TU WIEN Informatics

Automated Solution Methods for Complex Real-life Personnel Scheduling Problems

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Lucas Kletzander, BSc
Matrikelnummer 01225758

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Dr.techn. Nysret Musliu

Diese Dissertation haben begutachtet:

Elina Rönnberg	Hana Rudová

Wien, 5. September 2022

Lucas Kletzander

TU WIEN Informatics

Automated Solution Methods for Complex Real-life Personnel Scheduling Problems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Lucas Kletzander, BSc
Registration Number 01225758

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Dipl.-Ing. Dr.techn. Nysret Musliu

The dissertation has been reviewed by:

Elina Rönnberg	Hana Rudová

Vienna, 5th September, 2022

Lucas Kletzander

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

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Lucas Kletzander, 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, 5. September 2022

Lucas Kletzander

Acknowledgements

I would like to express my gratitude towards everyone who supported me during my PhD and the work on this thesis, especially my adviser Privatdoz. Dr. Nysret Musliu, for the constructive and supportive collaboration, advice and feedback, as well as to him and my colleagues both at the university and our industry partner for a very pleasant and positive working environment with interesting discussions.

I also want to thank my family for their ongoing encouragement, my friends for their continued support, and my reviewers for their time and valuable feedback.

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

Kurzfassung

Optimierungsprobleme in realen Anwendungen sind oft sehr schwer zu modellieren und lösen, da viele verschiedene Bedingungen und Ziele zu berücksichtigen sind, die aus unterschiedlichen Quellen kommen und unter Umständen häufig geändert und angepasst werden müssen. Dafür werden oft Lösungen speziell für eine bestimmte Version eines Problems erzeugt. Obwohl solche maßgeschneiderten Lösungen wichtig sind, um etwa das Erreichen einer optimalen Lösung zu beweisen oder Grenzen für den Abstand zum Optimum zu bekommen, ist es kaum möglich oder erfordert extremen Aufwand, diese für andere Probleme anzuwenden. Dies zeigt einen Bedarf für allgemeine Lösungsmethoden, die mit geringem Anpassungsaufwand für verschiedene Probleme angewandt werden können.

Diese Arbeit trägt sowohl zum Stand der Technik für problemspezifische als auch allgemeine Methoden für komplexe reale Beispielprobleme aus dem Anwendungsbereich der Personalzeitplanung bei, einem sehr herausfordernden Aufgabengebiet, in dem die Lösungen große Auswirkungen auf Leben und Wohlbefinden der Arbeiter*innen ebenso wie auf die Profite der Unternehmen haben.

Der Beitrag unter Verwendung des ersten Beispielproblems inkludiert die Erweiterung des rotierenden Schichtplanungsproblems, wo fixe Schichten einem rotierenden Plan zugeteilt werden müssen, mit zusätzlichen Bedingungen und Optimierungszielen aus der Praxis und einen exakten Lösungsansatz unter Verwendung eines Lösers für Constraint Programming. Weiter wird eine detailierte Analyse der verfügbaren Instanzen sowie der Stärken und Schwächen verschiedener Lösungsansätze mittels Instanzraumanalyse durchgeführt, wodurch ein neues, besser verteiltes Set von Instanzen für das Problem erzeugt wurde.

Im Aufgabengebiet des Designs minimaler Schichten, wo ein vorgegebener Bedarf mit Schichten unter bestimmten Einschränkungen so abgedeckt werden soll, dass auch die Anzahl unterschiedlicher Schichten minimiert wird, zeigt diese Arbeit einen Vergleich verschiedener neuer Modellierungsoptionen inklusive einem Model basierend auf dem Fluss durch ein Netzwerk, mit dem alle Instanzen des öffentlichen Vergleichssets in kurzer Zeit optimal gelöst werden können.

Mit dem Busfahrerplanungsproblem basierend auf einem österreichischen Kollektivvertrag bietet diese Arbeit die formale Definition eines neuen Problems mit sehr komplexen

Bedingungen für Pausen im Vergleich zu anderen Arbeiten in der Literatur. Dies beinhaltet eine detailierte Analyse der Problemeigenschaften sowie ein neues Vergleichsset von Instanzen basierend auf den Eigenschaften realer Instanzen. Lösungen werden zunächst mit Metaheuristiken erzielt, dann mit der Lösungsmethode Branch and Price, wodurch hochqualitative Lösungen und allgemein anwendbare Optimierungen für das hochdimensionale Unterproblem erreicht werden.

Auf der Seite der allgemeinen Methoden wird die Architektur für ein intervallbasiertes Gerüst präsentiert, in dem alle drei Beispielprobleme, aber auch andere Probleme, in einem modularen Rahmen dargestellt werden können, der es ermöglicht, Bedingungen flexibel auszutauschen und Algorithmen auf unterschiedliche Probleme mit geringem Anpassungsaufwand anzuwenden.

Wir verwenden dieses Gerüst um mehrere neue Heuristiken für die drei Probleme als algorithmische Bausteine einzuführen, die dann mit der allgemeinen Methode der Hyper-Heuristiken verwendet werden können, die aus diesen Bausteinen dynamisch neue Lösungsmethoden generiert. Damit werden mehrere Hyper-Heuristiken auf dem Stand der Technik evaluiert, die gute Ergebnisse bieten und gleichzeitig allgemein anwendbar sind.

Schließlich wird die neue Hyper-Heuristik LAST-RL präsentiert, die verstärkendes Lernen basierend auf einer umfangreichen Repräsentation des aktuellen Zustands der Suche in Kombination mit einer Erkundungsstrategie basierend auf der Methode Iterated Local Search verwendet. Damit werden sehr gute Ergebnisse auf den sechs Domänen eines Hyper-Heuristik-Wettbewerbs erreicht und die Ergebnisse anderer Methoden übertroffen, die ebenfalls auf verstärkendem Lernen basieren. Die neue Methode wird letztendlich erfolgreich auf den drei realen Anwendungsbereichen dieser Arbeit angewandt, wo neue beste Ergebnisse für zwei der drei Bereiche erzielt werden. Dies zeigt großes Potential für die Anwendung solcher allgemeiner Methoden für komplexe reale Problemstellungen.

Abstract

Optimization problems in real-life scenarios are often difficult to model and solve, as they have various different constraints and objectives coming from different sources, and these might require frequent changes and adaptations. This often leads to the requirement for highly customized solutions that are specifically made for a particular version of a problem. While such tailored solutions are important, e.g., to obtain proven optimal or bounded solutions, they can often not be applied to other problems or require extensive adaptation effort, which raises the need for general solution methods that can be applied to different problem variants or entirely different problems with minimum adaptation effort.

This thesis contributes to the state of the art both regarding problem-specific methods and general methods for complex real-life problems from the application area of employee scheduling, a very challenging domain where the solutions have a huge impact on the lives and well-being of employees as well as the profits of companies.

The contribution using the first example problem domain includes the extension of the Rotating Workforce Scheduling problem, where fixed shifts need to be assigned to a rotating schedule, with additional practical constraints and optimization goals, an exact solution approach using a Constraint Programming solver, and a detailed analysis of available instances as well as strengths and weaknesses of different solution approaches using Instance Space Analysis, leading to a new better distributed set of benchmark instances for the problem.

For the domain of Minimum Shift Design, where a given demand needs to be covered with a minimal set of shifts that adhere to some restrictions, this thesis provides a comparison of different novel modelling options including a network flow based model that allows to solve all instances of a public benchmark set to optimality in short time.

With Bus Driver Scheduling based on the Austrian collective agreement, the thesis provides the formalization of a new problem that has very complex break constraints compared to literature, including a detailed analysis of problem characteristics and a new benchmark data set based on features of real-world instances. Solutions are first obtained with metaheuristics, followed by a Branch and Price approach that both provides high-quality solutions and introduces optimizations that are generally applicable for high-dimensional subproblems.

On the side of general methods, the thesis provides the architecture of an interval-based framework that allows to model the three example problems above, but also further problems, in a common modular framework that allows to flexibly exchange constraints and apply algorithms to different problems with minimum adaptation effort.

We use this framework to introduce several new Low-Level Heuristics, which are algorithmic building blocks, for the three example domains, in order to be able to apply the general methodology of hyper-heuristics, which use these building blocks to create a solution method for an instance of a problem on the fly. With the new Low-Level Heuristics in place, a range of state-of-the-art hyper-heuristics are evaluated on these domains, providing very good results while being generally applicable.

Finally, the thesis introduces the new hyper-heuristic LAST-RL, which uses reinforcement learning based on a rich representation of the search state in combination with an exploration strategy based on Iterated Local Search. This new hyper-heuristic performs very well on an evaluation using six domains from a hyper-heuristic competition, outperforming previous approaches based on reinforcement learning. The new approach is finally applied successfully to the three real-life domains, providing new best results for two of those domains, which shows the great potential of applying such general methods for complex real-life domains.

Contents

CHAPTER 1

Introduction

Efficiently solving real-life optimization problems is notoriously difficult. Typically, a wide range of different constraints needs to be respected, coming from various sources like legislation, the working environment, company policies, or employee requirements. Moreover, these constraints are often hard to identify, as they are sometimes not written down at all, or at least not in a formal definition but rather in textual descriptions, and often have to be collected from various sources. This can include different stakeholders in the problem like company management, workforce representatives, and legal consultants, as well as human planners who include implicit constraints based on their experience. Further, not only constraints, but also optimization objectives can be ambiguous, difficult to define, and often result in multi-objective criteria.

These difficult scenarios often lead to the issue that no standard solution can be applied to such a problem since too many specialized factors need to be respected, and a custom-built solution is applied. Usually such a solution needs many rounds of constraint and objective refinement as the original formalizations are often not precise enough or contain misunderstandings. This requires a very time-consuming and therefore also costly process that might end with a highly specialized solution method that is adapted to provide high-quality solutions to a very specific problem. However, these solutions can only be applied in practice if the formalization process is executed well enough that the new solution method actually solves the problem that is required in real life.

Sometimes it is very important to have a high-quality problem-specific solution, e.g., to efficiently solve a problem to optimality or at least get very good solutions with known bounds to the optimum. Creating a solution that deals with a very specific problem, however, leads to another issue that is very debilitating in practice. Often a large amount of time is spent on generating a solution method that is only applicable to a very narrow problem formulation. However, requirements can easily change, leading to difficulties adapting the existing methods to a different scenario or even to a slight modification of the scenario. This problem highlights that more general methods, that can be adapted to

changing problem formulations or even different problems, would be highly beneficial to reduce the effort necessary for new solutions and greatly increase the benefits of method improvements.

One very large domain of problems that can come in many different, highly complex scenarios in practice, is personnel scheduling. In many professions different shifts are needed to cover varying requirements including areas like health care, protection services, transportation, manufacturing, call centres, and many more. Related problems can surface in many shapes. The demand can be based on shifts that are required to cover 24/7 operations like in a hospital, or it might be based on the expected number of incoming calls at different times of the day in a call center, and in public transport personnel might need to drive vehicles according to a fixed timetable. These different demands need to be fulfilled while conforming to a wide range of constraints based on legal requirements, collective agreements, and company policies.

Shift work deeply affects the lives of many employees. In Austria, according to the statistics published by the European Union [Eur], almost 20 % of the employees work in shifts (18.7 % in 2020), which is just slightly above the EU average of 17.7 %. In some other European countries this figure rises as high as almost 40 %, affecting millions of people for the majority of their life.

On the one hand, shift work is often very difficult to align with a healthy social life [ABIG+19], since irregular working hours can reduce the options for contact with family and friends working in regular hours or negatively impact child care. Further, many studies found negative health impacts [MMS+19], including strong links to a higher risk for cardiovascular diseases, gastrointestinal and metabolic disorders, and, to a smaller extent, also links to higher risk for cancer, mental health problems and reproduction issues. These factors make it imperative to consider rules that aim at reducing the negative impacts of shift work on employees. While some rules come in the form of legal requirements, in each specific application employee satisfaction criteria should be considered to provide a healthy work environment.

On the other hand personnel cost is one of the largest cost factors for the employers [VdBBDB+13], making it necessary to find and use optimized schedules in order to stay competitive. Therefore, cost is the main optimization goal for a majority of the personnel scheduling problems in literature. When combining the importance of the cost-related and ergonomic requirements, this can result in more complex optimization problems with multiple, sometimes opposing objectives that are required to find a balance that is cost-effective and satisfies the needs of the employees.

With the high impact and the complex setting that arises when dealing with employee scheduling problems, it is no surprise that a large body of research already exists. However, these results can be hard to apply in practice [KW07, PVB12, Pet19], especially when not all needs of the different stakeholders are met in problem formulations that abstract too much from the complexity of the real-world setting. In such scenarios, manual processes for solution generation or only partially automated systems that cannot deal with the

full complexity of the real-world situation are frequently encountered, indicating the need for further progress in the area.

Therefore, the aim of this thesis is to advance the knowledge and methodology to deal with complex real-life problem formulations. This is done by formalizing different problems in the area of personnel scheduling as an exemplary application domain, including their full scope of practically relevant constraints, designing solution methods to advance the state of the art for individual problems, and investigating the spectrum of methods by transitioning from problem-specific to problem-independent solution methods that can be applied to various problems with minimal adaptation effort.

1.1 Research Goals

This section provides the detailed sub-goals on the way to advance the field of generally applicable optimization methods for complex real-life problems based on the study of several such complex problems in the area of personnel scheduling.

1.1.1 Research Goal 1: Formal Specification of Complex Real-Life Problems

Problems in the real world are often complex and notoriously hard to put into a mathematical form that is unambiguous and fully represents the interests of different stakeholders of the problem. Rules might come from different laws, collective agreements, and company policies. These are usually not written with a mathematical formulation in mind, leading to formalization difficulties and different interpretation possibilities. Further rules might only exist in the mind of a human planner or are implicitly assumed to always hold without formally mentioning them.

Simpler rule sets might be easier to compare to other methods and benchmarks in literature, making it tempting to simplify and generalize the real-life scenario. However, this can easily lead to solutions that are not actually applicable in practice and greatly hinder the transfer of well-designed solution methods to industry, simply because the solution is not designed for the right problem.

In order to deal with problem formulations that actually allow to transfer the results to practice, and therefore to transfer the benefits to both the companies and the employees, the work in this thesis is conducted on problems provided by our industry partner. The specifics of this research goal are as follows:

- Investigate existing problem formulations to append them with additional constraints that are relevant in practice but not yet considered in literature.
- Formalize new problem domains that considerably differ from previous literature including all relevant constraints from practice.

- Include both the needs of the employers and the employees in the objectives that are optimized in order to deal with both the financial and the employee health concerns.

1.1.2 Research Goal 2: Solution Methods Improving the State of the Art

For the specified problems the next goal is to focus on solution methods specific to each problem in order to provide and improve state-of-the-art solutions. This investigation should consider solution methods from different solving paradigms. This includes exact methods if possible, like Constraint Programming (CP) or Mixed Integer Programming (MIP), or also advanced decomposition techniques like Branch and Price (B&P), but also metaheuristics for problems with large and complex instances.

The expected results are high-quality solutions for the different problems, but also insights about which algorithms can successfully be applied to which problems as stated in Research Goal 3 (1.1.3).

Comparison with existing work is expected to be done by using existing benchmark instances from literature when available, or otherwise by providing new public benchmark instances that are based on real-life scenarios for use in future research.

1.1.3 Research Goal 3: Algorithm Analysis and Selection

The next goal is to collect and combine valuable insights about the strengths and weaknesses of different methods on the different problems, including lessons learned from Research Goal 2 about the application of various state-of-the-art methods to real-life personnel scheduling problems, as well as to apply learning methods to select a solution approach to solve a problem either offline or online.

When differences in results are hard to explain, a systematic evaluation of both the benchmark instances and the solution methods will be conducted, leading to the generation of new benchmark instances if necessary, the description of new features that can explain the differences, and the application of algorithm selection methods.

1.1.4 Research Goal 4: General Solution Methods

While the problems used as examples to evaluate different solution methods all have different formulations, they can be viewed in a common setting based on intervals and sequences of such intervals. Based on the experience and the results from the previous steps, the next goal is to design and develop a more general framework for dealing with interval-based employee scheduling problems.

The first expected benefit is an infrastructure that is highly modular and allows an easy specification of different problem variants and even different problems in the area of employee scheduling, as well as applying a solution method to these different problems with minimal adaptation effort.

Based on this framework, the goal is to investigate the use of hyper-heuristics for the automated design of algorithms for the specified problems by using a set of Low-level Heuristics (LLHs) for each of the problem domains that are automatically combined in a meaningful way to provide high-quality solutions. Specific sub-goals are:

- Introduce new Low-level Heuristics for the employee scheduling domains under consideration.
- Investigate the use of existing hyper-heuristics on these domains by making use of the interval-based framework and the Low-level Heuristics from the previous step.
- Propose new methods for hyper-heuristics that can provide high-quality solutions on these domains as well as other domains outside of employee scheduling.

1.2 Contributions

This thesis investigates several different solution approaches ranging from very problem-specific to very general methods in order to analyze strengths and weaknesses of different methods and provide a range of tools to work with when dealing with complex real-life personnel scheduling problems. The methods are applied on three real-life example problems, which are either newly defined or extended versions of problems from literature, and are provided by our industry partner XIMES GmbH, who have a long history in providing consulting and software for various companies regarding the scheduling of their employees.

The contributions in this thesis will be presented as follows. First, more specific solutions methods are applied on the three real-life example problems for this thesis that are introduced in the next chapters. In Chapter 2 the Rotating Workforce Scheduling (RWS) problem will be introduced, which deals with the creation of repeating schedules that cover demand for different shifts that varies during the week, and is therefore of high practical relevance. This chapter investigates solving this real-life problem with several new practically relevant features. This includes early recognition of certain infeasibility criteria, complex rest time constraints regarding weekly rest time, and optimization goals to deal with optimal assignments of free weekends. We introduce a state-of-the-art constraint model and evaluate it with different extensions. The evaluation shows that many real-life instances can be solved to optimality using a constraint solver. Our approach has since been deployed in a state-of-the-art commercial solver for Rotating Workforce Scheduling.

In order to analyse strengths and weaknesses of different solution methods, we use Instance Space Analysis (ISA) to perform an in-depth evaluation on the RWS problem. At first we present a set of features aiming to describe hardness of instances. We create a new, more diverse set of instances based on a first analysis revealing gaps and possible extensions. The results of three algorithms on the extended instance set show different strong and weak areas in the instance space as well as a transition from feasible to

infeasible instances including a more challenging area of instances at this transition. Work on RWS was published in the following venues:

- Lucas Kletzander, Nysret Musliu, Johannes Gärtner, Thomas Krennwallner, and Werner Schafhauser. Exact methods for extended rotating workforce scheduling problems. In Proceedings of the 29th International Conference on Automated Planning and Scheduling, volume 29, pages 519–527, 2019. [KMG+19]

- Lucas Kletzander, Nysret Musliu, and Kate Smith-Miles. Instance space analysis for a personnel scheduling problem. Annals of Mathematics and Artificial Intelligence, 89(7):617–637, 2021. [KMSM21]

In Chapter 3 the Minimum Shift Design (MSD) problem will be introduced. When demand for employees varies throughout the day, the MSD problem aims at placing a minimum number of shifts that cover the demand with minimum overstaffing and understaffing. This chapter investigates different constraint models for the problem, using a direct representation, a counting representation and a network flow based model and applies both CP and MIP solvers. The results show that the model based on network flow clearly outperforms the other models. While a CP solver finds some optimal results, with MIP solvers it can for the first time provide optimal solutions to all existing benchmark instances in short computational time. This work was published in the following venue:

- Lucas Kletzander and Nysret Musliu. Modelling and solving the minimum shift design problem. In 16th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pages 391–408. Springer, 2019. [KM19]

In Chapter 4 the Bus Driver Scheduling (BDS) problem based on the Austrian collective agreement will be presented and analysed in detail. When scheduling drivers for public transport, in addition to covering the demand and dealing with the spatial dimension, a range of legal requirements, collective agreements and company policies need to be respected. The level of concentration required while driving leads to strict rules for break assignments. This results in a complex problem where creating cost-efficient and employee-friendly schedules is challenging. This chapter deals with Bus Driver Scheduling using the rules of the Austrian collective agreement for private omnibus providers. The contributions are the formalization of the complex Austrian rules for bus drivers, a new set of publicly available instances based on the characteristics of real-life instances, and a metaheuristic solution approach for the problem. The algorithm was able to significantly improve the solutions of real-life instances and is evaluated on the generated instances. Further we provide insight in the necessity of objectives for employee satisfaction and their effects. Our method can even be successfully applied to improve results on a problem with very different constraints from Brazil.

Next, the chapter presents a Branch and Price (B&P) approach for the problem. The column generation uses a set partitioning model as master problem and a Resource Constrained Shortest Path Problem (RCSPP) as subproblem. Due to the complex constraints, the B&P algorithm adopts several novel ideas to improve the column generation in the presence of a high-dimensional subproblem, including exponential arc throttling and a dedicated two-stage dominance algorithm. Evaluation on the set of benchmark instances shows that the approach provides the first provably optimal solutions for small instances, improving best-known solutions or proving them optimal for 48 out of 50 instances, and yielding an optimality gap of less than 1% for more than half the instances. The work on BDS covered in this thesis was published in the following venues:

- Lucas Kletzander and Nysret Musliu. Solving large real-life bus driver scheduling problems with complex break constraints. In Proceedings of the 30th International Conference on Automated Planning and Scheduling, volume 30, pages 421–429, 2020. [KM20b]
- Lucas Kletzander, Nysret Musliu, and Pascal Van Hentenryck. Branch and price for bus driver scheduling with complex break constraints. In Proceedings of the 35th AAAI Conference on Artificial Intelligence, volume 35, pages 11853–11861, 2021. [KMVH21]

Then the focus is shifted to general ways to solve personnel scheduling problems in Chapter 5. In real-life applications problems can frequently change or require small adaptations. Manually creating and tuning algorithms for different problem domains or different versions of a problem can be cumbersome and time-consuming. First, a general framework structure for specifying interval-based personnel scheduling problems is introduced, allowing to unite the definitions of various real-life problems and different solution methods in a common framework.

Then we consider our three example problems in this framework. Instead of designing very specific solution methods, we propose to use the more general approach based on hyper-heuristics which take a set of simpler Low-level Heuristics (LLHs) and combine them to automatically create a fitting heuristic for the problem at hand. This chapter first presents a major study on applying hyper-heuristics to these domains, which contributes in three different ways: First, it defines new LLHs for these scheduling domains, allowing to apply hyper-heuristics to them for the first time. Second, it provides a comparison of several state-of-the-art hyper-heuristics on those domains. Third, new best solutions for several instances of the different problem domains are found. These results show that hyper-heuristics are able to perform well even on very complex practical problem domains in the area of scheduling and, while being more general and requiring less problem-specific adaptation, can in several cases compete with specialized algorithms for the specific problems. These results help to improve industrial systems in use for solving different scheduling scenarios by allowing faster and easier adaptation to new problem variants.

Finally, the new hyper-heuristic LAST-RL is proposed in the last part of the chapter. It is based on a rich state representation for reinforcement learning, using tile coding as the state-action value function approximation method, SARSA-lambda as the learning agent, and an adaptive chain length selection based on the Luby sequence with a learned multiplication factor. This approach is used in combination with an epsilon-greedy policy, where the exploration policy is based on the very successful Iterated Local Search (ILS) approach with adaptive probabilities for different categories of LLHs. This new hyper-heuristic is then evaluated on six different domains from the Cross Domain Heuristic Search Challenge 2011 (CHeSC 2011), where it can outperform the original competitors and provide good results in comparison with the most recent developments in the area of hyper-heuristics. Finally, it is applied to the domains of this thesis, where it can outperform the other hyper-heuristics in comparison on two out of the three domains, allowing to further strengthen the results on these real-life problems, while providing a very general and adaptable solution method that can easily be applied to very different problems. Work on these general methods has been published at ICAPS 2022 so far, where it received the Best Industry and Applications Track Paper Award:

- Lucas Kletzander and Nysret Musliu. Hyper-heuristics for personnel scheduling domains. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 32, pages 462–470, 2022. [KM22]

Additional publications authored and co-authored by the author during the work on this thesis that are not part of the thesis themselves, including work on general employee scheduling based on the author's Master thesis, further work on the Bus Driver Scheduling problem, and a different problem in the area of personnel scheduling were published in the following venues:

- Lucas Kletzander and Nysret Musliu. Solving the general employee scheduling problem. Computers & Operations Research, 113:104794, 2020. [KM20c]
- Lucas Kletzander and Nysret Musliu. Scheduling Bus Drivers in Real-Life Multi-Objective Scenarios with Break Constraints. Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT, volume 1, 2021. [KM20a]
- Tobias Geibinger, Lucas Kletzander, Matthias Krainz, Florian Mischek, Nysret Musliu, and Felix Winter. Physician scheduling during a pandemic. In 18th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pages 456–465. Springer, 2021. [GKK+21]
- Lucas Kletzander, Tommaso Mannelli Mazzoli, and Nysret Musliu. Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints. In Proceedings of the Genetic and Evolutionary Computation Conference, pages 232–240, 2022. [KMM22]

CHAPTER 2

Constraint Modeling and Analysis for Rotating Workforce Scheduling

As stated before, personnel scheduling problems can surface in many shapes, using different demands and constraints. In several applications it can be beneficial to obtain a rotating schedule where each employee rotates through the same sequence of shifts and days off across several weeks, however, at different offsets within the rotation. As the design of shift schedules highly influences the work-life balance of the employees, such problems are subject to a wide range of constraints, dealing not only with the demand for employees in different shifts, but also legal and organizational constraints that determine allowed shift assignments.

Due to its importance there has been ongoing research on the Rotating Workforce Scheduling problem, and results have found their way into commercial software like the Shift Plan Assistant (SPA) by XIMES GmbH. The contributions of the first part of this chapter are twofold. We introduce and solve a new extended problem that includes several new additions based on the experience from working with these problems in practice. Extensions include new constraints for fast detection of infeasible instances, complex constraints to respect weekly rest times, as well as soft constraints optimizing free weekends in the schedule, turning the satisfaction problem into an optimization problem. To solve the problem we provide a new constraint model and implement it in the constraint modelling language MiniZinc.

Both the core model and the extended models are then evaluated on a standard set of benchmark instances based on real-life examples using the constraint solver Chuffed and compared to recent literature. The results show that the extended models greatly improve the handling of infeasible instances and allow to incorporate complex rest time constraints and optimization goals providing optimal solutions for the majority of the benchmark instances in short computational time. With our work, published at ICAPS

2019 [KMG+19], we further improve the state-of-the-art solver for RWS and enable this solver to be used in more complex real-life situations.

However, the proposed models show different behaviour on different instances. Therefore, for a detailed analysis of strengths and weaknesses of solution methods, as well as for a deeper understanding what makes certain problem instances easy or hard, it is not sufficient to just compare average results on arbitrary benchmarks.

In order to explain what causes hardness in instances and which algorithms work best on different parts of the instance space, we need a diverse set of instances, features that are able to explain the problem hardness and a representation of the instances where algorithms can be compared in a meaningful way. In the second part of this chapter we aim to achieve these goals using Instance Space Analysis (ISA).

We present a set of features for RWS aiming to describe the information required for algorithm selection and Instance Space Analysis. We perform ISA using the toolkit MATILDA, revealing gaps and possible extensions for the original set of 2000 instances for the problem. Therefore, new instances are created and extended analysis is performed on a more diverse selection of instances.

Then an instance space is generated based on the most important features and the full set of instances. The results for two different exact models and a metaheuristic approach show different strong and weak areas in the instance space as well as a good coverage of the instance space when combining the strengths of the algorithms. Further we describe a transition from feasible to infeasible instances in the space and a more challenging area of instances at this transition. The results can also be found in the corresponding journal publication [KMSM21].

2.1 Related Work

In this thesis three different employee scheduling problems are investigated in detail, using different solution methods. Before going to literature for specific solution methods and problems, which will be presented in each corresponding chapter, the following section will first provide a very general overview of work in the area of employee scheduling.

Due to the high practical relevance many different versions of employee scheduling problems have been investigated for several decades. Already in 1986 [GM86] an informal description of the General Employee Scheduling (GES) problem was provided, giving rise to the identification of several common notions in all problems of this kind. A general framework [KM20c] was recently proposed to model various GES problems in a modular and reusable way.

Several reviews provide an overview of various lines of work in employee scheduling. In the review on staff scheduling and rostering by Ernst et al. [EJKS04] several modules in the rostering process are identified. The combined scheduling of days off and assigning shift sequences to employees is known as the Tour Scheduling Problem, where a dedicated

review presents several different solution approaches [Alf04]. An annotated bibliography of work in many different areas of personnel scheduling and rostering was also published [EJK+04].

Methods in nurse rostering are reviewed by Burke et al. [BDCVBVL04]. The nurse rostering problem originates in hospital staff scheduling for nurses. It typically involves several different, predefined shift types with various staffing requirements and several constraints restricting the way nurses can be assigned to these shifts. It might also contain skills that are required for the assignment. The review categorizes different methods to approach such problems. There are also variations that consider cyclic or rotating schedules.

In the review by Van den Bergh et al. [VdBBDB+13] hundreds of papers are classified according to different characteristics that are described. Some important characteristics are summarized in the following. Frequent contractual constraints can refer to full time, part time or casual employments, they also frequently include skills. Scheduling often involves individual assignment, but can also rely on crew scheduling. Decisions often involve task scheduling, group scheduling, shift sequences or scheduling of time periods. Shifts can be placed differently across the day, either with fixed start and end times or with the requirement for shift design. Coverage constraints are often included as hard constraints, but can also be soft constraints. Overstaffing and understaffing might be allowed and treated in different ways.

Several different ways of including cost, e.g., per employee, per day or per task can be distinguished. A balanced workload as well as employee preferences are frequently used. Lots of different time-related constraints regarding the number and sequence of assignments, the workload, the time between assignments and much more are identified. Presented solution methods include several types of mathematical programming, constructive heuristics, improvement heuristics, simulation, constraint programming and others. Some problem variants also incorporate uncertainty.

The review by De Bruecker et al. [DBVdBBD15] focuses on work including skills. It distinguishes different skill classes, the hierarchical and categorical class, and deals with different ways to incorporate skill substitution. It investigates in detail how different papers deal with the definition and assignment of skills.

Further there are papers providing general modelling and complexity analysis like by Brucker et al. [BQB11] including some results that even some special cases in certain problems can already lead to NP-hardness.

2.1.1 Rotating Workforce Scheduling

The Rotating Workforce Scheduling (RWS) problem is an employee scheduling problem that can be classified as a single-activity tour scheduling problem with non-overlapping shifts and rotation constraints [Bak76, RGR16] and is known to be NP-complete [CL96].

So far the problem has been addressed with a range of different methods. Complete approaches include a network flow formulation [BW90], integer linear programming [LNB80], several constraint programming formulations [Lap99, MGS02, LP04, TM11] and an approach with satisfiability modulo theories [EM17]. There is also work on heuristic approaches [Mus05], including the metaheuristic we use for our evaluation [Mus06], further the creation of rotating schedules by hand [Lap99], and using algebraic methods [FBCL16].

A new constraint model using a formulation in MiniZinc was introduced by Musliu et al. [MSS18]. It uses a solver independent formulation in the MiniZinc constraint language, either with a direct representation or using a regular automaton, and applies both the lazy clause generation solver Chuffed and the MIP solver Gurobi. It is the first complete method able to solve the standard benchmark set of 20 instances and introduces new benchmark instances that we also use for comparison.

Existing work on RWS mostly deals with the standard version of the problem, requiring any feasible solution, or delegates the selection of preferred solutions to the user in an interactive process [MGS02]. While standard RWS already has practical relevance, the new extensions introduced in this thesis allow to deal with more complex issues and provide solutions that are of higher value in real-life applications.

Since the publication of the work presented in this thesis, a new approach based on Branch and Cut [BSW22] has been proposed. It models the schedules on a graph representation, where a schedule is represented as an Eulerian cycle of stints in a way that is compact regarding the number of employees. With this formulation they tackle the extended set of instances that resulted from the work in the second part of this chapter, and are able to outperform previous results both on the satisfaction version and some optimization versions like the number of free weekends by providing optimal results in seconds. However, they do not tackle the other optimization versions in this thesis dealing with the distribution of weekends, which can be more complex to model than just counting the number of free weekends.

2.2 Problem Definition for Standard RWS

A rotating workforce schedule consists of the assignment of shifts or days off to each day across several weeks for a certain number of employees.

Shift type	Mon	Tue	Wed	Thu	Fri	Sat	Sun
D	1	1	1	1	1	1	1
A	1	1	1	1	1	1	0
N	1	1	1	1	1	1	1

Table 2.1: Example demand for three shift types

Table 2.1 shows a simple demand matrix for the three shift types day shift (D), afternoon shift (A), and night shift (N), each requiring either 0 or 1 shift of this type per day of the week. For this example, the goal is to find a cyclic schedule for 4 employees that fulfils the following constraints:

- 5 to 7 consecutive days of work
- 2 to 4 consecutive free days
- Consecutive shifts per type: 2 to 5 for D, 2 to 4 for A, 2 to 3 for N
- No D after A or N, no A after N

Table 2.2 shows an example schedule for four employees (or four equal-sized groups of employees) that satisfies the given demand and constraints. Each employee starts their schedule in a different row, moving from row i to row i mod $n+1$ (where n is the number of employees) in the following week.

Employee	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 2.2: Example schedule for four employees

2.2.1 Problem Specification

We start by defining the basic version of the RWS problem and recall definitions and notation from literature [MGS02, MSS18]. Extensions to the formulation are introduced in the next section. We define:

- n: Number of employees.
- w: Length of the schedule, typically $w = 7$ as the demands repeat in a weekly cycle. The total length of the planning period is $n \cdot w$, as each employee rotates through all n rows.
- $\mathbf{A}$: Set of work shifts (activities), enumerated from 1 to m, where m is the number of shifts. A day off is denoted by a special activity O with numerical value 0 and we define $\mathbf{A}^+ = \mathbf{A} \cup \{O\}$.
- R: Temporal requirements matrix, an $m \times w$-matrix where each element $R_{i,j}$ corresponds to the number of employees that need to be assigned shift $i \in \mathbf{A}$ at day j. The number of employees o_j that need to be assigned a day off on day j can be calculated by $o_j = n - \sum_{i=1}^{m} R_{i,j}$.

