the violations of certain properties within a solution. We, therefore, model the soft constraints as an equality constraint that sets the corresponding decision variable $\mathtt{v}_s$ equal to the number of violations of the constraint. For some computations, we need additional decision variables that allow us to count the violations. Since decision variables should preferably be restricted to a small domain, we try to find tight lower and upper bound calculations for their domains.

Example 4.2. We know that the number of working weekends must be at least 0 and can be at most the number of all weekends. However, this is a relatively loose bound, especially for larger instances. Therefore, taking the demand into account we are able to further tighten the lower bound to allow for more efficient modelling and solving.

For $s^{\text{N}_{>3}}$ we use the decision variable $\mathtt{v}_{\text{N}_{>3}}$ with lower bound 0 and an upper bound of

$$\left\lceil \frac{\sum_{i=1}^{d} T_{3,i}}{\ell_3} \right\rceil \cdot max(0, |u_3 - 3|) \tag{4.13}$$

As not all instances have night shifts, we distinguish two cases. If the instance has no night shifts we set $\mathtt{v}_{\text{N}_{>3}} = 0$. Otherwise we define the constraint as follows.

$$\mathtt{v}_{\text{N}_{>3}} = \sum_{i=1}^{n \cdot d} \mathbf{1}_{\forall j \in \{0,\dots,3\} (\mathtt{S}[i-j]=3)} \tag{4.14}$$

Here $\mathbf{1}$ is an indicator variable, i.e. its value is 1 if $\forall j \in \{0, \dots, 3\} (\mathtt{S}[i-j] = 3)$ is true (evaluates to 1) and 0 otherwise. In our implementation this does not need to be modelled explicitly as boolean expressions can directly be treated as 0 or 1 valued integers.

The $s^{\ell_{\text{dev}}}$ constraint can not be modelled directly. We therefore introduce an additional decision variable $\mathtt{S}_{on}$ which is a list of the same dimensions as the schedule $\mathtt{S}$ that allows us to count the lengths of working sequences.

Example 4.3. For a schedule $\mathtt{S} = [1, 1, 2, 3, 3, 0, 0, 1, 1, 0]$ the corresponding sequence $\mathtt{S}_{on}$ is defined as $[1, 2, 3, 4, 5, 0, 0, 1, 2, 0]$, i.e. for each day i that corresponds to a working day $\mathtt{S}_{on}[i] = n$ indicates that day i is the n^{th} working day of the current sequence of working days.

We model $\mathtt{S}_{on}$ as a decision variable list of length $n \cdot d$ that can take values from 0 to u_{w}. As we fixed the first day of $\mathtt{S}$ to be a working day and the last day to be a day off we can safely set $\mathtt{S}_{on}[1] = 1$. For the remaining entries we make use of the following constraint.

$$\mathtt{S}_{on}[i] = \begin{cases} 0 & \text{if } \mathtt{S}[i] = 0 \\ \mathtt{S}_{on}[i-1] + 1 & \text{else} \end{cases} \qquad \forall i \in \{2, \ldots, n \cdot d\} \tag{4.15}$$

Using $\mathtt{S}_{on}$ we can now model the $s^{\ell_{\mathrm{dev}}}$ constraint. We use the decision variable $\mathtt{v}_{\ell_{\mathrm{dev}}}$ with lower bound 0 and an upper bound given by

$$\left(\left\lceil \frac{\sum_{a \in \mathbf{A}} \sum_{i=1}^{d} T_{a,i}}{\ell_{\mathrm{w}}} \right\rceil \cdot max(|5 - \ell_{\mathrm{w}}|, |5 - u_{\mathrm{w}}|) \right)^2$$

We model the constraint as an equality, again using an indicator variable $\mathbf{1}$.

$$\mathtt{v}_{\ell_{dev}} = \sum_{i=1}^{n \cdot d - 1} \mathbf{1}_{\mathtt{S}_{on}[i] \neq 0 \wedge \mathtt{S}_{on}[i+1] = 0} \cdot (5 - \mathtt{S}_{on}[i])^2 \tag{4.16}$$

In order to model the soft constraints involving working weekends we make use of a decision variable that keeps track of free weekends. We introduce the boolean variable list $\mathtt{W}_{\mathrm{off}}$ of length n that indicates for each weekend if it is free. To model this we use the following constraint.

$$\mathtt{W}_{\mathrm{off}}[i] = (\mathtt{S}[7 * i - 1] = 0 \wedge \mathtt{S}[7 * i] = 0) \qquad \forall i \in \{1, \ldots, n\} \tag{4.17}$$

We can now easily model the s^{ww} constraint using the decision variable $\mathtt{v}_{\mathrm{ww}}$ which has a lower bound of $max(\sum_{a \in \mathbf{A}} T_{a,6}, \sum_{a \in \mathbf{A}} T_{a,7})$ and an upper bound of n as follows.

$$\mathtt{v}_{\mathrm{ww}} = n - \sum_{i=1}^{n} \mathbf{1}_{\mathtt{W}_{\mathrm{off}}[i]} \tag{4.18}$$

For $s^{\mathrm{N_{ww}}}$ we extend the notion of working weekends to the case where Friday night shifts are considered too. For instances without night shifts we can set $s^{\mathrm{N_{ww}}}$ equal to s^{ww}. For all other instances, we can use information from $\mathtt{W}_{\mathrm{off}}$ to define a new boolean variable list $\mathtt{W}_{\mathrm{N_{off}}}$ that indicates whether a weekend is free and has no night shifts on Friday. The entries of $\mathtt{W}_{\mathrm{N_{off}}}$ are defined by the following constraint.

$$\mathtt{W}_{\mathrm{N_{off}}}[i] = (\mathtt{S}[7 * i - 2] \neq 3 \wedge \mathtt{W}_{\mathrm{off}}[i]) \qquad \forall i \in \{1, \ldots, n\} \tag{4.19}$$

Using this we can now model $s^{\mathrm{N_{ww}}}$ with the decision variable $\mathrm{v_{N_{ww}}}$ that has an upper bound of n and a lower bound given by

$$max(\sum_{a\in\mathbf{A}} T_{a,6} - b, \sum_{a\in\mathbf{A}} T_{a,7} - \mathbf{1}_{\ell_O>1}\cdot b, T_{3,5}) \tag{4.20}$$

where $b = max(T_{3,5} - \sum_{a\in\mathbf{A}\setminus\mathbf{F}_2[3]} T_{a,6}, 0)$ indicates the number of free Saturdays that are "blocked" by a Friday with a night shift. Using this additional variable we are able to further tighten the lower bound compared to the general working weekends constraint. We can now model $h^{\mathrm{N_{ww}}}$ analogously to h^{ww} as follows.

$$\mathrm{v_{N_{ww}}} = n - \sum_{i=1}^{n} \mathbf{1}_{\mathrm{W_{N_{off}}}[i]} \tag{4.21}$$

For the soft constraints concerned with the distance between free weekends, we introduce a new decision variable. We use the decision variable list $\mathrm{W_{dist}}$ that indicates for each weekend the distance to the next free weekend. To handle the case where no weekend is free we use the following constraint.

$$(0 = \sum_{i=1}^{n} \mathrm{W_{off}}[i]) \rightarrow \mathrm{W_{dist}}[n] = 1 \tag{4.22}$$

We instantiate all other entries with constraints as follows.

$$\mathrm{W_{dist}}[i] = \begin{cases} 0 & \text{if } \mathrm{W_{off}}[i+1] \\ \mathrm{W_{dist}}[i+1]+1 & \text{else} \end{cases} \qquad \forall i \in \{1,\dots,n\} \tag{4.23}$$

Using this we can model $s^{\mathrm{d_{max}}}$ using the corresponding decision variable $\mathrm{v_{d_{max}}}$. The upper bound for $\mathrm{v_{d_{max}}}$ is $n+1$ while the lower bound is

$$\left\lceil \frac{n-1}{n - max(\sum_{a\in\mathbf{A}} T_{a,6}, \sum_{a\in\mathbf{A}} T_{a,7})} \right\rceil \tag{4.24}$$

This allows us to model the equality for the $s^{\mathrm{d_{max}}}$ constraint.

$$\mathrm{v_{d_{max}}} = max(\mathrm{W_{dist}}) + 1 \tag{4.25}$$

For the last constraint, $s^{\mathrm{d_{rms}}}$, we again need an auxiliary list variable. We introduce $\mathrm{W_{R_{dist}}}$ that stores the values $\hat{d}_i$ used to define $s^{\mathrm{d_{rms}}}$. In order to define $\mathrm{W_{R_{dist}}}$ we make use of $\mathrm{W_{dist}}$ as follows.

$$\mathrm{W_{R_{dist}}}[i] = \begin{cases} \mathrm{W_{dist}} & \text{if } \mathrm{W_{off}}[i+1] \\ n & \text{else} \end{cases} \qquad \forall i \in \{1,\dots,n\} \tag{4.26}$$

We can now use $v_{\mathrm{d_{rms}}}$ to model the $s^{\mathrm{d_{rms}}}$ constraint. The variable $v_{\mathrm{d_{rms}}}$ has an upper bound of n^3 and a more intricate lower bound

$$n^2\cdot(n-w_{max}) + \left(\left\lfloor\frac{n}{w_{max}}\right\rfloor - 1\right)^2 \cdot \left(w_{max} - (n \,\%\, w_{max})\right) + \left\lfloor\frac{n}{w_{max}}\right\rfloor^2 \cdot (n \,\%\, w_{max}) \tag{4.27}$$

where $w_{max} = n - max(\sum_{a \in \mathbf{A}} T_{a,6}, \sum_{a \in \mathbf{A}} T_{a,7})$ is the maximal number of free weekends. We can now define the equality that models the $s^{\mathrm{d_{rms}}}$ given below.

$$\mathtt{v}_{\mathrm{d_{rms}}} = \sqrt{\frac{1}{n} \sum \mathtt{W}_{\mathrm{R_{dist}}}[i]^2} \tag{4.28}$$

Since many solvers do not support floating point operations we only consider the sum of squared distances within the model and evaluate the rooted mean outside of the model.

CHAPTER 5

Explaining Infeasibility

When employing algorithms to solve problems, it can happen that the algorithm cannot come up with a solution. This can be due to a lack of resources (e.g. timeout, memory overflow) or incorrect implementations. We are, however, interested in the case where algorithms cannot solve a problem because it is infeasible. Some algorithms can prove the infeasibility of a problem instance. However, even if we know that a problem is not solvable, we do not yet know why this is the case. Especially when working with stakeholders, simply stating that "there is no solution" is not enough. Therefore we are interested in *explaining* infeasibility. We can do so by providing explanations in different formats that isolate the reasons why a problem cannot be solved.

We will focus on three main explanation components, that allow us to understand what causes infeasibility, which part of the specified problem is solvable and what needs to be adapted to regain feasibility.

To do so, we implemented a Constraint Programming approach to the Rotating Workforce Scheduling Problem (RWS). Using our implementation we can provide explanations for infeasible configurations of RWS instances.

5.1 Explanations using Constraint Sets

We distinguish three types of Constraint Sets that provide explanations for infeasibility. Given a problem that is defined over a set of constraints $\mathcal{C}$, we define the following.

Definition 5.1 (MUS)**.** *Let $M \subseteq \mathcal{C}$. Then M is a Minimal Unsatisfiable Set (MUS) if M is unsatisfiable and for each $c \in M$ it holds that $M \setminus \{c\}$ is satisfiable.*

Minimal Unsatisfiability Sets help to determine where a conflict lies within a given set of constraints. Using MUSs we can explain which constraints cannot be satisfied in combination.

> **Definition 5.2** (MSS). *Let $M \subseteq \mathcal{C}$. Then M is a Maximal Satisfiable Set (MSS) if M is satisfiable and for each $c \in \mathcal{C} \setminus M$ it holds that $M \cup \{c\}$ is unsatisfiable.*

Maximal Satisfiability Sets depict the subset-maximal set of constraints which are feasible. Using MSSs we can explain which parts of the problem are not conflicting and do not need to be modified to regain feasibility.

> **Definition 5.3** (MCS). *Let $M \subseteq \mathcal{C}$. Then M is a Minimal Correction Set (MCS) if $M = \mathcal{C} \setminus M'$, where M' is a MSS.*

Minimal Correction Sets give us the set of constraints that need to be removed from $\mathcal{C}$ in order to restore feasibility. Removing only parts of an MCS from $\mathcal{C}$ will not restore feasibility. Using MCSs we can directly provide information on how the problem must be modified and relaxed to be solvable. The resulting problem from such a modification is an MSS of the original set of constraints.

The relationships between the set of all constraints (unsatisfiable problem) and the different constraint sets that we introduced are depicted in Figure 5.1. We see the hitting-set relation of the MUS and MCS as well as the complement relation of the MSS and MCS. Together the MSS and MCS form the set of all constraints, i.e. the unsatisfiable problem.

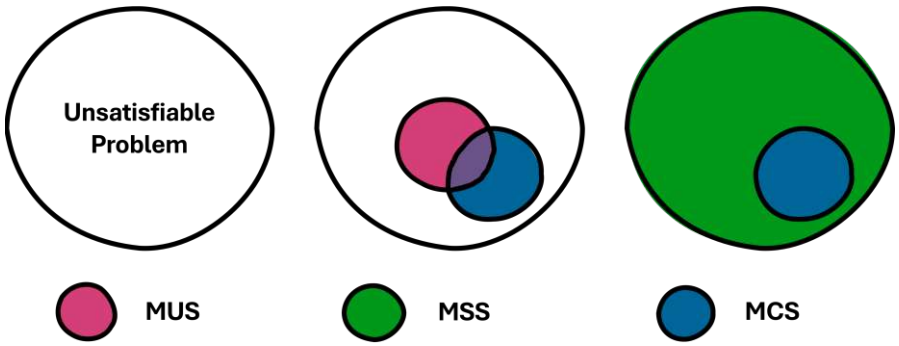


Figure 5.1: Relationship between unsatisfiable problem, MUS, MSS and MCS.

Using the above definitions we are able to give different kinds of explanations that allow stakeholders to understand what causes infeasibility and how feasibility can be restored. Even though all three forms of explanations can be useful, we focus on MCSs in our work since MCSs allow us to give explicit directions to stakeholders on which changes need to be made to their problem configuration in order to find a solution.

5.2 Explanations for Satisfaction Configurations

We used FairSubset (Ortell, Switonski, and Delaney 2019) in the online tool provided[1] to extract a representative set of 50 instances from a total of 1843 infeasible instances. These benchmarks were introduced by Kletzander, Musliu, and Smith-Miles 2021. For these instances, infeasibility was shown using MiniZinc. Some instances were proven infeasible later using branch&cut. As a case study, we additionally constructed a simple infeasible instance based on a feasible real-world instance.

5.2.1 Case Study

We present a case study for a small artificial instance in order to show how Minimal Correction Sets (MCS) can be retrieved for problems with incompatible hard constraint configurations. We define the instance as follows, where the demand T is given in 5.1.

$n = 4$ $\quad [\ell_0, u_0] = [1, 4]$ $\quad \mathbf{F}_2 = \{\langle 2, 1\rangle, \langle 3, 1\rangle, \langle 3, 2\rangle\}$

$d = 7$ $\quad [\ell_1, u_1] = [2, 7]$ $\quad \mathbf{F}_3 = \{\}$

$\mathbf{A} = \{1, 2, 3\}$ $\quad [\ell_2, u_2] = [2, 7]$

$[\ell_w, u_w] = [4, 7]$ $\quad [\ell_3, u_3] = [1, 3]$

Shift	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	1	1	1	1	2	2
2	1	1	1	1	1	1	1
3	1	1	1	1	2	0	0

Table 5.1: Daily demand T for $instance_0$ over all three shift types.

While the demand never exceeds the total amount of workers, and the remaining parameters are reasonably set, this instance is not feasible. Our framework using ortools (Perron and Furnon 2024) as a general solver can detect infeasibility within roughly 0.1 seconds. Using MARCO with the z3 solver (De Moura and Bjørner 2008) as general solver and ortools as a map solver we can enumerate all 462 Minimal Correction Sets (MCS) within roughly 2.5 minutes. The sizes of the MCSs vary from 2 constraints up to 11.