- ℓ_w and u_w: Minimal and maximal length of blocks of consecutive work shifts.
- ℓ_s and u_s: Minimal and maximal lengths of blocks of consecutive assignments of shift s given for each $s \in \mathbf{A}^+$.
- Forbidden sequences of shifts: Any sequences of shifts (like N D, a night shift followed by a day shift) that are not allowed in the schedule. This is typically required due to legal or safety concerns. In practice it is usually sufficient to forbid sequences of length 2 or sequences of length 3 where the middle shift is a day off. These are also the kind of restrictions used in the benchmark instances for RWS.

  In our model, we use a set $\mathbf{F}_s^2 \subseteq \mathbf{A}$ for each $s \in \mathbf{A}$ to denote forbidden sequences of length 2, such that $x \in \mathbf{F}_s^2$ declares that shift x must not follow shift s.

  Forbidden sequences of length 3 are given as a set $\mathbf{F}_3$ of arrays of length 3 containing elements of $\mathbf{A}^+$. However, our model allows arbitrary forbidden sequences, therefore we do not limit the length or arrangement and accept sets $\mathbf{F}_l$ for any length $3 \leq l \leq n \cdot w$ containing arrays of length l, where each array encodes a forbidden sequence of length l consisting of elements of $\mathbf{A}^+$. Note that sequences of length 2 could also be represented in this way, however, a specific treatment for those sequences will be used in the model for efficiency purpose.

The task is to construct a cyclic schedule S, represented as an $n \times w$-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. The schedule for employee i through the whole planning period consists of the cyclic sequence of all rows of S starting with row i.

Instead of the matrix representation, the same schedule can also be represented as an array T which is equal to the schedule of the first employee, where T_i denotes the shift assignment on day i with $1 \leq i \leq n \cdot w$. As the schedule is cyclic, we could choose any day in the schedule to correspond to the first element in T. We define the offset o with $0 \leq o < w$ to denote the position of the first element in T within the schedule, i.e., its day of the week.

Table 2.3 provides an overview of the most important variables introduced in the standard definition of RWS as well as in the extensions discussed later in this chapter for quick reference. The extensions will be discussed in more detail in their respective sections. Additional definitions that are not key concepts and not commonly reused are not part of this table and will be only mentioned in their sections.

2.2.2 Constraint Model

Our main model for standard RWS uses a direct representation of the constraints based on Musliu et al. [MSS18]. Some aspects are modelled in a different way, most notably the different way to deal with cyclicity using the offset o.

Variable	Meaning
n	Number of employees
$\mathbf{N}$	$\mathbf{N} = \{1, \dots, n\}$, set of employee indices
w	Length of the schedule
$\mathbf{W}$	$\mathbf{W} = \{1, \dots, w\}$, set of day indices
m	Number of shifts
$\mathbf{A}$	Set of shift types $1, \dots, m$
$\mathbf{A}^+$	Set of shifts including a day off O, $\mathbf{A}^+ = \mathbf{A} \cup \{O\}$
R	Temporal requirements matrix of dimension $m \times w$
ℓ_w, u_w	Minimal and maximal lengths of work blocks
ℓ_s, u_s	Minimal and maximal lengths of consecutive blocks for $s \in \mathbf{A}^+$
$\mathbf{F}_s^2$	Set of shifts not allowed to follow shift s
$\mathbf{F}_\ell$	Set of sequences of length ℓ that are forbidden
S	Solution matrix of dimension $n \times w$
T	S in array form of length $n \cdot w$
$\mathbf{NW}$	$\mathbf{NW} = \{1, \dots, n \cdot w\}$, set of indices of the solution array T
o	Offset of the first day of the schedule (its day of week)
C, C^x	Various arrays of length $n \cdot w$ counting different aspects of the schedule
r	Total sum of demands
g	Time granularity (number of time slots per day)
$start_s$, end_s	Start and end time of a shift s in minutes relative to its assigned day
wr	Minimum weekly rest time in minutes
wr_{red}	Reduced minimum weekly rest time in minutes
e	Number of exceptions per sp weeks
sp	Span in weeks for the number of exceptions and average calculation
f, f_e, f_l	Different definitions for the number of free weekends
d_m	Maximum distance between consecutive free weekends
d	Combined objective for optimization of free weekends

Table 2.3: Overview of important definitions for RWS

For any array or matrix the indices are modulo its dimension. Within this description, such modulo operations are omitted for better readability. We define $\mathbf{N} = \{1, \dots, n\}$, $\mathbf{W} = \{1, \dots, w\}$ and $\mathbf{NW} = \{1, \dots, n \cdot w\}$.

The following equations model the demand:

$$\sum_{i=0}^{n-1} (T_{d+w \cdot i} = s) = R_{s,d+o} \qquad \forall d \in \mathbf{W},\ s \in \mathbf{A} \quad (2.1)$$

$$\sum_{i=0}^{n-1} (T_{d+w \cdot i} = O) = n - \sum_{i=1}^{m} R_{i,d+o} \qquad \forall d \in \mathbf{W} \quad (2.2)$$

Equation (2.1) models the demand for each day d and each shift type s. The left side counts occurrences of s on day d, the right side uses the offset to access the correct column of the demand matrix. Equation (2.2) is a redundant constraint that counts

the number of day-off assignments for each day d. This is not necessary to obtain a complete model of the problem, however, constraint satisfaction solvers can benefit from such redundant definitions.

The next equations introduce symmetry breaking constraints which are also used to make dealing with the cyclic nature of the problem easier in following constraints:

$$T_1 \neq O \tag{2.3}$$
$$T_{n \cdot w} = O \tag{2.4}$$

Equations (2.3) and (2.4) declare that the first element of T has to hold a working shift, while the last element of T has to hold a day off. In principle any day of the planning period could be used as the first day as it is cyclic. Taking into account that every reasonable rotating workforce scheduling problem contains at least one working day and at least one day off, we can set the first element of T to align with the beginning of a working block.

This has two advantages. First, it eliminates several symmetric versions of the same solution, reducing cyclic occurrences of the same solution from $n \cdot w$ possible notations to the number of working blocks within the solution. Second, this guarantees that blocks of the same shift type or working blocks (consecutive days without day-off assignments) can never cycle across the end of T, eliminating the need to deal with cyclicity in their definition.

Block Lengths Constraints

Next, constraints for the lengths of shift blocks and working blocks are defined:

$$\forall j \in \{1, \ldots, \ell_s - 1\} : \; T_{i+j} = s \qquad \forall s \in \mathbf{A}^+,\; i \in \mathbf{NW},\; T_i = s,\; T_{i-1} \neq s \tag{2.5}$$
$$i + u_s > n \cdot w \vee \exists j \in \{\ell_s, \ldots, u_s\} : \; T_{i+j} \neq s \qquad \forall s \in \mathbf{A}^+,\; i \in \mathbf{NW},\; T_i = s,\; T_{i-1} \neq s \tag{2.6}$$
$$\forall j \in \{1, \ldots, \ell_w - 1\} : \; T_{i+j} \neq O \qquad \forall i \in \mathbf{NW},\; T_i \neq O,\; T_{i-1} = O \tag{2.7}$$
$$i + u_w > n \cdot w \vee \exists j \in \{\ell_w, \ldots, u_w\} : \; T_{i+j} = O \qquad \forall i \in \mathbf{NW},\; T_i \neq O,\; T_{i-1} = O \tag{2.8}$$

Equation (2.5) defines the minimum block length for all shift types including day-off assignments. For all elements T_i containing shift s, where the block starts at i (corresponding to $T_{i-1} \neq s$), the next elements of T until the minimum length must also contain shift s. Equation (2.6) defines the maximum block length, stating that no later than u_s elements after the block start a different shift type has to occur. Additionally, if i is too close to the end of T, the block will end anyway, giving rise to the inequality part.

Equations (2.7) and (2.8) define the same constraints for working blocks, using ℓ_w and u_w as bounds and checking for any working shift ($\neq O$) instead of a specific shift type s.

Alternative Block Lengths Constraints

The previous set of constraints is not the only way to model the block constraints of this problem. Next a different modelling approach for these block lengths is described. It uses additional counting arrays of length $n \cdot w$, the array C for individual shift types including days off and the array C^W for working blocks.

$$C_1 = 1 \tag{2.9}$$
$$C_i = C_{i-1} + 1 \wedge C_i \leq u_{T_i} \qquad \forall i \in 2, \ldots, n \cdot w,\ T_i = T_{i-1} \tag{2.10}$$
$$C_i = 1 \wedge C_{i-1} \geq l_{T_{i-1}} \qquad \forall i \in 2, \ldots, n \cdot w,\ T_i \neq T_{i-1} \tag{2.11}$$
$$C_{n \cdot w} \geq \ell_O \tag{2.12}$$
$$C^W_1 = 1 \tag{2.13}$$
$$C^W_i = C^W_{i-1} + 1 \wedge C^W_i \leq u_w \qquad \forall i \in 2, \ldots, n \cdot w,\ T_i \neq O \tag{2.14}$$
$$C^W_i = 0 \wedge (C^W_{i-1} = 0 \vee C^W_{i-1} \geq \ell_w) \qquad \forall i \in 2, \ldots, n \cdot w,\ T_i = O \tag{2.15}$$

In this formulation C counts the length of the current block for each position in T. Equation (2.10) models increasing the counter and checking whether the maximum is violated (where T_i determines which upper bound u_{T_i} is relevant for the current block) for all elements of T_i continuing the current block ($T_i = T_{i-1}$). Equation (2.11) models resetting the counter to 1 and checking whether the previous block violated its minimum length whenever a new block begins. Equations (2.9) and (2.12) apply to the first and last element of C, making use of the symmetry breaking constraints.

Equations (2.14) and (2.15) do the same for working blocks, while $C^W_i = 0$ is set for all i assigned to a day off. Equation (2.13) again sets the first element, as T ends with a day-off block, the last element does not need an additional constraint.

However, this version of the block lengths constraints did not perform as well and was therefore dropped from the evaluation.

Forbidden Sequence Constraints

Finally the forbidden sequences need to be modelled:

$$T_i \neq s \vee T_{i+1} \notin \mathbf{F}^2_s \qquad \forall s \in \mathbf{A},\ i \in \mathbf{NW} \tag{2.16}$$
$$\exists j \in \{1, \ldots, \ell\} : X_j \neq T_{i+j-1} \qquad \forall X \in \mathbf{F}_\ell,\ i \in \mathbf{NW} \tag{2.17}$$

Equation (2.16) models sequences of length 2, denoted by the set $\mathbf{F}^2_s$ of shift types not allowed to follow shift type s. Forbidden sequences of arbitrary length ℓ are modelled in (2.17), where for each possible match of each forbidden sequence at least one element must differ from the forbidden sequence.

Offset Symmetry Constraints

Further symmetry breaking constraints might be applied to determine the offset o if certain conditions hold:

$$o = \min \left\{ d \in \mathbf{W} \;\middle|\; \sum_{i=1}^{m} R_{i,d} > \sum_{i=1}^{m} R_{i,d-1} \right\} - 1 \tag{2.18}$$

$$(\forall s \in \mathbf{A}, d \in \mathbf{W} : R_{s,d} = R_{s,d+1}) \rightarrow o = 0 \tag{2.19}$$

Equation (2.18) models the case that there is any day d where the previous day $d-1$ has a lower total demand for shifts, then at least one new shift block has to start on day d, making it possible to fix the offset. Note that (2.18) cannot be applied if the condition does not hold for any d, however, in that case (2.19) might be applied if the demand is constant for each shift. In this case week days are completely symmetrical, allowing to fix the offset to 0.

2.3 Problem Extensions

This section introduces several new extensions to the problem and provides the formal constraint model for them. The extensions cover different aspects of the problem and its solving process, dealing with the detection of infeasible instances, the introduction of complex rest time constraints and the optimization towards better scheduling of free weekends.

2.3.1 Detecting Infeasible Instances

The standard benchmark data set consists of 20 instances derived from real life scenarios, all of them admitting feasible solutions. However, the larger instance set [MSS18] also includes infeasible instances. The results show that the solver Chuffed also used by us has difficulties identifying those instances. However, in practice it is important to provide fast feedback to the user when they give infeasible settings so that they can correct their input.

Two particular infeasibility tests are described in this section. As these are defined on input parameters only, several infeasible instances can be detected already while compiling the instance for the solver, while there is still one consistent formulation of the problem.

Infeasible Weekly Fluctuation

Consider the following demand for any shift s together with $\ell_s = 3$ and $u_s = 4$:

Table 2.4 defines a common pattern for weekly demand, weekdays with more demand and a weekend with less demand. The increase in demand from Sunday to Monday requires at least three new work blocks of shift s to start on Monday, similarly the decrease from

Mon	Tue	Wed	Thu	Fri	Sat	Sun
5	5	5	5	5	2	2

Table 2.4: Infeasible demand for shift s

Friday to Saturday requires at least three work blocks of shift s to end on Friday. Next, as the minimum block length is 3, all three blocks starting on Monday and all three blocks ending on Friday span across Wednesday. On the other hand, all six blocks are distinct as the maximum block length is 4 and therefore no block can cover Monday to Friday at once. Therefore, at least six shifts of type s are required on Wednesday, which is higher than the demand and results in infeasibility of the instance.

More formally, this observation can be generalized as follows.

$$\forall j \in \{u_s + 1, \ldots, 2 \cdot \ell_s - 1\},\ k \in \{j - \ell_s, \ldots, \ell_s - 1\} : \\ R_{s,i+k} \geq R_{s,i} - R_{s,i-1} + R_{s,i+j-1} - R_{s,i+j} \qquad \forall s \in \mathbf{A}, i \in \mathbf{W} \quad (2.20)$$

Equation (2.20) extends the example above for arbitrary block lengths. j iterates through possible distances between block starts and block ends (5 in the example), k iterates through days of guaranteed overlap (only Wednesday in the example). The check is performed for every shift type s and every possible start day i.

Bounding the Number of Blocks

Another observation is the fact that in a cyclic schedule the number of work blocks and the number of free blocks is equal. On the other hand, both for work blocks and free blocks a minimum and maximum number of blocks can be calculated from the required number of shifts and the allowed block lengths.

$$r = \sum_{i=1}^{m} \sum_{j=1}^{w} R_{i,j} \quad (2.21)$$

$$low_w = \left\lceil \frac{r}{u_w} \right\rceil \quad (2.22)$$

$$up_w = \left\lfloor \frac{r}{\ell_w} \right\rfloor \quad (2.23)$$

$$low_O = \left\lceil \frac{n \cdot w - r}{u_O} \right\rceil \quad (2.24)$$

$$up_O = \left\lfloor \frac{n \cdot w - r}{\ell_O} \right\rfloor \quad (2.25)$$

$$low = \max\{low_w;\ low_O\} \quad (2.26)$$

$$up = \min\{up_w;\ up_O\} \quad (2.27)$$

Equations (2.22) and (2.23) define lower and upper bounds for the number of work blocks, using the total demand for work shifts r defined in Equation (2.21). Equations (2.24)

and (2.25) define lower and upper bounds for the number of free blocks. As the numbers of blocks need to be equal, (2.26) and (2.27) define the common bounds. Instances with $low > up$ can immediately be classified as infeasible.

Using Block Bounds for Redundant Constraints

We introduce two possibilities to use the block bounds for redundant constraints. The first uses a global cardinality constraint that takes four arguments: an array to operate on, an array of values to count in the first array, and lower and upper bounds for the corresponding values.

$$gcc_{lu}([(T_i \neq O \wedge T_{i-1} = O) - (T_i = O \wedge T_{i-1} \neq O) \mid i \in \mathbf{NW}], \\ [-1, 1], [low, low], [up, up]) \quad (2.28)$$

Equation (2.28) defines an array holding -1 for the start of off-blocks and 1 for the start of work blocks and sets lower and upper limits for the counts of both of them. Note that this constraint does not enforce both counts to be equal, however, it propagates lower and upper bounds well to restrict the search space.

The second possibility uses counting arrays of length $n \cdot w$ for the number of days off C^O, the number of work days C^w, and the number of blocks C^b.

$$C_i^O = \begin{cases} 0 & \text{if } i = 1 \\ C_{i-1}^O + (T_i = O) & \text{otherwise} \end{cases} \qquad \forall i \in \mathbf{NW} \quad (2.29)$$

$$C_i^w = \begin{cases} 1 & \text{if } i = 1 \\ C_{i-1}^w + (T_i \neq O) & \text{otherwise} \end{cases} \qquad \forall i \in \mathbf{NW} \quad (2.30)$$

$$C_i^b = \begin{cases} 1 & \text{if } i = 1 \\ C_{i-1}^b + (T_{i-1} = O \wedge T_i \neq O) & \text{otherwise} \end{cases} \qquad \forall i \in \mathbf{NW} \quad (2.31)$$

$$(low - C_i^b) \cdot \ell_O \leq n \cdot w - r - C_i^O \leq (up - C_i^b) \cdot u_O \qquad \forall i \in \mathbf{NW}, C_i^b < C_{i+1}^b \quad (2.32)$$

$$(low - C_i^b) \cdot \ell_w \leq r - C_i^w \leq (up - C_i^b) \cdot u_w \qquad \forall i \in \mathbf{NW}, C_i^b < C_{i+1}^b \quad (2.33)$$

Equations (2.29), (2.30) and (2.31) count the number of days off, work days and blocks (counting every time a work block starts) up to each day $i \in \mathbf{NW}$. These counts are then used in (2.32) and (2.33) to bound the remaining number of off-blocks and work blocks after every day i where a new work block starts on the next day, using the bounds for the number of blocks.

2.3.2 Weekly Rest Time

While forbidden sequences of shifts can be used to handle minimum free time between consecutive shifts, work regulations often contain different, more complex regulations for free time. Real-world scenarios often need to consider a weekly rest time. Typically once

a week a certain minimum amount of time has to be free without interruption. Further, it might be possible to have exceptions once every few weeks where the weekly rest time might be shorter according to certain rules.

Definition

The following requirements are defined to consider weekly rest times:

- g: Time granularity, given as the number of time slots per day, in this thesis we use minute level granularity with $g = 1440$.
- $start_s$ and end_s: As weekly rest times consider the time between shifts, start and end times need to be defined for each shift. The times are given in minutes relative to the day the shift is assigned to, e.g., a night shift s ending at 6:00 on the next day has $end_s = 1800$.
- wr: Minimum weekly rest time (full weekly rest) in minutes, denoting the minimum time from the end of the last shift before the weekly rest time to the beginning of the next shift after the weekly rest time.
- A weekly rest needs to contain a full free day (0:00 to 24:00, i.e, no night shift from the previous day might overlap).
- A weekly rest is required in each calendar week (Monday to Sunday). Each rest period counts for the week where more than half of the rest is located, if the rest time is exactly split between two weeks, it counts for the later week.
- wr_{red}: Reduced minimum weekly rest time in minutes.
- e: Number of exceptions per sp weeks.
- sp: Span in weeks for the number of exceptions and the calculation of the average.

Every week in the planning period needs to have a weekly rest of length at least wr with the exception of e times in a rolling horizon of sp weeks where the weekly rest can be reduced to wr_{red}, however, the average across sp weeks always needs to be at least wr.

Constraint Model

To capture the rest time between shifts an array $Rest$ of length $n \cdot w$ is defined. For a day off we define $start_O = g$.

$$Rest_i = \begin{cases} Rest_{i-1} + start_{T_i} & \text{if } T_{i-1} = O \\ g - end_{T_{i-1}} + start_{T_i} & \text{otherwise} \end{cases} \qquad \forall i \in \mathbf{NW} \quad (2.34)$$

Equation (2.34) first considers the case $T_{i-1} = O$ where $Rest_{i-1}$ holds an intermediate rest time starting at the last shift and ending at the end of day $i-1$. In any case, $start_{T_i}$ holds the rest time from midnight until the start of the next shift, while $g - end_{T_{i-1}}$ holds the rest time from the previous shift until midnight.

Next, the individual requirements are modelled via a boolean matrix D of dimension $6 \times n \cdot w$.

$$D_{1,i} = (Rest_i \geq wr) \qquad \forall i \in \mathbf{NW} \quad (2.35)$$
$$D_{2,i} = (T_i \neq O) \qquad \forall i \in \mathbf{NW} \quad (2.36)$$
$$D_{3,i} = (T_{i-1} = O) \qquad \forall i \in \mathbf{NW} \quad (2.37)$$
$$D_{4,i} = (end_{T_{i-2}} \leq g) \qquad \forall i \in \mathbf{NW} \quad (2.38)$$
$$D_{5,i} = \left(\frac{Rest_i}{2} < g \cdot ((i + o - 1) \bmod 7) + start_{T_i}\right) \qquad \forall i \in \mathbf{NW} \quad (2.39)$$
$$D_{6,i} = (Rest_i \geq wr_{red}) \qquad \forall i \in \mathbf{NW} \quad (2.40)$$

Equation (2.35) models the minimum weekly rest time, (2.40) the reduced minimum weekly rest time. (2.36) ensures that only the end of a rest period is considered, not intermediate results that are used in (2.34). Equations (2.37) and (2.38) make sure that the full free day is respected. If less than half of the rest is located in the previous week, $D_{5,i}$ is set to *true* in (2.39), the right side of the inequality specifies the time from the start of the current week to the start of the shift on day i.

The content of matrix D is also used to allow users to understand why a certain rest period does or does not qualify as a weekly rest period.

Now the position P_i of the weekly rest period for each week i can be determined, using $x = (i-1) \cdot w + j - o$.

$$P_i = \max\{0;\ j \mid D_{1,x} \wedge D_{2,x} \wedge D_{3,x} \wedge D_{4,x} \wedge (j \leq w) = D_{5,x}\} \qquad \forall i \in \mathbf{N} \quad (2.41)$$

Equation (2.41) defines the position of the weekly rest period for each week i by taking the latest rest period that qualifies as a proper weekly rest using the elements from D and is still assigned to the correct week. Here, $(j \leq w) = D_{5,x}$ makes sure that either the weekly rest still ends in the current week and is also assigned to the current week or it ends in the next week, but is not yet assigned to the week where it ends. This also guarantees that no j with $j > w + \frac{u_o}{2} + 1$ needs to be considered as the corresponding rest would be counted towards the following week for sure. In case no proper weekly rest is found, P_i is set to 0.

Exceptions

Now regular weekly rest for each week could be enforced simply requiring $P_i \neq 0$ for all i, however, we want to consider exceptions as follows.

$$\sum_{j=0}^{sp-1} (P_{i+j} = 0) \leq e \qquad \forall i \in \mathbf{N} \quad (2.42)$$

Equation (2.42) counts the number of weeks for each span of length sp where no proper weekly rest can be found.

To check the reduced weekly rest the position P_i^{red} for each week i is defined, using $x = (i-1) \cdot w + j - o$.

$$P_i^{red} = \begin{cases} P_i & \text{if } P_i \neq 0 \\ \max\{0;\ j \mid D_{6,x} \wedge D_{2,x} \wedge (j \leq w) = D_{5,x}\} & \text{otherwise} \end{cases} \qquad \forall i \in \mathbf{N} \quad (2.43)$$

$$P_i^{red} \neq 0 \qquad \forall i \in \mathbf{N} \quad (2.44)$$

Equation (2.43) is built similar to (2.41), but uses the reduced rest time and does not enforce the full free day. If P_i already holds a valid weekly rest, it is simply transferred to P_i^{red}. (2.44) ensures that the reduced minimum weekly rest time is never violated.

Finally the average rest time needs to be checked for each span of sp weeks.

$$\sum_{j=0}^{sp-1} Rest_{(i+j-1)\cdot w + P_{i+j}^{red} - o} \geq wr \cdot sp \qquad \forall i \in \mathbf{N} \quad (2.45)$$

Equation (2.45) calculates the sum of the weekly rest times for the given span of weeks. Note that the index of P^{red} is modulo n and the index of $Rest$ modulo $n \cdot w$. Further the index of $Rest$ is not correct for $P_{i+j}^{red} = 0$, but in this case (2.44) is already violated anyway.

As a side effect, with the definition of $Rest$, it is also possible to enforce a minimum daily rest time defined via the time span rather than the forbidden sequences of length 2. This allows to define one common minimum daily rest time which is independent of the number of shift types and does not need modification when shift types are added or changed.

2.3.3 Optimizing Free Weekends

In the previous SPA implementation the user was presented a choice in several stages of the algorithm, potentially selecting from a large number of feasible solutions. However, defining properties of beneficial solutions beforehand and including these definitions in the model allows to transform the satisfaction problem into an optimization problem and to shift the selection process to the solver.

Shift work can be very disruptive to the social life of employees, e.g., social interactions with friends and family might be hard to schedule as free time is arranged in various different patterns compared to employees with regular free weekends.

Therefore, we chose to optimize the free time on weekends and provide different measurements of desirable weekend schedules. Note that for each optimization goal we also include bounds that allow the solver to immediately stop if the bound can be reached.

Maximizing the Number of Free Weekends

Note that our week length w could be any positive number, however, in this context we use regular 7-day weeks where Saturday and Sunday are considered weekend. Usually it is beneficial to have a whole free weekend compared to a weekend with only one free day. We first define the number of free weekends f as follows:

$$f = \sum_{i=1}^{n} (T_{w \cdot i - o - 1} = O \wedge T_{w \cdot i - o} = O) \tag{2.46}$$

$$u_f = n - \max \left\{ \sum_{s=1}^{m} R_{s,6};\ \sum_{s=1}^{m} R_{s,7} \right\} \tag{2.47}$$

Equation (2.46) counts the number of weeks where both Saturday and Sunday do not have a shift assigned. (2.47) defines an upper bound for the number of free weekends using the maximum number of shifts on either Saturday or Sunday. This allows to immediately stop the search if the upper bound is reached.

However, having Saturday and Sunday free might not be enough. Consider the case of having a night shift assigned to the Friday before a free weekend, where most of Saturday will probably be spent sleeping. Therefore, f_e defines an extended notion of a free weekend, additionally supplying a set $\mathbf{S}_e$ of shifts that are forbidden on Friday to consider the weekend free:

$$f_e = \sum_{i=1}^{n} (T_{w \cdot i - o - 2} \notin \mathbf{S}_e \wedge T_{w \cdot i - o - 1} = O \wedge T_{w \cdot i - o} = O) \tag{2.48}$$

$$b = \sum_{s \in \mathbf{S}_e} \max \left\{ R_{s,5} - \sum_{k \in \mathbf{A} \setminus \mathbf{F}_s^2} R_{k,6};\ 0 \right\} \tag{2.49}$$

$$u_{f_e} = n - \max \left\{ \sum_{s \in \mathbf{S}_e} R_{s,5};\ \sum_{s=1}^{m} R_{s,6} + b;\ \sum_{s=1}^{m} R_{s,7} + b \cdot (\ell_O > 1) \right\} \tag{2.50}$$

Equation (2.48) again counts the number of weeks satisfying the requirements. This time, however, the upper bound can be strengthened further by taking into account the forbidden Friday shifts. b in (2.49) holds the number of forbidden Friday shifts s that are guaranteed to be followed by a day off on Saturday. This is done by subtracting the

demand of shifts that are allowed to follow after s on Saturday from the demand for s on Friday. Then, three upper bounds are considered for the final bound, which are the demand for forbidden shifts on Friday, the total demand plus the free shifts blocked by forbidden Friday shifts for Saturday and finally the bound for Sunday considering the total demand plus the free shifts blocked by forbidden Friday shifts only if the minimum length of off-blocks is not 1.

The disadvantage of this version, however, is the fact that free weekends with a forbidden Friday shift count for nothing at all. The third version combines the previous objectives lexicographically into the objective f_l, allowing to maximize the number of weekends with Saturday and Sunday free, breaking ties with the maximum number of free weekends with no unwanted Friday shifts:

$$f_l = f \cdot (u_{f_e} + 1) + f_e \tag{2.51}$$

Equation (2.51) combines the previous objectives using the upper bound u_{f_e} to ensure lexicographic ordering. Note that both previous upper bounds are still valid for the corresponding parts of the objective.

Optimizing the Distribution of Weekends

While the number of free weekends is a good candidate for optimization, it is not the only possible measure regarding weekend schedules. Consider a schedule where two weekends are free in a row, followed by six weeks without a free weekend. Usually, a more regular distribution, e.g., a free weekend every four weeks, would be considered better.

In the following we use a boolean array *Free*, denoting for every week i whether the corresponding weekend is free. This array could be built using any of the definitions of free weekends above or a different one. We use free Saturday and Sunday as defined in (2.46). Next an array *Dist* of length n holding distances to the next free weekend is defined and used to minimize the maximum distance:

$$Dist_i = \begin{cases} \min\{j \in \mathbf{N} \mid Free_{i+j}\} & \text{if } Free_i \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \mathbf{N} \tag{2.52}$$

$$d_m = \begin{cases} n+1 & \text{if } \max(Dist) = 0 \\ \max(Dist) & \text{otherwise} \end{cases} \tag{2.53}$$

$$l_{d_m} = \left\lceil \frac{n}{u_f} \right\rceil \tag{2.54}$$

Equation (2.52) gathers the distance to the next free weekend. The index of *Free* is modulo n, therefore in the worst case the distance is n. In (2.53), the maximum distance d_m is defined, taking into account the case of no free weekends in the whole schedule, which gets assigned the worst possible value for d_m. The lower bound for d_m is given in (2.54) by the most even possible distribution of the upper bound for free weekends.

A different possibility to deal with weekend distances, not only optimizing the maximum distance, is presented in the following, using a slightly modified distance definition $\widehat{Dist}$:

$$\widehat{Dist}_i = \begin{cases} \min\{j \in \mathbf{N} \mid Free_{i+j}\} - 1 & \text{if } Free_i \\ n & \text{otherwise} \end{cases} \qquad \forall i \in \mathbf{N} \quad (2.55)$$

$$d = \sum_{i=1}^{n} \widehat{Dist}_i^2 \quad (2.56)$$

$$l_d = n^2 \cdot (n - u_f) + \left(\left\lfloor \frac{n}{u_f} \right\rfloor - 1 \right)^2 \cdot (u_f - n \bmod u_f) + \left\lfloor \frac{n}{u_f} \right\rfloor^2 \cdot (n \bmod u_f) \quad (2.57)$$

Equation (2.55) defines the distance to the next free weekend such that maximum values are assigned to weekends that are not free. Therefore, in (2.56) non-free weekends have the worst penalty, leading to the optimization of the number of free weekends, but due to the penalization of higher distances, at the same time the distribution of weekends is optimized. A lower bound can computed with (2.57) first taking into account the minimum number of non-free weekends, followed by the optimal distribution of the maximum number of free weekends with all distances of either $\lfloor \frac{n}{u_f} \rfloor$ or $\lfloor \frac{n}{u_f} \rfloor - 1$ due to integrality of the distances.

2.4 Evaluation

This section describes the implementation of the models and the choice for the solver and evaluates the different models on a set of benchmark instances that are based on real-life problems. All experiments were carried out on an Intel Core i7-7500 CPU with 2.7 GHz and 16 GB RAM.

2.4.1 Implementation and Solver

All models described above were implemented using the solver-independent modelling language MiniZinc 2.2.2 [NSB+07]. It allows to directly specify the constraint models and compiles them into a format called FlatZinc, which is understood by a wide range of solvers. This allows to evaluate the performance of different solvers or to switch to a new solver whenever new technologies advance without having to recreate the model.

A preliminary version of our model was tested with a range of different solvers. However, the lazy clause generation solver Chuffed [CSS+18] was clearly the best choice among the tested solvers for the presented model. Therefore, in the following evaluation Chuffed 0.10.3 is used for all experiments.

To increase the efficiency of the search, both MiniZinc and Chuffed offer settings to guide the search in the right direction. MiniZinc uses search annotations that allow to specify a custom search strategy. A wide range of different strategies was tested with different preliminary models. While details are omitted here, overall the best results

where achieved using the variable selection strategy `smallest` and the value selection strategy `indomain_min` on T, assigning the smallest possible values first.

Regarding Chuffed, a range of flags is available to influence the search. Again, a range of combinations was tried to find the best settings. The following flags turned out to be beneficial.

- `f`: Free search is the most important flag. This allows to switch on each restart between the user-specified search from the MiniZinc annotation and the activity-based search which is default for Chuffed.
- `restart-scale` 1000: While the default is 100 for free search, 1000 allows to go deeper into the search tree before each restart.
- `use-var-is-introduced no-decide-introduced`: Restricts the variable assignment decisions to the main variables instead of additional ones introduced during compilation.

2.4.2 Instances

For the experiments we focus on the standard RWS benchmark set of 20 instances to look at results in more detail. These instances stem from real life applications and cover a range of 7 to 163 employees (or employee groups), which includes instances considered very large in real settings, while most are in the range from 15 to 50 employees.

For the first comparison regarding the core model we use the extended set of 50 instances [MSS18] which also include infeasible instances that were generated in a similar range like the real life instances.

2.4.3 Core Model

First we evaluate the core model `CORE` using Equations (2.1) to (2.8) and (2.16) to (2.19) in comparison with the model `EXT1` including extensions for infeasible instances and the global cardinality constraint additionally using (2.20) to (2.28), as well as the model `EXT2` that uses all constraints from `EXT1` and additionally the block count constraints (2.29) to (2.33). The timeout is 3600 seconds, average values in seconds are calculated across all instances including timeouts.

Table 2.5 shows the results of the comparison. Clearly both `EXT1` and `EXT2` can provide a major reduction in runtime (avg rt) compared to `CORE` with drops in the average runtime of 49% and 54%. This is especially visible for the unsatisfiable instances, where runtime drops by 67% for `EXT1` and `EXT2` can determine infeasibility for all 8 infeasible instances in an average of 1.3 seconds.

Detailed comparison shows that 4 infeasible instances are caught by (2.20). While no instances have $low > up$, several have those two values equal or very close. Especially for those instances `EXT2` is often beneficial or even necessary to prevent a timeout, while for

	All Instances				Sat Instances		Unsat Instances	
Model	#total	Avg nodes	Avg mem	Avg rt	#sat	Avg rt	#unsat	Avg rt
CORE	45	3.6m	64MB	445.9	40	271.1	5	1364.1
EXT1	47	1.8m	50MB	226.9	40	184.3	7	450.8
EXT2	49	251k	128MB	206.9	41	246.1	8	1.3
BEST	50	82k	52MB	25.5	42	30.2	8	0.8

Table 2.5: Evaluation results using infeasibility and block count constraints

some other instances the additional constraints make EXT2 slower than EXT1, visible in the higher average regarding satisfiable instances. As EXT1 and EXT2 are strong on different instances, a combination BEST, executing both models until the first one has a result, can further provide a major improvement, allowing to solve all 50 instances with an average of 25 seconds. This is also a viable strategy in practice, as it can easily be done using two threads.

In comparison to previous work [MSS18], we can improve the number of instances solved by Chuffed from 48 to 50 and both EXT1 and EXT2 provide significantly better results for infeasible instances both regarding the number of finished instances and the runtime.

2.4.4 Weekly Rest Time

In this section we compare the runtimes for the models with weekly rest time constraints (2.34) to (2.45) added to the models EXT1 and EXT2, resulting in EXT1_WR and EXT2_WR. We use the following settings for weekly rest times.

- $start = [6 \cdot 60,\ 14 \cdot 60,\ 22 \cdot 60]$
- $end = [14 \cdot 60,\ 22 \cdot 60,\ 30 \cdot 60]$
- $wr = 36 \cdot 60$
- $wr_{red} = 24 \cdot 60$
- $e = 1$
- $sp = 4$

The shift types represent typical early, late and night shifts, the weekly rest time is set to 36 hours with the possible exception of 24 hours once within 4 weeks. Instances with only 2 shift types do not have a night shift. All time spans are in minutes.

Table 2.6 and Figure 2.1 show the results of the comparison. The table shows which instances were proven satisfiable or unsatisfiable with weekly rest constraints, together with runtimes for the different models. In the figure, the y-axis depicts the runtime in

	1	2	3	4	5	6	7	8	9	10
sat	✓	×	✓	✓	✓	✓	✓	✓	✓	✓
EXT1	0.3	0.4	0.6	0.5	0.3	0.4	0.7	0.5	1.3	0.8
EXT1_WR	0.9	1.3	2.5	1.4	1.2	0.9	3.8	2.6	58.9	4.9
EXT2	0.6	0.5	1.2	0.5	0.4	0.3	1.4	1.1	7.9	2.3
EXT2_WR	0.9	1.5	3.3	1.4	1.4	0.8	4.2	2.7	152.0	6.0

	11	12	13	14	15	16	17	18	19	20
sat	✓	✓	✓	✓	?	✓	✓	✓	?	✓
EXT1	2.2	0.5	0.7	0.4	8.7	0.9	0.7	8.0	3.0	19.4
EXT1_WR	854.6	3.6	5.1	1.8	TO	5.4	5.7	98.9	TO	404.1
EXT2	16.7	1.6	1.9	0.7	1280.4	2.3	2.6	59.1	113.3	669.7
EXT2_WR	111.3	6.0	4.7	2.0	TO	6.5	8.1	153.4	TO	1466.2

Table 2.6: Evaluation results with weekly rest time constraints

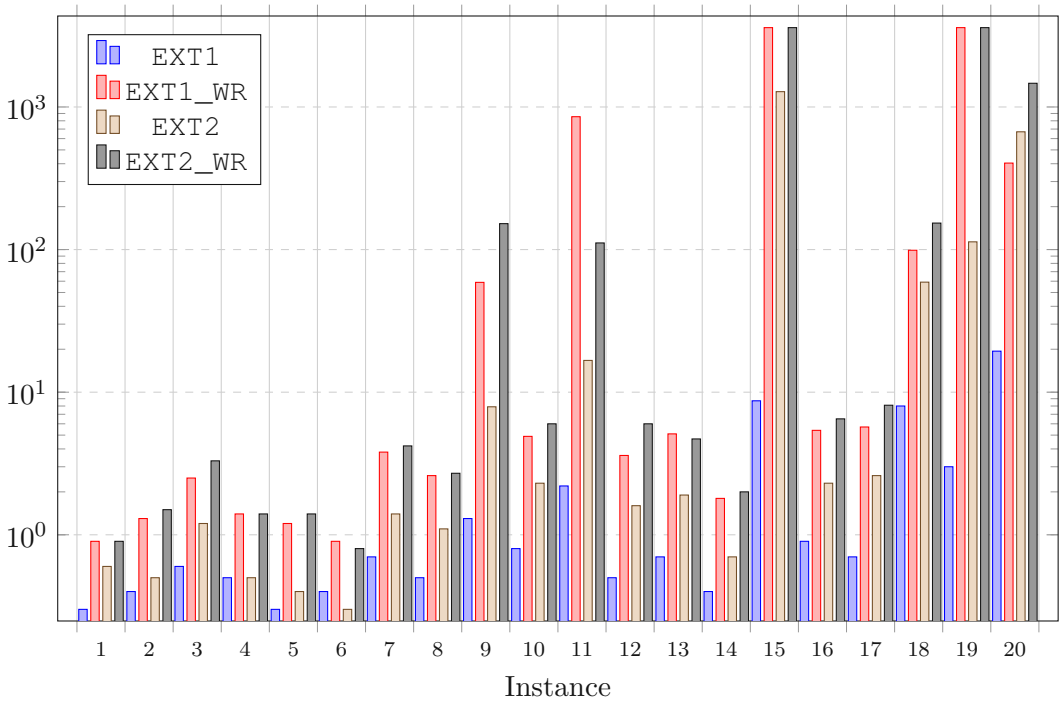


Figure 2.1: Runtime with weekly rest time constraints

seconds. With weekly rest constraints, instance 2 was proven unsatisfiable and instances 15 and 19 ran into timeout while for the other 17 instances a valid result was found.

The results provide several insights. First, for almost all instances it is still possible to find a valid solution with the additional constraints, which is welcome. On the other hand, infeasibility of instance 2 shows that we cannot assume to always find such a solution, therefore, integrating these constraints is useful and important. Further, most schedules actually need the reduced weekly rest exceptions to get feasible, which supports the decision to also model these additional constraints. While 2 large instances that admit a solution without weekly rest can now not be solved within 1 hour, for the majority of instances a solution can still be found within a few seconds, justifying the usage in many practical scenarios.

2.4.5 Optimizing Free Weekends

For the optimization, from a practical point of view, it is not only important to reach a proven best result, but often also to get results fast. This does not have to be a contradiction, as many solvers, including Chuffed, allow the delivery of intermediate solutions using the flag `a`. This allows users to already consider intermediate plans, either accepting them at some point or deciding to spend more runtime on potential further improvements, while, given enough runtime, the best solution is still guaranteed in the end. Therefore, this evaluation shows not only the best reached solutions, but also the first reached solution and how the solver approaches the best solution, using `EXT1` as the base that is extended by the optimization goals.

Figure 2.2 shows the optimization of free weekends maximizing f. Each instance generates one trace where each mark represents a new best solution, allowing to investigate how solutions improve over time. The last mark indicates proof of optimality or timeout. For 18 out of 20 instances a solution is found within the timeout of 3600 seconds, for 14 instances the best solution is found and proven optimal. For a few instances long horizontal lines indicate that no better solution is found, but optimality is not proven. However, for 15 instances a first solution is found within 10 seconds, for 13 of those even optimality is proven within 10 seconds, indicated by the cluster of traces in the lower left corner. This shows viability and practical applicability of the approach for the majority of problems.

Figure 2.3 shows the same graph for the objective to minimize d_m. Steep declines for many instances show that there is more potential for improvement starting from the first results and that results can be improved very fast. Here, 16 instances get a first solution within 10 seconds, 14 are solved to proven optimality within 10 seconds.

For the distance measure d the results are similar as shown in Figure 2.4, giving 15 first solutions within 10 seconds, for 9 instances the best solution is proven in 10 seconds. However, this time 7 instances run into the timeout before proving the optimum. As the definition of the objective is more complex, there is more room for small optimizations, resulting in many new best solutions during the search. With the majority of instances

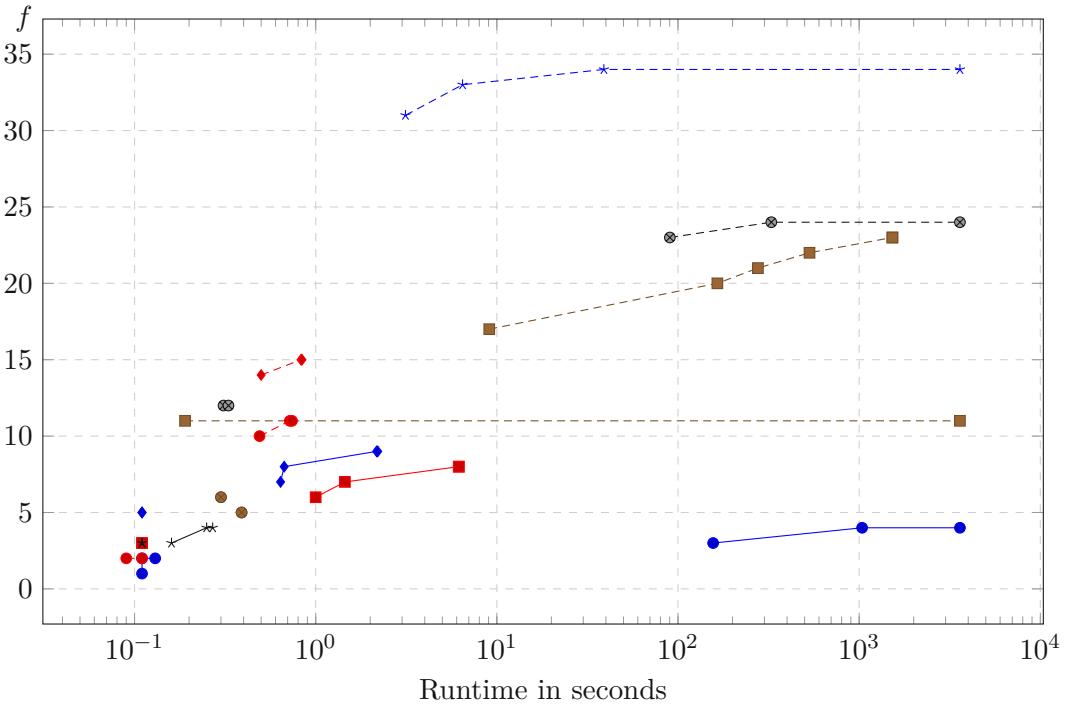


Figure 2.2: Maximizing the number of free weekends f

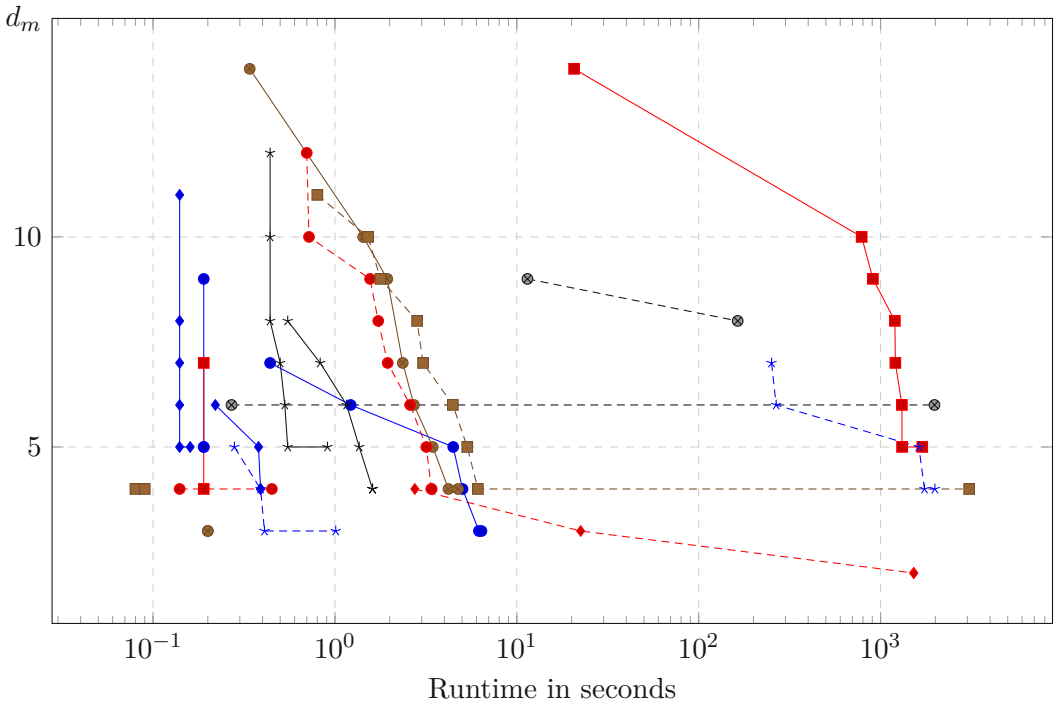


Figure 2.3: Minimizing the maximum distance d_m

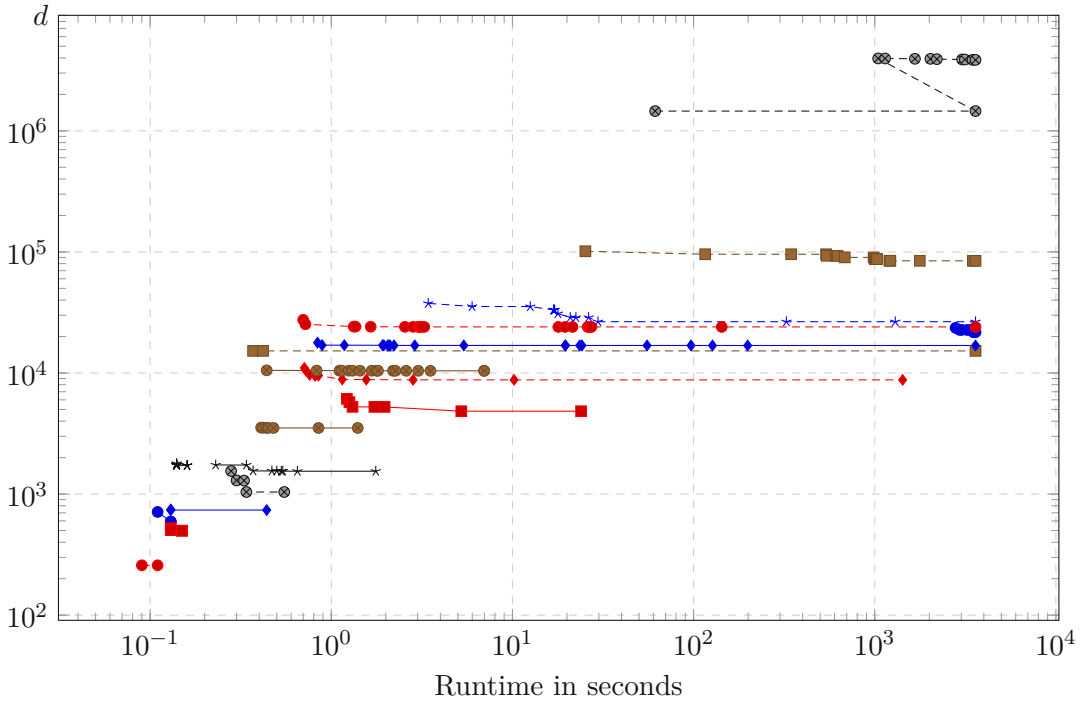


Figure 2.4: Minimizing the distance measure d

having early first solutions, the user can choose a good trade-off between solution quality and runtime.

2.4.6 Conclusion

We have presented a new exact model for Rotating Workforce Scheduling that is implemented using the modelling language MiniZinc and solved with the lazy clause generation solver Chuffed. First we added new model extensions that showed to be very efficient in detecting infeasible instances in very little computational time and used bounds for the number of blocks to define redundant constraints. Then we added new real-life requirements in the form of complex weekly rest time constraints and are still able to solve the majority of the benchmark instances in short computational time. We introduced new objectives to optimize the scheduling of free weekends and showed that for the majority of instances the optimum can be found and proven in short computational time. Moreover, the output of intermediate solutions allows the user to decide about the trade-off between runtime and quality while running the solver.

In total this provides several improvements to the state-of-the-art modelling of the problem that have since been integrated as a core component of the new iteration of the Shift Plan Assistant software by our industry partner XIMES GmbH as shown in Figure 2.5.

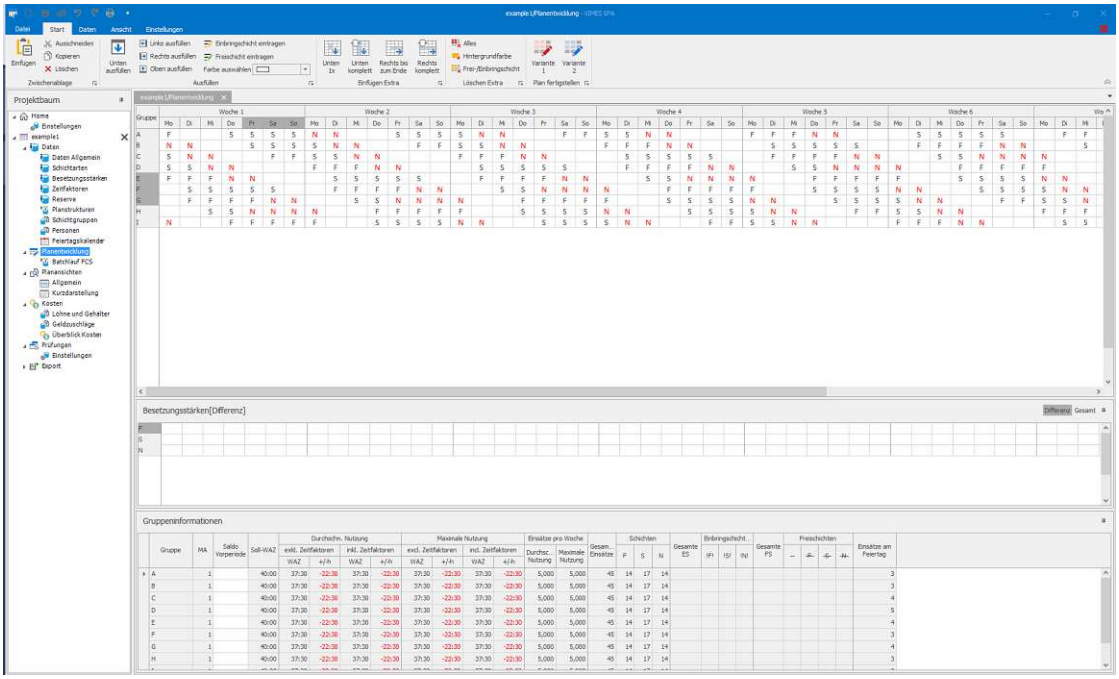


Figure 2.5: Screenshot of the SPA user interface

2.5 Strengths and Weaknesses of Different Models

In Table 2.5 there was a clear distinction of the performance of the models `EXT1` and `EXT2` on different sets of instances, in particular with `EXT1` showing stronger performance on feasible instances, and `EXT2` on infeasible instances. These differences became even more obvious looking at the best performance `BEST` which is significantly better than any of the two models on its own. Further, the results show that runtimes can vary significantly between instances even for the same model and feasibility. While instance size is expected to play a role, there are likely more factors that contribute to these distinctions. Therefore, a more detailed analysis of strengths and weaknesses of the different models, and a deeper investigation of what makes instances hard to solve would be highly beneficial for understanding RWS better.

Instance Space Analysis (ISA) is a methodology developed by Smith-Miles and co-workers [SMBWL14, SMB15, MSM17] in recent years, by extending the algorithm selection problem framework of Rice [Ric76, SM09]. Instances are represented as a feature vector that captures the intrinsic difficulty of instances for various algorithms (or models or parameter settings). By constructing a 2-d projection of a feature-vector representation of instances, ISA allows us to:

1. visualize the distribution and diversity of existing benchmark instances;
2. assess the adequacy of the features;

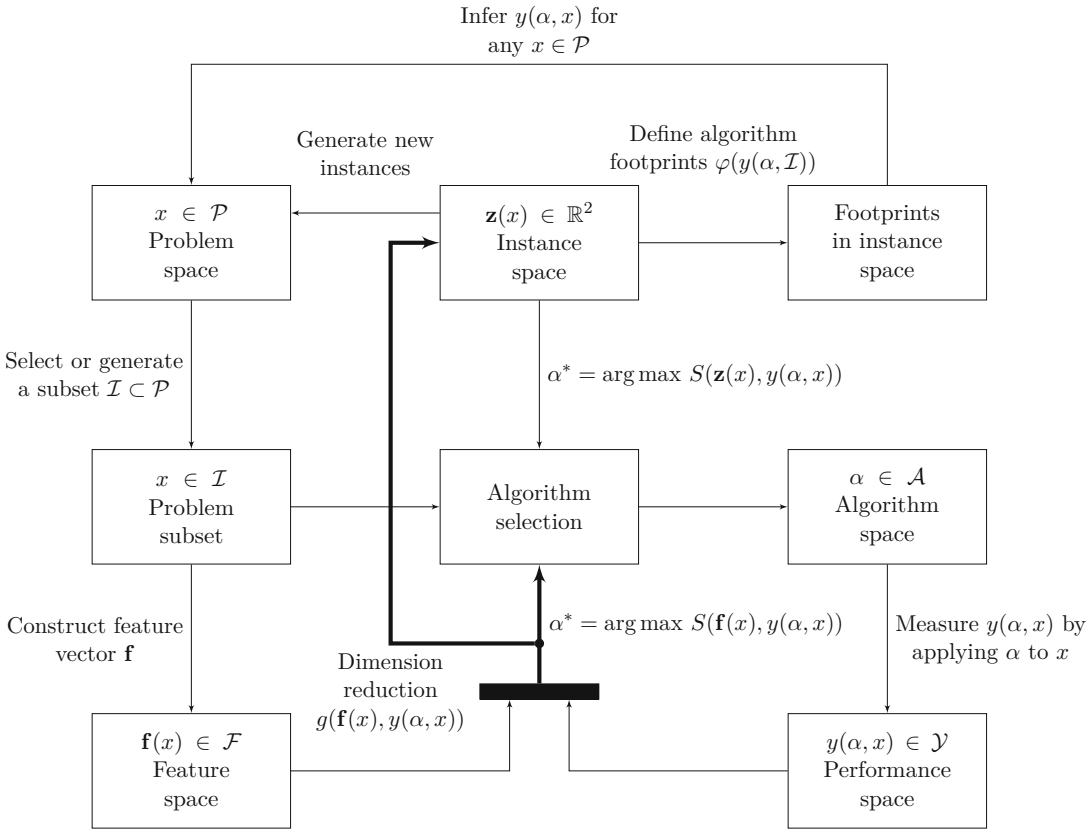


Figure 2.6: ISA framework [SMM21] extending the original by Rice [Ric76]

3. identify and measure the algorithm's regions of strength *footprint* and weaknesses; and

4. distinguish areas of the space where it may be useful to generate additional instances to support greater insights.

Figure 2.6 illustrates the framework and its component spaces. On the top left is the ill-defined problem space $\mathcal{P}$, which contains all the relevant problems to be solved. However, we only have computational results for a subset $\mathcal{I}$. Below is the feature space $\mathcal{F}$, which contains multiple measures that characterize the properties that make an instance in $\mathcal{I}$ difficult. These measures are represented by the vector $\mathbf{f}(x)$. On the center right is the algorithm space $\mathcal{A}$, which is composed of a portfolio of successful algorithms for the problems in $\mathcal{I}$. Below is the performance space, $\mathcal{Y}$, which is the set of feasible values of $y(\alpha, x)$, a measure of the performance of an algorithm $\alpha \in \mathcal{A}$ to solve a problem $x \in \mathcal{I}$.

The meta-data, composed of the features and algorithm performance for all the instances in $\mathcal{I}$, is used to learn the mapping $\mathbf{z}(x) = g(\mathbf{f}(x), y(\alpha, x))$ that projects an instance x from a high-dimensional feature space to a two-dimensional space, which we call the

instance space (bold arrow). Now algorithm selection can be performed, trying to learn a mapping S to find the best algorithm α^* for a given instance x, either on the feature vector $\mathbf{f}(x)$, or on the projection to the instance space $\mathbf{z}(x)$. The instance space can further be used to define footprints (regions of strength) for the algorithms, which can be used to infer the performance $y(\alpha, x)$ of an algorithm α on an unseen problem instance $x \in \mathcal{P}$. The information from the instance space can also be used to identify where new instances are needed to augment $\mathcal{I}$.

In earlier work, the projection $\mathbf{z}$ was achieved using principal component analysis, and applied to problems as diverse as graph colouring [SMBWL14], time series forecasting [KHSM17], and software test case generation methods [OAGSM18]. In this thesis, the methodology is adopted from an application to machine learning algorithms [MVBSM18], where a customized projection algorithm was developed to obtain an optimal projection that aims to expose linear trends in both features and algorithm performance to aid interpretability. The latest version of the evolving methodology can be found in a more recent publication [SMM21]. While the ISA methodology is broadly applicable, it needs to be customised through careful choice of instance features and an understanding of what makes the problem hard [SML12].

2.5.1 Problem and Algorithm Space

In this part of the chapter we will apply Instance Space Analysis on the Rotating Workforce Scheduling problem. ISA is especially useful when different solution methods seem to have different strengths and weaknesses when solving various instances, and when a large, but maybe not optimally distributed set of instances is available for evaluation. Both criteria are fulfilled for RWS. While an analysis would also be interesting for different optimization variants of the problem, this work focusses on the satisfaction version using the extended models `EXT1` and `EXT2`.

Instances

Since ISA needs a large set of instances for reliable results, we took all 2000 instances from the RWS benchmark set for the initial evaluation. These instances include the 20 real life instances, the 30 additional instances used for Table 2.5, and other randomly generated instances [Mus05, Mus06, MSS18]. The additional instances have the following properties:

- 9 to 51 employees
- 2 to 3 shift types
- Length of work blocks: 3 to 4 minimal and 5 to 7 maximal
- Length of days-off blocks: 1 to 2 minimal and 2 to 4 maximal
- Periods of consecutive shifts:

- D: 2 to 3 minimal and 5 to 7 maximal
- A: 2 to 3 minimal and 4 to 6 maximal
- N: 2 to 3 minimal and 4 to 5 maximal

The same forbidden sequences as for real-life examples are used. Initially the temporal requirements for shifts are distributed randomly between shifts based on the total number of working days and days-off (the number of days-off is set to $\lfloor n \times w \times 0.2857 \rfloor$). With probability 0.3 the temporal requirements during weekend are changed (half of these duties are distributed to the temporal requirements of the weekdays).

In the process of ISA we generated additional 4000 instances based on the generator for the first 2000 instances. The generator was modified to generate instances with a larger number of employees and more diverse distribution of workforce requirements during weekdays. As we will explain in later sections these new instances enabled us to cover a larger space of instances. The new instances are available online in the RWS extended benchmark set.

Algorithm Space

We use both the constraint models `EXT1` and `EXT2` in this comparison. Further, since metaheuristics are expected to show very different characteristics to the given exact models, we also include the metaheuristic solver for RWS from literature [Mus05, Mus06]. This solver includes several methods based on tabu search, min-conflicts heuristic and methods that combine min-conflicts heuristic with a tabu mechanism and random walk. The hybrid methods improved the performance of the commercial system for the generation of rotating workforce schedules and are currently state-of-the-art heuristic methods for this problem.

2.6 RWS Features

In order to apply Instance Space Analysis, a set of features needs to be defined. These features need to be able to explain the hardness of instances and the different performances of algorithms to obtain useful results from the analysis. While a subset of the features is eventually used for the instance space, at first a larger set of potentially useful features is created, using three categories.

First we use direct instance parameters or their minimum and maximum values. Second, we contribute new features calculated from the instance specification. Further we obtain numbers of variables and constraints using the conversion of the MiniZinc model as well as the initialization of Chuffed. We explain both the calculation of the features and the reason for including them as well as expectations for their usefulness. In total we present 38 potential features.

2.6.1 Direct Instance Features

While not necessarily expressive enough to explain instance hardness on their own, we include 13 features based on instance parameters:

- Number of employees n: This feature is expected to correlate with hardness is some way as larger n requires larger solutions.
- Number of shifts m: While instances with larger numbers of shifts are also expected to increase difficulty, the instances in the existing benchmarks focus on 2 or 3 shifts and still provide a wide range of difficulty. Therefore we keep the investigation in this area regarding the number of shifts.
- Minimum and maximum length of work blocks ℓ_w and u_w as well as blocks off shift ℓ_O and u_O.
- Minimum, maximum and average for each of the sets $\{\ell_s \mid s \in \mathbf{A}\}$ and $\{u_s \mid s \in \mathbf{A}\}$.
- Number of forbidden sequences f.

2.6.2 Advanced Instance Features

We select the following 14 new features based on observations working with the algorithms that might help to explain instance hardness:

- *workFraction* (2.58): Percentage of all days spent working. Equation (2.21) defines r as the sum of all requirements.

$$workFraction = \frac{r}{w \cdot n} \tag{2.58}$$

- Minimum and maximum of *shiftFraction* (2.59): Distribution of requirements between shifts.

$$shiftFraction = \left\{ \frac{\sum_{j=1}^{w} R_{s,j}}{w \cdot n} \;\middle|\; s \in \mathbf{A} \right\} \tag{2.59}$$

- *blockTightness* (2.62): Equations (2.60) and (2.61) recall earlier definitions regarding lower and upper bounds. Equation (2.26) defined *low* as a lower bound for the total number of working blocks in the solution. As the problem is cyclic, the number of work blocks and free blocks needs to be equal, therefore the maximum of the lower bounds for both types is calculated. Equation (2.27) did the same for *up* for the upper bound. This feature can be used to identify instances as infeasible when the value is negative. This fact is used in the constraint models. Further, low positive values seem to lead to harder instances according to the experience in preliminary

testing. This might be explained by the reduced possibilities in choosing block lengths for a valid solution.

$$low = \max\left\{\left\lceil \frac{r}{u_w} \right\rceil ; \left\lceil \frac{n \cdot w - r}{u_O} \right\rceil\right\} \tag{2.60}$$

$$up = \min\left\{\left\lfloor \frac{r}{\ell_w} \right\rfloor ; \left\lfloor \frac{n \cdot w - r}{\ell_O} \right\rfloor\right\} \tag{2.61}$$

$$blockTightness = up - low \tag{2.62}$$

- *minAvgBlockLength* (2.63) and *maxAvgBlockLength* (2.64): Lower and upper bound for the average block length (work block + consecutive free block) as a different way to use *blockTightness*.

$$minAvgBlockLength = \frac{w \cdot n}{up} \tag{2.63}$$

$$maxAvgBlockLength = \frac{w \cdot n}{low} \tag{2.64}$$

- Minimum and maximum of *shiftBlockTightness* (2.67): Freedom in choosing block lengths for individual shift types. Equations (2.65) and (2.66) calculate lower and upper bounds for the number of blocks for each type $s \in \mathbf{A}$.

$$low_s = \left\lceil \frac{\sum_{j=1}^{w} R_{s,j}}{u_s} \right\rceil \tag{2.65}$$

$$up_s = \left\lfloor \frac{\sum_{j=1}^{w} R_{s,j}}{\ell_s} \right\rfloor \tag{2.66}$$

$$shiftBlockTightness = \{up_s - low_s \mid s \in \mathbf{A}\} \tag{2.67}$$

- Minimum and maximum of *shiftDayFactor* (2.68): Regularity of shifts throughout the week.

$$shiftDayFactor = \left\{ \frac{\min\{R_{s,j} \mid j \in \mathbf{W}\}}{\max\{R_{s,j} \mid j \in \mathbf{W}\}} \;\middle|\; s \in \mathbf{A} \right\} \tag{2.68}$$

- Minimum and maximum of *dayFraction* (2.69): Workload in relation to the number of employees for individual days.

$$dayFraction = \left\{ \frac{\sum_{i=1}^{m} R_{i,j}}{n} \;\middle|\; j \in \mathbf{W} \right\} \tag{2.69}$$

- Minimum and maximum of *dailyChange* (2.70): Change in workload between consecutive days.

$$dailyChange = \left\{ \sum_{i=1}^{m} R_{i,j+1} - \sum_{i=1}^{m} R_{i,j} \right\} \tag{2.70}$$

2.6.3 Model Features

Further we used the standard constraint model of RWS modelled in MiniZinc and the initialization in Chuffed to obtain numbers of variables and constraints for 11 further features.

From the MiniZinc to FlatZinc conversion statistics we obtained the number of boolean and integer variables as well as the numbers of boolean and integer constraints. From the initialization of the instance in Chuffed we obtained the number of variables, the number of propagators, the number of SAT variables, the numbers of binary, ternary and long clauses as well as the average length of long clauses.

There could be an advantage in such features as they are rather problem independent, which might allow transferring results to other problems. On the other hand, these features might not capture the structure of the instances well enough to be useful.

2.6.4 Initial Experiments

While we could immediately try ISA with our features, we decided to start with experiments using classical machine learning techniques on a test run comparing the base constraint model with the metaheuristic on the original set of 2000 instances using a timeout of 200 seconds. These are meant to give us a good impression whether our features are helpful in learning. Table 2.7 provides results using Random Forests [Bre01] for various predictions. The result column reports accuracy (classification) or correlation coefficient (regression). The results were obtained using Weka with the default settings and 10 fold cross validation.

For the best method, 3 classes are possible (Chuffed, metaheuristic, tie). Solved (boolean) corresponds to predicting whether an instance could be solved by either Chuffed or the metaheuristic without timeout. Feasible (boolean) only uses instances which could be solved by either method, where the task is to predict whether the instance was solved or proven infeasible. Regarding the runtime, the distribution shows many instances with very short runtime, fewer with longer runtimes and a number of timeouts. Therefore, predicting the magnitude of the runtime worked much better than directly predicting the runtime.

The prediction results are overall promising that we have meaningful features to proceed with ISA. While the results on predicting the best method are below 80%, for the prediction of hard (not solved by any method) or feasible instances we can achieve more than 90% accuracy. For the runtimes, the result shows the correlation coefficient exceeding 0.85.