To restore feasibility of the instance, we need to choose a singular MCS and remove its constraints from the problem specification. Since we are faced with a large number of MCSs, choosing one is not straight-forward. Choosing an MCS should be done according to the preferences of stakeholders. While some might be interested in the smallest amount of changes, others may want to keep the demand intact. Using such preferences, we can use weights to give priorities to different constraints. Given a total weight function

[1] https://delaney.shinyapps.io/FairSubset/

$w : \mathcal{C} \mapsto \mathcal{N}$ that maps every constraint to a natural number we can find optimal MCSs. An MCS is optimal if the sum of the weights of its constraints is minimal.

Looking at our case study, we can apply different weight functions to express preferences over the constraints. Some examples of preferences and weighted functions are given below.

Example 5.1 (Minimal Size)**.** Assume we are interested in a minimal-sized MCS. We can employ $w(c) = 1\ \forall c \in \mathcal{C}$ to equally weight all constraints. An example of a resulting MCS is $\{T_{3,Fri} \geq 2,\ u_{\mathrm{w}} \leq 7\}$. This means that by reducing the demand on Fridays for shift 3 from 2 to only 1 shift and only requiring an upper bound of 8 on working sequences we can restore feasibility.

Example 5.2 (Only Demand)**.** Assume we are interested in a smallest MCS where only the demand should be changed. We can use the weight function that assigns $w(c) = 1\ \forall c \in \mathcal{C}_{\text{demand}}$ and $w(c) = 10$ for all other constraints. An example of a resulting MCS is $\{T_{1,Mon} \geq 2, T_{3,Fri} \geq 2\}$. This means that by reducing the demand on Fridays for shift 3 and on Mondays for shift 1 from 2 to only 1 shift we can restore feasibility.

Example 5.3 (No Demand)**.** Assume we are interested in a smallest MCS where the demand should not be changed. We can use the weight function that assigns $w(c) = 10\ \forall c \in \mathcal{C}_{\text{demand}}$ and $w(c) = 1$ for all other constraints. An example of a resulting MCS is $\{\ell_0 \geq 2, T_{3,Fri} \geq 2\}$. We see that the resulting MCS still contains a constraint regarding the demand. Even for increased weights for demand constraints we always get a MCS where a demand constraint is mentioned. This means that we can not restore feasibility unless the demand is changed.

5.2.2 Computational Study

We tested our framework on 50 infeasible instances by comparing approaches using different solvers. We evaluated results from MARCO, which generates all Minimal Correction Sets (MCS), as well as results for providing optimal MCSs.

We performed our experiments on a cluster with Ubuntu 22.04.2 LTS with 2× Intel Xeon CPU E5-2650 v4 (2.2GHz, 12 physical cores, no hyperthreading). We ran our experiments using Python 3.11. We employed the solvers ortools (v9.11, Perron and Furnon 2024), z3 (v4.13, De Moura and Bjørner 2008), Exact (v2.1, Elffers and Nordström 2018) and pysat (v1.8, Ignatiev, Morgado, and Marques-Silva 2018).

Each run was given a 1h timeout to prove the unsatisfiability of the instance using ortools. We additionally used a 1h timeout for providing explanations. Employing a timeout

on MARCO and finding an MCS might result in suboptimal MCSs, meaning that the resulting set of constraints is a Correction Set, but not necessarily subset minimal. If an MCS was provided before the timeout, minimality is guaranteed. Each run additionally had a memory limit of 16'384 megabytes.

solver	failed proof	memory limit	MCS found	#MCS found
z3-ortools	24%	0%	**20%**	**100**
z3-pysat	24%	8%	18%	51
Exact-ortools	24%	0%	16%	62
Exact-pysat	24%	0%	16%	64

Table 5.2: Comparison of running MARCO with the respective configuration. Percentages of instances in relation to all 50 instances are given for failed proofs of infeasibility, memory limit exhaustion and cases where MCSs could be found. Additionally, the total number of MCSs found is indicated.

We first evaluated MARCO using z3 or Exact as primary solvers and ortools or pysat as map solvers. This yields a total of four solver combinations. Full results can be found in the Appendix in Tables 1, 2, 3, and 4. A summary of our findings is depicted in Table 5.2. We observe that 24% (12 out of 50) of instances can not be proven unsatisfiable within the one hour time limit. Even if no proof of unsatisfiability was be found, we performed the MARCO algorithm. However, no solver combination could provide MCSs using MARCO for these instances within the time limit. For the combination z3-pysat we observe that the runs for 4 instances ran out of memory. No other solver combination had this issue. When we compare the percentage of instances for which MCSs could be found, the configuration z3-ortools was able to find MCSs for a total of 10 out of 50 instances. In total, using this configuration yielded 100 MCSs over all instances. For 7 out of the 10 instances z3-ortools found at least as many MCSs as all other solvers. Additionally, only one instance (instance 1798) that was solved by another solver (z3-pysat) was not solved by z3-ortools. When comparing the time MARCO used on the instance for both algorithms, we find that z3-pysat ran 2 minutes longer, while z3-ortools aborted at the 60 minute mark. In comparison to z3-ortools, Exact performed worse with two instances less for both ortools and pysat. However, while the Exact approaches could find MCSs for fewer instances than z3-pysat, Exact was able to provide more MCSs in total. Therefore, it is hard to compare these remaining three configurations especially because each configuration was able to solve at least one instance that none of the other two configurations could solve. Additionally, for some instances the number of MCSs found varied highly between the approaches. For example, using Exact-ortools provided 17 MCSs on instance 83 while none of the other two approaches could find any MCS. Using Exact-pysat yielded 27 MCSs on instance 456 while Exact-ortools could not find any MCS and z3-pysat only 3. Lastly, for instance 5720, Exact-pysat could find only 3 MCSs, Exact-ortools 9 and z3-pysat 19 MCSs. These examples show that while the configurations performed worse that z3-ortools, they are hard to compare with each other.

	solver	MCS found	mean size	stdv size	mean time	stdv time
base	ortools	**100%**	**1.86**	1.23	1933.88	1581.96
	Exact	18%	2.78	1.40	3162.28	1427.61
minimal	ortools	**100%**	1.88	1.24	**1897.67**	1590.07
	Exact	18%	2.78	1.48	3172.28	1434.20
no demand	ortools	**100%**	1.92	1.38	1983.00	1610.62
	Exact	18%	3.11	1.85	3168.04	1423.33

Table 5.3: Comparison for finding any MCS (base) minimal-sized MCS (minimal) or MCS without demand (no demand) within 1h using ortools or Exact. Percentage of instances for which an MCS was found, mean size of MCS and run times are indicated.

We could not enumerate MCSs using MARCO for a majority of the instances. However, we are usually only interested in a single MCS and do not need to enumerate all of them especially as this is computationally more expensive. We, therefore, focus on finding a single MCS using different weight configurations as introduced in the case study. The full results of the runs can be found in the Appendix in Tables 5, 6, and 7. A summary of the results can be found in Table 5.3. We compare results for finding any MCS (base), i.e. no weight function, minimal MCS (minimal), i.e. $w(c) = 1\ \forall c$, and minimal MCS without demand (no demand), for which we set a weight of 100 on demand constraints and a weight of 1 on all remaining hard bounds. We observe that ortools is able to find an MCS for each instance over all configurations, even for the 24% of instances where no proof of infeasibility could initially be generated. In contrast, Exact only finds an MCS for 9 out of 50 instances for each configuration. We also observe that the mean time used by ortools is roughly 30 minutes, which is half of the 1h timeout. In fact, only 40% of instances ran for the whole hour, indicating that the optimal MCS could be found for 60% of the instances. Additionally, for half of the instances that ran for the whole hour an MCS of size one could be found, indicating that they must also be subset optimal. In comparison, Exact could only solve 18% of the instances, which were all solved before the timeout of one hour. However, compared to ortools, Exact took roughly 10 times more time to provide an MCS. In the best case, Exact was about 4 times slower and in the worst case around 25 times slower than ortools. This clearly shows that ortools is not only able to provide MCSs for all instances but is also considerably faster than Exact.

We compare the sizes of MCSs found by ortools under the base, minimal or no demand configuration in Figure 5.2. We observe that there is no big difference in the sizes of MCS. However, when comparing the base and minimal configurations, we observe that the sizes of MCSs are the same for each instance except instance 3966. For this instance, an MCS of size 3 was found for the minimal configuration, but a true subset of size 2 for the base and no demand case. For all configurations, the instance was run until a timeout, indicating that under the minimal configuration a non-minimal correction set was returned. For the no demand case, we observe that for two instances an MCS of size 6 was provided (instead of sizes 4 and 5). In both cases, these were found in about 10-15 seconds, indicating their optimality. When comparing with the MCS found

for the base and minimal case, we find that the MCSs contained a demand constraint. Additionally, MCSs for four other instances contained demand bounds for the base and minimal configuration. In each case, a MCS of the same size without demand bounds could be found.

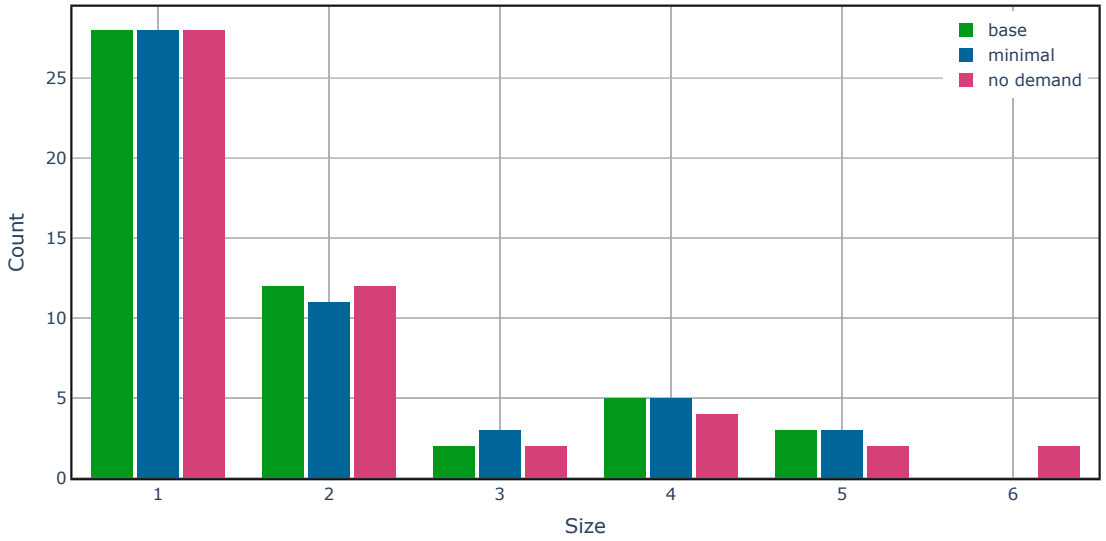


Figure 5.2: Histogram comparing sizes of MCS found using ortools for base, minimal and no demand MCS.

Lastly, we want to analyse if there is a relation between the MCS sizes and the coordinates of the instances within the x-y plane that was introduced by Kletzander, Musliu, and Smith-Miles 2021. We depict the sizes of MCS found by ortools and Exact for the base case mapped to the x-y space in Figure 5.3. As a reference all infeasible instances are depicted in the back. Here darker areas indicate a higher density of instances. The space is configured such that instances on the upper-left (inner) border are close to feasible instances, the centre area consists of hard infeasible instances and the lower-right (outer) border consists of instances which are clearly infeasible. We observe that Exact is not able to find MCSs for the hard centre area but mostly for clearly infeasible instances. These instances tend to have very large MCSs which is consistent with the fact that they are clearly infeasible. For the upper left area, which consists of instances that are very similar to feasible instances, MCSs are rather small. This is coherent, as only a few changes are necessary to gain a feasible instance. For the centre area consisting of hard infeasible instances we observe MCSs of differing sizes.

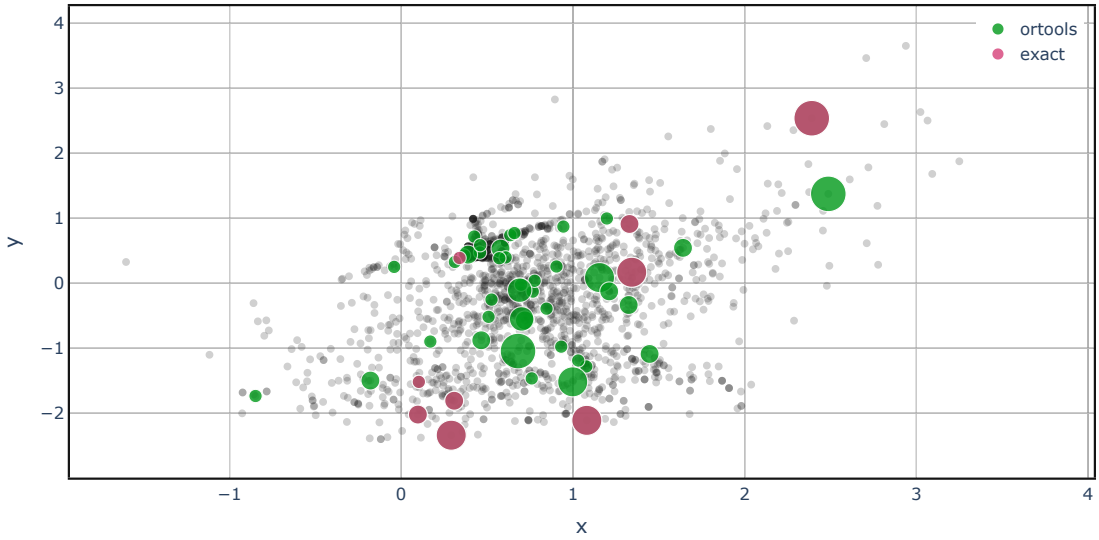


Figure 5.3: Visualisation of MCS sizes for ortools and Exact in infeasible x-y space for base MCS.

5.3 Explanations for Optimisation Configurations

We now want to analyse feasibility of configurations using thresholds on soft constraints for feasible instances. We want to test whether threshold combinations still allow for acceptable solutions. If not, we provide MCSs as explanations for infeasibility. To this end, we consider 20 feasible real-life instances of different sizes.

In order to determine suitable thresholds for each dimension we first determined the $opt(s)$ and $avg(s)$ value for each soft constraint s and instance i. The $opt(s)$ value corresponds to the expected violation of s for instance i if s is the only optimisation goal. On the other hand, $avg(s)$ corresponds to the expected violation of s if the sum of all soft constraint violations is set as the optimisation goal. We determined these values by running each of the 20 instances once optimising the sum of all soft constraints and once for each of the soft constraints, considering only a single soft constraint as the optimisation goal. We performed 5 runs per configuration using ortools. The best of five *opt* and *avg* values can be found in the Appendix in Tables 8 and 9. For the *avg* values, we took results from the runs with the least summed objective. If optimality of the objective could be proven, we show the values in green, and if optimality could not be proven, the corresponding values are shown in red.

We compared our optimal values *opt* with objective values obtained by Kletzander, Musliu, Gärtner, et al. 2019 for the same benchmark set. The authors use a MiniZinc model of the problem and solve the instances with the Chuffed solver (Chu et al. 2018), also using a timeout of one hour. We compared the objective values of the four soft constraints that are used by the authors as well. For the working weekends constraint s^{ww}, we could

prove optimal objectives for 5 additional instances and found a better objective value for instance 11, which is the only instance where our approach could not prove optimality. Similarly, for working weekends including Friday night $s^{\mathrm{N_{ww}}}$, we proved optimality for 4 additional instances and again found a better objective for instance 11, which was again the only instance where we could not prove optimality. For the maximal weekend distance $s^{\mathrm{d_{max}}}$, our approach could not prove optimality for instance 11, while Kletzander et al. could. However, we obtained the same objective value, indicating that the optimal objective was found but optimality could not be proven within the time limits. While we could not prove optimality for any additional instances, we were able to find solutions within the time limit for instances 15 and 20, which Chuffed failed to do. Lastly, for $s^{\mathrm{d_{rms}}}$, we could also not prove optimality on any new instances. However, we found better objective values for three instances and were again able to find a solution within the time limit for instance 15, which Chuffed was not able to do.