Regarding the features, we compared using all 38 features to skipping either the direct, advanced, or model features. The results show for all predictions that the best results are obtained not using all features, but excluding the model features, while skipping the advanced features leads to the worst results. Therefore, we are confident that the

Target	Features	Result
Best method	all	76.15%
Best method	no direct	75.55%
Best method	no advanced	70.90%
Best method	no model	77.85%
Solved	all	92.35%
Solved	no direct	90.90%
Solved	no advanced	88.70%
Solved	no model	92.40%
Feasible	all	97.97%
Feasible	no direct	97.28%
Feasible	no advanced	91.96%
Feasible	no model	98.35%
log(runtime Chuffed)	all	0.855
log(runtime Chuffed)	no direct	0.839
log(runtime Chuffed)	no advanced	0.766
log(runtime Chuffed)	no model	0.866
log(runtime Meta)	all	0.878
log(runtime Meta)	no direct	0.856
log(runtime Meta)	no advanced	0.788
log(runtime Meta)	no model	0.883

Table 2.7: Results for various predictions using Random Forests

advanced features are useful for ISA and proceed with 27 features, excluding the model features.

2.7 Instance Space Analysis

We perform the Instance Space Analysis using the Matlab toolkit MATILDA. It uses a configuration file to specify parameters for the analysis and then performs the following steps: It bounds extreme outliers and does normalization using a Box-Cox and Z transformation. Next, features with low diversity are removed. Features with high correlation to performance are retained and a clustering step is applied. Then it calculates the projection to the 2-dimensional instance space and the footprints of the algorithms within the space.

The algorithms are executed on an Intel Core i7-7500 CPU with 2.7 GHz and 16 GB RAM using a timeout of 1000 seconds. The metaheuristic is executed three times on each instance to account for slight variation in the results.

The settings for MATILDA are mostly the default settings, using all processing steps. The performance metric is set to the runtime (lower is better). The diversity threshold is

reduced to 0.01 (eliminate features where the number of unique values is less than 1% of the number of instances), as some integer features do not have high diversity, but might still be interesting for the evaluation, e.g., *blockTightness*. The correlation threshold is 5, the silhouette threshold for clustering is 0.45.

In the following we first report results for the original instances and discuss our conclusions from these results. Then we evaluate the extended set of instances generated based on those conclusions and discuss the new insights we get.

2.7.1 Original Instances

For the first set of experiments we used the existing dataset of 2000 instances. We evaluate the results from the base constraint model as well as the metaheuristic.

Instance Distribution

Figure 2.7 shows the resulting instance space and the distribution of the selected features. Note that values on the scale correspond to normalized feature values. Equation (2.71) shows the projection matrix applied to the Box-Cox transformed and normalized feature values.

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} -0.45 & -0.39 \\ 0.45 & 0.40 \\ 0.50 & 0.08 \\ -0.32 & 0.37 \\ 0.23 & -0.63 \end{pmatrix}^{\mathsf{T}} \cdot \begin{pmatrix} maxShiftDayFactor' \\ maxDayFraction' \\ employees' \\ minAvgBlockLength' \\ blockTightness' \end{pmatrix} \tag{2.71}$$

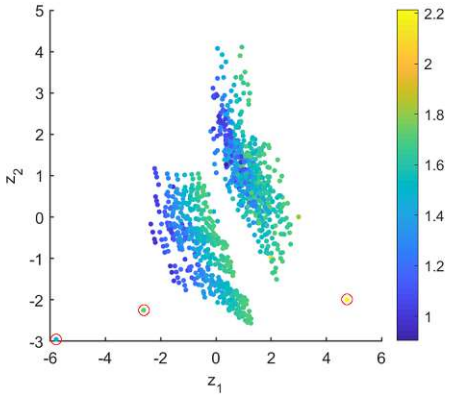
With *blockTightness* and *minAvgBlockLength*, two features describing possible distributions of block lengths were selected by the analysis. Further, the number of employees representing the size of the instances is present. The other two features represent the distribution throughout the week (*maxShiftDayFactor*) and the daily workload (*maxDayFraction*).

One immediate observation is that the instances cluster in two areas with a separating gap in between. The feature most clearly describing the difference is *maxShiftDayFactor*, where the left cluster almost uniformly has the highest value. In fact, this cluster consists of instances with constant demand on different days of the week while the other cluster has varying demand on some days.

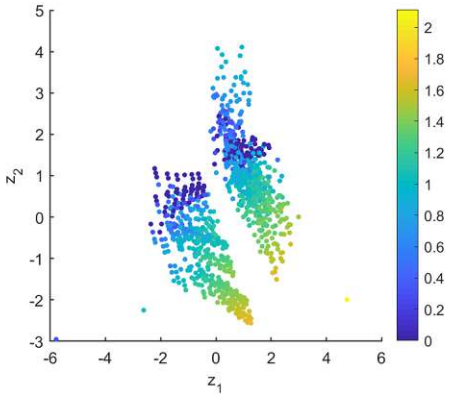
Further, the first 20 instances are not randomly generated, but stem from real-world scenarios. Some of these show as outliers in the results (see highlights in Figure 2.7a), hinting that the randomly generated instances do not cover all of those scenarios.

Algorithm Evaluation

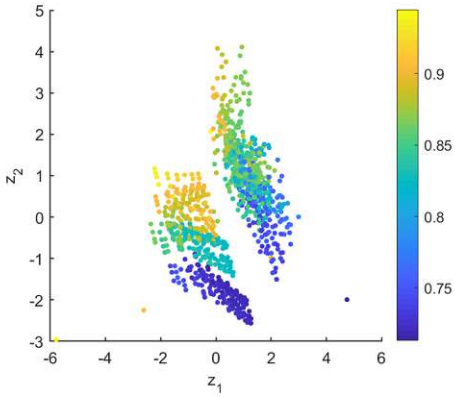
Figure 2.8 shows the results of the evaluated algorithms on the original instances. Runtime values are also normalized, lower is better. The results show two trends. First,



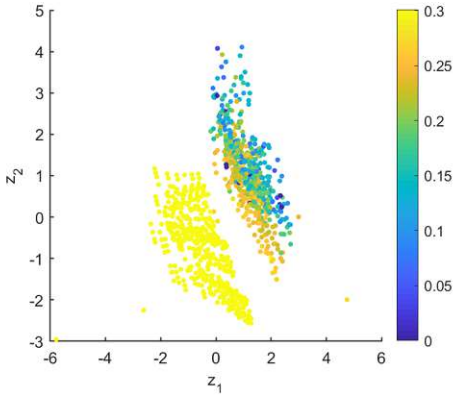
(a) *employees*



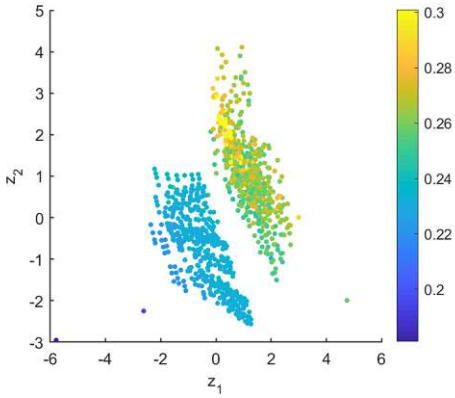
(b) *blockTightness*



(c) *minAvgBlockLength*

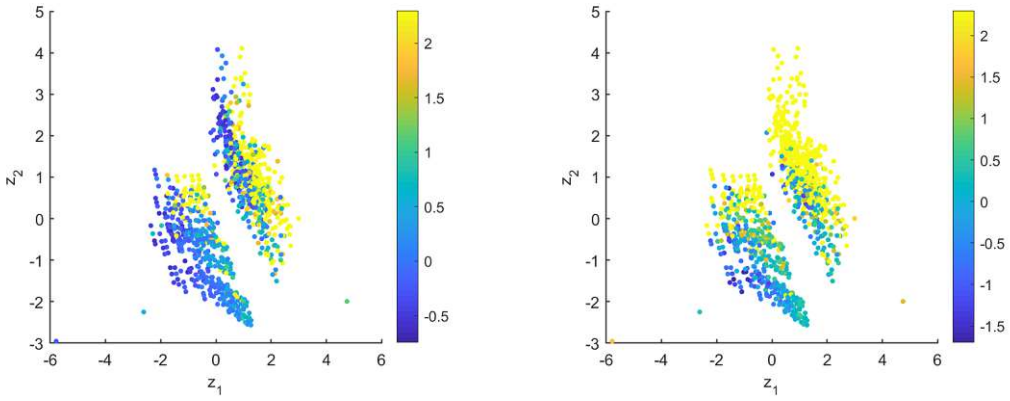


(d) *maxShiftDayFactor*



(e) *maxDayFraction*

Figure 2.7: Selected features for the original instance set



(a) Early Chuffed model (b) Metaheuristic

Figure 2.8: Algorithm runtimes for the original instance set

instances higher along the z_2 coordinate seem to be harder. This distribution is especially visible for the metaheuristic. A lower z_2 coordinate, on the other hand, corresponds to higher *blockTightness* and lower *minAvgBlockLength*, translating to more possibilities for choosing the lengths of blocks. Second, within each cluster, instances with higher z_1 coordinate seem to be harder. This mostly corresponds to higher numbers of employees.

2.7.2 Extended Instances

Due to the gaps and outliers explained in the previous section as well as the fact that algorithm performance seems similar in the two clusters of instances, we decided to extend the set of instances with two goals.

First, we want to close the gap between the clusters of instances, mainly by changing the way the instance generator handles the distribution of shifts across days. Second, we want to expand the borders of the clusters in general in order to explore a wider region of the instance space and cover the real life instances more comprehensively.

However, some features like the number of employees are unbounded or may have natural restrictions on values that make sense for practical instances, such as avoiding extremely low demand relative to the number of employees. We increased the maximum number of employees within the generated instances from 51 to 108. For several other generator options we used wider ranges of values than before.

In total we generated 4000 new instances, dropping 520 of those with *blockTightness* lower than the previous minimum, as these instances are infeasible anyway and can be identified rather easily. Together with the original instances, we evaluated 5480 instances for the following results.

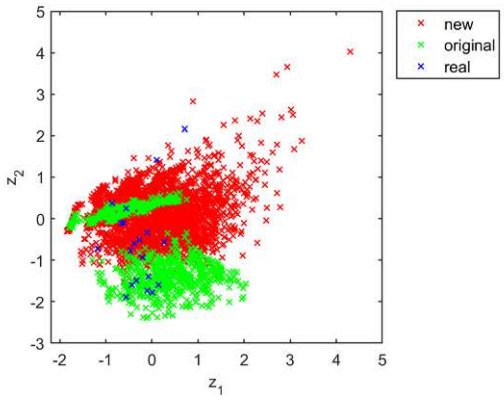


Figure 2.9: Instance distribution for the extended instance set

Instance Distribution

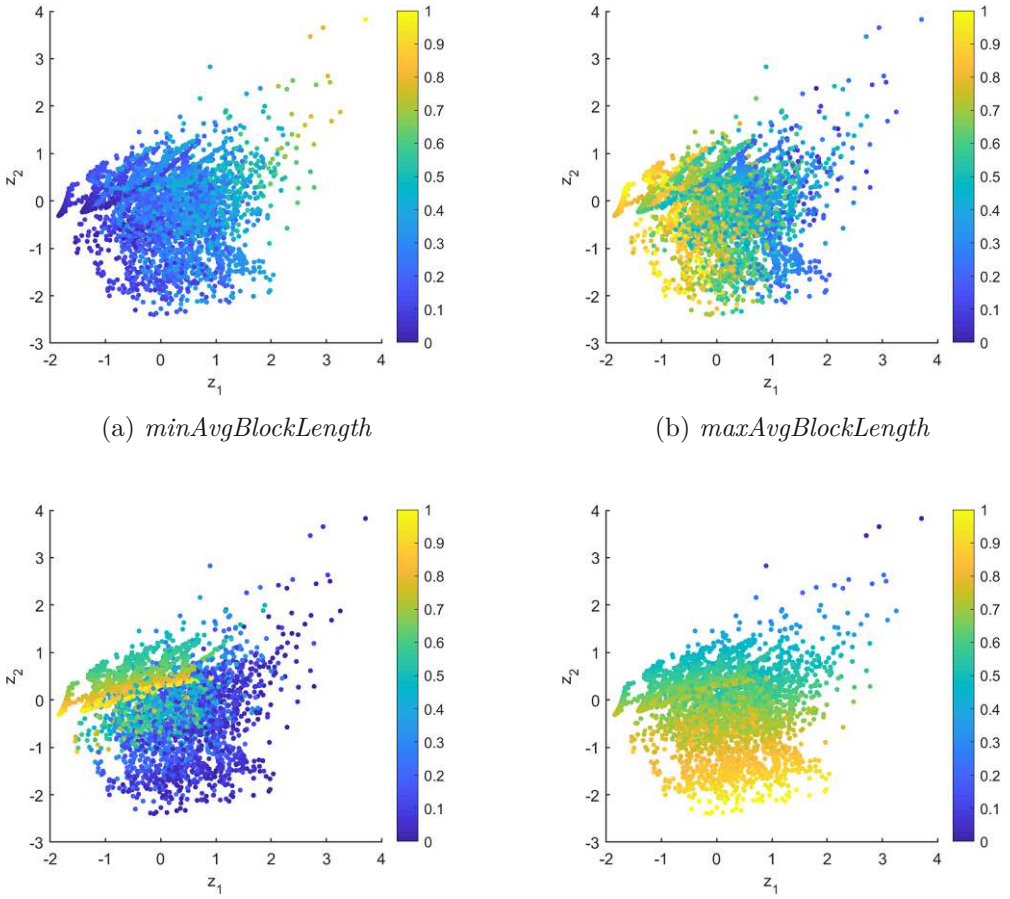
Figure 2.9 shows the distribution of original and new instances in the new instance space. Note that due to the re-computation of the space with more instances and all three algorithms the selection of features changed slightly. Also, the whole projection is now rotated roughly 90 degrees clockwise from the original projection. The new instances mostly fill the gap between the previous clusters and extend the instances towards the area with high values in both coordinates. Almost all original instances (19 out of 20) are now within the more densely populated areas of the instance space, indicating good coverage of real-world scenarios.

Figure 2.10 shows the selected features for the extended instance set. The projection matrix is given in Equation (2.72).

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} -0.31 & 0.31 \\ 0.02 & -0.57 \\ -0.47 & -0.08 \\ 0.44 & 0.15 \end{pmatrix}^{\mathsf{T}} \cdot \begin{pmatrix} minDayFraction' \\ maxDayFraction' \\ maxAvgBlockLength' \\ minAvgBlockLength' \end{pmatrix} \tag{2.72}$$

We see that this time there is no obvious gap between the instances. There is a trail of thinly distributed instances with high values on both coordinates identified by very high values of *minAvgBlockLength*. Instances at this extreme of the instance space are not practical (very long blocks of consecutive work days) and are also clearly not feasible as seen in Figure 2.13, therefore, we saw no need to create further instances in this region.

Regarding the selected features the representation of *blockTightness* was replaced by *maxAvgBlockLength*. The number of employees is not present any more, even though a wider range of values is used in the new instances, hinting that other factors are more critical for instance hardness in this more diverse dataset. *maxShiftDayFactor*, previously identifying a whole cluster of instances, is not present either. However, its distribution is very similar to the ratio between minimum and maximum of *dayFraction*.



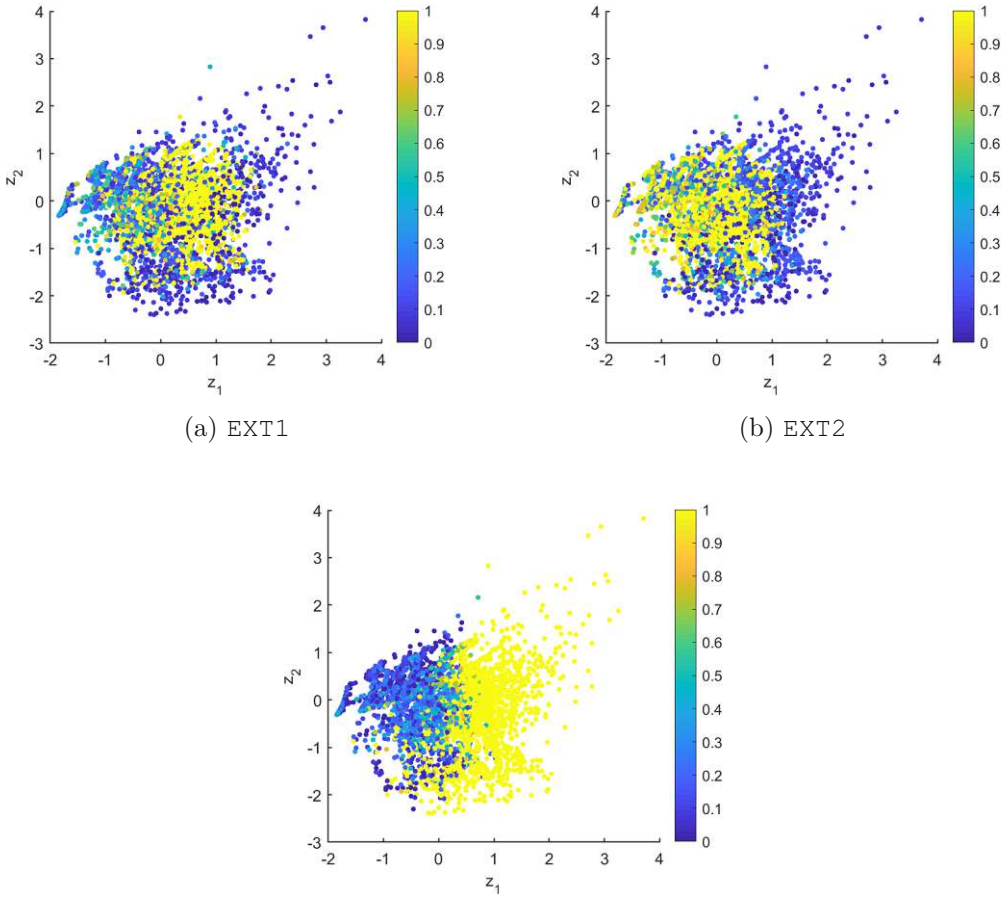
(a) *minAvgBlockLength* (b) *maxAvgBlockLength*

(c) *minDayFraction* (d) *maxDayFraction*

Figure 2.10: Selected features for the extended instance set

In total, we can identify two main axes in the instance space. In the z_1 direction, we can identify instances with low minimum and high maximum average block length for low z_1 values, while high minimum and low maximum can be found for high z_1 values. Therefore, low z_1 corresponds to a wide range of possibilities for choosing block lengths, while high z_1 corresponds to a narrow range of possibilities or even infeasibility, as seen later.

In the z_2 direction, low values identify regions with low minimum and high maximum *dayFraction*. This corresponds to large differences in demand between different days. High values identify regions with high minimum and low maximum *dayFraction*, corresponding to little to no difference between different days.



(a) EXT1

(b) EXT2

(c) Metaheuristic

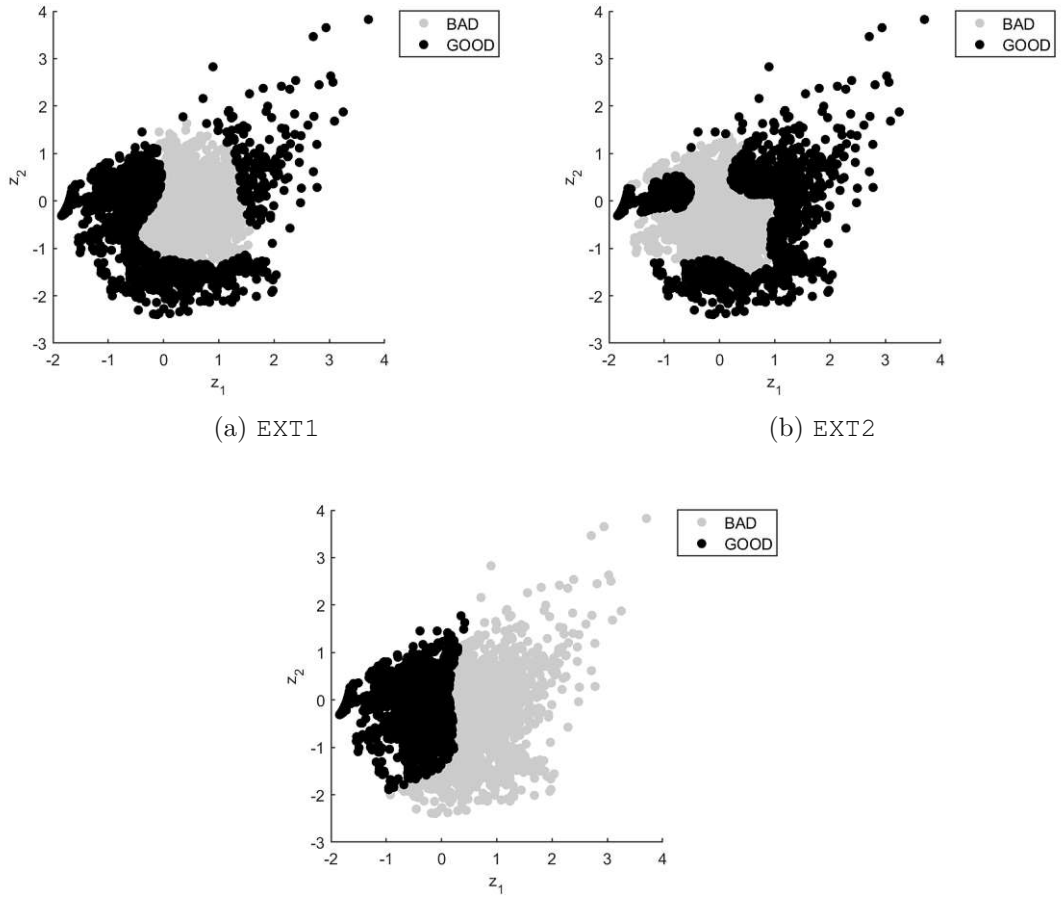
Figure 2.11: Algorithm runtimes for the extended instance set

Algorithm Evaluation

Figure 2.11 shows the runtimes of the algorithms on the extended instances, again for a timeout of 1000 seconds. We can draw several conclusions from these figures. First, both Chuffed versions are very fast in the low z_2 and high z_1 areas, while the metaheuristic cannot solve instances in this area. Regarding the central area, the Chuffed versions show different distributions. For EXT1 the bad area is more to the right of the center (higher z_1), while for EXT2 it is more to the left of the center (lower z_1). The metaheuristic shows fast runtimes in the area of low z_1 and high z_2.

Figure 2.12 shows this information in the form of footprints, generalizing from individual instances to a more uniform representation of good and bad areas for the algorithms.

Figure 2.13 shows the algorithm results in the categories feasible solution (green), proven infeasible (red) and timeout (yellow).



(a) EXT1

(b) EXT2

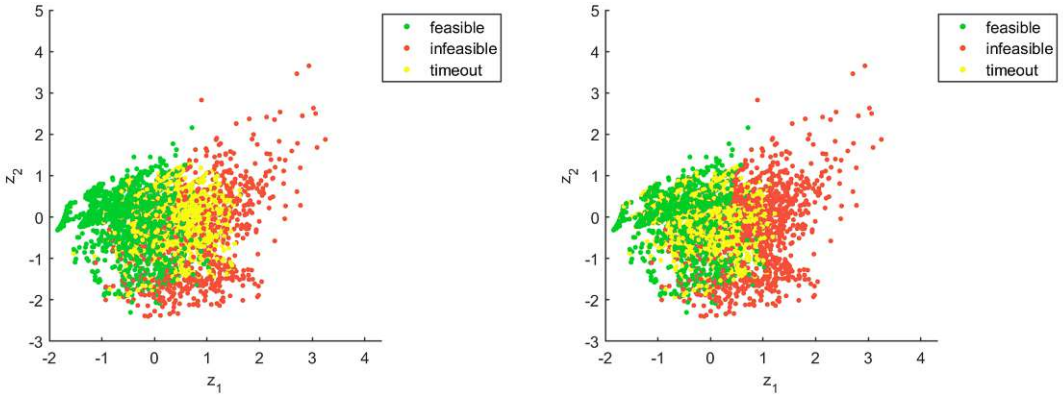
(c) Metaheuristic

Figure 2.12: Algorithm footprints for the extended instance set

For the results from Chuffed, both versions show mostly feasible results for low z_1 coordinates and mostly infeasible results for high z_1 and low z_2 coordinates. However, we can see clear differences in the distribution of the timeout instances. For EXT1, the timeouts are more towards the infeasible side, while for EXT2, they are more towards the feasible side. This indicates that EXT2 is stronger at detecting infeasibility at the cost of lower performance in the feasible area while EXT1 is stronger in the feasible area.

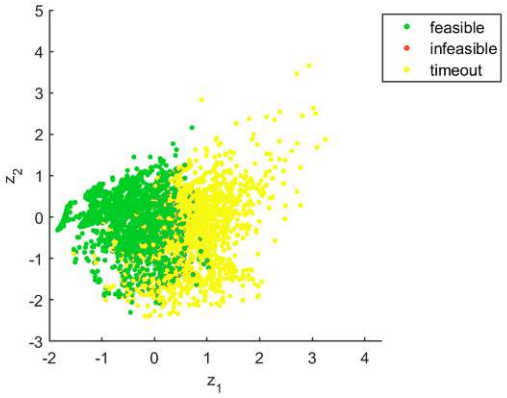
For the metaheuristic the result is different as it is not able to identify infeasible instances, but rather searches for a result until timeout. In comparison to the combined chart the metaheuristic seems to work very well for most of the feasible instances.

Several conclusions can be drawn from the next charts, showing the distribution of feasible and infeasible instances as well as the number of successful algorithms. The combined feasibility chart shows instances where at least one algorithm found a feasible solution (green), at least one algorithm proved infeasibility (red), or timeout on all algorithms

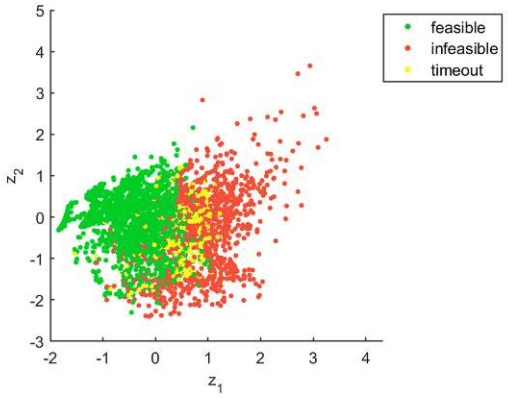


(a) EXT1

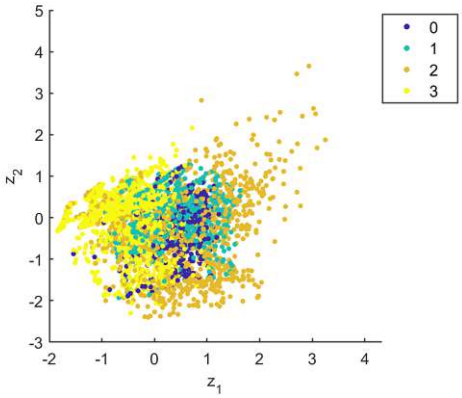
(b) EXT2



(c) Metaheuristic



(d) Combined results



(e) Number of obtained results

Figure 2.13: Algorithm results for the extended instance set

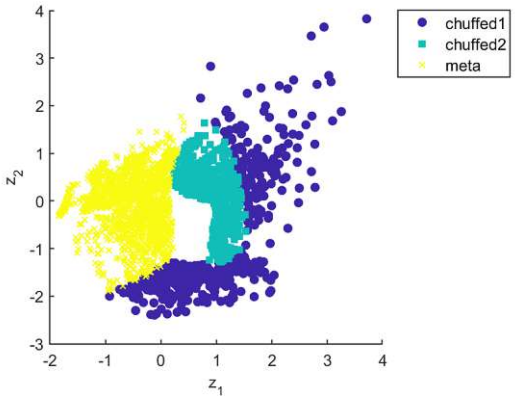


Figure 2.14: Algorithm portfolio

(yellow). First, we notice a clear distribution of feasible and infeasible instances using our features. Further, there is a low number of unsolved instances, also compared to individual algorithm results, showing that the portfolio of algorithms can combine the different strengths of the algorithms.

Regarding the number of solutions, in the low z_1 region, for most instances all algorithms could find a solution. For high z_1 or low z_2, both Chuffed models could prove infeasibility for most instances. Along the border, however, there is a cluster of instances with a low number of successful algorithms, for several instances actually no algorithm found a solution. This tells us that instances on the border of feasibility are usually harder. Among those, especially instances with medium z_2 are hard. Furthermore, our set of instances covers both the hard border region and easier regions on either side, indicating a good diversity of instances.

Algorithm Portfolio

In order to combine the strengths of the different algorithms, a portfolio approach can be used. Figure 2.14 shows that for clearly feasible instances the metaheuristic is the best choice, while for the infeasible space and feasible instances close to the feasibility border the Chuffed models are the best choice. `EXT2` is better in the central infeasible area close to the border with medium z_2 values, while `EXT1` is best for the outer areas. For the white area at the center no particular method has a high confidence to succeed. For these instances longer computation times or trying multiple methods at once might be necessary.

2.8 Conclusion

In this chapter we have studied the Rotating Workforce Scheduling (RWS) problem in detail, including several model extensions and a detailed analysis of strengths and

weaknesses of different exact models in comparison with a metaheuristic approach.

First we have presented Constraint Programming models that extend previous state of the art in several ways, including new constraints to identify infeasible instances faster, additional real-life requirements like weekly rest time, and several different ways to turn the satisfaction problem into an optimization problem to improve the distribution of free weekends for the employees. Including these extensions based on real-life experiences and requirements from our industry partner contributed to Research Goal 1 (Section 1.1.1) of this thesis.

The evaluation of the new models showed that the new constraints enable faster solutions for the standard version of RWS, as well as fast solutions of high quality for the extended versions of the problem, contributing to Research Goal 2 (Section 1.1.2) of the thesis.

Since different models showed different strengths and weaknesses, we used Instance Space Analysis (ISA) to evaluate the performance of two constraint models and a metaheuristic on the RWS problem. We defined new sets of features and evaluated them by using them for machine learning with Random Forests. With the results of the ISA using the original dataset, we were able to produce a larger, more diverse dataset to cover a larger part of the instance space covering the real life instances and closing gaps in the generated instances, also contributing to Research Goal 2 (Section 1.1.2).

Using the extended dataset, we identified four features that provide a good projection for the analysis of the instance space, corresponding to the possibilities for block lengths and the distribution differences between different days. We found different strong and weak areas for the individual algorithms and showed that in combination the algorithms cover the overall instance space well. We also identified areas of feasible and infeasible instances and linked the hardness of instances with the transition between these areas. We further presented a portfolio that uses the most suitable algorithm based on the position of the instance in the instance space. The contributions using ISA on the problem add to Research Goal 3 (Section 1.1.3).

While much progress has been made on the problem, including the practical application of the problem, further work on RWS could for example extend explanations for failure further than simple cases like individual constraints detecting infeasibility or the output of intermediate variables. To look into infeasibility in more detail, e.g., minimal unsatisfiable subsets could be used to explain infeasibilities in a more general way. Future work using ISA could also include the analysis of further dimensions of the problems like larger numbers of shifts or extended versions of the problem like optimization variants.

CHAPTER 3

Network Modeling for Minimum Shift Design

In the previous chapter the goal was to assign employees to predetermined shifts. However, to get to this step, first the shifts themselves need to be specified. In some cases this might be easy, e.g., when a 24-hour schedule is split into three 8-hour shifts and the number of needed employees per shift is known. On the other hand, when demand fluctuates throughout the day, a decision has to be made regarding the placement of various shifts along the day to cover the demand as well as possible. This could be done in an integrated fashion, performing the assignment of employees and the placement of shifts at the same time, which has very high computational complexity, or the steps could be considered separately, first determining the shifts and the required number of employees per shift before assigning concrete employees in the next step.

The Minimum Shift Design (MSD) problem aims at this first step of designing the required shifts while minimizing the amount of understaffing and overstaffing regarding the given demand and in addition keeping the total number of different shifts to a minimum as well, as schedules with fewer shifts are deemed to be easier to read and manage. As the problem has high practical importance, there has been work in literature as well as implementations in commercial software like the Operating Hours Assistant (OPA) [GMS01] from XIMES GmbH.

Although previous techniques in the literature give good results in practice, solving of some challenging instances to optimality is still an open question. This chapter presents new models for the problem built in the solver independent modelling language MiniZinc [NSB+07]. The first uses a direct representation of the problem and serves for comparison. Another model is built based on counting remaining shift times. The final model turns some of the constraints into a network flow constraint to model the assignment of shifts including understaffing and overstaffing.

All models are then evaluated using both the lazy clause generation solver Chuffed [CSS+18] and the MIP solvers Gurobi [GO18] and CPLEX [CPL09] on the standard benchmark datasets that range from instances allowing a perfect cover to hard instances where optima were not known so far. The results show that the model based on network flow clearly outperforms the other models and that MIP solvers, especially Gurobi, perform very well using the new model. With this model, for the first time we can solve all benchmark instances to optimality within short computational time. These results have been published at CPAIOR 2019 [KM19].

3.1 Related Work

The MSD problem is described in [MSS04] where a tabu search approach with several neighbourhood relations is proposed. The first dataset of the MSD benchmark set was introduced by this work. This work also considers a forth objective that is not used in the following works, setting bounds for the average number of duties per week. While the models presented in this chapter also exclude this forth objective, as the comparison is done on the most commonly used version of this problem, it is actually very important in many practical applications. Therefore, in Chapter 5, the full version of the problem including the forth objective is used.

Di Gaspero et al. [DGGK+07] provide an improved local search technique and analyse the complexity of the problem. They describe the relation of the problem to a min-cost max-flow problem, therefore raising the connection to the usage of network flow. More precisely, the problem could be represented as a min-cost max-flow problem as long as the minimization of shifts is not taken into account. They use this relation to provide a hybrid method combining min-cost max-flow with local search. Datasets 2 and 3 were introduced by this work.

Recent approaches include a formulation in answer set programming [AGM+16] that is able to provide several optimal results, while it struggles with instances that do not admit an exact cover of the demand. Good results have been achieved using MIP [Koc15], providing optimal solutions for most instances with Gurobi and CPLEX using a runtime of two hours. Those approaches also consider dataset 4.

A related problem is shift scheduling [Dan54, BJ90, Tho95, Ayk00, BKP08], which deals with one or more days, potentially including activities, skills or breaks. In contrast, MSD deals with the minimization of the number of shifts across several days including cyclicity. Recently, regular and context-free languages [QR10, CGQR11, CGR11, CGR13, HLBSAR19] as well as column generation techniques [LBD+12, RLM12] have been applied to shift scheduling successfully. In further work [LBD+12, RLM12], networks are used where paths correspond to feasible assignments of activities. There has also been work integrating shift design with other stages of employee scheduling [SN19].

Further the scheduling of breaks within shifts [BGM+08, BGM+10, WM14] or the combined problem of shift and break scheduling [DGGM+10, DGGM+13, APU18] are

considered by several authors. In [APU18] results for the first MSD benchmark dataset are given as well, however, applying weights for under- and overstaffing per timeslot instead of per minute.

3.2 Problem Description

The MSD problem aims at generating a set of shifts and determining the required number of employees for each shift on each day in the planning period. The specification is mainly based on Musliu et al. [MSS04]. The problem input is defined as follows:

- n consecutive timeslots $[a_1, a_2), [a_2, a_3), \ldots, [a_n, a_{n+1})$, each slot $[a_i, a_{i+1})$ with the same slotlength sl in minutes and with a requirement for workers R_i indicating the optimal number of employees required at that timeslot. A full day consists of a whole number of slots, therefore, sl divides $24 \cdot 60$. The time from a_1 to a_{n+1} is called the planning period and consist of D days, where D is integer.
- S shift types $t_1, \ldots, t_S$, each type t_s associated with a minimum and maximum start time $smin_s$ and $smax_s$, given in timeslots of the current day, as well as minimum and maximum length $lmin_s$ and $lmax_s$, given in timeslots. Note that shifts can extend to the following day. We consider a cyclic schedule where shifts extending beyond a_{n+1} continue from a_1. For simplicity, we assume that shift types never overlap, i.e, no shift can be of two types at the same time. If necessary, new shift types can be introduced to resolve overlap, as a redefinition of overlapping shift types, potentially with more types, but no overlap, is always possible.

Note that the concept of a shift type for the purpose of this problem is slightly different from RWS. E.g., an early shift in RWS is already fixed, while for MSD it defines windows for the possible design of shifts. After the shifts are fixed in MSD, an RWS problem might be solved for the assignment to individual employees, where either all instances of early shifts are grouped, or different instances might be treated as different shift types for RWS.

For illustration purpose throughout the chapter consider the following example `TOY` with $D = 1$ and $sl = 180$. The two shift types are given in Table 3.1. Corresponding values are provided both in timeslots and hour:minute notation. The demand is given as $R = [1, 1, 4, 3, 5, 5, 2, 3]$.

In order to provide a feasible solution, shifts are only allowed within the time windows specified by the shift types. Among feasible solutions, the following criteria have to be minimized:

- T_1: Sum of excesses of workers across the whole planning period in minutes.
- T_2: Sum of shortages of workers across the whole planning period in minutes.

	$smin_s$		$smax_s$		$lmin_s$		$lmax_s$	
Shift type	Slots	Time	Slots	Time	Slots	Time	Slots	Time
t_1	2	6:00	2	6:00	2	6:00	4	12:00
t_2	4	12:00	7	21:00	2	6:00	4	12:00

Table 3.1: Shift types for TOY

- T_3: Number of shift classes in use. A shift class is considered in use if there is any day in the planning period where employees are assigned to this shift. (I.e., shifts belong to the same shift class if they have the same length and the same start time relative to the day they are assigned to.)

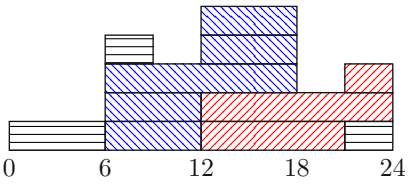
- T_4: Deviation of the average shift length from the defined window $[w_{min}, w_{max}]$. This window is derived from a minimum and maximum number of shifts per week that are supposed to be assigned to each employee in average, and the number of working hours per week per employee. E.g., if each employee is supposed to work 40 hours per week and is supposed to work 4 to 5 shifts per week, then the average shift length should be between $w_{min} = 8$ and $w_{max} = 10$ hours to fulfil these requirements.

The difficulty of the problem is related to the conflicting objectives: Optimizing the demand as well as possible, especially if it fluctuates frequently, is easier with many different short shifts. However, the minimization of different shifts often reduces the possibilities to choose many short shifts to cover the demand. Further, the required average shift lengths will often prevent the usage of too many short shifts and might force the use of longer shifts even if the demand is more difficult to cover with those. This is the reason why T_4 is very important in practice - a good cover with many short shifts is hard to assign to full-time employees in the following assignment step, e.g., using RWS.

$$T = \sum_{i=1}^{v} \alpha_i \cdot T_i \tag{3.1}$$

The result is a multi-criteria optimization problem where the individual objectives are combined according to given weights α_i. Equation (3.1) provides the combined objective T, where $v = 3$ is used in this chapter for the default version from literature, and $v = 4$ is used in Chapter 5 for most practical utility.

A solution is given as a set of feasible shifts and their corresponding numbers of assigned employees for each day. For TOY, assuming $\alpha_1 = \alpha_2 = 1$ and $\alpha_3 = 180$, an optimal solution with no understaffing or overstaffing and three used shift classes is easily found and given in Figure 3.1. Each line in the table represents a shift class. Remember that usually instances consider multiple days.



Shift start	Shift end	Employees
6:00	18:00	3
12:00	24:00	2
21:00	9:00	1

Figure 3.1: Optimal solution for TOY

Variable	Meaning
n	Number of timeslots
$\mathbf{N}$	$\mathbf{N} = \{1, \ldots, n\}$, set of timeslot indices
$[a_i, a_{i+1})$	Timeslots ($i \in \mathbf{N}$)
sl	Length of a timeslot in minutes
D	Number of days in the planning period
$\mathbf{D}$	$\mathbf{D} = \{1, \ldots, D\}$, set of day indices
S	Number of shift types
$\mathbf{S}$	$\mathbf{S} = \{1, \ldots, S\}$, set of shift type indices
t_s	Shift types ($s \in \mathbf{S}$)
$smin_s$, $smax_s$	Minimum and maximum start of shift type s in timeslots of the current day
$lmin_s$, $lmax_s$	Minimum and maximum length of shift type s in timeslots
R	Demand array of length n
T_i	Objectives ($i \in \{1, 2, 3, 4\}$)
α_i	Weights for the objectives ($i \in \{1, 2, 3, 4\}$)
T	Total weighted objective
w_{min}, w_{max}	Window for the average shift length (T_4) in timeslots
$\mathbf{I}$	$\mathbf{I} = \{1, \ldots, int\}$, set of timeslot indices assuming int timeslots per day
$\mathbf{St}$	Possible range of shift starting time slots
$\mathbf{L}$	Possible range of shift lengths
$Check_{t,\ell}$	Checks whether a shift of length ℓ can start at timeslot t of the day
E	Matrix of dimension $\lvert\mathbf{St}\rvert \times \lvert\mathbf{L}\rvert \times D$ denoting numbers of shifts
C	Matrix of dimension $n \times \lvert\mathbf{L}\rvert$ counting shifts with particular remaining length
$\mathbf{V}$	Set of vertices for the network model, $\mathbf{V} = \mathbf{N}$
$\mathbf{A}$	Set of arcs (i, j) for the network model
B_i	Balance of a node in the network (flow consumed or generated) for $i \in \mathbf{N}$
$F_{(i,j)}$	Flow along arc (i, j) in the network model

Table 3.2: Overview of important definitions for MSD

Again, Table 3.2 provides an overview of the important definitions from both the problem description presented above as well as the different models that are presented in the remainder of this chapter for a quick reference. The definitions regarding individual models will be described in more detail in their corresponding sections.

3.3 Constraint Models

In this section we describe three different ways to provide a constraint model for the problem. This includes a direct model `DIRECT`, a model using a counting representation `COUNT` and a model based on a network flow constraint `NETWORK`. For all models, we define the number of intervals per day $int = \frac{24 \cdot 60}{sl}$, and the sets $\mathbf{I} = \{1, \ldots, int\}$, $\mathbf{D} = \{1, \ldots, D\}$, $\mathbf{S} = \{1, \ldots, S\}$ and $\mathbf{N} = \{1, \ldots, n\}$.

3.3.1 Direct Model

The direct model `DIRECT` represents a solution directly by using decision variables that model the number of employees assigned to each possible shift class on each day. More precisely, for each combination of a shift start time and shift length of any shift type, the number of employees is represented for each day. This can be done by a 3-dimensional matrix E of size $int \times \max_{s \in \mathbf{S}}(lmax_s) \times D$. Here, the first dimension represents all possible shift start times in the day, the second all possible lengths and the third all days. All elements that are not allowed by any shift type can be set to 0 immediately. To reduce the size of the matrix, the following sets are defined:

$$\mathbf{St} = \{\min_{s \in \mathbf{S}}(smin_s), \ldots, \max_{s \in \mathbf{S}}(smax_s)\} \tag{3.2}$$

$$\mathbf{L} = \{\min_{s \in \mathbf{S}}(lmin_s), \ldots, \max_{s \in \mathbf{S}}(lmax_s)\} \tag{3.3}$$

Equation (3.2) defines the set of indices in use for the first dimension of E from the earliest minimum start time to the latest maximum start time, (3.3) defines the set of indices for the second dimension from the smallest minimum length to the largest maximum length. Next, we deal with infeasible shifts as follows:

$$Check_{t,\ell} = (\exists s \in \mathbf{S} : t \geq smin_s \wedge t \leq smax_s \wedge \ell \geq lmin_s \wedge \ell \leq lmax_s) \tag{3.4}$$

$$\neg Check_{t,\ell} \rightarrow (\forall d \in \mathbf{D} : E_{t,\ell,d} = 0) \qquad \forall t \in \mathbf{St}, \ell \in \mathbf{L} \tag{3.5}$$

For convenience, Equation (3.4) defines a predicate that is true if the shift starting at t with length ℓ is a valid shift for any type s. In Equation (3.5), for any combination of a start time t and a length ℓ, if there is no shift type allowing this combination, then for all days the number of employees is set to 0. Note that this only depends on input parameters and is already used during compilation of the model.

	i							
ℓ	1	2	3	4	5	6	7	8
2	0	1	0	0	3	0	2	0
3	1	0	0	3	0	2	0	0
4	0	0	3	0	2	0	0	1

Table 3.3: Counting representation for TOY (transposed)

In order to track the number of employees working at any specific timeslot, we introduce the workforce array W of length n:

$$W_{(d-1)\cdot int+i} = \sum_{\substack{t\in\mathbf{St}\\ t<i}} \sum_{\substack{\ell\in\mathbf{L}\\ t+\ell\geq i}} E_{t,\ell,d} + \sum_{\substack{t\in\mathbf{St}\\ t-int<i}} \sum_{\substack{\ell\in\mathbf{L}\\ t+\ell-int\geq i}} E_{t,\ell,d-1} \qquad \forall d \in \mathbf{D}, i \in \mathbf{I} \quad (3.6)$$

Equation (3.6) sums up the shifts active at timeslot i on day d. The first sum spans all shifts that start before i and end not earlier than i. The second sum deals with shifts that are active after midnight. Those are still assigned to day $d-1$, but contribute to the timeslot on day d if start and end, corrected by int, match the requirements.

Now the optimization goals can be defined in terms of the model:

$$T_1 = sl \cdot \sum_{i\in\mathbf{N}} \max\{W_i - R_i;\ 0\} \quad (3.7)$$

$$T_2 = sl \cdot \sum_{i\in\mathbf{N}} \max\{R_i - W_i;\ 0\} \quad (3.8)$$

$$T_3 = \sum_{t\in\mathbf{St}} \sum_{\ell\in\mathbf{L}} (\exists d \in \mathbf{D} : E_{t,\ell,d} > 0) \quad (3.9)$$

Equations (3.7) and (3.8) hold the sums corresponding to workforce over respectively under the demand. The factor sl is used to count overstaffing and understaffing in minutes. Equation (3.9) counts all shift classes that have an employee assigned on at least one day.

3.3.2 Counting Model

The counting model COUNT uses a similar idea to residual lengths [AGM+16] to keep track of the remaining lengths of shifts for each timeslot instead of just storing which shifts are in use. For this purpose, the decision variables in E are replaced with a 2-dimensional counting matrix C of size $n \times |\mathbf{L}|$. Now, each element $C_{i,\ell}$, with $i \in \mathbf{N}$ and $\ell \in \mathbf{L}$, states that in timeslot i there are $C_{i,\ell}$ active shifts of remaining length ℓ. Note that index i is considered cyclic. For illustration, Table 3.3 shows the matrix C for the optimal solution for TOY.

We can now define the counting behaviour as follows.

$$Prev_{i,\ell} = \begin{cases} 0 & \text{if } \ell = \max_{s \in \mathbf{S}}(lmax_s) \\ C_{i-1,\ell+1} & \text{otherwise} \end{cases} \tag{3.10}$$

$$\begin{cases} C_{i,\ell} \geq Prev_{i,\ell} & \text{if } Check_{(i-1) \bmod int,\ell} \vee Check_{(i-1) \bmod int+int,\ell} \\ C_{i,\ell} = Prev_{i,\ell} & \text{otherwise} \end{cases} \quad \forall i \in \mathbf{N}, \ell \in \mathbf{L} \tag{3.11}$$

Equation (3.10) defines the previous element in the matrix along the secondary diagonal. Shifts counted by $C_{i,\ell}$ either had a remaining length of $\ell+1$ in timeslot $i-1$ or are newly introduced in timeslot i. This is captured in Equation (3.11), where this inequality is expressed whenever it is possible to start a new shift at $C_{i,\ell}$. Note that the existence of shifts is checked both for the current day and shifts overlapping from the previous day. The second line deals with infeasible shifts, as in this case values are propagated without change along the diagonal, corresponding to Equation (3.5) in the direct model.

Now we have an easier time defining the workforce W.

$$W_i = \sum_{\ell \in \mathbf{L}} C_{i,\ell} + \sum_{\ell \in \{1,\ldots,\min_{s \in \mathbf{S}}(lmin_s)-1\}} C_{i-\ell,\min_{s \in \mathbf{S}}(lmin_s)} \quad \forall i \in \mathbf{N} \tag{3.12}$$

In Equation (3.12) we first sum up the row of C. Note that we could extend the lengths in C to start from index 1, however, below $\min_{s \in \mathbf{S}}(lmin_s)$ we would only have direct propagation along the diagonal as no new shifts can start in this region. Omitting this region makes C smaller and we can simply continue the sum along the first column of C to get shifts with remaining length $< \min_{s \in \mathbf{S}}(lmin_s)$.

Equations (3.7) and (3.8) still define T_1 and T_2, only T_3 needs to be redefined.

$$T_3 = \sum_{i \in \mathbf{I}} \sum_{\ell \in \mathbf{L}} (\exists d \in \mathbf{D} : C_{(d-1) \cdot int+i,\ell} > Prev_{(d-1) \cdot int+i,\ell}) \tag{3.13}$$

Equation (3.13) now uses the fact that new shifts introduce a difference along the diagonal, while overall still counting all shift classes that have an employee assignment on at least one day.

3.3.3 Network Flow Model

There is a known relation between the MSD problem and the min-cost max-flow problem [DGGK+07]. In fact, without taking the minimization of shifts into account, the problem can be formulated using a flow network. However, we want to combine utilization of network flow with the minimization of shifts by formulating parts of the problem using a global constraint for network flow, while additional constraints are enforced on the flow variables to represent the whole problem in our model `NETWORK`. In the following the definition of the network graph is presented, first for exact cover and then extending it for over- and understaffing. The whole network will later be illustrated using `TOY` in Figure 3.3.

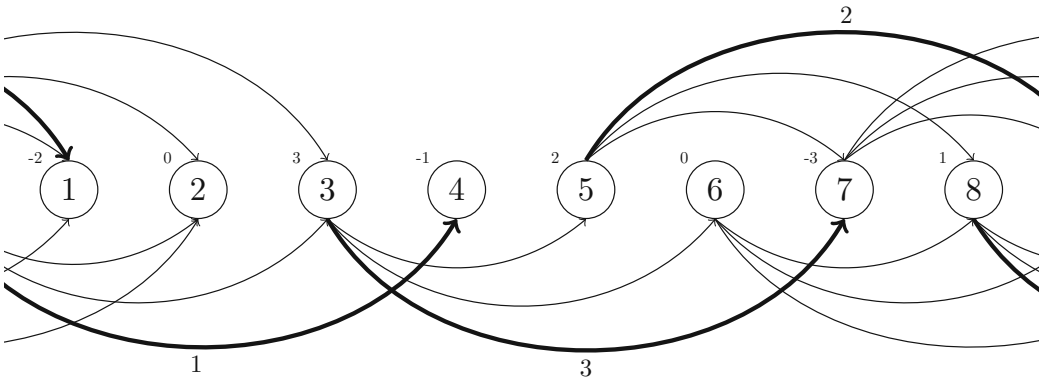


Figure 3.2: Network flow representation for TOY for exact cover

- $\mathbf{V} = \mathbf{N}$: Vertices correspond to timeslots.
- $\mathbf{A} = \{(i,j) \mid i,j \in \mathbf{N},\ \mathit{Check}_{(i-1) \bmod \mathit{int}, j-i} \vee \mathit{Check}_{(i-1) \bmod \mathit{int}+\mathit{int}, j-i}\}$: Arcs correspond to feasible shifts as defined by *Check*.
- $B_i = R_i - R_{i-1}$: The balance, i.e., the flow that is consumed or produced per node corresponds to the difference in demand from a timeslot to the previous one.
- $F_{(i,j)}$: The decision variables in this model represent the flow along the arcs, corresponding to the number of employees working the represented shift.

Figure 3.2 shows this network for TOY. Since the instance has a slotlength of 3 hours, node 1 corresponds to 0:00, node 2 to 3:00, node 3 to 6:00, etc. At 3:00, there is no change in demand, while at 6:00 the demand raises by $B_3 = 3$, which is indicated by the small number above the node. No shifts can start at 0:00 or 3:00, therefore no arcs leave these nodes. At 6:00 shifts can start with 6, 9, or 12 hours of lengths, represented by the arcs to nodes 5, 6, and 7. The bold arcs and the numbers next to them represent the optimal solution, e.g., the arc from node 3 to node 7 corresponds to the shift from 6:00 to 18:00 with 3 assigned employees represented by the flow $F_{(3,7)} = 3$.

Note that this network does not constitute a usual flow network as it is cyclic. Due to cyclicity of the whole problem it rather forms a ring-like structure. Also it does not have a single source and target node. However, the network flow constraint in MiniZinc can deal with the given network as it essentially models the flow conservation constraints taking into account the balance.

Note that so far we only model the demand changes, not the total demand. This has to be done using additional constraints.

$$\mathit{Span}_{i,j,k} = j \leq i < k \vee (j > k \wedge (i \geq j \vee i < k)) \tag{3.14}$$

$$R_i = \sum_{\substack{(j,k) \in \mathbf{A} \\ \mathit{Span}_{i,j,k}}} F_{(j,k)} \qquad \forall i \in \mathbf{N} \tag{3.15}$$

Equation (3.14) is a predicate defined to check whether arc (j, k) spans across timeslot i. This usually indicated by $j \leq i < k$, but due to cyclicity the second case needs to be considered as well. Equation (3.15) sums up the flows of all arcs that span a given timeslot i, which has to match the demand for each timeslot.

$$T_3 = \sum_{\substack{(i,j)\in\mathbf{A} \\ Check_{i-1,j-i}}} (\exists d \in \mathbf{D} : F_{((d-1)\cdot int+i,(d-1)\cdot int+j)} > 0) \tag{3.16}$$

Equation (3.16) defines the number of shifts in the usual way, counting all shifts that have at least one day with positive flow. This time the sum can be represented going through all arcs assigned to the first day in the planning period (via the *Check* condition) and then adding the required number of days to both start and end node.

However, so far the model only represents exact cover without the possibility for overstaffing or understaffing. To include these in the network itself, we introduce new arcs and switch to a weighted network:

- $\mathbf{A}_{OC} = \{(i, i+1) \mid i \in \mathbf{N}\}$
- $\mathbf{A}_{UC} = \{(i, i+1) \mid i \in \mathbf{N}\}$
- $\mathbf{A}' = \mathbf{A} \cup \mathbf{A}_{OC} \cup \mathbf{A}_{UC}$

New arcs are defined from each node to its immediate successor, both for undercover and overcover. The total set of arcs $\mathbf{A}'$ is the union of all arc sets.

$$weight_a = \begin{cases} 0 & \text{if } a \in \mathbf{A} \\ -\alpha_1 & \text{if } a \in \mathbf{A}_{OC} \\ \alpha_2 & \text{if } a \in \mathbf{A}_{UC} \end{cases} \tag{3.17}$$

Equation (3.17) defines the weights for the arcs. They are 0 for all shift arcs. Overcover arcs are defined to hold a certain overflow capacity that should stay on the overcover arcs, therefore, we use negative weights. If overcover is needed on shift arcs, the corresponding capacity is removed from the negatively weighted overcover arcs, resulting in higher cost. As for any demand peak $\max(R)$ is an upper bound for the number of assigned employees and for a slot of maximum overcover shifts from both sides (earlier demand peaks and later demand peaks) might overlap, an upper bound of $2 \cdot \max(R)$ provides enough overcover capacity for the worst case. Undercover arcs are used if demand cannot be fulfilled using shift arcs, as the additional demand uses the undercover arcs and increases the cost.

Figure 3.3 shows the network flow representation for `TOY` including overcover and undercover arcs. The main flow corresponding to the optimal solution is highlighted using bold arcs. As the cover is exact, all undercover arcs with weight α_2 have flow 0, all overcover arcs with weight $-\alpha_1$ have flow $2 \cdot \max(R) = 10$.

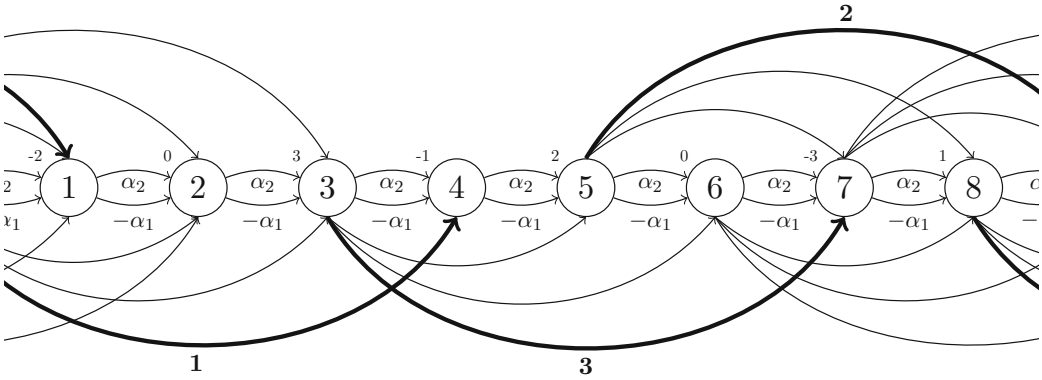


Figure 3.3: Full network flow representation for TOY

The flows for the new arcs, denoted by F^{OC} and F^{UC} and the overcover capacity also need to be included in the demand sum.

$$R_i + 2 \cdot \max(R) = \sum_{\substack{(j,k) \in \mathbf{A} \\ Span_{i,j,k}}} F_{(j,k)} + F^{OC}_{(i,i+1)} + F^{UC}_{(i,i+1)} \qquad \forall i \in \mathbf{N} \quad (3.18)$$

Equation (3.18) sums up all relevant arc flows for each timeslot i.

The global constraint used for weighted network flow includes summing up the total network cost, represented by *cost*. Finally, the combined objective T needs to be defined.

$$T = (cost + 2 \cdot \max(R) \cdot \alpha_1 \cdot n) \cdot sl + \alpha_3 \cdot T_3 \quad (3.19)$$

Equation (3.19) corrects the network cost by the corresponding value for the negatively weighted overcover capacity and takes the slotlength into account.

Note that there is another way to describe overcover arcs by changing their direction and using positive weights instead. This allows to get rid of overcover capacity, instead routing overcover backwards through the network, but introduces a lot of small cycles in the graph.

3.4 Evaluation

This section presents the evaluation of the different models on the standard MSD benchmark set. The instances are organized in four datasets. The first three use a set of four shift types with different start times across the day and a duration of 7 to 9 hours. The slotlength is either 15, 30 or 60 minutes, the length of the planning period is 7 days. The first two datasets are designed in a way that a solution with exact cover exists. However, dataset 3 is constructed to make the existence of such exact covers very unlikely. These three datasets contain 30 instances each. Note that dataset 1 actually

contains 200 instances, but in related work only the first 30 of them are usually evaluated. Since datasets 3 and 4 are more interesting, we also keep our evaluation of dataset 1 to the first 30 instances. Dataset 4 contains a real-life instance using three shift types and two modified instances from dataset 3. All experiments were executed on an Intel Core i7-7500 CPU with 2.7 GHz and 16 GB RAM.

The models were implemented using the solver-independent modelling language MiniZinc 2.2.2 [NSB+07]. It allows to directly specify the constraint models including a range of global constraints like network flow and compiles them into a format called FlatZinc, which is understood by a wide range of solvers. This allows to compare the performance of different solvers. Regarding CP, the lazy clause generation solver Chuffed [CSS+18] was used in version 0.10.3. For MIP, we compare Gurobi [GO18] version 8.0.1 and CPLEX [CPL09] version 12.8.

3.4.1 Search Efficiency with Chuffed

To increase the efficiency of the search, both MiniZinc and Chuffed offer settings to guide the search in the right direction. MiniZinc uses search annotations that allow to specify custom variable and value selection strategies.

In all models the goal is to minimize T. However, while this goal will directly be used with the MIP solvers in the comparison, Chuffed struggles within a large search space filled with a lot of feasible solutions that are far off the optimum. E.g., an empty schedule is always a feasible solution with zero shifts and no overstaffing, but a huge amount of understaffing.

Typically, good solutions will be found when the actual workforce is somewhere around the demand. Therefore, we define an array WD for the workforce difference.

$$WD_i = |R_i - W_i| \qquad \forall i \in \mathbf{N} \quad (3.20)$$

Equation (3.20) defines the distance as the absolute difference between workforce and demand. For Chuffed, we can now use free search, alternating between the internal search and the strategy to first assign the smallest possible values to the workforce distance. As preliminary evaluations of our models showed, the variable selection strategy `first_fail`, assigning the variable with the smallest domain first, together with the value selection strategy `indomain_min`, assigning the smallest value in the domain first, used on WD, provided the best results.

In a similar way, this is also done for the network flow model. Here, no new variables need to be defined, instead the flow variables are used. First, the smallest possible flow values are assigned to the undercover arcs. Then, the largest possible flow values are assigned to the overcover arcs and finally also to the regular flow arcs. All search strategies use the variable selection strategy `first_fail`, the value selection strategy is either `indomain_min` or `indomain_max`. The reason for using the maximum for regular flow arcs is to go towards the minimization of the number of shifts by rather assigning

more employees to the same shift than splitting assignments across many shifts. The annotations for the individual flows are combined using the sequential search annotation.

3.4.2 Results

We executed all three models with all three solvers. As NETWORK performed best for all solvers, we report results for NETWORK on all datasets. The comparison to the other models is presented on selected datasets.

Dataset 1

The first dataset contains instances which admit solutions with exact cover of the demand. Table 3.4 provides the results for this set of instances. The summary shows solved (S), optimal (O) and proven optimal (P) solutions. We compare all three models for Chuffed to highlight the differences in performance for this solver, further the results for the MIP solvers using NETWORK.

For Chuffed we can see a clear ranking of the models in terms of performance. For DIRECT, 11 optimal solutions can be found and proven within the timeout of 3600 seconds, while for 17 instances no feasible solution is found at all. The results for COUNT are already much better, giving 18 proven optimal solutions within the timeout, while only 7 instances do not reach any feasible solution within the runtime. However, best results are achieved with NETWORK, finding feasible solutions for all instances (even though the solution for instance 23 is far off), and for 23 instances the optimum can be proven within the timeout. This relation between the different models using Chuffed also holds for the following datasets, where actually apart from NETWORK hardly any feasible solutions are found within the timeout.

Using either Gurobi or CPLEX, all instances from dataset 1 can be solved within less than 14 seconds per instance. While optimal solutions for these instances were already known, the best approaches so far still needed up to 56 seconds for some instances [Koc15], resulting in a major runtime improvement using NETWORK. Note that all three models reach optimal solutions for all instances in this dataset, however, the other models need more runtime for several instances. We will present this comparison in detail for dataset 3, as these instances are more challenging and therefore show more differences between the models.

A major factor contributing to runtime needs to be discussed here as well. In fact, Gurobi never spends more than 4 seconds on any instance of dataset 1. However, the conversion from MiniZinc to FlatZinc also takes some time, in case of the network model combined with Gurobi actually more than than the solver itself. The scaling factor that decides the needed compilation time is clearly the time granularity, as it also determines the number of possible shifts when shift types are otherwise unmodified. Table 3.5 shows average compilation times for different models, independent of the solver. Note that these values show very small deviations across all datasets.

	Chuffed						Gurobi		CPLEX	
	DIRECT		COUNT		NETWORK		NETWORK		NETWORK	
Inst.	Res.	Time	Res.	Time	Res.	Time	Res.	Time	Res.	Time
1	480	13.80	480	6.44	480	1.83	480	0.60	480	0.50
2	-	3600	300	3600	300	102.26	300	2.56	300	2.82
3	600	295.13	600	1732.00	600	2.02	600	0.60	600	0.50
4	-	3600	-	3600	570	3600	450	2.62	450	4.12
5	480	28.21	480	8.48	480	1.05	480	0.60	480	0.50
6	420	3.47	420	3.94	420	1.17	420	0.77	420	0.53
7	-	3600	270	624.17	270	21.90	270	2.32	270	1.93
8	-	3600	-	3600	150	1462.39	150	11.60	150	9.28
9	-	3600	150	3600	150	1262.67	150	10.57	150	9.17
10	-	3600	330	3600	330	42.60	330	2.37	330	2.07
11	30	2.89	30	4.09	30	10.27	30	10.83	30	8.78
12	-	3600	90	45.72	90	46.20	90	10.32	90	8.78
13	-	3600	105	374.69	105	64.36	105	10.32	105	9.05
14	-	3600	-	3600	285	3600	195	13.68	195	12.64
15	180	0.44	180	0.77	180	0.78	180	0.57	180	0.48
16	-	3600	-	3600	225	3600	225	12.88	225	12.28
17	-	3600	-	3600	630	3600	540	3.94	540	3.45
18	720	3600	720	916.80	720	7.58	720	0.83	720	0.54
19	-	3600	180	3600	180	3600	180	12.44	180	9.44
20	540	27.98	540	17.20	540	1.26	540	0.72	540	0.52
21	-	3600	120	2102.38	120	118.82	120	11.61	120	9.05
22	75	1372.65	75	18.95	75	21.70	75	10.99	75	9.03
23	-	3600	-	3600	655335	3600	150	10.48	150	9.22
24	480	18.06	480	22.36	480	1.19	480	0.81	480	0.65
25	-	3600	-	3600	720	3600	480	5.17	480	13.03
26	-	3600	600	599.08	600	44.47	600	1.16	600	0.92
27	480	68.06	480	40.58	480	1.15	480	0.68	480	0.52
28	5550	3600	270	996.53	270	17.83	270	2.38	270	1.90
29	-	3600	360	3600	360	396.86	360	2.21	360	1.85
30	75	927.67	75	16.27	75	14.81	75	10.31	75	9.00
S/O/P	13/12/11		23/23/18		30/25/23		30/30/30		30/30/30	

Table 3.4: Results for dataset 1

Model	$sl = 60$	$sl = 30$	$sl = 15$
DIRECT	0.2	0.5	2.4
COUNT	0.6	1.4	3.6
NETWORK	0.5	1.8	10

Table 3.5: Approximate compilation times in seconds

We can see that DIRECT compiles fastest, while NETWORK needs the most time. This actually allows some faster results for easy instances, where compilation constitutes most of the runtime, using DIRECT or COUNT. The compilation times show a quadratic correlation to the inverse of the slotlength *sl*.

We also evaluated the results for dataset 1 applying α_1 and α_2 per timeslot instead of per minute to compare to Akkermans et al. [APU18], where we can reach all optimal solutions using an average of only 17.2 instead of 108 seconds per instance. They do not present comparable results for other datasets.

Dataset 2

The second dataset consists of further instances with exact cover solutions. However, it requires increasing numbers of shifts for later instances. This makes the dataset harder than set 1, otherwise the results are quite similar. The results for NETWORK are presented in Table 3.6.

Regarding Chuffed, 5 instances do not reach a feasible solution, rather towards the larger instances. 16 optimal solution can be proven. Gurobi and CPLEX both find the optimal solutions for all instances. Note that previous solution methods were not able to find the optimum for instance 27.

Chuffed only finds one feasible solution using DIRECT and 11 feasible (5 proven optimal) solutions using COUNT. Gurobi can still find and prove all optimal solutions using DIRECT, however, using significantly more runtime on several instances, and all but two with COUNT. CPLEX finds all but one with DIRECT and all but two with COUNT.

Datasets 3 and 4

Next the results for dataset 3 are presented. As these instances are constructed such that it is very unlikely an exact cover exists, this dataset is the most challenging for solvers. The results, including the detailed comparison of different models using Gurobi, are presented in Table 3.7.

The results show that Chuffed can find 24 feasible solutions for this dataset, however, no optimal solutions are found and the gap to the optimal solutions is rather large, never less than 25%. Using the other models, Chuffed does not find feasible solutions before the timeout.