Using the *opt* and *avg* values we can implement thresholds of the following form: *Reduce constraint s_1 by x% compared to the average value and increase constraint s_2 by at most 10%* . This is implemented as follows. Given a factor $f(s) \in [-1, 1]$ by which the value of the soft constraint s should be decreased or increased, we can calculate the threshold $t(s)$ using the following.

$$t(s) = max\Big(opt(s), avg(s) + \lceil d(s) \cdot |f(s)| \rceil \cdot sgn(f(s))\Big)$$

where $sgn(\ )$ is the sign function and $d(s)$ is the distance between $avg(s)$ and $opt(s)$ (and 1 if $avg(s) = opt(s)$). We decided on this formulation for rounding for multiple reasons. First of all, rounding is necessary as we deal with integer valued soft constrains. Secondly, traditional rounding always rounds up/down in the same direction for positive and negative numbers, i.e. 0.6 will be rounded to 1 and -0.6 will be rounded to 0. However as we use positive and negative factors to set a threshold relative to the average value, we want thresholds with the same absolute distance to the average value (mirrored rounding), i.e. for $avg = 0$ and $d = 1$ a factor +/-0.6 should result in +/-1. Lastly, we use ceiling rounding as we want a difference of at least 1 to the average value for thresholds with a factor other than 0.

Example 5.4 (Threshold)**.** Let us consider instance 1. For the length deviation constraint we have an optimal value $opt(s^{\ell_{dev}}) = 5$ and an average value $avg(s^{\ell_{dev}}) = 17$. Assume we want to reduce the length deviation by 50% while not increasing the number of working weekends (s^{ww}) by more than 20%, we get a configuration with the following thresholds. We have $d(s^{\ell_{dev}}) = |17 - 5| = 12$ and $f(s^{\ell_{dev}}) = -0.5$. This results in $t(s^{\ell_{dev}}) = 17 + \lceil 12 \cdot 0.5 \rceil \cdot -1 = 17 - 6 = 11$. For the working weekends we have $d(s^{ww}) = 1$ (as $opt(s^{ww}) = avg(s^{ww}) = 7$) and $f(s^{ww}) = +0.2$. This yields $t(s^{ww}) = 7 + \lceil 1 \cdot 0.2 \rceil \cdot 1 = 8$.

We are interested in finding a soft constraint threshold configuration that will lead to infeasibility. This allows us to evaluate the MCSs provided for these infeasible

configurations and to learn which combinations of soft constraints are the most conflicting. For this purpose, we look at six different threshold configurations. We test for each soft constraint reducing its violation by 100% (i.e. setting the threshold to *opt*) while increasing each of the remaining 5 soft constraints by at most 10%. In the following, we identify the configuration by the constraint that is reduced to the optimum. I.e. *reduced* s^{ww} refers to the configuration where $f(s^{\mathrm{ww}}) = -1$ and $f(s) = 0.1$ for all remaining soft constraints. For each of these configurations, there are two MCSs, namely one MCS consisting of the reduced constraint s and one MCS which is a subset of the remaining constraints. Intuitively, there is a solution with $v(s) = opt(s) = t(s)$, so that removing at most all of the other thresholds must yield a solution. On the other hand, we know that there exists a solution with $v(s') = avg(s') \leq t(s')$ for all $s' \neq s$, since we have $f(s') = 0.1$. Therefore removing the threshold on s must also yield a solution.

As we are interested in which constraints stand in conflict with the reduced constraint we generate MCSs with a weight function $w(s) = 10$ if s is being reduced and $w(s') = 1$ for all $s' \neq s$. We again use ortools to solve our instances within a 1h time limit. If the configuration on the instance was proven unsatisfiable or its status is not known (i.e. neither a solution was found nor a proof of unsatisfiability), we proceed to finding an MCS with the weight function specified above comparing ortools and Exact. The detailed results for each configuration can be found in the Appendix in Tables 10,11,12,13,14, and 15. We discuss the summarised results in Table 5.4. For each configuration of reduced soft constraint we depict the percentage of instances that could be solved feasibly or optimally (sat/opt), instances for which a proof of infeasibility was found (unsat) as well as the percentage of instances for which neither was the case (unknown/?). For unsatisfiable and unknown instances, we proceed by generating MCSs using either ortools or Exact. We observe that with ortools we are able to find (potentially suboptimal) MCSs even for the unknown instances. For each of the configurations, ortools can find an MCS for all instances except one that are either unsatisfiable or unknown. This one instance is usually instance 15, which has been proven hard to solve and in one case instance 20, which is the largest instance and therefore inherently difficult to solve. In comparison, Exact mostly fails to find an MCS even for some instances that are proven unsatisfiable.

reduced	sat	opt	unsat	?	mcs ortools	mcs Exact
$s^{\mathrm{N}>3}$	40%	30%	5%	25%	25%	0%
$s^{\ell_{\mathrm{dev}}}$	5%	20%	25%	50%	70%	10%
s^{ww}	35%	35%	0%	30%	25%	0%
$s^{\mathrm{d}_{\mathrm{max}}}$	5%	30%	15%	50%	60%	10%
$s^{\mathrm{d}_{\mathrm{rms}}}$	15%	30%	5%	50%	50%	5%
s^{Nww}	30%	35%	0%	35%	30%	0%

Table 5.4: Results for finding MCSs with thresholds of using ortools or Exact. Percentages of instances where solutions were found (sat/opt) and where no solution was found (unsat/?) are indicated. Percentages of instances where ortools or Exact could find an MCS under 1h are also given.

While ortools shows promising results we must keep in mind that by using the timeout of 1h we are not guaranteed that the resulting MCS are indeed subset optimal. In fact, ortools mostly uses the whole hour for computing the MCS. The only exceptions are instances 1,2 and 8 for which shorter runtimes of roughly 10 to 20 seconds apply. Additionally, for instance 3 and 14 an MCS can be found in 350 and 1720 seconds, respectively, but only for the configurations where length deviation is reduced. We also expect runtimes below one hour for finding an MCS for other small instances such as instances 4 to 7. However, these instances remain feasible under all configurations, as their *opt* and *avg* values typically coincide or differ only by one.

To evaluate the influence of the soft constraints on each other, we evaluate how often a constraint s' is mentioned in a MCS for the configuration reducing s. A visualisation of this can be found in Figure 5.4. We depict the constraint to be reduced on the x-axis and the relative occurrences of constraints in MCSs on the y-axis. We count the occurrences in an MCSs in percentages relative to the total number of MCSs found for this configuration. As multiple constraints can occur in one MCS the summed percentages in each column do not necessarily sum up to 100%. We observe that the night shift constraint ($s^{\mathrm{N}>3}$) occurs the least often in a conflict set. The length deviation constraint ($s^{\ell_{dev}}$), on the other hand, occurs very often in an MCS. In fact, almost every MCS for reducing distances between weekends ($s^{\mathrm{d}_{\max}}$, $s^{\mathrm{d}_{\mathrm{rms}}}$) contains the length deviation constraint as part of the problem. However, if we compare how often the weekend distance constraints occur in an MCS for reducing length deviation, we find that while $s^{\mathrm{d}_{\max}}$ occurs in roughly 70% of the MCSs, the $s^{\mathrm{d}_{\mathrm{rms}}}$ constraint is only problematic in about 50% of MCSs. In general, we observe that there is not necessarily a direct correspondence between the percentage with which constraint s' occurs in MCSs when reducing s and the percentage with which constraint s occurs in MCSs when reducing s'.

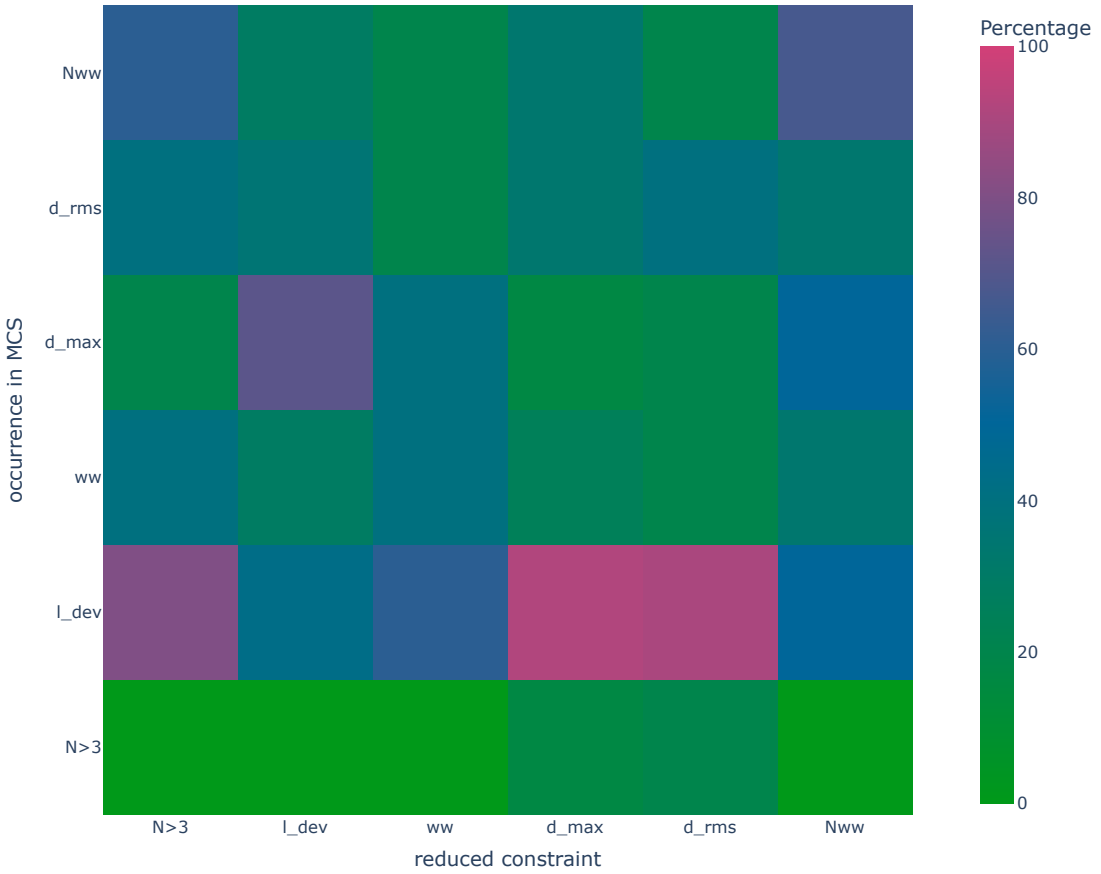


Figure 5.4: Heat map indicating the percentage of occurrences of a constraint in an MCS for a configuration using thresholds.

CHAPTER 6

Providing Multiple Solutions

Previously, the Rotating Workforce Scheduling (RWS) problem was mainly solved with methods that generated a single solution. While this can be sufficient, sometimes stakeholders are not content with the solution. In this case, an iterative process of refining the problem specification or tuning weights for soft constraints needs to be started. To circumvent this process, we are interested in generating many non-dominated solutions from which the decision-maker can select a suitable solution. If no solution that matches the stakeholder's preferences can be found, it is still possible to learn about trade-offs between different objectives. This allows us to show what combinations of preferences are possible and which goals are not realisable. To achieve this, we adapt the Pareto Simulated Annealing (PSA) framework (Czyżak and Jaszkiewicz 1997) for RWS to generate sets of solutions that approximate the Pareto fronts in different multidimensional objective spaces.

We implement and adapt the general PSA framework for the RWS problem. A pseudo-code description of the algorithm can be found in Algorithm 6.1. We will now describe the steps of the algorithm in detail, indicating the corresponding lines in Algorithm 6.1

6.1 Algorithm Description

PSA performs Simulated Annealing (SA) runs in parallel with a set of generating solutions G while keeping a non-dominated solutions archive S. Initially, k distinct generating solutions are constructed, for example using a construction heuristic (line 2). All non-dominated construction solutions are added to the solution archive S (line 4). The algorithm performs iterations as long as no aborting condition is fulfilled, for example, timeout or iteration limit (line 5). In each iteration, a move is applied to every solution $x \in G$, generating neighbouring solutions (line 11). A new solution y that is generated is compared to the current set of non-dominated solutions S (line 12). The set S is updated

according to the hard constraint violations and soft objectives of y. If y is not dominated by any solution, it is added to S, and any solution dominated by y is removed.

The applied move is accepted, resulting in the replacement of x by y as generating solution in G if one of the following applies:

- y is added to the set of non-dominated solutions S. (This criterion was added and is not proposed in the original implementation.) (line 13)
- y dominates x. (line 15)
- with probability sl (given in Equation 6.1), which depends on the current temperature t. (line 17)

$$sl(x, y, t) = \min\left(1, e^{\sum_{c \in \mathcal{H} \cup \mathcal{S}} \frac{\lambda_c(x) \cdot (v_c(x) - v_c(y))}{t}}\right) \tag{6.1}$$

Each solution x has its own weight vector $\lambda(x)$, which is updated in each iteration (line 22). This should lead to solutions diverging into different directions within the solution space and should allow for better coverage of the space. For each solution x, the closest neighbouring solution $x' \in S$ which is not dominated by x is determined. This is classically done by comparing the absolute distance in terms of violations of two solutions, i.e. $\|v(x) - v(x')\|_1$. The weights of x are updated for each constraint $c \in \mathcal{H} \cup \mathcal{S}$ according to the violations of solution x' using the update factor α (see Equation 6.2).

$$\lambda_c(x) = \begin{cases} \lambda_c(x) \cdot \alpha & \text{if } v_c(x) \leq v_c(x') \\ \lambda_c(x)/\alpha & \text{otherwise} \end{cases} \tag{6.2}$$

We additionally propose an alternative approach to updating the weight vectors, where the closest neighbour is determined by comparison of the weight vectors, i.e. x' such that $\|\lambda(x) - \lambda(x')\|_1$ is minimal. This can lead to a more diverse distribution of solutions in the search space, especially when considering a small number of dimensions.

Lastly, the weight vectors are normalised such that the sum of weights equals 1. We enforce a minimum weight of 0.001, an addition recently introduced by Mischek and Musliu 2024. This ensures that every constraint is still relevant during the search.

A further modification, initially proposed by Drexl and Nikulin 2008, consists in restarting generating solutions $x \in G$ (line 19). For this variant of PSA, solutions x which have not contributed new solutions to the set of non-dominated solutions S for r consecutive iterations, are replaced by a random solution from S. The aim of this addition is to replace 'bad' generating solutions with new generating solutions that are more promising.

Algorithm 6.1: Parallelised implementation of Pareto Simulated Annealing including modifications

Input: Instance I,
number of generating solutions k,
abortion criteria a,
restart frequency r,
weight update policy w,
weight update factor α,
Result: Non-dominated solution set S

```
1  Function Main(I, k, a, r, w, α):
2      G ← init(I, k)
3      S ← {}
4      update_pareto(S, G)
5      while ¬ abort(a) do
6          map(G, r, w, α, apply_iteration)
7      end
8      return S
9
10 Function apply_iteration(x, r, w, α):
11     y ← move(x)
12     update_pareto(S, y)
13     if y ∈ S then
14         x ← y
15     else if dominates(x,y) then
16         x ← y
17     else if random([0,1])<sl(x, y, t) then
18         x ← y
19     else if no_contribution(x) ≥ r then
20         x ← random(S)
21     end
22     update_weights(x, w, α)
23     return
```

6.2 Computational Study

We now present the application of PSA to the RWS problem. These results have been presented at the Multi-Objective Decision Making Workshop[1] (MODeM) of the 27th European Conference on Artificial Intelligence[2] (ECAI 2024) under the title *Towards Multi-Objective Optimisation for Rotating Workforce Scheduling*.

We performed our experiments on the same cluster as the Explainability experiments. The PSA algorithm was implemented in Python 3.9 and run with the fast PyPy[3] interpreter.

We evaluate our approach on three different real-life RWS instances[4] of different size and difficulty. The first instance (instance 10) only considers $n = 27$ employees, while the second (instance 15) and third (instance 20) deal with 64 and 163 employees. As initial construction solutions, we use solutions generated using the branch&cut (Becker, Schiffer, and Walther 2022) approach.[5] This ensures that all our construction solutions are feasible, allowing us to concentrate on the optimisation towards soft constraints. Additionally, this ensures that all solutions added to the set of non-dominated solutions are feasible, as no infeasible solution can dominate a feasible solution. For our experiments, we use $k = 8$ generating solutions, which should allow for a good trade-off between runtime and quality of the solution set (Czyżak and Jaszkiewicz 1997). Additionally, all experiments are run with a starting temperature $t = 1$, cooling factor $\beta = 0.999$ and 1 million iterations. When the temperature falls below 10^{-5}, t is reheated to the starting temperature of 1. We use a fixed weight of 5 for hard constraints and update weights only for soft constraints using the factor $\alpha = 1.05$. The initial soft constraint weights are chosen at random for each generating solution.

For the application of moves, we consider *PeriodIntervalSwaps*. An application swaps two intervals of length 1 to 7 (number of days) that differ by a period of 7, meaning two sequences of consecutive shifts are swapped between two employees. Thereby, the number and type of shifts assigned to each day remain the same, as only shifts between the same days of the week are swapped.

We have conducted brief experiments, including all of the proposed soft constraints. Based on the most conflicting constraints, length deviation ($s^{\ell_{\text{dev}}}$), working weekends (s^{ww}) and weekend distance ($s^{d_{\max}}$), we have decided to focus on the quality of the solutions for the 3-dimensional case.

The quality of the generated solution is evaluated by using the hypervolume metric. The hypervolume indicates the percentage of the multidimensional objective space which is dominated by the approximated Pareto front. The size of the whole objective space is determined by the ideal(min) and anti-ideal (max) points. The dominated space spans

[1] https://modem2024.vub.ac.be/
[2] https://www.ecai2024.eu/
[3] https://www.pypy.org/
[4] https://www.dbai.tuwien.ac.at/staff/musliu/benchmarks/
[5] https://github.com/tribec/bac-rwsp

	$s^{\ell_{\text{dev}}}$		s^{ww}		$s^{\text{d}_{\max}}$	
instance	$[min,$	$max]$	$[min,$	$max]$	$[min,$	$max]$
10	[1,	48]	[12,	18]	[1,	18]
15	[20,	154]	[45,	54]	[3,	54]
20	[2,	962]	[120,	163]	[3,	164]

Table 6.1: Ranges from ideal to anti-ideal points for soft constraints on different instances.

from the anti-ideal point to the approximated front. The (anti-)ideal point in every dimension is given minimal (maximal) violation value of the corresponding constraint that a solution can have. The ideal points for s^{ww} and $s^{\ell_{\text{dev}}}$ were calculated using the branch&cut framework by Becker, Schiffer, and Walther 2022, to which we added the squared length deviation minimisation. The anti-ideals for s^{ww} and $s^{\ell_{\text{dev}}}$ could be determined by using the branch&cut approach, too, but by aiming to maximise the violation. Lastly, the ideal for $s^{\text{d}_{\max}}$ is given by the ratio between working and free weekends $\lceil \frac{min(s^{\text{ww}})}{n-min(s^{\text{ww}})} \rceil$, while the anti-ideal points correspond to the anti ideal of s^{ww} (except for instance 20 where no weekend is free in the worst case). The ideals (max) and anti-ideals (min) for the three dimensions and instances can be found in Table 6.1. We have used these points for the calculation of the normalised hypervolume (further referred to simply as hypervolume).

6.2.1 Visualising Solutions

We use parallel coordinate plots to visualise the approximated fronts. This allows stakeholders to investigate individual solutions while providing insights into correlations between different constraints. The plot can be studied interactively by filtering regions for each dimension. Thereby only solutions which satisfy the filters in all dimensions are shown. This shows clearly which trade-offs have to be made in which dimensions when a stakeholder wants to improve a specific objective. Such a plot for instance 20 can be found in Figure 6.1. Each line corresponds to a non-dominated solution, indicating the violation values by the intersection with the corresponding axis. The solutions are colour-coded with respect to the length deviation dimension.

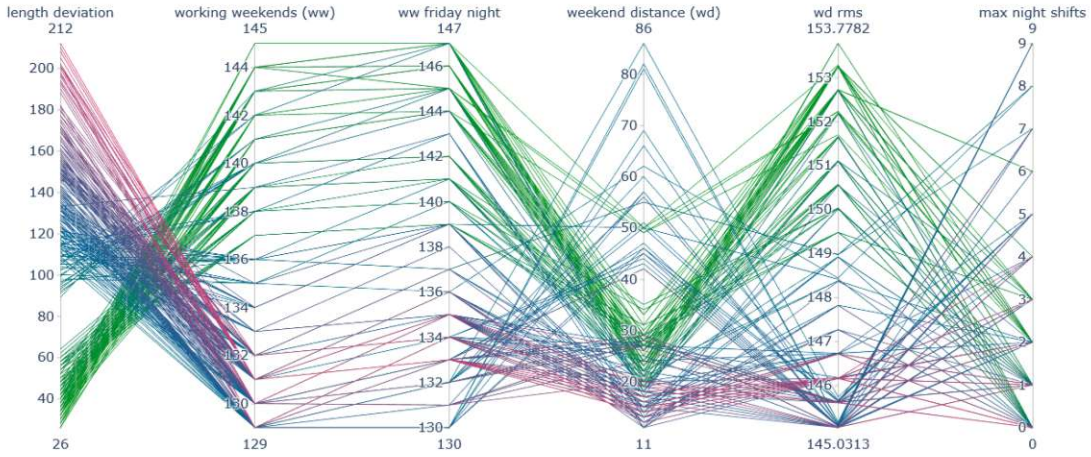
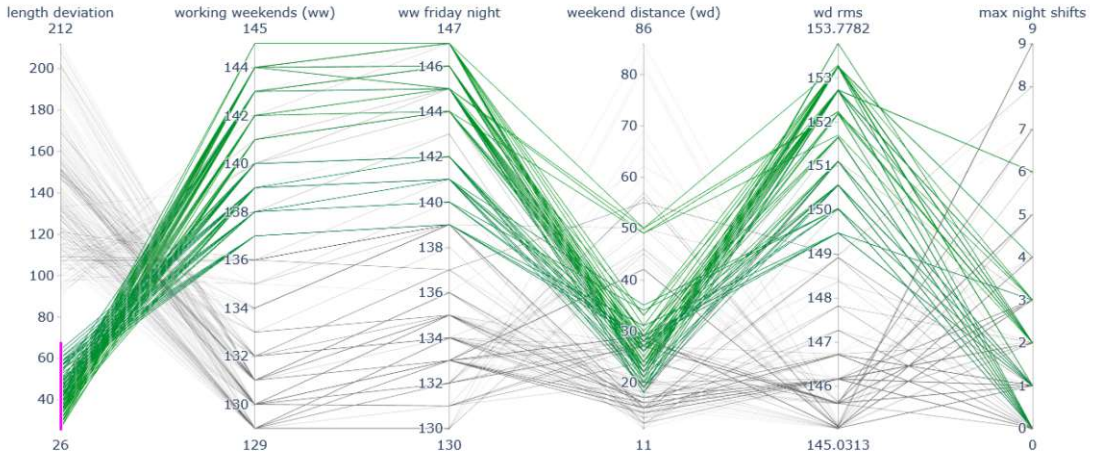


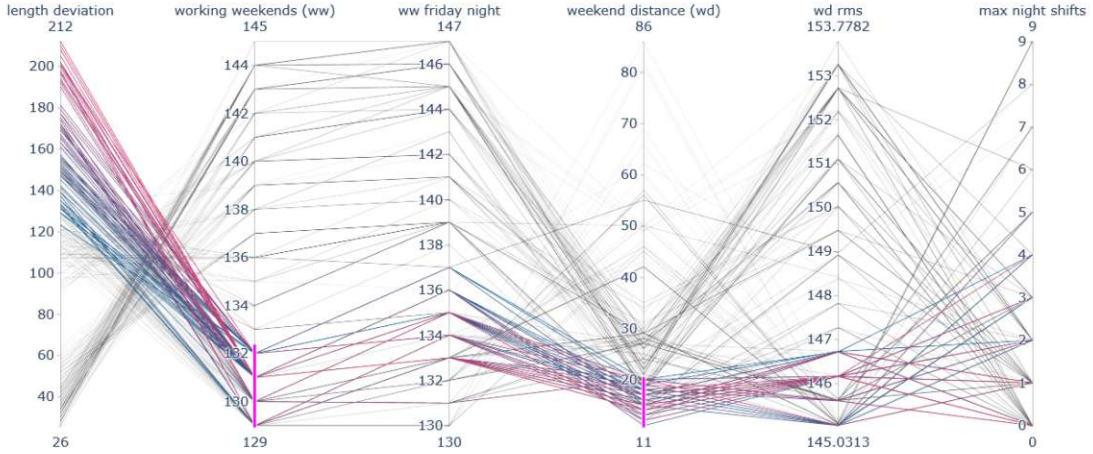
Figure 6.1: Approximated Pareto front with 305 solutions after 1M iterations for instance 20 with all dimensions, making weight updates according to closest violation neighbours

These plots also make it possible to filter solutions in an interactive manner. Examples of filters for instance 20 can be found in Figure 6.2. By applying filters on the axis of different dimensions, all solutions not matching the criteria are greyed out. This allows stakeholders to specify their preferences and to observe which solutions remain. In Figure 6.2a, a filter is set that allows only solutions with violations of $s^{\ell_{\text{dev}}}$ in the range from 26 to about 70. As a result, we can clearly see that any solution adhering to that criteria will have at least 137 working weekends. In Figure 6.2b, a filter is set on the number of working weekends, allowing at most 132 working weekends. Additionally, the weekend distance is limited to about 20. This results in solutions that will have violations in the $s^{\ell_{\text{dev}}}$ dimension of at least 120.

For instance 20, we can observe that while the number of working weekends, working weekends including Friday night and equal distance between free weekends show a strong correlation, the length deviation constraint seems to be in conflict with optimising free weekends.



(a) Filter on length deviation ($s^{\ell_{\text{dev}}}$).



(b) Filter on working weekends (s^{ww}) and weekend distance ($s^{\text{d}_{\max}}$)

Figure 6.2: Filtered approximated Pareto front with 305 solutions after 1M iterations for instance 20 with all dimensions, making weight updates according to closest violation neighbours.

For instances 10 and 15, we provide similar plots which can be found in Figure 6.3 and 6.4, respectively. While we can observe a conflicting relation between the weekend distance ($s^{\mathbf{d_{max}}}$ and the length deviation ($s^{\ell_{\text{dev}}}$) for instance 10, it is more difficult to pinpoint conflicting constraints directly.

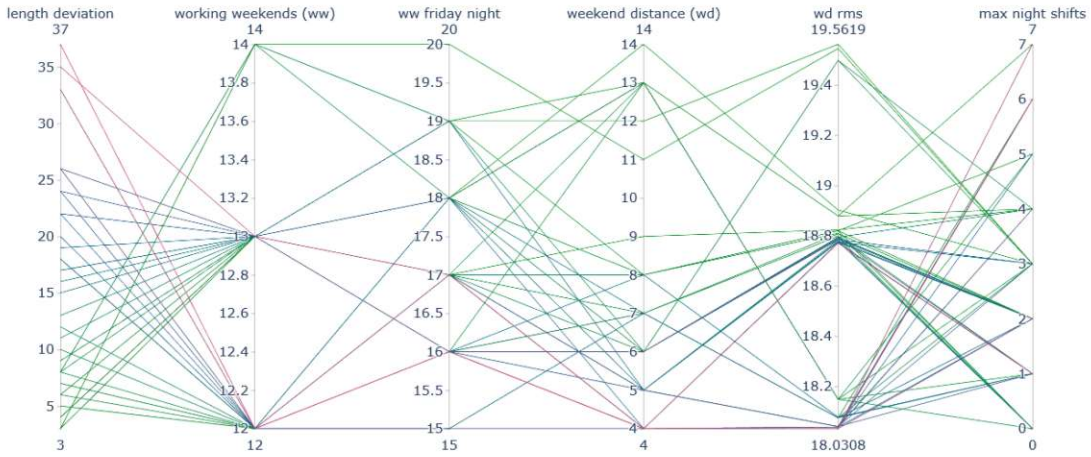


Figure 6.3: Approximated Pareto front with 52 solutions after 1M iterations for instance 10 with all dimensions, making weight updates according to closest violation neighbours

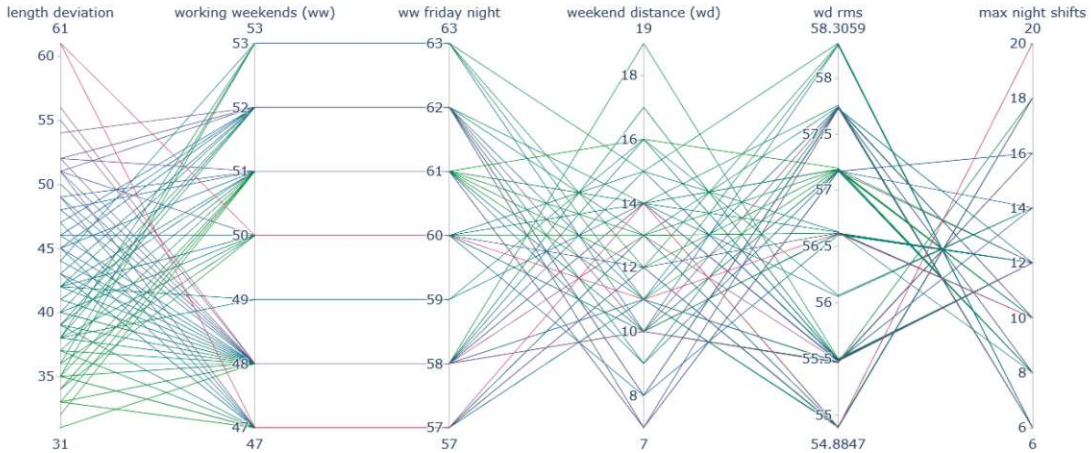


Figure 6.4: Approximated Pareto front with 88 solutions after 1M iterations for instance 15 with all dimensions, making weight updates according to closest violation neighbours

We validate our observations using the sample Pearson correlation coefficient $r(dim1, dim2)$. We compared the dimensions length deviation, working weekends and weekend distance for all three instances. For this we took the violation vectors for each non-dominated solution generated by runs considering all dimensions and making weight updates using closest-violation neighbours. Table 6.2 shows that while the correlations of the objectives differ for each instance, a general tendency of negative correlation can be observed. In addition, each instance provides a strong negative correlation for a different combination of the three dimensions.

instance	$r(s^{\ell_{\text{dev}}}, s^{\text{ww}})$	$r(s^{\text{ww}}, s^{\text{d}_{\max}})$	$r(s^{\text{d}_{\max}}, s^{\ell_{\text{dev}}})$
10	-0.368	0.109	**-0.736**
15	-0.013	**-0.414**	-0.201
20	**-0.873**	-0.134	-0.179

Table 6.2: Sample Pearson correlation coefficients for three objectives.

6.2.2 Testing Improvements

In addition to showing how Pareto Simulated Annealing can be used for RWS to provide stakeholders with multiple solutions, we want to investigate if adaptions of the general algorithms can lead to a better performance. We therefore evaluate which of the modifications introduced before help to provide better solution sets. Additionally, we compare the PSA approach to the state-of-the-art branch&cut approach.

Weight Neighbors

Our motivation for introducing weight-neighbour weight updates stems from the observation, that for the 2-dimensional case the weight vectors of the generating solution converged toward 1 in one of the dimensions. This led to solutions which mostly optimised only one of the two dimensions. As a result, only few non-dominating solutions could be found that optimise both constraints in equal manner. When using weight-neighbour updates this gap could be closed. However, fewer solutions that mainly optimise one of the constraints could be obtained. A visualisation of the observation for instance 20 and the dimensions length deviation and working weekends can be found in Figure 6.5. Comparing the hypervolume for all three instances over 10 runs considering length deviation and working weekends, yielded similar average hypervolume for both weight updating procedures. Nevertheless, we investigated both approaches also for multi-dimensional cases.