Regarding Gurobi, again all instances can be solved to optimality using NETWORK. While previous work was not able to solve all instances using a timeout of 7200 seconds, the new model can provide a solution within 1 minute for all instances except instance 23, which can still be solved in less than 8 minutes. This constitutes a major speedup compared to previous work and, for the first time, allows to optimally solve all benchmark instances, even within short runtime.

In comparison to the other models, DIRECT is able so prove all optimal results but one, however, for several instances using significantly longer runtime. While for some

	Chuffed		Gurobi		CPLEX	
Instance	Result	Time	Result	Time	Result	Time
1	720	34.44	720	0.67	720	0.76
2	720	11.25	720	0.92	720	0.55
3	360	323.19	360	2.46	360	2.56
4	360	778.56	360	2.33	360	1.90
5	720	30.42	720	1.08	720	1.03
6	360	460.39	360	2.90	360	1.86
7	720	4.19	720	1.83	720	0.68
8	315	3600	180	13.66	180	13.38
9	360	174.22	360	2.49	360	1.98
10	660	5.98	660	1.29	660	1.67
11	840	3600	480	7.17	480	20.87
12	900	3109.58	900	0.99	900	1.10
13	900	572.12	900	2.03	900	1.90
14	840	45.50	840	1.09	840	0.85
15	480	3600	480	8.52	480	17.06
16	240	3600	240	10.83	240	9.01
17	960	2889.60	960	0.90	960	0.86
18	840	178.92	840	1.99	840	0.94
19	-	3600	240	10.96	240	9.50
20	960	423.70	960	1.15	960	0.91
21	690	3600	600	2.78	600	2.62
22	1080	3600	1080	1.47	1080	2.13
23	-	3600	300	34.65	300	31.72
24	780	3600	600	3.97	600	6.90
25	-	3600	600	3.96	600	7.31
26	1020	1337.78	1020	1.76	1020	2.69
27	-	3600	300	25.87	300	323.70
28	-	3600	300	11.76	300	17.52
29	1140	3600	1140	1.78	1140	2.26
30	1140	3600	1020	2.08	1020	2.60
S/O/P	25/20/16		30/30/30		30/30/30	

Table 3.6: Results for dataset 2 (NETWORK)

	Chuffed		Gurobi						CPLEX	
	NETWORK		DIRECT		COUNT		NETWORK		NETWORK	
Inst.	Res.	Time	Res.	Time	Res.	Time	Res.	Time	Res.	Time
1	-	3600	2385	5.02	2385	5.34	2385	10.80	2385	9.07
2	10500	3600	7590	23.52	7590	149.61	7590	3.66	7590	11.43
3	20700	3600	9540	4.46	9540	5.54	9540	3.28	9540	2.73
4	10410	3600	6540	101.84	6540	32.80	6540	5.87	6540	66.58
5	12240	3600	9720	2.31	9720	3.68	9720	0.87	9720	7.40
6	2940	3600	2070	5.74	2070	4.91	2070	10.41	2070	8.99
7	434595	3600	6075	6.25	6075	16.96	6075	10.67	6075	9.34
8	11820	3600	8580	2.71	8580	3.73	8580	2.52	8580	2.32
9	147660	3600	6000	16.49	6000	34.65	6000	11.22	6000	12.29
10	4920	3600	2940	4.94	2940	7.79	2940	2.69	2940	7.71
11	19500	3600	5190	74.73	5190	137.62	5190	6.80	5190	28.91
12	-	3600	4110	122.88	4110	841.17	4110	14.44	4110	47.79
13	-	3600	4605	1274.54	4605	3600	4605	35.84	4605	205.46
14	13800	3600	9600	7.42	9600	13.11	9600	1.93	9600	2.09
15	21840	3600	11250	49.97	11250	106.73	11250	7.36	11250	20.35
16	14940	3600	10620	8.40	10620	3.08	10620	2.16	10620	6.06
17	1198110	3600	4680	21.23	4680	204.38	4680	13.78	4680	14.34
18	9210	3600	6540	24.05	6540	57.99	6540	4.27	6540	15.60
19	-	3600	4890	2489.77	4890	3600	4890	39.28	4890	143.03
20	-	3600	8910	38.28	8910	46.49	8910	4.61	8910	18.34
21	-	3600	5910	3600	5910	3600	5910	444.50	5910	2558.94
22	20460	3600	12600	23.05	12600	89.91	12600	4.16	12600	32.28
23	15000	3600	8280	13.54	8280	12.78	8280	1.87	8280	6.02
24	17520	3600	10260	4.28	10260	9.69	10260	1.29	10260	3.20
25	17040	3600	13020	6.72	13020	5.84	13020	1.76	13020	1.22
26	24330	3600	12780	39.23	12780	94.53	12780	4.11	12780	25.31
27	22620	3600	10020	6.06	10020	3.92	10020	1.52	10020	0.93
28	15000	3600	10440	7.36	10440	11.07	10440	1.83	10440	8.30
29	14190	3600	6510	109.94	6510	332.04	6510	54.56	6510	102.83
30	17760	3600	13320	2.63	13320	5.79	13320	1.07	13320	0.80
S/O/P	24/0/0		30/30/29		30/30/27		30/30/30		30/30/30	

Table 3.7: Results for dataset 3

	Chuffed		Gurobi		CPLEX	
Instance	Result	Time	Result	Time	Result	Time
1	42090	3600	18420	0.569	18420	0.768
2	12540	3600	9720	3.375	9720	64.194
3	24660	3600	18780	0.871	18780	4.027
S/O/P	3/0/0		3/3/3		3/3/3	

Table 3.8: Results for dataset 4 (NETWORK)

instances COUNT is faster than DIRECT, in the average it is worse, also running into timeout on three instances. Note that for some instances both DIRECT and COUNT are faster than NETWORK. This is the result of the longer compilation time as already mentioned before. On such easy instances DIRECT and COUNT are better, however, on more difficult instances they cannot keep up with NETWORK.

The results obtained by CPLEX are similar to the previous datasets. Again, CPLEX can prove optimal solutions for all instances, on some instances slightly faster than Gurobi, but on several instances significantly slower. Again, DIRECT and COUNT are mostly slower and cannot prove optimality for a few instances (DIRECT 1, COUNT 2) similar to the comparison using Gurobi.

Dataset 4 consist of one real-life instance and two modified versions of instance 5 from dataset 3. The results are similar to those of dataset 3 and are presented in Table 3.8. The results from Gurobi and CPLEX are again optimal.

Summary

To sum up, our new model based on network flow NETWORK clearly outperforms the other models across different datasets and solvers. The CP solver Chuffed can provide optimal solutions for several instances where exact cover solutions exist, but struggles with instances admitting no such solution. However, compared to [AGM+16], it can improve the results for many instances across all datasets when working only with CP-related methods.

The MIP solvers Gurobi and CPLEX can both provide optimal solutions for all instances. Further, without timeout this is only possible using NETWORK, showing its superiority compared to the other models. The only drawback is longer compilation time, resulting in more runtime for easy instances, however, more difficult instances greatly benefit from the new model. In direct comparison, while most results are close, CPLEX can provide slightly faster results on several instances, but Gurobi can provide significantly faster results on some difficult instances.

Compared to previous work we can, for the first time, provide a model that can solve all benchmark instances to optimality in short computational time. While comparison of

runtimes across different machines always needs to be considered carefully, we can solve all instances but one within less than a minute of runtime, and all instances within less than 8 minutes, while previous approaches timed out after one or two hours on some instances without an optimal solution. Therefore, we can provide a significant speedup with our approach.

3.5 Conclusion

In this chapter we presented three different constraint models for the Minimum Shift Design problem. Those models were implemented in MiniZinc and evaluated on the standard benchmark datasets using three different solvers, the lazy clause generation solver Chuffed and the MIP solvers Gurobi and CPLEX.

The results show that our new model `NETWORK`, using a network flow based formulation, clearly outperforms the other models. Further, while Chuffed can provide optimal solutions for several instances in the first two datasets, all optimal solutions can be found using Gurobi or CPLEX. This conclusion is similar to [MSS18], which also applies global constraints in MiniZinc and Gurobi, however, in a different way to a different problem, using a regular constraint for RWS. Comparing Gurobi and CPLEX, most results are close, but Gurobi provides significantly shorter runtimes on several difficult instances.

This is the first time in literature that all available instances can be solved to optimality within short runtime, using less than 1 minute for all instances but one, which can be solved in less than 8 minutes, highlighting the strength of our approach. Across different instances from all datasets, runtimes can be reduced significantly in comparison to previous work. Therefore, our approach constitutes the new state of the art for Minimum Shift Design and the results contribute to Research Goal 2 (Section 1.1.2) of this thesis.

While the additional objective T_4 has not been addressed in this chapter, it will be included in Chapter 5, where according to Research Goal 1 (Section 1.1.1) we want to include this constraint due to its practical relevance. Future work regarding the models given in this chapter might involve the integration of T_4 or other additional constraints into the network flow model or trying to apply a network flow based model to break scheduling or combined shift and break scheduling problems.

CHAPTER 4

Metaheuristics and Branch and Price for Bus Driver Scheduling

In the previous chapter, the demand was given in the form of required numbers of employees for each time slot in the schedule. However, in some applications the demand might be specified in different ways, e.g., when shifts need to be designed around given stretches of non-preemptive work. In a competitive environment, it is important to have efficient schedules in order to cover the demand with minimal cost. On the other hand, as usual there is a range of legal requirements, collective agreements and company policies that need to be taken into account to create feasible schedules. Further, especially with very complex rules, not every schedule that is feasible will be readily accepted by the employees, purely optimizing cost might result in reduced employee satisfaction and potential conflicts with labour unions.

An area that is especially restricted by various constraints is scheduling for drivers in public transport. As these employees have a great responsibility keeping their passengers safe, legal requirements enforce strict break assignments in order to maintain concentration. In addition to that a spatial component needs to be considered. This makes the goal to create cost-efficient and employee-friendly schedules even more challenging. This chapter deals with optimizing schedules for bus drivers in Austria, using the regulations from the Austrian collective agreement for employees in private omnibus providers serving regional lines.

The contributions of this chapter are as follows. Based on real-life problems from different cities in Austria we generate a publicly available set of benchmark instances that can be reused for further research. We formalize and optimize the rules from the Austrian collective agreement which are more complex than most other regulations presented in literature so far. We present a metaheuristic based on Simulated Annealing (SA) that was able to significantly improve the results in real-life scenarios and evaluate this method

on the newly generated instances. We explain the practical relevance of optimizing not just cost, but also objectives for employee satisfaction. Then, we also include a comparison with work on Brazilian bus driver scheduling, highlighting that our approach can be adapted to driver scheduling problems with completely different objectives and still improve some of the best known results. These contributions have been presented at ICAPS 2020 [KM20b].

In order to get some exact solutions or at least bounds on the quality of the solutions, we further present a Branch and Price (B&P) approach for the problem, where the column generation uses set partitioning as the master problem and the Resource Constrained Shortest Path Problem (RCSPP) as the subproblem. However, due to the complex set of constraints, the RCSPP has to deal with a large number of high-dimensional labels. We propose several novel contributions to improve the generation of new columns for high dimensional pricing problems, including the splitting of the subproblem, a throttling scheme, and the use of k-d trees and bounding boxes in a two stage dominance algorithm.

Branch and Price provides the first provably optimal solutions for small instances and improves previous best-known solutions or proves them optimal for 48 out of 50 instances, resulting in an optimality gap of less than 1% for more than half the instances. It also provides a lower bound for the quality of the metaheuristic approaches which are mostly within 4% of the optimum. These results have been presented at AAAI 2021 [KMVH21].

4.1 Related Work

A survey for the different objectives in operating bus transport systems is provided by Ibarra-Rojas et al. [IRDGM15]. Driver scheduling is located between vehicle scheduling and driver rostering in a six step process. Driver scheduling belongs to the area of crew scheduling problems [EJKS04] that is also frequently applied to airline [GJ05] and train crew scheduling.

Research on bus driver scheduling problems has a long history [WR95] and uses a variety of solution methods. Exact methods mostly use column generation with a set covering or set partitioning master problem and a Resource Constrained Shortest Path Problem as the subproblem [SW88, DS89, PLP09, LH16]. Heuristic methods like greedy [MT86, DLFM11, TK13] or exhaustive [CSSC13] search, tabu search [LPP01, SK01], genetic algorithms [LPP01, LK03], or iterated assignment problems [CdMNdA$^+$17] are used in different variations. This last approach is used on published benchmarks from Brazil that we compare with.

Our first solution method is based on Simulated Annealing [KGV83]. We focus on a combined objective that includes aspects for generating practically usable solutions, while most work so far focuses mainly on cost, only sometimes minimizing idle time and vehicle changes [IRDGM15], [CdMNdA$^+$17]. Further, break constraints are mostly simple, often just one meal break. However, break scheduling within shifts has been considered by authors in different contexts [BGM$^+$08, BGM$^+$10, WM14]. There is not much work on

multi-objective bus driver scheduling [LPP01], but multi-objective approaches are used in other bus operation problems [RMVP13].

Branch and Price is a decomposition technique for large mixed integer programs [BJN+98]. This chapter uses set partitioning [BP76] as the master problem and the RCSPP [ID05] as the subproblem. Resources are modelled via resource extension functions (REF) [Irn08]. Different techniques for dealing with pareto-front calculation can mostly be found as maxima-finding algorithms in a geometrical context [Ben80, CHT12].

4.2 Problem Description

The Bus Driver Scheduling (BDS) Problem deals with the assignment of bus drivers to vehicles that already have a predetermined route for one day of operation. The shifts that are generated need to respect a range of constraints regarding length and complex break assignment rules.

The specification presented here is taken from the Austrian collective agreement for employees in private omnibus providers [WKO19], using the rules for regional lines (up to 50 km per line). In case of ambiguities we use an interpretation from practice.

4.2.1 Problem Input

The bus routes are given as a set $\mathbf{L}$ of individual bus legs, each leg $\ell \in \mathbf{L}$ is associated with a tour $tour_\ell$ (corresponding to a particular vehicle), a start time $start_\ell$, an end time end_ℓ, a starting position $startPos_\ell$, and an end position $endPos_\ell$. The actual driving time for the leg is denoted by $drive_\ell$. The given instances use $drive_\ell = length_\ell = end_\ell - start_\ell$.

ℓ	$tour_\ell$	$start_\ell$	end_ℓ	$startPos_\ell$	$endPos_\ell$
1	1	360	395	0	1
2	1	410	455	1	2
3	1	460	502	2	1
4	1	508	540	1	0

Table 4.1: A Bus Tour Example

Table 4.1 shows a short example of one particular bus tour. The vehicle starts at time 360 (6:00 am) at position 0, does multiple legs between positions 1 and 2 with waiting times inbetween and finally returns to position 0. A valid tour never has overlapping bus legs and consecutive bus legs satisfy $endPos_i = startPos_{i+1}$. A tour change occurs when a driver has an assignment of two consecutive bus legs i and j with $tour_i \neq tour_j$.

A distance matrix specifies, for each pair of positions p and q, the time $d_{p,q}$ a driver takes to get from p to q when not actively driving a bus. If no transfer is possible, then $d_{p,q} = \infty$. $d_{p,q}$ with $p \neq q$ is called the passive ride time. $d_{p,p}$ represents the time it takes to switch tour at the same position, but is not considered passive ride time.

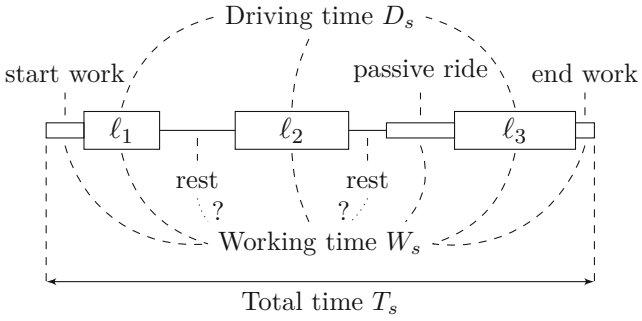


Figure 4.1: Example shift

Finally, each position p is associated with an amount of working time for starting a shift ($startWork_p$) and ending a shift ($endWork_p$) at that position. The instances in this thesis use $startWork_p = 15$ and $endWork_p = 10$ at the depot ($p = 0$). These values are 0 for other positions.

4.2.2 Solution

A solution to the problem is an assignment of exactly one driver to each bus leg. Criteria for feasibility are:

- No overlapping bus legs are assigned to any driver.
- Changing tour or position between consecutive assignments i and j requires $start_j \geq end_i + d_{endPos_i, startPos_j}$.
- Each shift respects all hard constraints regarding work regulations as specified in the next section.

4.2.3 Work and Break Regulations

Valid shifts for drivers are constrained by work regulations and require frequent breaks. First, different measures of time related to a shift s containing the set of bus legs $\mathbf{L}_s$ need to be distinguished, as visualized in Figure 4.1:

- The total amount of driving time: $D_s = \sum_{i \in \mathbf{L}_s} drive_i$.
- The span from the start of work until the end of work T_s with a maximum of $T_{max} = 14$ hours.
- The working time $W_s = T_s - unpaid_s$, which does not include certain unpaid breaks.

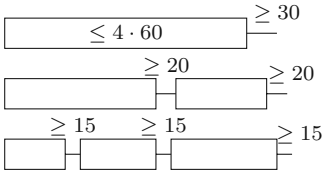


Figure 4.2: Driving break options

Driving Time Regulations

The maximum driving time is restricted to $D_{max} = 9$ hours. The whole distance $start_j - end_i$ between consecutive bus legs i and j qualifies as a driving break, including passive ride time. Breaks from driving need to be taken repeatedly after at most 4 hours of driving time. In case a break is split in several parts, all parts must occur before a driving block exceeds the 4 hour limit. Once the required amount of break time is reached, a new driving block starts. The following options are possible, as shown in Figure 4.2.

- One break of at least 30 minutes
- Two breaks of at least 20 minutes each
- Three breaks of at least 15 minutes each

Working Time Regulations

The working time W_s has a hard maximum of $W_{max} = 10$ hours and a soft minimum of $W_{min} = 6.5$ hours. If the employee is working for a shorter period of time, the difference has to be paid anyway. The actual paid working time is $W'_s = \max\{W_s;\ 390\}$.

A minimum rest break is required according to the following options:

- $W_s < 6$ hours: no rest break
- 6 hours $\leq W_s \leq 9$ hours: at least 30 minutes
- $W_s > 9$ hours: at least 45 minutes

The rest break may be split into one part of at least 30 minutes and one or more parts of at least 15 minutes. The first part has to occur after at most 6 hours of work.

Whether rest breaks are paid or unpaid depends on break positions according to Figure 4.3. Every period of at least 15 minutes of consecutive rest break is unpaid as long as it does not intersect the first 2 or the last 2 hours of the shift (a longer rest break might be partially paid and partially unpaid). The maximum amount of unpaid rest is limited:

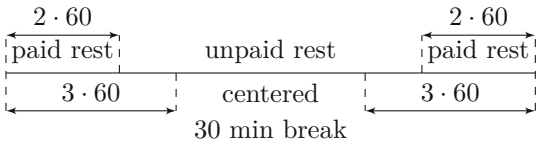


Figure 4.3: Rest break positioning

- If 30 consecutive minutes of rest break are located such that they do not intersect the first 3 hours of the shift or the last 3 hours of the shift, at most 1.5 hours of unpaid rest are allowed;
- Otherwise, at most one hour of unpaid rest is allowed.

Rest breaks beyond this limit are paid.

Split Shifts

If a rest break would be at least 3 hours long, it is instead considered a shift split, which is unpaid and does not count towards W_s. However, such splits are typically regarded badly by the drivers. A shift split counts as a driving break, but does not contribute to rest breaks.

4.2.4 Objectives

There are several optimization criteria, setting a different and often conflicting focus on the resulting schedules. Typical minimization objectives regarding operation cost are:

- Number of employees
- Sum of working times $\sum_s W_s$ or $\sum_s W'_s$
- Sum of extra hours

However, there are additional minimization objectives that need to be considered in order to get schedules that are actually workable in practice:

- Sum of total times $\sum_s T_s$
- Sum of passive ride times
- Number of tour changes
- Number of shift splits

While some of these additional objectives correlate with operational objectives (e.g., reducing the number of tour changes typically requires less change time, contributing to the working time objective), others like the number of shift splits actually reduce options like the possibility to use the same driver at two different peak times. However, schedules without such objectives typically are not acceptable in practice as they completely disregard driver needs.

We use a linear combination of several objectives in this work. Note that there is hardly a universal notion which objectives to consider, much less a perfect set of weights. In practice, depending on the scenario, often many different schedules are evaluated to figure out what works best. In this thesis, we propose the following objective function that stems from practical experience:

$$objective = \sum_s 2 \cdot W'_s + T_s + ride_s + 30 \cdot change_s + 180 \cdot split_s \tag{4.1}$$

In Equation (4.1) we use the sum of working times as the main objective, but also the sum of total times to reduce long unpaid periods for employees. The further objectives aim to reduce passive ride time $ride_s$ and the number of tour changes $change_s$ that are beneficial for employees, but also for efficient schedules. The last objective aims to reduce the number of split shifts $split_s$ as they are very unpopular. The weight of 30 for tour changes corresponds to 10 minutes of paid working time (paid working time contributes to W_s and T_s), the weight of 180 for a shift split corresponds to 1 hour of paid working time.

Table 4.2 summarizes the definitions from this section. Note that both solution methods in this chapter need several additional definitions that differ in some details. These are not part of the table, but rather explained in detail directly in the respective sections.

4.3 Instances

This section describes the kind of instances that are typically seen in real-life applications and highlights their most relevant features for the optimization. The features were drawn from analysing different real-life scenarios in Austria, looking at local bus lines both in urban and rural areas.

While the original instances cannot be shared in public, an instance generator was created to generate instances with the same characteristics that are shared publicly to allow comparison in further work in the BDS benchmark set. There are 50 instances, 5 each for 10 different sizes ranging from 10 tours (about 70 legs) to 100 tours (almost 1000 legs).

4.3.1 Instance Size

Real-life instances come in a variety of sizes. This is due to different scenarios that are considered. Small instances are typically composed of a few lines that have one or more

Variable	Meaning
$\mathbf{L}$	Set of bus legs
$tour_\ell$	Tour of a bus leg (assigned vehicle)
$start_\ell$, end_ℓ	Start and end time of a bus leg
$startPos_\ell$, $endPos_\ell$	Start and end position of a bus leg
$length_\ell$	Length of a bus leg, $length_\ell = end_\ell - start_\ell$
$drive_\ell$	Driving time of a bus leg
$d_{p,q}$	Distance between bus legs changing from position p to position q
$startWork_p$, $endWork_p$	Additional working time starting or ending a shift at position p
D_s	Total driving time of shift s
D_{max}	Maximum total driving time of a shift
T_s	Total span of shift s
T_{max}	Maximum span of a shift
W_s	Total paid working time of shift s
W'_s	Actual paid working time after considering the minimum W_{min}
W_{min}, W_{max}	Minimum and maximum paid working time of a shift
$ride_s$	Total passive ride time of shift s
$change_s$	Number of tour changes of shift s
$split_s$	Number of shift splits of shift s
$objective$	Total objective as a weighted sum of the individual objectives
$diff_{ij}$	Distance between legs i and j, $diff_{ij} = start_j - end_i$

Table 4.2: Overview of important definitions for BDS

common or at least closely located stations. When bus companies have to compete for providing service in an area, they often have to apply with a cost calculation for such line clusters. Typically, the vehicle routes are already predetermined, leaving the personnel calculation to the competitors. Therefore, the small instances reflect such clusters of lines.

On the other hand, some scenarios involve larger bus companies optimizing their whole bus network for a town or city, giving rise to very large instances with more than 1000 legs, making the problem very challenging to solve. While the problem is very large and there are many constraints to take into account, in practice still much work is done manually.

4.3.2 Demand Distribution

The instances encountered in real-life scenarios have a very typical shape highlighted in Figure 4.4. There is a large demand peak in the morning, where both employees and pupils require many buses in short amount of time, while there is a major depression until lunch time. This results in a problematic conflict with the constraints for valid shifts as a minimum work time of 6.5 hours needs to be paid, while after the peak there is not enough demand.

The other parts of the demand distribution are less of a problem for the optimization.

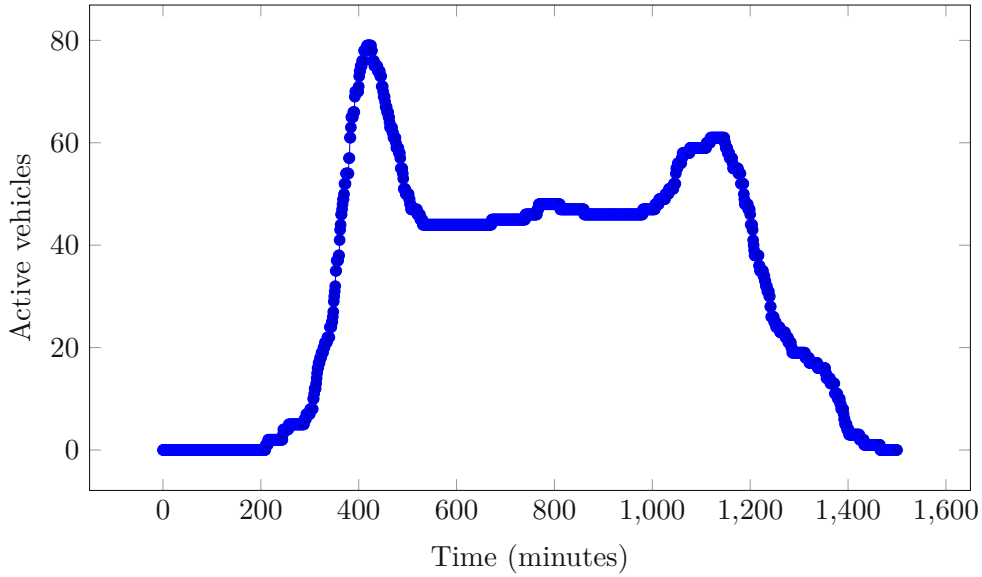


Figure 4.4: Demand distribution for instance 100_50.

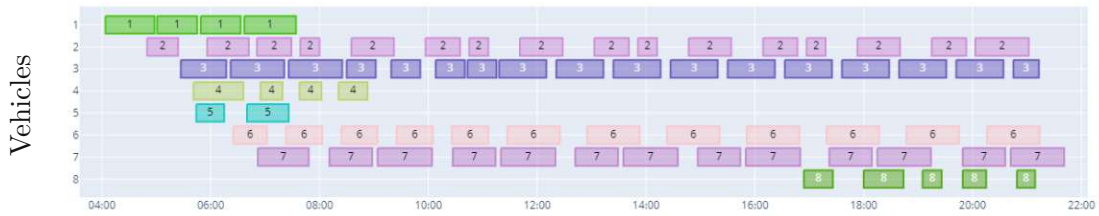


Figure 4.5: Bus tours for instance 10_1

There are often a few shifts that start very early and end very late, this might rarely also include night buses running the whole night. We do not include night buses in the generated instances. At lunch time there is often a slight peak, in particular when schools need to be served, further there is a significant peak in the evening, however, far less pronounced compared to the morning peak.

4.3.3 Waiting Times

Figure 4.5 shows the bus legs for one of the small instances. Depending on the instance, real-life scenarios might include significant waiting times within the tours. This typically occurs when the bus arrives at the final stop of a line and has to wait before starting the next trip. There might be longer waiting times on lines with less demand, especially in rural areas, while some lines have only very short waiting times. E.g., tour 2 in the example repeatedly shows long waiting times, while tour 3 has mostly short waiting times.

The crucial importance of these waiting times is the ability to use them for breaks in the schedule. Lines with waiting times make it very important to consider all break options, as splitting breaks in many parts often allows to place those naturally within the schedule of the line, reducing the need to exchange the driver on a tour.

Therefore, the generated instances exhibit a range of different waiting times that might allow usage as breaks. The evaluation section will present more detail on how that influences optimal schedules for the problem.

4.4 Metaheuristic Solution Methods

This section presents the metaheuristic solution methods used to handle the complex constraints and solve the problem, in particular the representation of the problem-specific constraints, a construction heuristic, and Simulated Annealing (SA) including the proposed moves and their selection.

4.4.1 Constraint Representation

The general idea for the representation of the complex constraints is to parse shift candidates from start to end, maintaining accumulators for several resources. These can then be used to check the constraints and, if violated, add penalties to the objective function.

Accumulators are initially assigned to 0. Assume that i and j are two bus legs assigned consecutively to shift s in the step of calculating the values at leg j. We define $diff_{ij} = start_j - end_i$.

Drive time constraints use the following accumulators:

$$dt_j = dt_i + drive_j \tag{4.2}$$

$$block_j = diff_{ij} \geq 30 \vee (diff_{ij} \geq 20 \wedge b20_i = 1) \vee (diff_{ij} \geq 15 \wedge b15_i = 2) \tag{4.3}$$

$$dc_j = \begin{cases} drive_j & \text{if } block_j \\ dc_i + drive_j & \text{else} \end{cases} \tag{4.4}$$

$$b20_j = \begin{cases} 0 & \text{if } block_j \\ 1 & \text{if } diff_{ij} \geq 20 \\ b20_i & \text{else} \end{cases} \tag{4.5}$$

$$b15_j = \begin{cases} 0 & \text{if } block_j \\ b15_i + 1 & \text{if } diff_{ij} \geq 15 \\ b15_i & \text{else} \end{cases} \tag{4.6}$$

Equation (4.2) sums up total drive time, (4.4) the drive time in the current driving block. The driving block resets depending on (4.3), formalizing the different splitting options

for driving breaks. Equations (4.5) and (4.6) keep track of the number of driving breaks of length at least 20 respectively 15 minutes in the current driving block.

The problem constraints are enforced with $dt_i \leq D_{max}$ and $dc_i \leq 240$.

Next we need to specify work time and rest break constraints:

$$diff'_{ij} = \begin{cases} 0 & \text{if } diff_{ij} - r_{ij} \geq 180 \\ diff_{ij} - r_{ij} & \text{else} \end{cases} \tag{4.7}$$

Equation (4.7) denotes the length of a potential rest break between legs i and j, using r_{ij} for the passive ride time necessary between legs i and j. In case passive ride time is needed, it is always placed after the rest break and before the start of leg j. $diff''_{ij}$ denotes the potentially unpaid break time by reducing $diff'_{ij}$ by parts located within the first 2 or last 2 hours.

$$wt_j = wt_i + diff'_{ij} + r_{ij} + length_j \tag{4.8}$$

$$rest_j = rest_i + \begin{cases} diff'_{ij} & \text{if } diff'_{ij} \geq 15 \\ 0 & \text{else} \end{cases} \tag{4.9}$$

$$first15_j = first15_i \vee (diff'_{ij} \geq 15 \wedge end_i - start'_s \leq 360) \tag{4.10}$$

$$break30_j = break30_i \vee diff'_{ij} \geq 30 \tag{4.11}$$

$$unpaid_j = unpaid_i + diff''_{ij} \tag{4.12}$$

$$center30_j = center30_i \vee \text{isCentered30}(diff_{ij}) \tag{4.13}$$

Equation (4.8) sums up the work time, currently still including unpaid rest breaks. Qualified rest breaks, whether unpaid or not, are summed up in Equation (4.9). Equations (4.10) and (4.11) are required to monitor split rest breaks, ensuring that the first part occurs no later than 6 hours into the shift (using $start'_s$ as shorthand for the start of the shift) and that there is a break of at least 30 minutes.

Equation (4.12) sums up potential unpaid rest time. The maximum amount might depend on breaks later in the schedule, therefore the final calculation is done at the end of the shift. Equation (4.13) keeps track whether there is a centred break of at least 30 minutes (according to the problem definition).

The values from the last bus leg ℓ of shift s are then used as follows:

$$upmax_s = \begin{cases} 90 & \text{if } first15_\ell \wedge break30_\ell \wedge center30_\ell \\ 60 & \text{if } first15_\ell \wedge break30_\ell \wedge \neg center30_\ell \\ 0 & \text{else} \end{cases} \tag{4.14}$$

$$W_s = wt_\ell + startWork'_s + endWork'_s - \min\{unpaid_\ell;\ upmax_s\} \tag{4.15}$$

Equation (4.14) calculates the maximum rest time that can be unpaid based on having a valid rest time at all and the location of the 30 minute part. Equation (4.15) calculates

the total paid working time. Keep in mind that for the objective W_{min} still needs to be taken into account. $startWork'_s$ and $endWork'_s$ are used to denote the extra working time at the start and end of the shift (depending on the starting position of the first leg in the shift and the end position of the last leg in the shift).

$$\begin{cases} W_s < 360 & \text{if } \neg first15_\ell \vee \neg break30_\ell \\ W_s \leq 540 & \text{if } first15_\ell \wedge break30_\ell \wedge rest_i < 45 \\ W_s \leq 600 & \text{else} \end{cases} \tag{4.16}$$

Equation (4.16) bounds the maximum working time based on the rest breaks that are present.

Further we need to check the maximum total time that can be calculated without an accumulator and maintain the following accumulators for the remaining objectives:

$$ride_j = ride_i + r_{ij} \tag{4.17}$$

$$change_j = change_i + \begin{cases} 1 & \text{if } tour_i \neq tour_j \\ 0 & \text{else} \end{cases} \tag{4.18}$$

$$split_j = split_i + \begin{cases} 1 & \text{if } diff_{ij} - r_{ij} \geq 180 \\ 0 & \text{else} \end{cases} \tag{4.19}$$

4.4.2 Construction Heuristic

The first part of the solution mechanism is to construct an initial solution by using a greedy assignment heuristic. While this heuristic is not meant to produce solutions of top quality, it is an important base for the use of Simulated Annealing in the later part of the algorithm.

Algorithm 4.1 shows the high-level view of the algorithm. The first loop in lines 1 to 10 goes through all bus legs and assigns them to the best employee in line 3, identified in line 2 as the employee where the assignment results in the lowest objective after assignment. This might either be an existing employee or a new, previously empty employee. Then as many legs from the same tour as possible without violations (checked in line 5) are assigned to the same employee in lines 4 to 9. These legs are skipped in the outer loop as assigning them removes them from $\mathbf{L}_{unassigned}$. This saves much effort compared to building combinations for the same tour in later stages of the algorithm.