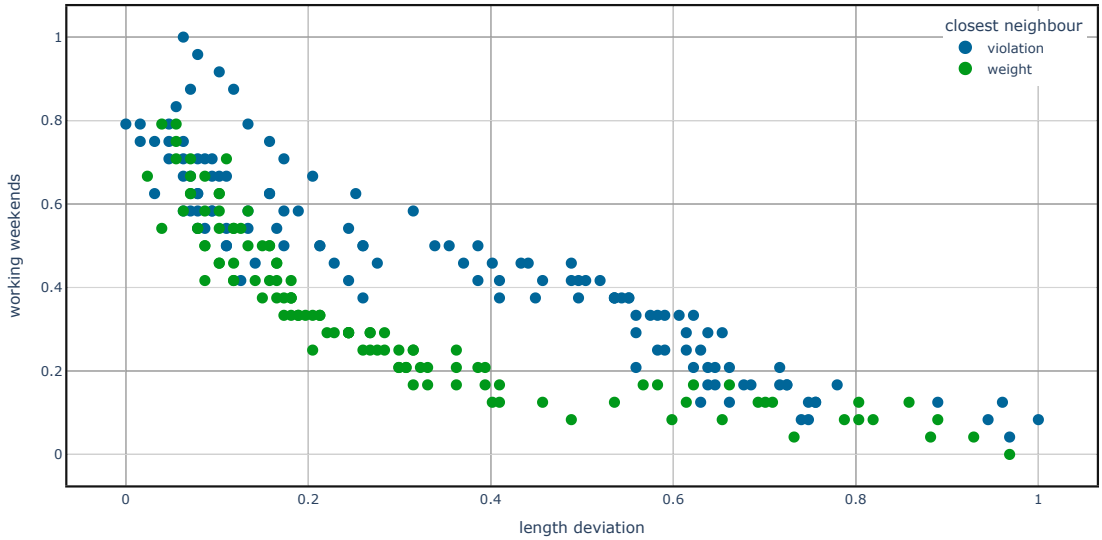


Figure 6.5: Combined solution sets over 10 runs using closest weight and violation neighbour for weight updates for instance 20. The solutions are given in the min/max normalised space over both dimensions.

Restarting Solutions

We calculated exact 2D Pareto fronts for s^{ww} and $s^{\ell_{\mathrm{dev}}}$ with the ε-constraint method using the branch&cut (b&c) approach. There, the optimisation of $s^{\ell_{\mathrm{dev}}}$ is converted into a constraint with an upper bound starting from its ideal iteratively optimising s^{ww} while increasing the constraint bound stepwise, until the ideal of s^{ww} is hit. This could be performed within a couple of minutes runtime. A visualisation of this front compared to a Pareto front approximation using PSA with weight-vectors to determine the closest neighbouring solution can be found in Figure 6.6. The generating solutions are given as arrows, directed in the direction within the 2-dimensional space toward which they are optimising. It can be observed that while the solution set generated with b&c provides better solutions for length deviation and working weekend constraints, weekend distance violations are comparably high.

We evaluated the hypervolume for the different configurations of PSA over all three instances, considering the length deviation, working weekends and weekend distance constraints. For each configuration, 10 runs were performed using the same random generator seeds for the different configurations. The hypervolume is calculated with regard to the ideal (*min*) and anti-ideal (*max*) for each of the three dimensions. The development of the average hypervolume when restarting solutions after a certain number of non-improvement iterations (consecutive iterations where a solution does not contribute to the set of non-dominated solutions) can be found in Figure 6.7. As a reference, the average hypervolume for random weight updates, where weights are assigned at random

in each iteration, as well as the average hypervolume for classical weight updates using the weight or violation vectors to determine closest neighbours, are indicated.

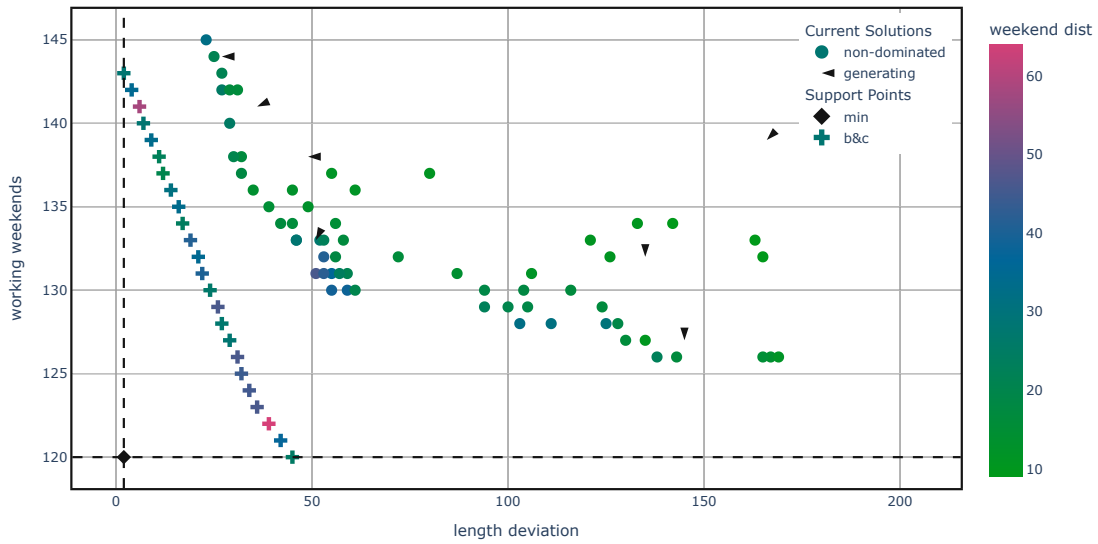


Figure 6.6: Approximated Pareto front over 3 dimensions after 1M iterations for instance 20. The 2-dimensional ideal and non-dominated solutions using b&c are given as reference.

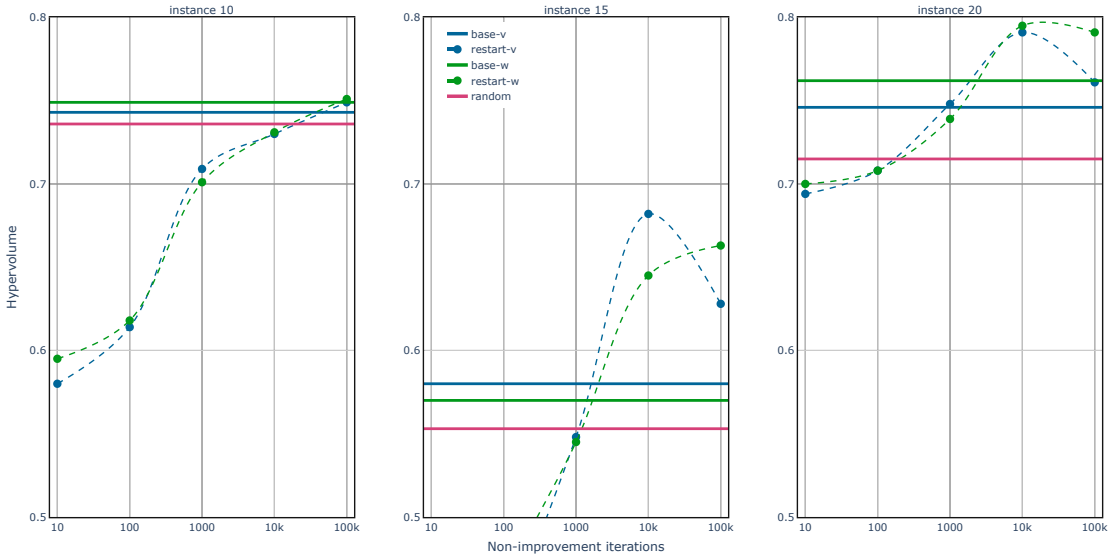


Figure 6.7: Comparison of hypervolume for random and classic weight updates with spline hypervolume for restarting approach.

It can be observed that frequent restarts (after 10 to 1k non-improvement iterations) lead to worse hypervolume. This can be explained by the fact that multiple generating solutions can be replaced by the same non-dominated solutions, leading to a lack of variety within the generating solutions. However, by increasing the threshold on non-improvement iterations, the hypervolume can be increased compared to the classical PSA approach. For instances 15 and 20, restarting after 10k non-improvement iterations leads to better results than for a threshold of 100k iterations, where the hypervolume tends to decline. In contrast, the hypervolume for instance 10 can hardly be increased by using the restarting method. This implies that a suitable threshold depends on the instance but should not be set too low, as this reduces the diversity among the generating solutions.

instance	configuration	hv - avg	hv - max	hv - stdv	t[h]
10	random	0.736	0.748	0.009	1.160
	classic-v	0.743	**0.757**	0.008	1.307
	classic-w	0.749	0.755	0.004	1.343
	restart-v10	0.580	0.678	0.074	1.760
	restart-w100k	**0.751**	**0.757**	0.005	1.406
	b&c	0.561			
15	random	0.553	0.644	0.050	1.575
	classic-v	0.580	0.648	0.037	1.726
	classic-w	0.570	0.663	0.054	1.758
	restart-v10	0.340	0.497	0.067	2.370
	restart-v10k	**0.682**	**0.838**	0.082	1.659
	b&c	0.782			
20	random	0.715	0.750	0.014	2.261
	classic-v	0.746	0.796	0.024	2.425
	classic-w	0.769	0.794	0.017	2.481
	restart-v10	0.694	0.741	0.019	3.360
	restart-w10k	**0.795**	**0.841**	0.024	2.597
	b&c	0.872			

Table 6.3: Average, max and standard deviation for hypervolume and average runtime over 10 runs for different PSA configurations. Weight update configuration (v/w) and restart thresholds (10, 10k, 100k) are indicated. Hypervolume of b&c is given as reference.

In Table 6.3 the average, maximal and standard deviation for the hypervolume over the 10 runs for different PSA configurations are given. For restarting runs, the configurations with the best and worst average hypervolume are indicated. Again, it is evident that choosing the threshold is crucial since low thresholds lead to worse hypervolume than the approach using no restarting and random weight updates. In addition to the hypervolume, the average runtime over 10 runs for the PSA procedure is given. It can be observed that the configuration using random weight updates takes the least runtime. This is obvious, as no closest neighbour must be determined to update weights, hence decreasing the number of comparisons that have to be made. However, random weight updates lead to

worse hypervolume than the classical approach. It can also be observed that frequent restarts (after 10 iterations) increase the runtime while high thresholds for restarting yield only slightly increased runtimes compared to the classic approach but can improve the hypervolume of the obtained solution sets. This indicates that rare restarting is a feasible addition to the classical implementation of PSA.

Comparing the performance of the presented approaches to the branch&cut reference points calculated with the ε-constraint method, we observe that while PSA can provide solution sets with more hypervolume for instance 10 and for some runs on instance 15, it is outperformed by branch&cut on instance 20. Since the b&c approach can not explicitly optimise weekend distance, we also analyse the mean violation for the best PSA configuration and b&c over the non-dominated solutions. The results can be seen in Table 6.4.

instance	configuration	$s^{\ell_{\text{dev}}}$	s^{ww}	$s^{\text{d}_{\max}}$
10	restart-w100k	7.02	12.89	8.18
	b&c	2.25	13.25	12.75
15	restart-v10k	33.84	47.79	11.34
	b&c	23.55	47.22	17.33
20	restart-w10k	51.04	130.51	16.52
	b&c	22.98	130.84	41.70

Table 6.4: Mean values for three objectives comparing PSA configurations with highest avg hypervolume to branch&cut (b&c).

As expected, the branch&cut approach has difficulties in optimising $s^{\text{d}_{\max}}$, especially in comparison to the PSA configurations. While the results for s^{ww} are very similar for both algorithms, b&c yields better results for $s^{\ell_{\text{dev}}}$, which is to be expected as this objective can be optimised using branch&cut.

These results show that while the hypervolume provides a good general indication of the quality of the solution sets, this does not allow for insights on the individual dimensions. We have shown that branch&cut provides very good solutions considering local objectives such as length deviation or working weekends. However, PSA can outperform branch&cut on non-local properties. In general, PSA with the right configuration manages to produce results comparable with branch&cut, while it also benefits from the initial feasible solutions provided by the b&c solver. Therefore, interesting future work would be a further hybridisation of both by (re-)starting PSA from partially optimised branch&cut solutions.

CHAPTER 7

Conclusion

We have proposed two frameworks for the extended Rotating Workforce Scheduling Problem (RWS) that includes several novel soft constraints. We tackle the scenarios where (1.) no solution can be found by an algorithm and (2.) if the stakeholder is not content with a solution and requires a better one.

To do so, we created a constraint model of the RWS problem using CPMpy. We showed how soft and hard constraints as well as thresholds can be modelled. This model was then used for providing explanations. Using a small example we showed how Minimal Correction Sets (MCS) can be generated and how preferences can be specified to obtain preferred MCSs. We first conducted experiments on infeasible RWS instances. While using MARCO to enumerate all Minimal Correction Sets (MCS) did not yield the best results, we could employ ortools to provide MCSs in under an hour for all infeasible instances. In a second step, we tested threshold combinations for soft constraints on feasible instances. While we could not find optimal MCSs especially for larger instances, we were generally able to provide MCSs in cases where no solution could be found. While some inferences could be made regarding conflicting soft constraints, no clear image of the relationships between soft constraints could be established.

We therefore used the second approach to evaluate this further. To do so, we implemented the Pareto Simulated Annealing (PSA) algorithm and proposed different modifications and adaptions of the algorithm which we evaluated on three instances of varying difficulty. We showed how large sets of solutions can be represented, allowing a stakeholder to choose among them. Additionally, we were able to determine the most conflicting constraints for the instances at hand. Lastly, we evaluated modifications of the PSA algorithm and their influences on the solution set. We studied the influence of using closest weight neighbour updates in the 2-dimensional space (length deviation and working weekends). While the hypervolumes obtained were very similar, we observed that using the closest weight-neighbours generates better compromising solutions, while violation-based updates enhance solutions favouring one of the dimensions. We have

shown that sporadically resetting generating solutions that do not contribute to the set of non-dominating solutions can improve the hypervolume compared to the classical PSA approach. Additionally, these sparse resets do not increase runtime significantly. Lastly, we compared our approach to the current state-of-the-art branch&cut. While PSA could not consistently outperform branch-and-cut, it yielded substantially better results for the non-local weekend distance objective.

In the future, it would be interesting to further evaluate our constraint model. Testing the performances of other solvers as well as running experiments with higher time-outs, could yield new insights. Additionally, large-scale testing on bigger benchmark sets could be performed. In a next step, the development of a graphical user interface that provides explanations and highlights problematic regions of a schedule could be addressed.

Regarding Pareto Simulated Annealing, it would be interesting to further analyse how PSA and branch&cut compare on other (non-local) constraints and to improve our approach by making use of the strengths that b&c provides. It would be interesting to use parameter tuning to determine optimal configurations, to investigate how instance characteristics influence suitable configurations, and what other metrics should be considered to evaluate solution quality. Finally, we also want to validate our results on a larger set of benchmark instances.

Appendix

instance	$t(p)$	status	$t(mcs)$	$\#mcs$	min(mcs)	max(mcs)
instance83	1.5	unsat	3,860.1	7	1	2
instance109	352.7	unsat	3,787.0	0		
instance100	2.0	unsat	3,685.5	1	1	1
instance163	2.1	unsat	33.0	3	1	1
instance446	2.7	unsat	3,626.7	0		
instance456	0.4	unsat	5,084.5	15	4	8
instance512	3,600.5	?	3,612.2	0		
instance684	470.0	unsat	3,622.8	0		
instance753	0.8	unsat	3,800.8	5	2	7
instance1245	731.6	unsat	3,732.7	0		
instance1279	3,600.5	?	4,373.6	0		
instance1490	2,225.7	unsat	3,616.6	0		
instance1628	42.6	unsat	3,706.5	0		
instance1705	3,600.1	?	3,672.1	0		
instance1757	1.1	unsat	49.7	5	2	4
instance1798	44.2	unsat	3,616.1	0		
instance1807	0.3	unsat	22.9	9	4	8
instance1921	6.5	unsat	3,652.7	0		
instance1950	105.0	unsat	3,640.7	0		
instance2190	704.6	unsat	3,791.0	0		
instance2465	3,600.5	?	3,771.0	0		
instance3125	19.4	unsat	3,619.4	0		
instance3143	3,600.3	?	3,789.2	0		
instance3328	63.1	unsat	3,644.7	0		
instance3377	3,600.1	?	3,869.2	0		
instance3819	0.2	unsat	3,636.5	11	5	8
instance3966	1,712.5	unsat	3,649.8	0		
instance4014	3,600.7	?	5,302.2	0		
instance4098	3,600.2	?	3,643.5	0		
instance4194	3,600.3	?	6,246.3	0		
instance4500	8.0	unsat	3,661.5	0		
instance4519	3,600.5	?	6,321.0	0		
instance4583	338.3	unsat	4,100.8	0		
instance4602	1.6	unsat	5,066.4	0		
instance4700	295.7	unsat	4,059.1	0		
instance4720	0.2	unsat	3,610.6	39	4	8
instance4725	2,640.8	unsat	3,657.3	0		
instance4759	9.3	unsat	3,666.1	0		
instance4763	8.9	unsat	4,521.2	0		
instance4779	232.3	unsat	4,011.6	0		
instance4953	3,600.9	?	4,395.6	0		
instance5018	3,600.4	?	3,784.2	0		
instance5043	727.8	unsat	3,651.1	0		
instance5363	338.7	unsat	3,609.6	0		
instance5486	351.7	unsat	3,635.9	0		
instance5621	1,404.2	unsat	3,878.8	0		
instance5706	20.8	unsat	3,725.6	0		
instance5756	1,157.8	unsat	3,620.6	0		
instance5833	642.3	unsat	4,042.5	0		
instance5965	1.6	unsat	1,543.9	5	2	4

Table 1: Number and sizes (min/max) of MCSs found using MARCO with z3-ortools. Time needed to prove unsatisfiability ($t(p)$) and time used by MARCO to find MCSs ($t(mcs)$) are given in seconds.

instance	$t(p)$	status	$t(mcs)$	$\#mcs$	min(mcs)	max(mcs)
instance83	1.33	unsat	4,548.35	0		
instance109	340.90	unsat	3,609.37	0		
instance100	1.95	unsat	4,039.20	1	1	1
instance163	1.69	unsat	42.69	3	1	1
instance446	2.50	unsat	5,497.43	0		
instance456	0.35	unsat	3,832.63	3	4	8
instance512	3,600.28	?	3,605.04	0		
instance684	363.51	unsat	3,614.55	0		
instance753	0.90	unsat	3,693.39	0		
instance1245	833.92	unsat	3,612.23	0		
instance1279	3,600.36	?	3,618.64	0		
instance1490	2,184.09	unsat	3,606.10	0		
instance1628	47.91	unsat	3,609.91	0		
instance1705	3,600.10	?	3,608.79	0		
instance1757	1.06	unsat	41.97	5	2	4
instance1798	56.55	unsat	3,745.49	1	1	1
instance1807	0.30	unsat	12.27	9	4	8
instance1921	6.25	unsat	4,110.33	0		
instance1950	126.80	unsat	3,608.69	0		
instance2190	627.08	unsat	3,611.73	0		
instance2465	3,600.45	?	3,617.04	0		
instance3125	25.34	unsat	3,609.53	0		
instance3143	3,600.47	?	3,619.66	0		
instance3328	61.33	unsat	3,615.91	0		
instance3377	3,600.21	?	3,621.56	0		
instance3819	0.23	unsat	3,641.16	5	6	8
instance3966	465.54	unsat	oom	0		
instance4014	3,600.80	?	oom	0		
instance4098	3,240.80	?	3,620.38	0		
instance4194	3,600.38	?	3,621.94	0		
instance4500	7.82	unsat	3,657.90	0		
instance4519	3,600.85	?	3,624.73	0		
instance4583	388.69	unsat	3,618.74	0		
instance4602	1.47	unsat	4,192.59	0		
instance4700	304.13	unsat	3,620.92	0		
instance4720	0.16	unsat	3,613.48	19	4	8
instance4725	2,733.95	unsat	3,611.60	0		
instance4759	10.04	unsat	3,624.28	0		
instance4763	8.79	unsat	3,622.61	0		
instance4779	176.15	unsat	oom	0		
instance4953	3,600.15	?	3,625.11	0		
instance5018	3,600.20	?	3,621.28	0		
instance5043	984.96	unsat	3,610.99	0		
instance5363	426.07	unsat	3,604.69	0		
instance5486	222.66	unsat	3,608.06	0		
instance5621	1,750.51	unsat	3,616.71	0		
instance5706	20.57	unsat	3,626.51	0		
instance5756	436.25	unsat	3,614.63	0		
instance5833	377.50	unsat	oom	0		
instance5965	1.24	unsat	2,477.94	5	2	4

Table 2: Number and sizes (min/max) of MCSs found using MARCO with z3-pysat. Time needed to prove unsatisfiability ($t(p)$) and time used by MARCO to find MCSs ($t(mcs)$) are given in seconds.

instance	$t(p)$	status	$t(mcs)$	$\#mcs$	min(mcs)	max(mcs)
instance83	1.40	undsat	3,709.59	17	1	2
instance109	522.48	undsat	4,518.76	0		
instance100	1.95	undsat	4,185.92	0		
instance163	1.56	undsat	230.22	3	1	1
instance446	2.73	undsat	4,209.52	0		
instance456	0.35	undsat	3,763.95	0		
instance512	3,600.20	?	6,746.63	0		
instance684	514.47	undsat	3,605.44	0		
instance753	0.84	undsat	375.06	9	2	9
instance1245	707.26	undsat	6,720.89	0		
instance1279	3,600.34	?	4,100.50	0		
instance1490	2,219.42	undsat	4,034.37	0		
instance1628	52.18	undsat	5,443.56	0		
instance1705	3,600.22	?	7,158.63	0		
instance1757	1.18	undsat	375.68	5	2	4
instance1798	37.79	undsat	4,669.38	0		
instance1807	0.30	undsat	37.90	9	4	8
instance1921	6.90	undsat	6,682.55	0		
instance1950	123.39	undsat	6,790.32	0		
instance2190	645.07	undsat	6,548.42	0		
instance2465	3,600.12	?	4,508.70	0		
instance3125	24.03	undsat	4,623.30	0		
instance3143	3,600.37	?	5,603.14	0		
instance3328	104.77	undsat	4,818.86	0		
instance3377	3,600.53	?	4,000.19	0		
instance3819	0.24	undsat	5,583.24	5	6	8
instance3966	510.80	undsat	4,620.91	0		
instance4014	3,600.62	?	4,236.21	0		
instance4098	3,600.19	?	3,943.36	0		
instance4194	3,600.31	?	6,001.61	0		
instance4500	8.42	undsat	6,657.30	0		
instance4519	3,600.43	?	3,918.49	0		
instance4583	594.78	undsat	5,869.79	0		
instance4602	1.55	undsat	3,940.67	0		
instance4700	236.60	undsat	3,906.29	0		
instance4720	0.16	undsat	3,676.22	9	4	7
instance4725	1,738.82	undsat	5,424.51	0		
instance4759	10.38	undsat	4,570.36	0		
instance4763	9.03	undsat	4,445.01	0		
instance4779	196.31	undsat	4,728.98	0		
instance4953	3,600.28	?	5,131.26	0		
instance5018	3,600.83	?	6,876.46	0		
instance5043	550.72	undsat	3,915.78	0		
instance5363	401.88	undsat	3,741.47	0		
instance5486	322.07	undsat	4,673.47	0		
instance5621	1,588.55	undsat	4,402.29	0		
instance5706	22.37	undsat	6,150.29	0		
instance5756	411.31	undsat	6,105.48	0		
instance5833	584.22	undsat	6,587.46	0		
instance5965	1.34	undsat	605.90	5	2	4

Table 3: Number and sizes (min/max) of MCSs found using MARCO with Exact-ortools. Time needed to prove unsatisfiability ($t(p)$) and time used by MARCO to find MCSs ($t(mcs)$) are given in seconds.

instance	$t(p)$	status	$t(mcs)$	$\#mcs$	min(mcs)	max(mcs)
instance83	1.34	unsat	3,747.70	0		
instance109	291.00	unsat	3,666.38	0		
instance100	1.63	unsat	3,654.77	0		
instance163	2.26	unsat	662.36	3	1	1
instance446	2.84	unsat	3,991.10	0		
instance456	0.38	unsat	4,823.08	27	4	8
instance512	3,600.19	?	3,625.54	0		
instance684	556.87	unsat	3,919.40	0		
instance753	0.81	unsat	178.96	9	2	9
instance1245	758.91	unsat	3,842.86	0		
instance1279	3,600.38	?	3,879.07	0		
instance1490	2,077.87	unsat	3,634.01	0		
instance1628	53.48	unsat	3,700.36	0		
instance1705	3,600.19	?	3,786.17	0		
instance1757	1.07	unsat	341.29	5	2	4
instance1798	42.67	unsat	3,646.24	0		
instance1807	0.31	unsat	39.82	9	4	8
instance1921	7.05	unsat	3,801.61	0		
instance1950	200.35	unsat	3,845.37	0		
instance2190	626.83	unsat	3,667.33	0		
instance2465	3,600.19	?	3,770.23	0		
instance3125	19.40	unsat	3,788.61	0		
instance3143	3,600.34	?	3,730.69	0		
instance3328	64.85	unsat	3,683.87	0		
instance3377	3,600.79	?	3,774.72	0		
instance3819	0.23	unsat	7,060.15	3	6	8
instance3966	1,889.03	unsat	3,718.84	0		
instance4014	3,601.02	?	3,835.49	0		
instance4098	3,600.42	?	3,749.44	0		
instance4194	3,600.43	?	3,687.57	0		
instance4500	7.26	unsat	4,445.38	0		
instance4519	3,600.99	?	3,923.40	0		
instance4583	401.34	unsat	3,892.82	0		
instance4602	1.50	unsat	5,634.08	0		
instance4700	461.36	unsat	3,998.29	0		
instance4720	0.16	unsat	3,688.84	3	4	7
instance4725	1,847.39	unsat	3,720.01	0		
instance4759	10.43	unsat	4,799.08	0		
instance4763	8.74	unsat	4,534.43	0		
instance4779	175.54	unsat	3,736.37	0		
instance4953	3,600.61	?	4,007.52	0		
instance5018	3,600.49	?	4,257.87	0		
instance5043	821.22	unsat	3,907.67	0		
instance5363	287.23	unsat	3,630.51	0		
instance5486	697.03	unsat	3,691.82	0		
instance5621	1,487.23	unsat	3,939.52	0		
instance5706	20.06	unsat	3,724.51	0		
instance5756	734.30	unsat	3,830.63	0		
instance5833	576.05	unsat	3,697.26	0		
instance5965	1.29	unsat	649.22	5	2	4

Table 4: Number and sizes (min/max) of MCSs found using MARCO with Exact-pysat. Time needed to prove unsatisfiability ($t(p)$) and time used by MARCO to find MCSs ($t(mcs)$) are given in seconds.

instance	$t(p)$	status	$t(ortools)$	$s(ortools)$	$t(Exact)$	$s(Exact)$
instance83	1.17	unsat	10.85	1.0	123.19	1.0
instance109	292.65	unsat	774.54	1.0	3,644.75	
instance100	2.02	unsat	28.77	1.0	3,640.38	
instance163	1.76	unsat	19.80	1.0	375.42	1.0
instance446	2.76	unsat	135.12	2.0	3,666.68	
instance456	0.35	unsat	9.77	4.0	89.15	4.0
instance512	3,600.85	?	3,609.41	1.0	3,632.67	
instance684	495.38	unsat	3,626.98	2.0	3,807.52	
instance753	0.82	unsat	8.92	2.0	71.32	2.0
instance1245	680.33	unsat	2,299.95	1.0	3,770.08	
instance1279	3,600.32	?	3,627.68	1.0	3,689.51	
instance1490	2,606.27	unsat	2,084.13	1.0	3,624.84	
instance1628	47.09	unsat	153.87	1.0	3,659.05	
instance1705	3,600.42	?	3,616.81	1.0	3,771.75	
instance1757	1.11	unsat	13.04	2.0	245.02	2.0
instance1798	30.86	unsat	46.06	1.0	3,642.86	
instance1807	0.30	unsat	8.23	4.0	31.04	4.0
instance1921	6.81	unsat	3,628.30	4.0	3,702.36	
instance1950	116.71	unsat	184.77	1.0	3,854.59	
instance2190	792.97	unsat	1,241.41	1.0	3,672.85	
instance2465	3,600.36	?	3,633.48	2.0	3,764.51	
instance3125	30.04	unsat	3,618.70	2.0	3,824.02	
instance3143	3,600.24	?	3,638.04	1.0	3,767.36	
instance3328	57.13	unsat	161.58	1.0	3,693.62	
instance3377	3,600.29	?	3,650.62	2.0	3,980.51	
instance3819	0.23	unsat	17.04	5.0	243.13	5.0
instance3966	1,792.74	unsat	3,649.21	2.0	3,833.23	
instance4014	3,600.20	?	3,654.78	3.0	4,169.27	
instance4098	2,888.00	unsat	3,648.54	1.0	3,958.14	
instance4194	3,600.26	?	3,639.11	2.0	3,817.34	
instance4500	7.88	unsat	3,645.35	4.0	3,894.54	
instance4519	3,600.87	?	3,653.48	1.0	4,800.12	
instance4583	586.86	unsat	3,634.12	2.0	3,697.86	
instance4602	1.50	unsat	2,607.11	5.0	3,634.24	
instance4700	230.27	unsat	788.57	1.0	3,931.90	
instance4720	0.16	unsat	8.09	4.0	35.09	4.0
instance4725	1,535.46	unsat	1,943.55	1.0	3,725.86	
instance4759	9.80	unsat	3,656.99	5.0	4,543.62	
instance4763	8.50	unsat	923.55	2.0	4,091.60	
instance4779	165.15	unsat	790.19	1.0	3,742.10	
instance4953	3,600.60	?	3,646.18	1.0	3,988.69	
instance5018	3,600.27	?	3,650.85	1.0	3,951.99	
instance5043	683.22	unsat	2,195.01	1.0	3,756.32	
instance5363	425.01	unsat	427.64	1.0	3,629.78	
instance5486	711.54	unsat	1,568.08	1.0	3,701.19	
instance5621	1,515.91	unsat	3,633.32	1.0	3,749.45	
instance5706	21.65	unsat	3,651.30	3.0	3,723.70	
instance5756	304.33	unsat	463.77	1.0	3,747.08	
instance5833	645.56	unsat	1,358.31	1.0	3,863.16	
instance5965	1.27	unsat	9.00	2.0	139.42	2.0

Table 5: Results for finding any MCS. Time for unsat proof $t(p)$ and status are indicated. Run times (1h bound) for finding an MCS using ortools/Exact ($t(ortools)/t(Exact)$) and size of the MCS are given ($s(ortools)/s(Exact)$).