The second loop in lines 11 to 17 performs some reassignments from employees with earlier shifts to those with later shifts. The last leg from an earlier employee e is moved to a later employee f (iterated in line 12) if this does not worsen the solution in line 13. The reason is that the greedy assignment of legs results in late legs often forming very short, expensive shifts. Therefore, reassigning legs from the end of earlier shifts to the start of later shifts results in a more balanced schedule. Reassignments are only done if they improve the overall objective.

Algorithm 4.1: Construction Heuristic

```
1  for leg in L_unassigned do
2  |  e ← bestEmployee(leg);
3  |  assignLegToEmployee(leg, e);
4  |  while leg ← nextTourLeg(leg) do
5  |  |  if ¬evaluateAssign(leg, e) then
6  |  |  |  break;
7  |  |  end
8  |  |  assignLegToEmployee(leg, e);
9  |  end
10 end
11 for e in Employees do
12 |  for f in startLater(Employees, e) do
13 |  |  if ¬reassignLast(e, f) then
14 |  |  |  break;
15 |  |  end
16 |  end
17 end
```

4.4.3 Simulated Annealing

The improvement stage of the algorithm is based on Simulated Annealing (SA), as this is a flexible method that has provided very good results for other scheduling problems.

Algorithm 4.2 shows the structure of the algorithm. The moves used for the improvement phase are different kinds of swaps. They apply to a pair of employees and exchange either one or more bus legs in a selected time window between those employees. The selection where to apply happens in `chooseMove` in line 5. While the base move of reassigning one bus leg could be applied repeatedly, moving multiple legs at once is critical as it avoids intermediate penalties.

As the larger instances have up to hundreds of employees, it is important to focus the application of moves towards areas that have high objective values. Therefore, with 50% probability the first employee for a move is selected among the 10 employees with the highest objective values, otherwise randomly. The second employee is always selected randomly. The split allows potential improvements to occur everywhere, while the focus allows faster convergence by higher chance of eliminating highly penalized duties.

A move is accepted in `acceptMove` in line 7, where *change* is the change in objective, if it does not make the solution worse or otherwise with probability $\exp\left(-\frac{change}{t}\right)$. The termination criterion is *maxCount* iterations without improvement, ensuring the algorithm settles to a stable solution.

Algorithm 4.2: Simulated Annealing Implementation

```
1  t ← t_start;
2  changeCount ← 0;
3  while changeCount < maxCount do
4      for j ← 0 to innerIterations do
5          move ← chooseMove();
6          change ← move.evaluate();
7          if acceptMove(change, t) then
8              move.execute(solution);
9              solution.value ← solution.value + change;
10             if change < 0 then
11                 changeCount ← 0;
12             end
13         else
14             move.abort();
15         end
16     end
17     changeCount ← changeCount + 1;
18     t ← t · coolingRate;
19 end
```

4.5 Metaheuristic Evaluation

This section presents the evaluation of our method. First it presents the setup of the experiments, then it provides the results for the new benchmark instances. The results are compared to using just paid work time as the objective, highlighting the importance of additional objectives. Finally, we provide a comparison with publicly available instances from Brazil, improving some best known solutions.

4.5.1 Experimental Setup

The experiments are run on a PC with an Intel Core i7-7500U at 2.7 GHz. The algorithm receives a maximum number of employees that it is allowed to use and typically settles on a good number due to the working time objective. However, the algorithm can often find slightly better solutions when restricting the employees even further, therefore, several exploration runs decreasing the employee limit are performed per instance.

Based on experiments we set the parameters for the algorithm as follows. Hard constraint violations are penalized by a cost of 1000 per minute of violation. We report the results from the construction heuristic and compare Simulated Annealing (SA) with randomized hill climbing (temperature set to 0). The starting temperature is set to 100, the exploration runs use a cooling rate of 0.9 and 10 as the number of iterations without change. More precise runs use a cooling rate of 0.99 and 100 iterations without

	Construction Heuristic			Simulated Annealing				
Instances	Time	Empl.	Value	Time	Empl.	Best	Mean	Std.dev.
10	0.2	12.2	15747.2	22.8	11.6	**14717.4**	14739.6	37.8
20	0.5	24.2	32627.8	62.2	22.6	**30860.6**	30970.8	147.5
30	1.7	41.0	54141.6	108.8	38.4	**50947.4**	51257.8	318.2
40	3.1	55.0	73417.0	267.0	52.2	**69119.8**	69379.9	489.5
50	6.3	68.2	91372.8	329.0	66.0	87013.2	87557.3	700.8
60	10.6	80.8	109293.8	543.6	78.8	103967.6	104333.1	364.6
70	18.0	92.8	130024.2	751.4	90.4	**122753.6**	123225.6	544.5
80	26.5	107.0	148889.0	1140.2	104.6	140482.4	140913.8	604.2
90	37.6	120.2	165171.6	1453.0	118.0	156385.0	157426.0	1224.0
100	48.0	130.4	183456.4	1449.4	128.2	173524.0	174501.6	1197.4

Table 4.3: Results with construction heuristic and Simulated Annealing

	Hill Climbing				
Instances	Time	Empl.	Best	Mean	Std.dev.
10	7.8	12.0	14904.4	14988.4	90.1
20	28.0	22.8	30931.4	31275.6	389.9
30	99.4	38.8	51544.2	51917.3	352.6
40	151.2	52.0	69533.6	71337.6	541.7
50	295.4	65.8	**86718.6**	87262.5	585.1
60	432.8	79.0	**103780**	104296.2	637.4
70	718.6	90.4	122912.8	123303.8	582.3
80	959.4	104.0	**139765.2**	140508.0	716.2
90	1516.6	117.2	**156239.4**	156862.5	808.2
100	1483.2	127.4	**172327.8**	172909.0	645.4

Table 4.4: Results with randomized hill climbing

change as they trade speed for result quality. The number of inner iterations (moves per temperature level) is set to 1000. We repeat the best configuration three times to get an understanding of the variation introduced by random decisions in the algorithm.

4.5.2 Results

Tables 4.3 and 4.4 show the results for the 50 benchmark instances, using the average over each size category of instances since instances in the same category show very similar results. Runtimes are in seconds. The construction heuristic produces fast results of reasonable quality. In comparison, the best obtained results are typically 5 to 7 percent better at the cost of higher runtime. Greedy strategies work well as taking the best option to finish the current employee's schedule yields a range of very good schedules. However, in the end there are typically several parts that do not fit together, leading to

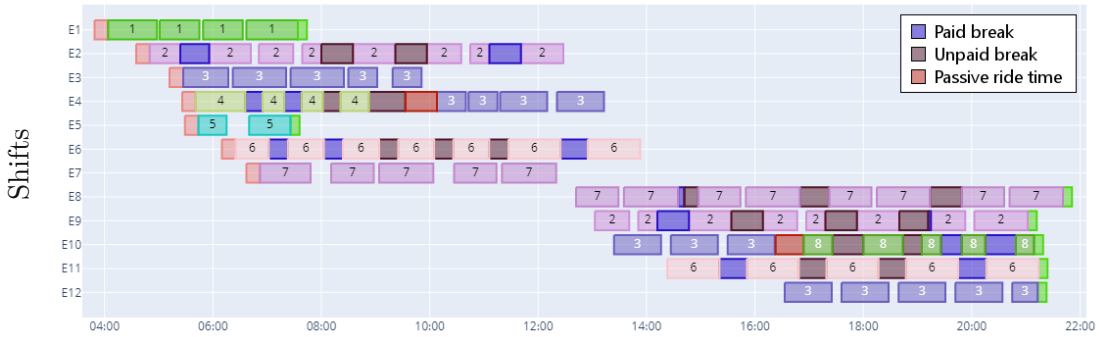


Figure 4.6: Best solution for instance 10_1 using objective (4.1)

the need for more sophisticated improvement methods.

After the construction heuristic, a few fast runs are performed to determine if better results can be obtained with lower employee limits. The results show that the best found solutions use only a few employees less than the original number. Runtime and number of employees for Simulated Annealing and hill climbing are reported from the run producing the best result, the mean result from three runs with equal configuration per instance.

In comparison, Simulated Annealing and hill climbing produce similar results. While differences are not significant, a tendency shows better results for SA on smaller instances and better performance of hill climbing on larger instance. The most likely cause is that SA helps to avoid local optima in smaller instances while in larger instances the search space is so large that the effort of repairing the solution after deteriorating moves is higher than the benefit from escaping local optima. Further, hill climbing is faster on smaller instances, but looses the advantage on larger instances. The standard deviation seems to grow more slowly for hill climbing.

4.5.3 Detailed Solution Analysis

In order to understand how the characteristic features of the instances influence the results, Figure 4.6 shows the best found solution for instance 10_1 using objective (4.1). Note that this instance is chosen to illustrate observations that are found across the whole set of instances.

The high peak in the morning typically leads to several short shifts in the morning, despite the minimum of at least 6.5 hours. Some of them might be combined with later parts of the schedule using shift splits, however, as their use is discouraged, at least some short shifts will usually remain.

The shifts use the waiting time available on the tours to fill their breaks. Therefore, most long tours only need to be handed over once. Note that tour 3 does not have enough breaks, therefore it needs to be handed over more frequently. The algorithm combines

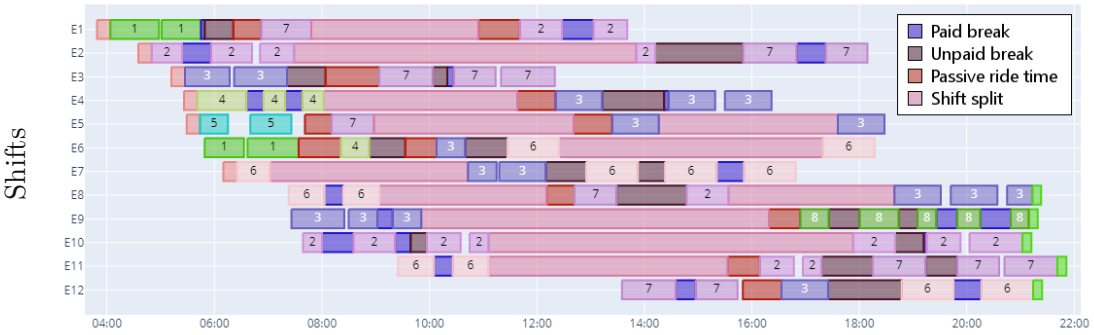


Figure 4.7: Best solution for instance 10_1 using only working time as objective

	Working time	Span	Passive ride	Tour changes	Shift splits
Combined	4840	4611	66	2	0
Working time	4680	8357	480	20	12
Change	−3.3%	+81.2%	+627.3%	+900%	+∞%

Table 4.5: Objective values of the different solutions

it with short tours in the morning and evening in order to utilize employees well, or generates shorter shifts.

When talking about objectives for optimization, the primary motivation for bus companies is to reduce cost. However, we are using the combined objective (4.1) that includes several aspects that create better schedules for employees. While on first sight this narrows the possibilities to reduce the cost as much as possible, in practice reasonable schedules for employees not only benefit them, but can also contribute to a better work atmosphere that might lead to lower personnel fluctuations and fewer sick leaves.

In order to understand how tremendous the differences in generated schedules can be, Figure 4.7 shows the best found solution for instance 10_1 when only using paid working time (W'_s) as the objective.

Both schedules are feasible according to all constraints and represent the best found solutions for their respective objective. However, it would be very hard to convince any operating company to adopt the schedule from Figure 4.7. Here, almost every employee has at least one shift split, as these do not add any cost to the objective and are therefore easily used to combine bus legs at different times of the day.

Table 4.5 compares the results for the objective values of the different solutions. The paid working time could indeed be reduced by 3.3% percent using only the working time objective. However, this is in a strong contrast to an increase in total span by 81.2% and even more extreme increases in the remaining objective values.

	CC130	CC251	CV412	CC512	CC761
GraphBDSP	**8389.4**	17600.0	**29512.5**	35105.0	**47532.9**
SA	8432.0	**17009.7**	30395.2	**33743.0**	48862.8

	CC1000	CC1253	CC1517	CC2010	CC2313
GraphBDSP	**64873.6**	82842.9	99852.8	**128964.2**	**147215.0**
SA	65015.6	**82332.6**	**99439.0**	130410.0	150215.5

Table 4.6: Comparison on Brazil instances

When comparing all 50 instances, the average reduction in work time is very considerable with 7.9%. However, the average increase is 60.9% in total span, 209% in passive ride time, 219% in tour changes and 1060% in shift splits.

This comparison clearly shows why it is very important to use a balanced set of objectives that actually produce solutions that are well suited to be used in practice.

4.5.4 Comparison

We were able to successfully improve schedules for a large-scale real-life application with slightly different constraints using more than 2700 legs by about 4% compared to their previous schedule, which was incrementally optimized by a human expert over a long period of time. As personal cost is typically the highest cost factor, this directly results in savings for the company while maintaining the same or even slightly better level of service and employee satisfaction. We also successfully applied the method to several smaller problems for clusters of lines in a major city.

Comparing with other work in the area is difficult for this problem as most often each work uses a different set of constraints, and applies their method on their own set of instances that are not publicly available. However, for a problem from Brazil [CdMNdA+17], where instances with more than 2300 bus legs are available online, we are able to compare our approach to their work.

However, their constraints are different from the ones we use. There are no driving breaks and only at most one rest break as well as no passive ride times. There is a minimum working time like in our problem, but also overtime cost as soon as the time is above the minimum, leading to a much more pronounced balancing objective in this problem. As we developed our solution method with potential application to similar problems in mind, we only need to exchange the part for the constraint representation and can then directly apply our method. Additionally we added a term to the objective function penalizing a combination of paid time between legs and overtime for speeding up convergence.

Table 4.6 shows the comparison of our method SA with their method GraphBDSP [CdMNdA+17]. We can improve 4 out of 10 benchmark instances with our method,

showing that it is applicable to problems with very different constraints and still able to provide good results.

4.5.5 Conclusion

In this first part of the chapter, we presented the Bus Driver Scheduling (BDS) problem using the complex set of rules in the Austrian collective agreement. We analysed characteristic features of real-life instances, in particular the demand distribution and waiting times and discussed their implications on scheduling. We provided new publicly available benchmark instances for further research and presented a solution method based on a construction heuristic and Simulated Annealing (SA). We successfully applied these methods to the new benchmark instances and discussed the importance of a well-designed objective function. We provided insights in the successful application in real-life scenarios and compared with a problem from Brazil, where we could improve several benchmark instances.

4.6 Branch and Price

The methods described in the previous section have provided good results, both in practical applications and on a benchmark problem from literature. Still, it is difficult to estimate how close to an optimal solution these results are since metaheuristic methods cannot provide any optimality guarantee or bounds for the solution. Therefore, the next step was to investigate exact methods for the BDS problem. However, the problem has high complexity and large instances that make it difficult to provide competitive exact methods.

As a first step we looked at direct modelling of the problem using CP Optimizer. However, even without the full complexity of the break requirements, it became clear that this approach will not be able to provide a solution that will scale for anything beyond the very small instances.

Therefore, the next step was to proceed to Branch and Price (B&P) [BJN+98], a decomposition technique that splits the problem into a master problem that generates the best schedule from a set of given candidate shifts, and a subproblem that generates new shifts. As mentioned before (Section 4.1), B&P is the classical way to approach BDS with exact methods. However, as our version of the problem shows much higher complexity regarding the break constraints compared to most problems from literature, this makes solving the subproblem much more complicated in our case, requiring the introduction of several improvements to make the approach applicable to our problem.

The column generation uses set partitioning as the master problem and the Resource Constrained Shortest Path Problem (RCSPP) as the subproblem. However, due to the complex set of constraints, the RCSPP has to deal with a large number of high-dimensional labels. The following sections propose several novel contributions to improve the generation of new columns for high dimensional pricing problems, including the

splitting of the subproblem, a throttling scheme, and the use of k-d trees and bounding boxes in a two stage dominance algorithm.

The new method is evaluated on the publicly available set of benchmark instances introduced before. It provides the first provably optimal solutions for small instances and improves previous best-known solutions or proves them optimal for 48 out of 50 instances, resulting in an optimality gap of less than 1% for more than half the instances. It also provides a lower bound for the quality of existing metaheuristic approaches which are mostly within 4% of the optimum.

4.6.1 Overview

Figure 4.8 shows an overview of the solving process using Branch and Price. To get the optimal integer solution, column generation is applied on each node of a branching tree. The master problem to be solved with the current set of shifts, called the Restricted Master Problem (RMP), has its integrality constraints relaxed and is solved using a MIP solver. The duals of the RMP are used to find new shifts that have the potential to improve the solution of the master problem. This is the case when they have negative reduced cost. If such shifts are found, they are added to the set of shifts, and the relaxed RMP is solved again. Otherwise, the solution of the relaxed RMP is optimal for the current node in the branching tree and provides a local lower bound for the solution of the integer problem.

In some cases, the resulting solution might already be integer, and the current branch is done. Usually, however, the result will be fractional. Then, the integer version of the RMP is solved with the current set of columns. While there are no guarantees regarding the quality of the integer solution in this case, in practice, solutions are often very close to the lower bound already. In any case, the result from the integer master problem is a feasible solution that provides a global upper bound for the problem. If the solution is not integer, branching will be done. In any case, next an open branch is selected to continue. The process keeps going until all branches either result in integer solutions or are cut off by the current objective bounds.

4.6.2 Master Problem

The goal of the master problem is to select a subset of the shift set $\mathbf{S}$, such that each bus leg is covered by exactly one shift while minimizing total cost. This corresponds to the set-partitioning problem:

$$\text{minimize} \sum_{s \in \mathbf{S}} cost_s \cdot x_s \tag{4.20}$$

$$\text{subject to} \sum_{s \in \mathbf{S}} cover_{s\ell} \cdot x_s = 1 \qquad \forall \ell \in \mathbf{L} \tag{4.21}$$

$$x_s \in \{0, 1\} \qquad \forall s \in \mathbf{S} \tag{4.22}$$

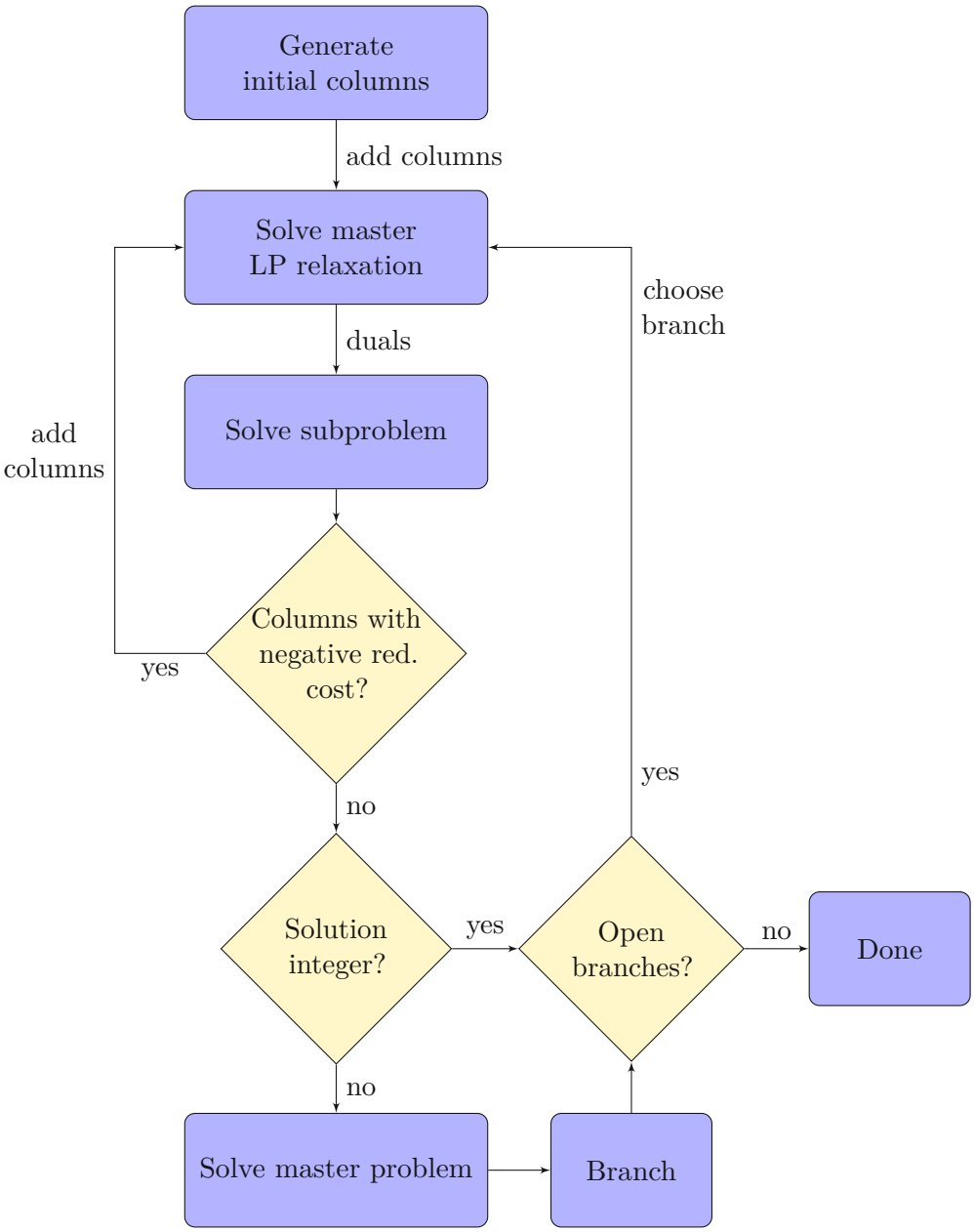


Figure 4.8: Branch and Price flow

Here x_s is the variable for the selection of shift s. The objective (4.20) minimizes the total cost, Equation (4.21) states that each bus leg needs to be covered exactly once (using $cover_{s\ell} \in \{0, 1\}$ to indicate whether shift s covers leg ℓ), and Equation (4.22) states the integrality constraint. This constraint is relaxed to $0 \leq x_s \leq 1$ for the relaxed master problem which is repeatedly solved at each node of the branching tree.

4.6.3 Subproblem

The goal of the subproblem is to find the column (shift) with the lowest reduced cost. For BDS this corresponds to the Resource Constrained Shortest Path Problem (RCSPP) on an acyclic graph. In this problem, a graph $G = (\mathbf{N}, \mathbf{A})$ is given where $\mathbf{N}$ is the set of n nodes, corresponding to all bus legs, a source node s, and a target node t, and $\mathbf{A}$ is the set of arcs, corresponding to connections between bus legs. As the bus legs are naturally ordered by time, the RCSPP is acyclic.

Each node and arc is associated with a cost and an r-dimensional resource vector representing the resource consumption when using the node or arc. A shift is defined as a path from s to t such that the path satisfies all feasibility criteria associated with the resources. Each node corresponding to a bus leg is associated with a dual from the master problem. The reduced cost of a path is therefore the cost of a path minus the sum of the duals along the path. The RCSPP aims at finding the least-cost feasible path from s to t.

There is an arc from node s to each node i except $i = t$, and there is an arc from each node i except $i = s$ to node t. Nodes corresponding to bus legs i and j are only connected by an arc ij if chaining the bus legs is feasible according to their times and the distance matrix d. Building the graph is done once per instance in $O(n^2)$. The rest of this section goes into the details of this graph, its costs, and its constraints. Note that these are similar to the accumulators used in Section 4.4.1 for Simulated Annealing, but they still differ in several ways due to the representation as an RCSPP.

Figure 4.9 shows an example for the RCSPP graph of a small toy instance, where the numbers next to nodes and arcs show the costs, while resources and duals are not shown. This instance represents three very short tours with three or four bus legs each, one per row in the figure, e.g., node 0 represents the first bus leg of the first tour, followed by nodes 1, 4, and 7. Arcs between nodes from different tours show possibilities to change tour, arcs might also allow to skip legs on the same tour. Note that for instances of relevant size each leg is connected to the majority of later legs, making the graphs very dense.

Costs

The costs for nodes and arcs are based on objective (4.1). The cost c_i for each node i corresponding to a bus leg i is defined as $c_i = 3 \cdot length_i$ as each bus leg contributes its length to both W_s (weight 2) and T_s (weight 1). By definition, $c_s = c_t = 0$.

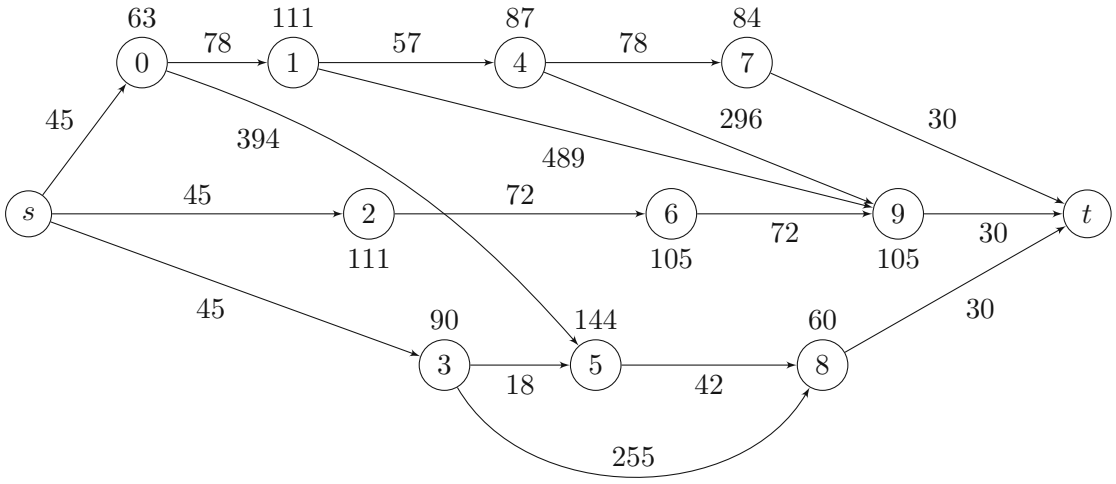


Figure 4.9: The RCSPP graph for a toy instance

Several helpful properties of arcs from node i to node j are defined and used for defining the arc costs:

$$length_{ij} = \begin{cases} start_j - end_i & \text{if } i \neq s \wedge j \neq t \\ startWork_{startPos_j} & \text{if } i = s \\ endWork_{endPos_i} & \text{if } j = t \end{cases} \tag{4.23}$$

$$ride_{ij} = \begin{cases} d_{endPos_i, startPos_j} & \text{if } i \neq s \wedge j \neq t \wedge endPos_i \neq startPos_j \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

$$change_{ij} = \begin{cases} 1 & \text{if } i \neq s \wedge j \neq t \wedge tour_i \neq tour_j \\ 0 & \text{otherwise} \end{cases} \tag{4.25}$$

$$split_{ij} = \begin{cases} 1 & \text{if } i \neq s \wedge j \neq t \wedge length_{ij} - ride_{ij} \geq 3 \cdot 60 \\ 0 & \text{otherwise} \end{cases} \tag{4.26}$$

$$remain_{ij} = \begin{cases} length_{ij} - ride_{ij} & \text{if } split_{ij} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.27}$$

$$rest_{ij} = \begin{cases} remain_{ij} & \text{if } i \neq s \wedge j \neq t \wedge remain_{ij} \geq 15 \\ 0 & \text{otherwise} \end{cases} \tag{4.28}$$

Equation (4.23) defines the length of an arc, taking into account start and end arcs. Equation (4.24) states the passive ride time, Equation (4.25) whether a tour change occurs, Equation (4.26) whether the arc corresponds to a shift split, Equation (4.27) captures the remaining arc length after removing the passive ride time and the shift split time, and Equation (4.28) captures a potential rest break.

The cost c_{ij} for arc ij (i.e., bus leg i to j) is defined as follows:

$$c_{ij} = 2 \cdot remain_{ij} + length_{ij} + 3 \cdot ride_{ij} + 30 \cdot change_{ij} + 180 \cdot split_{ij} \tag{4.29}$$

Note that unpaid rest cannot be determined at this point, therefore all rest is treated as paid for this computation. Unpaid rest is separately treated when solving the subproblem. $ride_{ij}$ contributes to both working time W_s and the additional passive ride penalty.

Resources

The solution method is a label-setting algorithm [ID05]. Due to the acyclic nature of the graph, each node can be processed in temporal order, starting with an initial label representing an empty path at s where each resource usage is 0. For each node i and each label x on that node, all arcs ij are processed and a label y is placed at the end node j of the arc unless some constraints are violated. Therefore, each label x represents a path from s to its node i, capturing the nodes in the path, the cost, and the resource usage. In total, eleven resources need to be tracked in order to satisfy the BDS constraints. The first resources are classical additive resources with a maximum allowed usage:

$$d_y = d_x + drive_j \tag{4.30}$$

$$s_y = s_x + length_{ij} + length_j \tag{4.31}$$

Equation (4.30) tracks driving time, Equation (4.31) the span.

$$rd_y = \begin{cases} true & \text{if } length_{ij} \geq 30 \vee (length_{ij} \geq 20 \wedge b20_x \geq 1) \vee \\ & (length_{ij} \geq 15 \wedge b15_x \geq 2) \\ false & \text{otherwise} \end{cases} \tag{4.32}$$

$$dc_y = \begin{cases} 0 & \text{if } rd_y \\ dc_x + drive_j & \text{otherwise} \end{cases} \tag{4.33}$$

$$b15_y = \begin{cases} 0 & \text{if } rd_y \\ b15_x + 1 & \text{if } \neg rd_y \wedge length_{ij} \geq 15 \\ b15_x & \text{otherwise} \end{cases} \tag{4.34}$$

$$b20_y = \begin{cases} 0 & \text{if } rd_y \\ b20_x + 1 & \text{if } \neg rd_y \wedge length_{ij} \geq 20 \\ b20_x & \text{otherwise} \end{cases} \tag{4.35}$$

Resources for monitoring drive breaks need to be reset at each full drive break. Equation (4.32) defines a helping flag to indicate whether the current drive block is finished. Equation (4.33) uses this flag to reset or increase the current driving time. Equation (4.34) tracks the number of 15-minute breaks in the current driving block and Equation (4.35) the number of 20-minute breaks. 30-minute breaks do not need to be tracked as each of them resets the driving block.

Constraints regarding rest breaks need to consider different sums of rest breaks and their positioning:

$$w_y = w_x + u_x - u_y + remain_{ij} + ride_{ij} + length_j \tag{4.36}$$
$$r_y = \min\{r_x + rest_{ij};\ 45\} \tag{4.37}$$
$$b30_y = b30_x \vee rest_{ij} \geq 30 \tag{4.38}$$
$$rest'_{ij} = rest_{ij} - \max\{2 \cdot 60 - s_x;\ 0\} - \max\{end_i + rest_{ij} - (end_y - 2 \cdot 60);\ 0\} \tag{4.39}$$
$$u_y = \min\left[u_x + \begin{cases} rest'_{ij} & \text{if } rest'_{ij} \geq 15 \\ 0 & \text{otherwise} \end{cases};\ 90\right] \tag{4.40}$$
$$bc30_y = bc30_x \vee rest''_{ij} \geq 30 \tag{4.41}$$

Equation (4.36) tracks working time, assuming that all of u_y is unpaid so far, which constitutes a lower bound for the actual value. Equation (4.37) tracks the amount of required rest time, capping it at 45 as higher values do not matter. Equation (4.38) deals with the occurrence of a 30-minute rest break.

Equation (4.39) captures the part of the rest break that can be unpaid depending on the position. However, this requires knowing the end time of the shift end_y which violates the general assumption of the algorithm that nodes can be processed in temporary order. Therefore, for each node j, with known end time end_j if j is the last bus leg in the shift, new nodes are added to the graph: $\mathbf{N}_j$ is the set of nodes reachable within 3 hours from end_j when traversing the network backwards from j (including j itself). For each node $i \in \mathbf{N}_j$ a new node i_j is created. For each pair of nodes i and k both in $\mathbf{N}_j$, an arc i_jk_j is added if ik is in the original graph. For $i \notin \mathbf{N}_j$ and $k \in \mathbf{N}_j$, an arc ik_j is added if ik is in the original graph. The arc j_jt replaces the arc jt. Each new node is associated with the end time end_j. All original nodes have end time ∞. This solves the problem, but increases the graph size by a factor of 6 for small instances up to more than 30 for large instances.

The potential total amount of unpaid rest is captured in Equation (4.40). Equation (4.41) tracks the existence of a centred 30-minute break using $rest''_{ij}$ defined like in (4.39) except with $3 \cdot 60$ instead of $2 \cdot 60$.

Finally, the cost needs to be computed:

$$cost'_y = cost'_x + 2 \cdot (u_x - u_y) + c_{ij} + c_j - dual_j \tag{4.42}$$
$$cost_y = cost'_y + 2 \cdot \max\{W_{min} - w_i;\ 0\} \tag{4.43}$$

Equation (4.42) takes into account the unpaid rest and the duals for the bus legs. Equation (4.43) further considers the minimum working time.

Constraints

A shift is a path from s to t that does not violate any resource constraints. The following resource constraints need to be satisfied for every label x:

$$d_x \leq D_{max} \tag{4.44}$$
$$s_x \leq T_{max} \tag{4.45}$$
$$dc_x \leq 4 \cdot 60 \tag{4.46}$$
$$w_x \leq W_{max} \tag{4.47}$$
$$r_x \geq 15 \text{ if } w_x \geq 6 \cdot 60 \tag{4.48}$$

Additionally, at node t, all labels x need to satisfy:

$$r_x \geq 45 \text{ if } w_x > 9 \cdot 60 \tag{4.49}$$
$$b30_x \text{ if } w_x \geq 6 \cdot 60 \tag{4.50}$$

Dominance

In order to track only paths with the potential to become minimum-cost paths, only pareto-optimal labels with respect to both cost and resource usage are kept at each node, i.e., labels which are not dominated. Regarding more complex constraints, it is important to observe that, if label x dominates label y at a node i, extensions of x to future nodes must still dominate extensions of y. Finally, the least-cost label at node t represents the minimum-cost path.