instance	$t(p)$	status	$t(ortools)$	$s(ortools)$	$t(Exact)$	$s(Exact)$
instance83	1.36	unsat	10.69	1.0	122.65	1.0
instance109	438.95	unsat	841.09	1.0	3,660.28	
instance100	2.16	unsat	28.41	1.0	3,638.75	
instance163	1.97	unsat	17.79	1.0	376.49	1.0
instance446	2.97	unsat	184.04	2.0	3,758.44	
instance456	0.35	unsat	10.30	4.0	90.16	4.0
instance512	3,600.28	?	3,609.58	1.0	3,649.50	
instance684	541.71	unsat	3,626.90	2.0	3,752.14	
instance753	0.85	unsat	8.14	2.0	73.06	2.0
instance1245	867.76	unsat	1,881.04	1.0	3,736.99	
instance1279	3,600.44	?	3,627.32	1.0	3,779.74	
instance1490	2,180.28	unsat	2,044.53	1.0	3,655.53	
instance1628	34.83	unsat	127.40	1.0	3,659.15	
instance1705	3,600.26	?	3,616.85	1.0	3,771.03	
instance1757	1.21	unsat	11.62	2.0	240.41	2.0
instance1798	27.90	unsat	54.83	1.0	3,654.20	
instance1807	0.30	unsat	6.64	4.0	31.08	4.0
instance1921	6.43	unsat	3,628.50	4.0	3,776.70	
instance1950	79.78	unsat	164.89	1.0	3,809.63	
instance2190	708.81	unsat	1,183.62	1.0	3,750.29	
instance2465	3,600.27	?	3,632.98	2.0	3,810.23	
instance3125	16.79	unsat	3,618.76	2.0	3,647.67	
instance3143	3,600.21	?	3,638.24	1.0	3,820.05	
instance3328	62.07	unsat	194.74	1.0	3,745.68	
instance3377	3,600.48	?	3,651.35	2.0	3,981.34	
instance3819	0.24	unsat	15.82	5.0	244.24	5.0
instance3966	1,662.00	unsat	3,649.93	3.0	3,927.78	
instance4014	3,600.18	?	3,654.98	3.0	4,196.56	
instance4098	3,010.58	unsat	3,574.67	1.0	4,065.44	
instance4194	3,600.25	?	3,639.54	2.0	3,828.13	
instance4500	8.30	unsat	3,645.81	4.0	3,824.86	
instance4519	3,600.80	?	3,654.92	1.0	4,778.65	
instance4583	989.98	unsat	3,633.96	2.0	3,745.19	
instance4602	1.71	unsat	2,575.39	5.0	3,634.24	
instance4700	294.78	unsat	834.78	1.0	3,924.59	
instance4720	0.16	unsat	7.23	4.0	35.16	4.0
instance4725	1,916.69	unsat	1,991.60	1.0	3,731.77	
instance4759	10.13	unsat	3,656.62	5.0	4,669.58	
instance4763	9.11	unsat	1,299.46	2.0	4,233.79	
instance4779	160.79	unsat	619.54	1.0	3,697.79	
instance4953	3,600.30	?	3,646.43	1.0	4,049.18	
instance5018	3,601.07	?	3,648.71	1.0	3,918.10	
instance5043	855.02	unsat	1,953.33	1.0	3,657.46	
instance5363	342.21	unsat	431.52	1.0	3,636.83	
instance5486	827.02	unsat	526.60	1.0	3,701.58	
instance5621	1,731.82	unsat	3,633.38	1.0	3,727.82	
instance5706	21.77	unsat	3,654.00	3.0	3,724.10	
instance5756	412.53	unsat	433.09	1.0	3,719.86	
instance5833	578.39	unsat	1,070.59	1.0	3,810.36	
instance5965	1.51	unsat	11.56	2.0	139.70	2.0

Table 6: Results for finding minimal MCS. Time for unsat proof $t(p)$ and status are indicated. Run times (1h bound) for finding an MCS using ortools/Exact ($t(ortools)/t(Exact)$) and size of the MCS are given ($s(ortools)/s(Exact)$).

instance	$t(p)$	status	$t(ortools)$	$s(ortools)$	$t(Exact)$	$s(Exact)$
instance83	1.20	unsat	10.37	1.0	105.33	1.0
instance109	437.01	unsat	842.96	1.0	3,658.63	
instance100	1.81	unsat	27.95	1.0	3,659.96	
instance163	1.85	unsat	18.91	1.0	322.91	1.0
instance446	2.84	unsat	45.45	2.0	3,696.33	
instance456	0.35	unsat	10.13	6.0	70.70	6.0
instance512	3,600.09	?	3,609.79	1.0	3,643.51	
instance684	512.51	unsat	3,627.37	2.0	3,777.99	
instance753	0.85	unsat	8.74	2.0	90.77	2.0
instance1245	635.10	unsat	1,657.73	1.0	3,812.10	
instance1279	3,600.29	?	3,627.37	1.0	3,697.04	
instance1490	1,893.08	unsat	2,155.51	1.0	3,645.06	
instance1628	50.25	unsat	127.58	1.0	3,666.80	
instance1705	3,600.19	?	3,616.84	1.0	3,741.20	
instance1757	1.11	unsat	13.47	2.0	285.83	2.0
instance1798	27.82	unsat	61.34	1.0	3,656.44	
instance1807	0.31	unsat	6.94	4.0	23.16	4.0
instance1921	6.89	unsat	3,628.75	4.0	3,727.96	
instance1950	110.40	unsat	199.59	1.0	3,718.99	
instance2190	740.06	unsat	1,262.69	1.0	3,861.75	
instance2465	3,600.21	?	3,633.35	2.0	3,815.25	
instance3125	26.93	unsat	3,618.70	2.0	3,708.89	
instance3143	3,600.24	?	3,637.85	1.0	3,842.77	
instance3328	59.13	unsat	197.43	1.0	3,686.99	
instance3377	3,600.97	?	3,653.18	2.0	4,016.43	
instance3819	0.23	unsat	14.54	6.0	357.10	6.0
instance3966	2,124.70	unsat	3,649.57	2.0	3,909.72	
instance4014	3,600.63	?	3,654.44	3.0	4,134.20	
instance4098	3,600.60	?	3,648.73	1.0	4,044.56	
instance4194	3,600.60	?	3,640.35	2.0	3,856.79	
instance4500	7.87	unsat	3,646.98	4.0	3,805.11	
instance4519	3,601.05	?	3,653.01	1.0	4,621.55	
instance4583	395.05	unsat	3,633.72	2.0	3,897.94	
instance4602	1.62	unsat	3,323.34	5.0	3,629.70	
instance4700	296.83	unsat	868.36	1.0	3,758.34	
instance4720	0.16	unsat	7.13	4.0	39.08	4.0
instance4725	1,447.76	unsat	2,672.91	1.0	3,757.00	
instance4759	10.33	unsat	3,656.25	5.0	4,589.43	
instance4763	8.79	unsat	3,433.60	2.0	4,016.62	
instance4779	141.52	unsat	557.44	1.0	3,748.71	
instance4953	3,600.87	?	3,647.38	1.0	4,082.17	
instance5018	3,601.07	?	3,652.01	1.0	4,046.49	
instance5043	817.22	unsat	1,015.01	1.0	3,717.21	
instance5363	394.43	unsat	308.77	1.0	3,629.61	
instance5486	790.87	unsat	2,124.69	1.0	3,680.65	
instance5621	1,697.96	unsat	3,633.66	1.0	3,763.71	
instance5706	19.75	unsat	3,651.94	3.0	3,782.00	
instance5756	363.46	unsat	555.99	1.0	3,696.68	
instance5833	498.66	unsat	1,189.38	1.0	3,741.04	
instance5965	1.48	unsat	10.63	2.0	163.60	2.0

Table 7: Results for finding minimal MCS without demand constraints. Time for unsat proof $t(p)$ and status are indicated. Run times (1h bound) for finding an MCS using ortools/Exact ($t(ortools)/t(Exact)$) and size of the MCS are given ($s(ortools)/s(Exact)$).

instance	$val(s^{\ell_{\text{dev}}})$	$val(s^{\text{ww}})$	$val(s^{\text{d}_{\max}})$	$val(s^{\text{d}_{\text{rms}}})$	$val(s^{\text{N}_{\text{ww}}})$	$val(s^{\text{N}_{>3}})$
instance1	5	7	5	8.11	7	0
instance2	10	6	3	7.44	6	5
instance3	2	12	5	14.38	12	0
instance4	24	10	5	11.52	11	0
instance5	8	6	4	8.19	9	2
instance6	8	5	4	6.07	7	3
instance7	20	18	6	22.91	18	0
instance8	6	4	3	8.06	6	0
instance9	16	12	2	23.75	18	1
instance10	1	12	3	18.02	15	0
instance11	10	23	5	26.32	23	0
instance12	14	12	4	15.55	12	0
instance13	3	18	4	20.84	18	0
instance14	5	9	4	10.90	11	0
instance15	35	45	8	53.69	55	2
instance16	0	20	4	24.12	21	0
instance17	1	22	4	26.97	22	0
instance18	0	30	4	39.91	30	0
instance19	56	85	4	101.00	85	0
instance20	11	120	5	139.86	120	0

Table 8: Best optimal values for soft constraints per instance over five runs of one hour. The optimal values $val(s)$ are given, where $val(s)$ indicates that optimality was proven and $val(s)$ indicates that only feasibility was proven within the time limit.

instance	$val(s^{\ell_{\text{dev}}})$	$val(s^{\text{ww}})$	$val(s^{\text{d}_{\max}})$	$val(s^{\text{d}_{\text{rms}}})$	$val(s^{\text{N}_{\text{ww}}})$	$val(s^{\text{N}_{>3}})$
instance1	16	7	5	8.11	7	0
instance2	17	6	4	7.48	6	5
instance3	12	13	5	14.38	13	2
instance4	24	11	5	11.52	11	0
instance5	8	9	4	8.19	9	2
instance6	8	7	4	6.07	7	3
instance7	20	18	6	22.91	18	1
instance8	6	6	5	8.06	6	2
instance9	20	18	4	23.76	18	5
instance10	18	16	4	18.04	16	1
instance11	33	24	6	26.90	24	0
instance12	48	12	4	15.55	12	0
instance13	19	18	5	20.84	18	0
instance14	13	11	4	10.90	11	0
instance15	47	60	15	56.61	60	10
instance16	24	21	5	24.14	21	0
instance17	27	22	5	26.99	22	0
instance18	20	31	6	39.91	31	2
instance19	132	91	9	104.52	91	0
instance20	96	124	10	141.61	124	0

Table 9: Best avg value combinations for soft constraints per instance over five runs of one hour. The optimal values $val(s)$ are given, where $val(s)$ indicates that optimality was proven and $val(s)$ indicates that only feasibility was proven within the time limit.

instance	$t(p)$	status	$t(or)$	$mcs(or)$	$t(ex)$	$mcs(ex)$
instance1	29.91	opt				
instance2	26.05	opt				
instance3	3,600.12	sat				
instance4	75.31	opt				
instance5	27.91	opt				
instance6	1.30	opt				
instance7	3,600.17	sat				
instance8	2.62	unsat	12.71	$[v_{\ell_{\text{dev}}}]$	3,640.79	
instance9	3,600.24	?	3,628.35	$[v_{\ell_{\text{dev}}}]$	3,758.40	
instance10	3,600.22	sat				
instance11	3,600.18	sat				
instance12	3,600.13	sat				
instance13	3,600.12	sat				
instance14	2,946.90	opt				
instance15	3,600.26	?	3,639.08		3,711.77	
instance16	3,600.14	sat				
instance17	3,600.15	sat				
instance18	3,600.26	?	3,632.65	$[v_{\text{Nww}}]$	3,727.32	
instance19	3,600.57	?	3,669.66	$[v_{\ell_{\text{dev}}}, v_{\text{d}_{\text{rms}}}, v_{\text{ww}}, v_{\text{Nww}}]$	3,915.38	
instance20	3,600.50	?	3,697.52	$[v_{\text{d}_{\text{rms}}}, v_{\text{d}_{\text{max}}}, v_{\text{Nww}}, v_{\ell_{\text{dev}}}, v_{\text{ww}}]$	4,168.26	

Table 10: Results for finding minimal MCS for reduced max night shift ($s^{\text{N}>3}$) constraint. Time for unsat proof $t(p)$ and status are indicated. Run times for finding an MCS using ortools/Exact ($t(ortools)/t(Exact)$) and MCS are given ($mcs(or)/mcs(ex)$).

instance	$t(p)$	status	$t(or)$	$mcs(or)$	$t(ex)$	$mcs(ex)$
instance1	6.57	unsat	20.06	$[v_{\mathrm{d_{max}}}]$	128.62	$[v_{\mathrm{d_{max}}}]$
instance2	21.23	unsat	20.61	$[v_{\mathrm{d_{max}}}]$	882.49	$[v_{\mathrm{d_{max}}}]$
instance3	335.55	unsat	352.66	$[v_{\mathrm{d_{max}}}]$	3,654.62	
instance4	86.74	opt				
instance5	23.24	opt				
instance6	1.10	opt				
instance7	3,600.23	sat				
instance8	64.88	opt				
instance9	3,600.21	?	3,629.73	$[v_{\ell_{\mathrm{dev}}}]$	3,749.75	
instance10	3,600.21	?	3,616.48	$[v_{\mathrm{d_{rms}}}, v_{\mathrm{d_{max}}}]$	3,764.59	
instance11	3,600.48	?	3,617.31	$[v_{\ell_{\mathrm{dev}}}]$	3,647.22	
instance12	3,600.17	?	3,609.62	$[v_{\ell_{\mathrm{dev}}}]$	3,636.34	
instance13	3,600.64	?	3,613.94	$[v_{\mathrm{d_{max}}}]$	3,687.95	
instance14	824.57	unsat	1,719.29	$[v_{\mathrm{d_{max}}}]$	3,660.23	
instance15	3,600.38	?	3,639.05		3,717.20	
instance16	493.70	unsat	3,616.94	$[v_{\mathrm{Nww}}, v_{\mathrm{d_{max}}}, v_{\mathrm{ww}}, v_{\mathrm{d_{rms}}}]$	3,670.72	
instance17	3,600.26	?	3,615.76	$[v_{\mathrm{ww}}, v_{\mathrm{Nww}}, v_{\mathrm{d_{max}}}, v_{\mathrm{d_{rms}}}]$	3,684.34	
instance18	3,600.20	?	3,632.99	$[v_{\ell_{\mathrm{dev}}}]$	3,792.49	
instance19	3,600.60	?	3,669.99	$[v_{\mathrm{ww}}, v_{\mathrm{d_{max}}}, v_{\mathrm{Nww}}, v_{\ell_{\mathrm{dev}}}, v_{\mathrm{d_{rms}}}]$	3,789.42	
instance20	3,600.85	?	3,698.52	$[v_{\mathrm{d_{rms}}}, v_{\mathrm{d_{max}}}, v_{\mathrm{Nww}}, v_{\ell_{\mathrm{dev}}}, v_{\mathrm{ww}}]$	3,857.27	

Table 11: Results for finding minimal MCS for reduced length deviation ($s^{\ell_{dev}}$) constraint. Time for unsat proof $t(p)$ and status are indicated. Run times for finding an MCS using ortools/Exact ($t(ortools)/t(Exact)$) and MCS are given ($mcs(or)/mcs(ex)$).

instance	$t(p)$	status	$t(or)$	$mcs(or)$	$t(ex)$	$mcs(ex)$
instance1	39.86	opt				
instance2	23.76	opt				
instance3	3,600.64	sat				
instance4	87.81	opt				
instance5	30.27	opt				
instance6	0.86	opt				
instance7	3,600.26	sat				
instance8	55.96	opt				
instance9	3,600.23	?	3,628.42	$[v_{\ell_{\text{dev}}}]$	3,806.40	
instance10	3,600.26	sat				
instance11	3,600.23	?	3,617.45	$[v_{\ell_{\text{dev}}}]$	3,684.54	
instance12	3,600.15	sat				
instance13	3,600.39	sat				
instance14	2,588.11	opt				
instance15	3,600.24	?	3,638.53	$[v_{\text{d}_{\max}}, v_{\text{ww}}]$	3,741.26	
instance16	3,600.18	sat				
instance17	3,600.15	sat				
instance18	3,600.23	?	3,632.78	$[v_{\ell_{\text{dev}}}]$	3,796.35	
instance19	3,600.44	?	3,669.27	$[v_{\text{d}_{\max}}, v_{\text{ww}}, v_{\text{d}_{\text{rms}}}, v_{\text{Nww}}]$	3,849.75	
instance20	3,600.57	?	3,698.47		4,186.24	

Table 12: Results for finding minimal MCS for reduced working weekends (s^{ww}) constraint. Time for unsat proof $t(p)$ and status are indicated. Run times for finding an MCS using ortools/Exact ($t(ortools)/t(Exact)$) and MCS are given ($mcs(or)/mcs(ex)$).

instance	$t(p)$	status	$t(or)$	$mcs(or)$	$t(ex)$	$mcs(ex)$
instance1	30.90	opt				
instance2	0.77	unsat	6.85	$[v_{\ell_{\text{dev}}}, v_{\text{N}_{>3}}]$	149.76	$[v_{\ell_{\text{dev}}}, v_{\text{N}_{>3}}]$
instance3	3,600.55	?	3,610.24	$[v_{\ell_{\text{dev}}}]$	3,675.97	
instance4	87.67	opt				
instance5	29.72	opt				
instance6	1.13	opt				
instance7	3,600.18	sat				
instance8	2.91	unsat	17.62	$[v_{\text{d}_{\text{rms}}}, v_{\ell_{\text{dev}}}, v_{\text{Nww}}]$	2,786.49	$[v_{\text{d}_{\text{rms}}}, v_{\ell_{\text{dev}}}, v_{\text{Nww}}]$
instance9	323.86	unsat	3,628.98	$[v_{\text{N}_{>3}}, v_{\ell_{\text{dev}}}]$	3,839.06	
instance10	3,600.20	?	3,616.81	$[v_{\ell_{\text{dev}}}]$	4,012.93	
instance11	3,600.28	?	3,617.27	$[v_{\ell_{\text{dev}}}]$	3,655.37	
instance12	3,145.23	opt				
instance13	3,601.03	?	3,613.72	$[v_{\ell_{\text{dev}}}]$	3,669.95	
instance14	1,586.60	opt				
instance15	3,600.43	?	3,638.64		3,731.92	
instance16	3,600.21	?	3,617.13	$[v_{\ell_{\text{dev}}}]$	3,846.99	
instance17	3,600.21	?	3,615.77	$[v_{\ell_{\text{dev}}}]$	3,643.36	
instance18	3,600.23	?	3,633.06	$[v_{\text{ww}}, v_{\text{Nww}}, v_{\text{d}_{\text{rms}}}]$	3,692.31	
instance19	3,600.44	?	3,669.52	$[v_{\ell_{\text{dev}}}, v_{\text{ww}}, v_{\text{d}_{\text{rms}}}, v_{\text{Nww}}, v_{\text{d}_{\text{max}}}]$	3,846.23	
instance20	3,600.50	?	3,697.49	$[v_{\text{Nww}}, v_{\text{d}_{\text{rms}}}, v_{\ell_{\text{dev}}}, v_{\text{ww}}, v_{\text{d}_{\text{max}}}]$	3,973.54	

Table 13: Results for finding minimal MCS for reduced weekend distance ($s^{\text{d}_{\text{max}}}$) constraint. Time for unsat proof $t(p)$ and status are indicated. Run times for finding an MCS using ortools/Exact ($t(ortools)/t(Exact)$) and MCS are given ($mcs(or)/mcs(ex)$).

instance	$t(p)$	status	$t(or)$	$mcs(or)$	$t(ex)$	$mcs(ex)$
instance1	30.71	opt				
instance2	0.80	unsat	6.86	$[v_{\ell_{\text{dev}}}, v_{\text{N}_{>3}}]$	181.93	$[v_{\ell_{\text{dev}}}, v_{\text{N}_{>3}}]$
instance3	3,601.06	sat				
instance4	63.35	opt				
instance5	17.75	opt				
instance6	1.26	opt				
instance7	3,600.17	sat				
instance8	63.11	opt				
instance9	3,600.39	?	3,628.36	$[v_{\text{N}_{>3}}, v_{\ell_{\text{dev}}}]$	3,789.98	
instance10	3,600.19	?	3,616.60	$[v_{\ell_{\text{dev}}}]$	3,855.50	
instance11	3,600.61	?	3,618.11	$[v_{\text{d}_{\text{rms}}}]$	3,680.59	
instance12	3,600.12	sat				
instance13	3,600.18	?	3,613.75	$[v_{\ell_{\text{dev}}}]$	3,639.29	
instance14	1,166.20	opt				
instance15	3,600.25	?	3,638.32		3,689.52	
instance16	3,600.19	?	3,616.59	$[v_{\ell_{\text{dev}}}]$	3,906.66	
instance17	3,600.25	?	3,615.67	$[v_{\ell_{\text{dev}}}]$	3,662.81	
instance18	3,600.26	?	3,632.85	$[v_{\text{d}_{\text{rms}}}, v_{\ell_{\text{dev}}}]$	3,730.29	
instance19	3,600.41	?	3,669.24	$[v_{\text{ww}}, v_{\text{Nww}}, v_{\text{d}_{\text{max}}}, v_{\ell_{\text{dev}}}, v_{\text{d}_{\text{rms}}}]$	3,936.24	
instance20	3,600.49	?	3,701.61	$[v_{\ell_{\text{dev}}}, v_{\text{d}_{\text{max}}}, v_{\text{Nww}}, v_{\text{d}_{\text{rms}}}, v_{\text{ww}}]$	4,140.76	

Table 14: Results for finding minimal MCS for reduced root mean squared weekend distance ($s^{\text{d}_{\text{rms}}}$) constraint. Time for unsat proof $t(p)$ and status are indicated. Run times for finding an MCS using ortools/Exact ($t(ortools)/t(Exact)$) and MCS are given ($mcs(or)/mcs(ex)$).

instance	$t(p)$	status	$t(or)$	$mcs(or)$	$t(ex)$	$mcs(ex)$
instance1	40.42	opt				
instance2	18.30	opt				
instance3	3,600.31	?	3,610.19	$[v_{\mathrm{d_{max}}}]$	3,678.52	
instance4	77.15	opt				
instance5	32.26	opt				
instance6	2.75	opt				
instance7	3,600.43	sat				
instance8	68.05	opt				
instance9	3,600.43	?	3,628.69	$[v_{\ell_{\mathrm{dev}}}]$	3,815.23	
instance10	3,600.13	sat				
instance11	3,600.22	?	3,617.59	$[v_{\mathrm{Nww}}]$	3,668.63	
instance12	3,600.13	sat				
instance13	3,600.14	sat				
instance14	2,505.51	opt				
instance15	3,600.32	?	3,638.81		3,770.18	
instance16	3,600.17	sat				
instance17	3,600.14	sat				
instance18	3,600.29	?	3,632.85	$[v_{\mathrm{Nww}}]$	3,732.25	
instance19	3,600.43	?	3,669.42	$[v_{\mathrm{d_{max}}}, v_{\mathrm{d_{rms}}}, v_{\mathrm{Nww}}, v_{\ell_{\mathrm{dev}}}, v_{\mathrm{ww}}]$	3,810.38	
instance20	3,600.48	?	3,697.94	$[v_{\mathrm{d_{max}}}, v_{\ell_{\mathrm{dev}}}, v_{\mathrm{ww}}, v_{\mathrm{Nww}}, v_{\mathrm{d_{rms}}}]$	4,162.70	

Table 15: Results for finding minimal MCS for reduced working weekends with friday night (s^{Nww}) constraint. Time for unsat proof $t(p)$ and status are indicated. Run times for finding an MCS using ortools/Exact ($t(ortools)/t(Exact)$) and MCS are given ($mcs(or)/mcs(ex)$).

List of Figures

List of Tables

List of Algorithms

Bibliography

Agrawal, Jagriti, Amruta Yelamanchili, and Steve Chien (2020). "Using Explainable Scheduling for the Mars 2020 Rover Mission". In: *arXiv preprint arXiv:2011.08733*.

Amine, Khalil (2019). "Multiobjective simulated annealing: Principles and algorithm variants". In: *Advances in Operations Research* 2019.1.

Arlinghaus, Anna et al. (2019). "Working Time Society consensus statements: Evidence-based effects of shift work and non-standard working hours on workers, family and community". In: *Industrial Health* 57.2, pp. 184–200.

Bailey, James and Peter J. Stuckey (2005). "Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization". In: *Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages (PADL)*, pp. 174–186.

Baker, Kenneth R. (1976). "Workforce allocation in cyclical scheduling problems: A survey". In: *Journal of the Operational Research Society* 27.1, pp. 155–167.

Becker, Tristan, Maximilian Schiffer, and Grit Walther (2022). "A general branch-and-cut framework for rotating workforce scheduling". In: *INFORMS Journal on Computing* 34.3, pp. 1548–1564.

Bleukx, Ignace, Tias Guns, and Dimos Tsouros (Feb. 2024). *Explainable Constraint Solving: A hands-on tutorial*. Version v1.0. URL: https://doi.org/10.5281/zenodo.10694139.

Chu, Geoffrey et al. (2018). *Chuffed, a lazy clause generation solver*. URL: URL:%20https://github.%20com/chuffed/chuffed.

Čyras, Kristijonas, Myles Lee, and Dimitrios Letsios (2021). "Schedule explainer: An argumentation-supported tool for interactive explanations in makespan scheduling". In: *Proceedings of the 3rd International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems (EXTRAAMAS)*, pp. 243–259.

Čyras, Kristijonas, Dimitrios Letsios, et al. (2019). "Argumentation for explainable scheduling". In: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pp. 2752–2759.

Czyżak, Piotr and Andrzej Jaszkiewicz (1997). "Pareto simulated annealing". In: *Proceedings of the 12th International Conference on Multiple Criteria Decision Making (MCDM)*, pp. 297–307.

Dall'Ora, Chiara et al. (2016). "Characteristics of shift work and their impact on employee performance and wellbeing: A literature review". In: *International journal of nursing studies* 57, pp. 12–27.

De Bruecker, Philippe et al. (2015). "Workforce planning incorporating skills: State of the art". In: *European Journal of Operational Research* 243.1, pp. 1–16.

De Moura, Leonardo and Nikolaj Bjørner (2008). "Z3: An efficient SMT solver". In: *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 337–340.

Deb, Kalyanmoy et al. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 182–197.

Drexl, Andreas and Yury Nikulin (2008). "Multicriteria airport gate assignment and Pareto simulated annealing". In: *IIE Transactions* 40.4, pp. 385–397.

Duh, Jiunn-Der and Daniel G. Brown (2007). "Knowledge-informed Pareto simulated annealing for multi-objective spatial allocation". In: *Computers, Environment and Urban Systems* 31.3, pp. 253–281.

Elffers, Jan and Jakob Nordström (2018). "Divide and Conquer: Towards Faster Pseudo-Boolean Solving." In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1291–1299.

Ernst, Andreas T. et al. (2004). "Staff scheduling and rostering: A review of applications, methods and models". In: *European Journal of Operational Research* 153.1, pp. 3–27.

Folkard, Simon and David A. Lombardi (2006). "Modeling the impact of the components of long work hours on injuries and "accidents"". In: *American journal of industrial medicine* 49.11, pp. 953–963.

Folkard, Simon, Karen A. Robertson, and Mick B. Spencer (2007). "A Fatigue/Risk Index to assess work schedules". In: *Somnologie* 11.3, pp. 177–185.

Gärtner, Johannes et al. (2018). "Scheduling matters-some potential requirements for future rostering competitions from a practitioner's view". In: *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pp. 33–42.

Gleeson, John and Jennifer Ryan (1990). "Identifying minimally infeasible subsystems of inequalities". In: *ORSA Journal on Computing* 2.1, pp. 61–63.

Guns, Tias (2019). "Increasing modeling language convenience with a universal n-dimensional array, CPpy as python-embedded example". In: *Proceedings of the 18th workshop on Constraint Modelling and Reformulation (ModRef) at CP 2019*.

Hamm, Matthias, Ulrike Beißert, and Markus König (2009). "Simulation-based optimization of construction schedules by using pareto simulated annealing". In: *Proceedings of the 18th International Colloquium on the Application of Computer Science and Mathematics in Architecture and Civil Engineering (ICRAMCS)*.

Health and Safety Executive (2006). *Managing shiftwork-Health and safety guidance*.

Ignatiev, Alexey, Antonio Morgado, and Joao Marques-Silva (2018). "PySAT: A Python Toolkit for Prototyping with SAT Oracles". In: *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pp. 428–437.

Jarosław, Pempera, Smutnicki Czesław, and Żelazny Dominik (2013). "Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm". In: *Procedia Computer Science* 18, pp. 936–945.

Kirkpatrick, Scott, Daniel Gelatt, and Mario P. Vecchi (1983). "Optimization by Simulated Annealing". In: *Science* 220.4598, pp. 671–680.

Kletzander, Lucas, Nysret Musliu, Johannes Gärtner, et al. (2019). "Exact methods for extended rotating workforce scheduling problems". In: *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 519–527.

Kletzander, Lucas, Nysret Musliu, and Kate Smith-Miles (2021). "Instance space analysis for a personnel scheduling problem". In: *Annals of Mathematics and Artificial Intelligence* 89, pp. 617–637.

Leo, Kevin and Guido Tack (2017). "Debugging unsatisfiable constraint models". In: *Proceedings of the 14th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pp. 77–93.

Liffiton, Mark H. and Ammar Malik (2013). "Enumerating infeasibility: Finding multiple MUSes quickly". In: *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pp. 160–175.

Liffiton, Mark H. and Karem A. Sakallah (2005). "On finding all minimally unsatisfiable subformulas". In: *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pp. 173–186.

— (2008). "Algorithms for computing minimal unsatisfiable subsets of constraints". In: *Journal of Automated Reasoning* 40, pp. 1–33.

— (2009). "Generalizing core-guided Max-SAT". In: *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pp. 481–494.

Lock, Alistar M., Darrell L. Bonetti, and A.D.K. Campbell (2018). "The psychological and physiological health effects of fatigue". In: *Occupational Medicine* 68.8, pp. 502–511.

Martínez Gasca, Rafael et al. (2007). "NMUS: Structural Analysis for Improving the Derivation of All MUSes in Overconstrained Numeric CSPs". In: *Proceedings of the 12th Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, pp. 160–169.

Mischek, Florian and Nysret Musliu (2024). "Preference Explanation and Decision Support for Multi-Objective Real-World Test Laboratory Scheduling". In: *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 378–386.

Moreno, Claudia R.C. et al. (2019). "Working Time Society consensus statements: Evidence-based effects of shift work on physical and mental health". In: *Industrial Health* 57.2, pp. 139–157.

Musliu, Nysret, Johannes Gärtner, and Wolfgang Slany (2002). "Efficient generation of rotating workforce schedules". In: *Discrete Applied Mathematics* 118.1-2, pp. 85–98.

Musliu, Nysret, Andreas Schutt, and Peter J. Stuckey (2018). "Solver independent rotating workforce scheduling". In: *Proceedings of the 15th International Conference on International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pp. 429–445.

Nethercote, Nicholas et al. (2007). "MiniZinc: Towards a standard CP modelling language". In: *proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 529–543.

Ortell, Katherine K., Pawel M. Switonski, and Joe Ryan Delaney (2019). "FairSubset: A tool to choose representative subsets of data for use with replicates or groups of different sample sizes". In: *Journal of Biological Methods* 6.3.

Perron, Laurent and Vincent Furnon (May 7, 2024). *OR-Tools*. Version v9.11. Google. URL: `https://developers.google.com/optimization/`.

Petrovic, Sanja, Jane Parkin, and David Wrigley (2021). "Personnel scheduling considering employee well-being: insights from case studies". In: *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. Vol. 1, pp. 10–23.

Rosekind, Mark et al. (2010). "The Cost of Poor Sleep: Workplace Productivity Loss and Associated Costs". In: *Journal of Occupational and Environmental Medicine / American College of Occupational and Environmental Medicine* 52, pp. 91–98.

Suresh, R.K. and K.M. Mohanasundaram (2006). "Pareto archived simulated annealing for job shop scheduling with multiple objectives". In: *The International Journal of Advanced Manufacturing Technology* 29, pp. 184–196.

Van den Bergh, Jorne et al. (2013). "Personnel scheduling: A literature review". In: *European Journal of Operational Research* 226.3, pp. 367–385.

Van Loon, J.N.M. (1981). "Irreducibly inconsistent systems of linear inequalities". In: *European Journal of Operational Research* 8.3, pp. 283–288.