Therefore, a label x dominates label y if all following conditions are true:

$$d_x \leq d_y \tag{4.51}$$
$$s_x \leq s_y \tag{4.52}$$
$$dc_x \leq dc_y \tag{4.53}$$
$$b20_x \geq b20_y \tag{4.54}$$
$$b15_x \geq b15_y \tag{4.55}$$
$$w_x + u_x \leq w_y \tag{4.56}$$
$$r_x \geq r_y \tag{4.57}$$
$$b30_x \vee \neg b30_y \tag{4.58}$$
$$bc30_x \vee \neg bc30_y \tag{4.59}$$
$$c_x + 2 \cdot u_x \leq c_y \tag{4.60}$$

In general, lower values dominate for resources with upper bounds and higher values dominate for resources with lower bounds. The most complex case arises from the fact that the maximum value of unpaid break might be 0, 60, or 90 depending on the existence and position of a 30-minute rest break. This results in neither higher nor lower values of

u_i being dominant. Rather, the uncertain amount of unpaid rest weakens the domination power of cost and working time, as those might vary in the amount of unpaid rest until the end of the path. However, in the next section a more useful way to deal with this problem is described.

4.6.4 Branching

Branching is performed on the connections between bus legs in a shift that correspond to arcs in the subproblem. The branching considers a connection between bus legs i and j that appears in a fractional shift in the solution to the master problem and selects the most fractional one (selection value closest to 0.5) to maximize impact. In the left branch, all columns containing i or j, but not both consecutively, are removed. In the RCSPP graph all outgoing arcs from i except for the one connecting to j are removed. In the right branch, all columns with i and j assigned consecutively are removed, as well as the arc from i to j in the RCSPP graph.

4.6.5 Lagrangian Bound

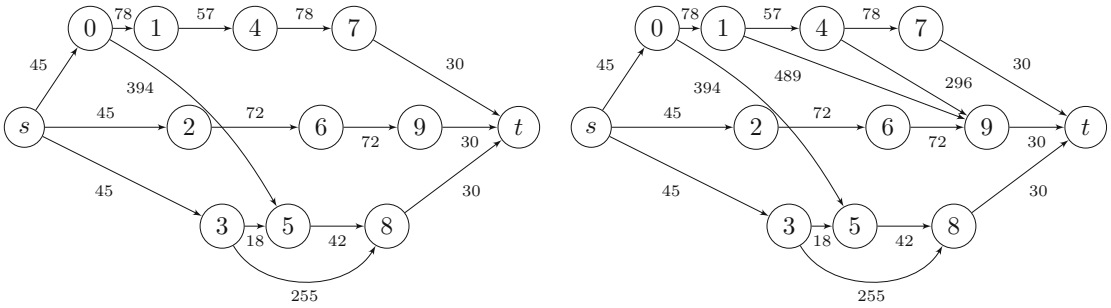
In the case when the column generation terminates at the root node, small optimality gaps are typically achieved. However, when there is not enough time to complete the column generation at the root node, a Lagrangian lower bound is computed as $O(RMP)+\kappa \cdot O(PP)$, where $O(RMP)$ is the optimum of the relaxed RMP, κ is an upper bound on the number of shifts and $O(PP)$ is the optimum of the pricing problem. The implementation uses $\kappa = \lfloor O(RMP)/minCost \rfloor$, where $minCost = 2 \cdot W_{min} + \min_{\ell \in \mathbf{L}} length_\ell$. The minimum reduced cost is only known when no more throttling (see next section) is in play. Otherwise a lower bound for the minimum reduced cost is computed where the constraints regarding break positions for unpaid breaks are relaxed.

4.7 Handling the Subproblem Dimensionality

As described in the previous section, due to the complex constraints for valid shifts, the subproblem complexity is very high. In particular, eleven resources are tracked in the labels. However, the efficiency of the label-setting algorithm depends on its ability to keep a small number of non-dominated labels. The more dimensions the labels have, the easier it is for labels to be non-dominated, drastically increasing the number of labels for processing. Therefore, the increase in the number of processed labels turns out to be the bottleneck of scaling the solution method to larger instances. Several novel improvements to reduce this bottleneck are introduced. These are generally applicable to other problems with similar characteristics.

4.7.1 Subproblem Partitioning

Instead of generating all possible shifts from one graph, the subproblem is split into three similar, but disjoint RCSPP problems, removing the uncertainty for unpaid breaks



(a) No rest breaks

(b) With rest breaks

Figure 4.10: Subproblem partitioning

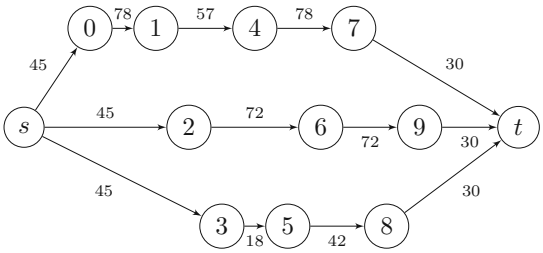
depending on whether the maximum amount of rest break is 0, 60, or 90. This depends on the existence and position of a 30-minute rest break:

- *No 30-minute rest break*: All arcs corresponding to rest breaks of at least 30 minutes can be removed, making the graph very spare and therefore fast to process. Unpaid rest is guaranteed to be 0, all rest break resources are ignored.

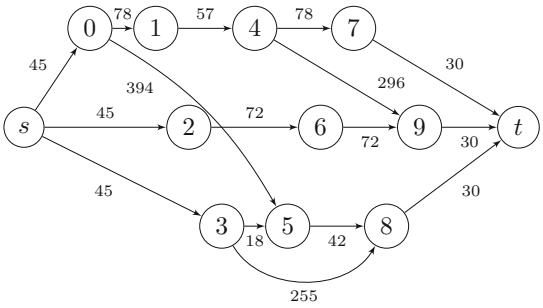
- *Uncentred 30-minute rest break*: $b30_y$ must be true at t. The maximum of u_y is changed to 60, but the current amount of u_y is guaranteed to be unpaid. Therefore, $w_x \leq w_y$ and $w_x + u_x \leq w_y + u_y$ can be used instead of $w_x + u_x \leq w_y$ as the domination criterion (same change for c_x and c_y), reducing the number of undominated labels.

- *Centred 30-minute rest break:* $b30_y$ and $b30c_y$ must be true at t. The maximum of u_y is set to 90, but the improved dominance criteria from the previous graph still apply as again all of u_y is guaranteed to be unpaid.

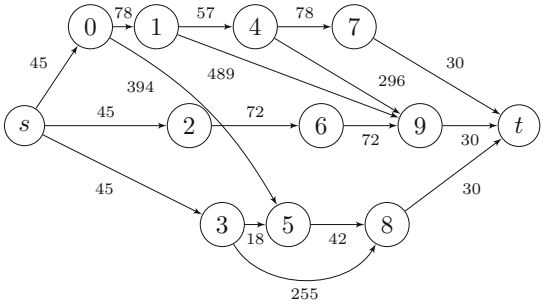
Figure 4.10 shows the different graphs for the toy instance. While here the effect is small, the majority of arcs can be removed for the graph without rest breaks in realistically sized instances. The pricing subproblem is used to add multiple columns (up to 1000 per graph) at once. Indeed, solving the relaxed master problem even with thousands of columns is much faster than solving the pricing subproblem. Therefore, accepting some unnecessary columns while reducing the number of times the subproblem is solved pays off. The different graphs are ordered by their complexity, measured by the number labels they expand. The pricing subproblem avoids searching more complex graphs, as long as those with less complexity still produce enough shifts with negative reduced costs (usually 10, 100 if the objective did not change, 1000 if the objective did not change repeatedly).



(a) Throttling limit 100 and 200



(b) Throttling limit 400



(c) Throttling limit 800

Figure 4.11: Throttling stages

4.7.2 Cost Bound

An upper bound for the reduced cost for each node can be computed by processing the RCSPP graph backwards with the current duals. The upper bound describes the maximum reduced cost at each node, such that there is still a path to arrive at t with reduced cost < 0, disregarding resource constraints. All labels above this bound can be discarded during the solution process.

4.7.3 Exponential Arc Throttling

While it might be hard to find the best column, at least in the beginning of the solving process, there are many columns with negative reduced cost. Two options were considered to reduce the size of the subproblem in early iterations. The first option is to reduce the number of labels at each node. This can be done either by rejecting new labels after a certain threshold or only retaining labels according to certain criteria, e.g., the best 100 by cost.

The better option is to exploit the fact that good columns are unlikely to include very costly arcs. It proposes a new throttling mechanism that imposes a maximum cost per arc, starting at 100 (just enough for a 30 min break). This greatly reduces the size of the graph and therefore the number of labels. Once the number of new columns is small, this

factor is multiplied by 2. This is repeated until all arcs are included. While a lot of arcs are not present in the early stage, the focus on arcs that are likely to appear in good columns leads to very fast convergence to good solutions, even if there is not enough time to finish the column-generation process. Figure 4.11 shows the approach for the toy instance.

4.7.4 Improved Elimination of Non-Dominated Labels

Even with the previously mentioned improvements, the majority of runtime is spent figuring out which labels are non-dominated. The naive approach is to compare each new label on a node with each label in the current set of non-dominated labels. However, the runtime complexity of this method is quadratic in the number of non-dominated labels. Therefore, two different methods are explored to speed up the dominance checks. The first method is the multi-dimensional divide-and-conquer [Ben80]: It is based on recursively solving the k-dimensional problem with n labels by two k-dimensional problems of size $n/2$ and one $k-1$-dimensional problem of size n.

The second method is a two-stage approach based on k-d trees and bounding boxes [CHT12]. The two-stage approach is more effective for several reasons. First, the algorithm is based on two stages. Instead of comparing each new label with all previously non-dominated labels both ways and deleting dominated labels in the process, two passes are performed. Labels are added as long as they are not dominated, but no removal operations are executed. The second pass is performed in opposite order before expanding the labels at a node, ensuring that only non-dominated labels are expanded. Second, instead of storing labels in a list, a k-dimensional tree is used for storage. For each node r on level ℓ, the left child is better with respect to dimension ℓ mod k and the right child is worse (or equal). Finally, each node r is associated with a bound u_r where each dimension contains the best value among all nodes in the subtree rooted at r. Therefore, if a label is not dominated by u_r, it is not dominated by any label in the subtree rooted at r, saving several comparisons.

4.8 Branch and Price Evaluation

The implementation uses Java (OpenJDK 14.0.1), the master problem is solved by CPLEX 12.10. The evaluation was performed on Windows 10 Professional using an Intel Core i7-6700K with 4x4.0 GHz and 32 GB RAM. The Java implementation is running on a single thread, CPLEX uses the default of all four cores. The application is deterministic: Repeated runs produce the same result. The method was evaluated on the same set of benchmark instances as SA.

4.8.1 Results

Table 4.7 shows the results of using the best configuration on the benchmark instances. Results are presented as average over the respective size category. The runtime for B&P

	Branch and Price				Metaheuristics			
Inst.	Time	LB	Best	Gap	Time	Best	Gap LB	Gap best
10	7.2	14709.2	14709.2	0 %	22.8	14717.4	0.06 %	0.06 %
20	1201.4	30290.3	30294.8	0.01 %	62.2	30790	1.65 %	1.63 %
30	3610.6	49674.4	49846.4	0.35 %	108.8	50947.4	2.56 %	2.21 %
40	3605.8	66780.3	67000.4	0.33 %	267.0	69072.2	3.43 %	3.09 %
50	3674.4	84042.1	84341.0	0.36 %	295.4	86539.6	2.97 %	2.61 %
60	4373.2	99334.0	99727.0	0.40 %	432.8	103445.2	4.15 %	3.74 %
70	6460.4	108083.1	118524.2	9.66 %	751.4	122531.4	13.4 %	3.38 %
80	5912.4	106499.2	134513.8	26.3 %	959.4	139518.2	31.0 %	3.72 %
90	7390.4	104848.1	150370.8	43.4 %	1516.6	156046.0	48.8 %	3.77 %
100	7395.8	102858.6	172582.2	67.8 %	1483.2	172269.6	67.5 %	-0.18 %

Table 4.7: Results using Branch and Price

was limited to one hour and up to one more hour to finish the last ongoing computation. This is needed to find a feasible solution even if the column generation at the root node does not terminate.

Results show that small instances can be optimally solved in very short time. For size 20, 4 out of 5 instances can be solved optimally within an hour, with a gap of only 0.07 % for the remaining instance. For all instances up including size 60, the root node terminates within one hour and the optimality gap of the solution is always less than 1%, a similar result also occurs for 2 instances of size 70 and one instance of size 80. For larger instances, the lower bound is provided by the traditional Lagrangian bound, whose quality depends on the column generation upon termination. However, due to the exponential arc throttling, the results obtained in this time are still of high quality, improving or proving the optimality of previous best known solutions on 48 out of 50 instances. It is expected that the new results are very close to the optimum as the lower bound shows slower convergence compared to the upper bound. For the larger instances (starting with some of size 70), solving the integer set-partitioning problem with around 50000 generated columns also times out with a gap of a few percent. Those are the instances with sightly above 2 hours of runtime.

There are two main contributions provided by this part of the research. On the one hand, the approach provides lower bounds to evaluate the quality of the previous solutions obtained by metaheuristics. In particular, they show that, for 4 out of 5 instances of size 10, the optimum was previously found (but not proven optimal). On the other hand, except for size 100, the Branch and Price algorithm improves existing results by 1-4%, with results for sizes 60–90 being close to 4%. This is very significant for such logistics and transportation problems.

Note that the metaheuristic solution methods have the advantage of lower runtimes (except for size 10) and are useful for even larger instances. However, in practical scenarios like providing cost calculations when competing for new bus lines, or investigating the

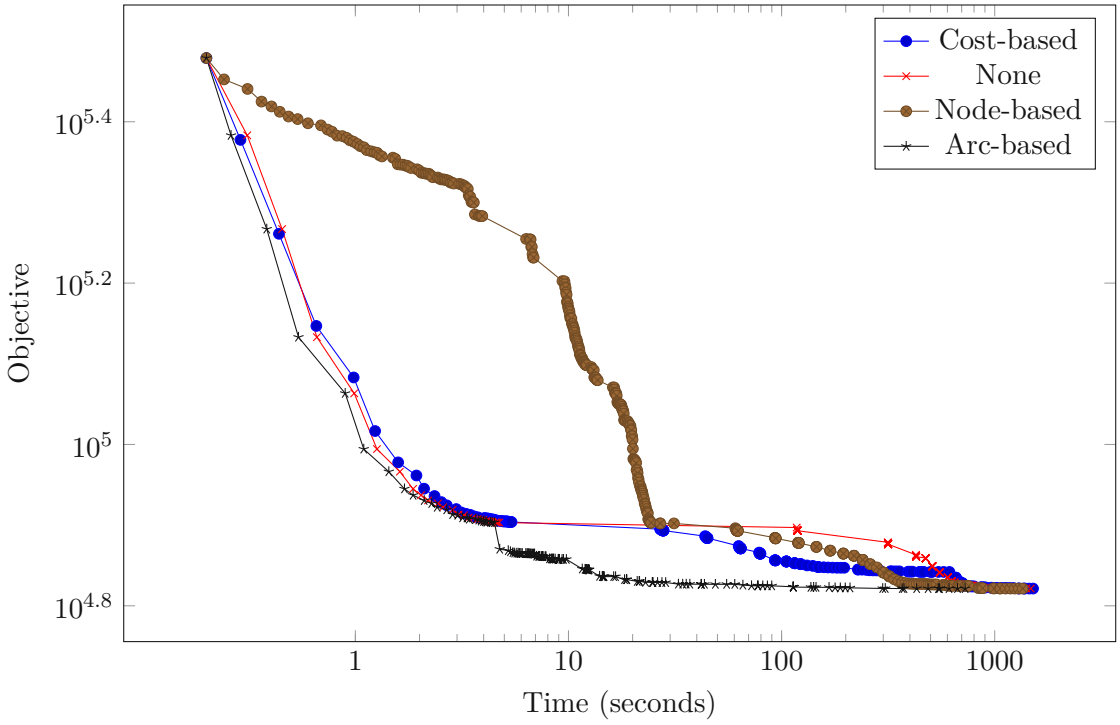


Figure 4.12: Throttling approaches for 40_16

potential improvement for the current bus network, spending some extra time for better and bounded solutions is very important, making B&P very valuable for for such scenarios.

4.8.2 Effects of Subproblem Improvements

While all the design choices and parameter settings have been carefully tested, this section focuses on the effects of our crucial subproblem improvements.

We highlight the benefits for splitting the subproblem on the example of instance 40_16. At the root node the subproblem is solved 197 times. Only 76 times the uncentred 30-min break graph is solved and only 14 times the centred 30-min break graph. However, the runtimes for the different graphs are 4 ms, 1.5 s and 1.9 s respectively at the beginning (arc throttling) and 16 ms, 7.7 s and 51.1 s at the end (all arcs). Even though easier subproblems only provide parts of the new shifts, their runtime advantage makes it worth only going to the more complex subproblems when necessary.

Figure 4.12 shows the comparison of cost-based throttling (100 best regarding cost per node, factor 10 increase), no throttling, node-based throttling (100 first per node, factor 10 increase), and arc-based throttling (arc cost limited to 100, factor 2 increase) on the objective of the relaxed RMP over time. The results show a significant overhead of cost-based throttling, and weak improvements early but more efficiency later for node-based throttling. Arc-based throttling, however, clearly shows superior performance

starting to drop significantly in objective value compared to other approaches in less than 10 seconds. Its overall time is 727 seconds compared to 1354 seconds for node-based throttling, the second best approach. Overall, using node-based throttling would provide improvement compared to the metaheuristic approaches on 16 fewer instances than arc-based throttling and terminate the root node for 5 fewer instances.

Regarding the dominance algorithm, one run of the largest subproblem on instance 40_16 takes 290 seconds with the default dominance algorithm, 293 seconds with the multidimensional divide-and-conquer algorithm, and 51 seconds with the two-stage algorithm. Overall, for default dominance the root node terminates in only 25 compared to 33 instances for the two-stage algorithm within the time limit.

4.9 Conclusion

This chapter provided a detailed analysis of problem characteristics and solution methods for the Bus Driver Scheduling (BDS) problem. The problem specification is directly taken from practice, modelling the full extent of the complex constraints from an Austrian collective agreement, which provides much higher complexity regarding break assignments than previous work in literature regarding BDS, which contributes to Research Goal 1 (Section 1.1.1). The formal specification of the problem was further aided by providing an analysis of real-life demand distributions and the design and evaluation of an objective function that incorporates both the need to reduce cost and several aspects to provide schedules that respect the needs of the drivers, making the results applicable in practice.

Since hardly any benchmark instances are available for BDS problems in literature, we generated new instances that reflect the properties of the real-life instances and made them publicly available for further research. Solutions to the problem were first provided using a greedy construction heuristic and metaheuristics based on Simulated Annealing (SA) and randomized hill climbing, providing good results to the new benchmark instances, as well as real-life applications and a comparison to a BDS problem with different constraints from Brazil, highlighting both the quality of the results and the flexibility to apply the methods to different versions of the problem.

Next we aimed at getting exact results to provide optimal solutions for smaller instances and bounds for larger instances, using Branch and Price (B&P) with set partitioning as the master problem and the Resource Constrained Shortest Path Problem (RCSPP) as the subproblem. The B&P approach includes a range of novel improvements for solving the high-dimensional subproblem that can be applied to other problems with similar characteristics. Experimental results showed that the approach provides strong lower bounds for the metaheuristics and improves or proves optimality for 48 out of 50 instances. The novel results have optimality gaps below 1% for small to medium size instances and represent high-quality solutions for larger instance sizes in reasonable time. Therefore the Branch and Price approach may be considered the new state-of-the-art exact method for this complex scheduling problem.

The high-quality results achieved using the different solution methods as well as the availability of new benchmark instances contribute to Research Goal 2 (Section 1.1.2). Future work regarding this problem might include applying the methods to more different versions of BDS problems, or incorporating robustness features like break buffers to increase robustness of the schedules when facing uncertainty in actual travel times. Further, if instances with different properties are encountered in further real-life applications, methods like Instance Space Analysis (ISA) might also be interesting for the BDS problem to gain more insight into relations between instance features and algorithm performances.

CHAPTER 5

Hyper-Heuristics for Personnel Scheduling

In the previous chapters, this thesis has presented several different approaches, both exact and heuristic, using different solvers and different methodologies to tackle personnel scheduling problems with varying demands and optimization goals. Problem-specific approaches are often necessary to reach solutions of very high quality, especially optimal solutions or solutions with a known small gap to a lower bound.

However, it can be very cumbersome to design and implement new solution approaches every time for different problems or even for other variants of problems that have some different constraints or optimization objectives. While for the academic discussion a fixed benchmark is required to be able to compare different approaches and reliably draw conclusions about their performance, and it is very important that the academic version of a problem still captures the full complexity of the practical problem (otherwise solution methods will not be able to perform adequately in practice), there are often some constraints that get changed or added in real-life applications.

Requirements can frequently change, as the exact constraints used in the end are often the result of a long refinement process with many iterations of changing constraints, adapting their weights, or even trying to figure out implicit constraints of previous (even manual) scheduling systems, and the needs of different stakeholders in the process, e.g., company and labour union representatives. Further, there might be applications for different customers in different lines of business with varying work regulations, using other collective agreements, or augmenting them with different company agreements. It might also be the case that companies want to compare different sets of constraints to find the best set to update their company agreements.

These factors lead to very dynamic problem specifications. In order to solve new problem variants efficiently and reuse as much as possible from existing solution methods, a clean

separation of problem constraints and solution methods should be pursued. This can be harder for fine-tuned exact methods. While adding additional constraints for the models presented for Rotating Workforce Scheduling or Minimum Shift Design is possible by formulating them in MiniZinc (and several options for that were shown in Chapter 2), it might still be difficult or slow down solving significantly. For the highly specialized Branch and Price method for Bus Driver Scheduling the adaptation to new rules is much more challenging due to the need to model them as resources in the RCSPP.

Therefore, this chapter aims to extend the spectrum of methods in the other direction, instead of presenting highly specialized methods that are harder to adapt to new problems, we present an interval-based framework for personnel scheduling problems that can model various different personnel scheduling problems in a common notation, and provides a clear separation between different components like problem-specific constraints, general moves and algorithmic components that provide a high degree of flexibility to deal with different variants of problems as well as entirely different problems.

With this new general framework, we can apply very general methods, using hyper-heuristics to solve the problems. These rely on smaller building blocks, called Low-level Heuristics and repeatedly choose an LLH to change the solution. This chapter presents a deep investigation of hyper-heuristics to the three challenging personnel scheduling domains presented in the previous chapters for the first time. We first propose several new LLHs for these domains. Then we present a major comparison of 10 different versions of state-of-the-art hyper-heuristics on all three domains. We obtain very good results on these domains, including several new best known solutions, and including practically relevant optimization goals that have been excluded previously. This provides strong results to deploy hyper-heuristics in practice to allow for easier adaptation to changing requirements and new problem domains. These contributions have been published at ICAPS 2022 [KM22].

Then we move to the development of our own hyper-heuristic method LAST-RL, introducing a new rich state representation to characterize the current state of the search with new features and use these for reinforcement learning in combination with an adaptive chain length selection based on the Luby sequence, and an exploration strategy based on adaptive selection probabilities according to Iterated Local Search to provide high-quality solutions. This new approach is evaluated on the well-known benchmark set of six different domains from the Cross Domain Heuristic Search Challenge 2011, where it can provide good results with high generality, outperforming the original competitors and providing good results in comparison with recent hyper-heuristic developments, outperforming previous hyper-heuristics mainly based on reinforcement learning. Then we apply LAST-RL on our three real-life domains where the previous high-quality results can further be improved on two out of the three domains, showing the high potential of using hyper-heuristics based on reinforcement learning for complex practical problems.

5.1 Interval-based Framework

A good way to start thinking about common methods for different problems is to consider what the problems themselves have in common. Hyper-heuristics do not need such a common formalism, as described later in this chapter, they only need a set of Low-level Heuristics for each specific problem (called a problem domain in this context). However, a common formalism for not only the personnel scheduling problems that are analysed in more detail in this thesis, but all kinds of personnel scheduling problems in general, has major benefits when new or different problems need to be investigated, since it is much more likely that different parts of previous problem formulations or solution methods can be reused. This is very important for the progress of research on general optimization methodology, but also in an industrial context like the work of our industry partner XIMES GmbH, where customers with very different scheduling problems want to have solutions to their problems in short time.

The idea behind the framework designed for personnel scheduling problems in the context of this thesis is to have some basic building blocks that are used to construct all schedules, which are intervals and sequences of intervals. An interval in its basic form has a start time, an end time, and an identification. Multiple intervals, ordered by their start times, form a sequence of intervals. These building blocks are used to build a very modular structure that allows to separate problem specifications, instances, and solution methods, represent different personnel scheduling problems, and adapt to different problem variants or implement new solution methods with minimal adaptation effort.

There has been previous work on general languages and frameworks for personnel scheduling problems and other domains. TEMPLE [GMSS11] is a domain-specific language for modelling and solving staff scheduling problems that uses some similar building blocks like intervals and moves, however, the use of a specific language requires a compiler to produce the optimization code, while our framework provides the reusable structure directly in Python, allowing much more flexibility. TEMPLE also uses a fixed set of possible algorithms for optimization. A framework for General Employee Scheduling (GES) [KM20c] also provides many general possibilities, but it is based on a specific problem and solution structure provided by the GES format, while the use of intervals provides less predetermined structure, but even more flexibility in the formulation of different problems. There is also work on frameworks for more specific domains, for example a Java framework for scheduling for deep space missions [MJS21]. The concept of modelling scheduling problems using intervals is also used in commercial software like CP Optimizer.

Figure 5.1 shows the overview of the connections between the most important parts of the framework. This thesis will not provide detailed information about implementation details, including many optimizations for efficient evaluation of changes using intelligent delta evaluation of the solution objective preventing unnecessary recomputations as much as possible, or details about input and output handling like the connection to the systems of our industrial partner, which would be outside the scope of this thesis. However, the

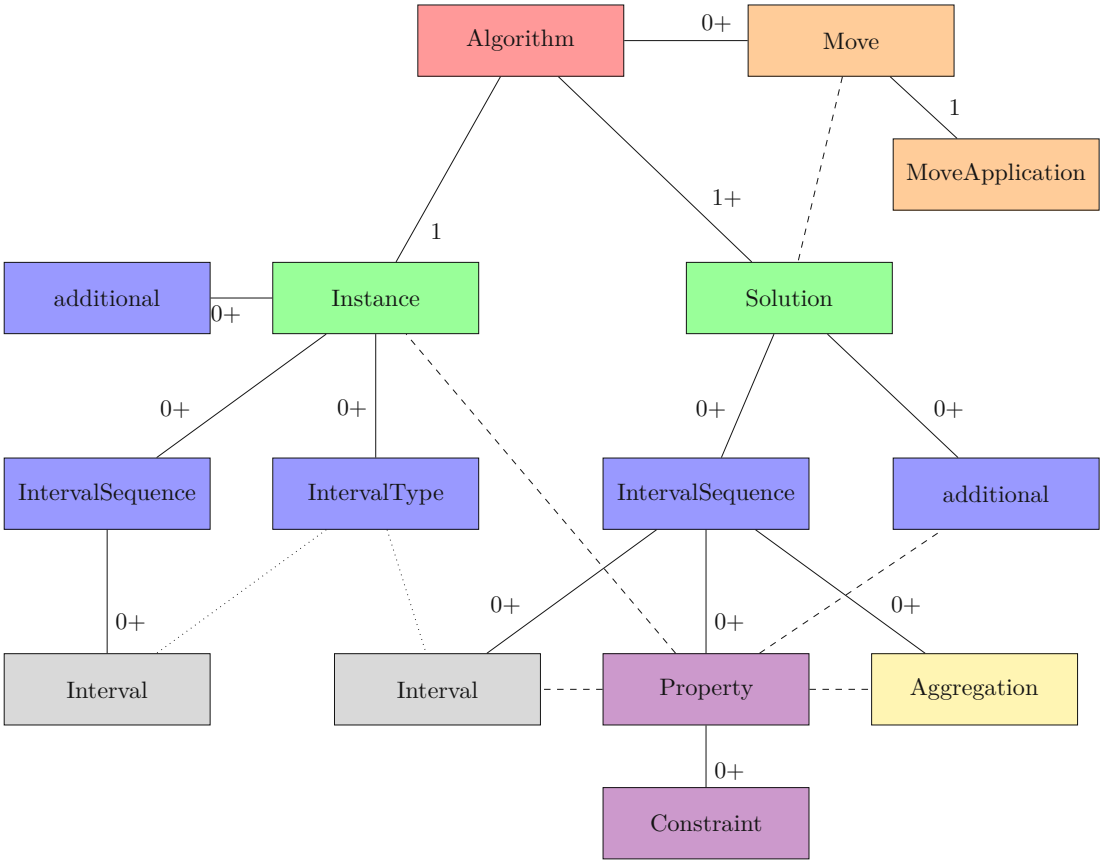


Figure 5.1: Interval-based framework

remainder of this section will provide an overview of the most important components of the framework and their interaction, and describe how to map the problems discussed in the previous chapters into this framework.

The core elements of the framework are connected in a tree-like structure (ignore the dashed and dotted lines in the figure for now, the numbers on the continuous lines represent possible quantities). The optimization methods are encapsulated in *Algorithms* (red), which are applied on an *Instance* and a *Solution* (green, a solution can either be empty or an existing solution to be modified). Constraints and their violations are tracked via certain *Properties* of parts of the solutions, that are then weighted by *Constraints*. The concept of *Intervals* as parts of *IntervalSequences* can be seen as the fundamental building blocks of both instances and solutions. In the framework, each interval is optionally also associated with an *IntervalType* (dotted lines), and a period of the scheduling horizon. For most problems the period corresponds to the day of assignment, however, while start and end time are absolute, a night shift starting after midnight might sometimes still count as a shift of the previous day, which makes this separate notion of a period necessary.

5.1.1 Instances and Solutions

An *Instance* represents the input data that is specific to one particular instance of a problem. The data is structured using different building blocks. For some problems, specific *Intervals* are already given as an input, and are grouped together in *IntervalSequences*, e.g., the fixed bus legs for BDS. Note that intervals can be extended by problem specific data if necessary. For example, for BDS a bus leg is an extension of an interval that does not only have a certain start and end time, but also start and end position, tour, and driving time.

While for BDS the task is to assign already existing intervals, an *IntervalType* is used as part of the instance if creating intervals is part of the problem. They define various limits for the creation of intervals, including, but not limited to windows for start and end time as well as lengths. For example, the definitions of shift types in MSD are mapped to this part of the instance definition.

Finally, *additional* contains all instance data that do not fit into the previous definitions. This allows arbitrary problem-specific data that might be necessary for problem-specific constraints. One type of additional data that is frequently used are demand curves like in MSD, but also the distance matrix for BDS or definitions for minimum and maximum sequence lengths for RWS are stored this way.

A *Solution* is actually represented in a very similar way to an instance. A solution also consists of (usually several) interval sequences and can have additional data, even though not used frequently (as a solution is usually some kind of schedule and a schedule can be represented using intervals and sequences of intervals). While an algorithm is not allowed to modify anything about the instance, the solution can be modified arbitrarily.

5.1.2 Properties and Constraints

The key elements to represent a problem specification in the framework are *Properties* that are based on *IntervalSequences*, and *Constraints*. A property is a numeric value that represents some aspect of an interval sequence, e.g., it could be the number of intervals in this sequence, or the sum of all lengths of intervals in this sequence.

For this purpose, the properties connect several sources of information (dashed lines), foremost using the intervals of the sequence. For more complicated properties, parts of the instance might be considered, e.g., the property that represents the total time for BDS needs the amount of extra work at the start and end of the shift from the additional data of the instance.

An *Aggregation* (yellow) is a specific construct that holds a value for each time in the scheduling horizon and allows a property to aggregate this information into a single value. E.g., for MSD an aggregation holds the difference between the currently assigned number of shifts and the demand specified by the instance for each point of time. Then the property aggregates the sum of the positive differences to get the amount of overcover for the current solution.

The value of a property can be used for one or more constraints. A constraint uses the value of the property to compute a violation, e.g., by simply taking the value if the property already holds an amount of violation, or by comparing the property value to a given maximum and taking the difference if the value is above the maximum. The violation is then multiplied with a weight and added to the objective function in one of multiple categories.

In this way, a property value can be used for multiple constraints, e.g., the working time of an employee for BDS can be weighted by 2 and added to the category for soft constraints, and also the part of the working time exceeding W_{max} can be weighted by a penalty weight for hard constraint violations and added to the category for hard constraints. The solution could then be discarded if hard constraint violations are not allowed, or the combined values of soft and hard constraints are used if an algorithm allows it. The categories could also be used in combination with multi-objective algorithms.

5.1.3 Algorithms

An *Algorithm* in the context of the framework is any method that takes an *Instance* and a *Solution* (or multiple solutions, e.g., to represent a crossover), and does any change to the given solution. The initial state of the solution can vary greatly, e.g., a construction heuristic might be an algorithm that works on a completely empty solution, while an improvement heuristic might expect an already feasible solution to work on.

The framework is mainly built and optimized with the intent to use (meta-)heuristic methods. These methods make use of *Moves*, which are structured ways to apply a small change to a solution, e.g., by moving one interval from one interval sequence to another. The whole structure used for the solution representation as well as properties and constraints is optimized to efficiently evaluate small changes to the solution (delta evaluation), which is very important for achieving good runtimes with the framework. Still, the design also allows for the use of exact methods for part of the solution process or solving the complete problem, as an algorithm is free to use any method internally to decide on changes to the solution.

For (meta-)heuristics the algorithm decides on a move to execute, while a *MoveApplication* is responsible for choosing where to apply the move in the context of the current problem and algorithm. E.g., for the move previously mentioned that moves one interval from one sequence to another, one such application could choose two random interval sequences and one random interval from the first sequence, while another one could systematically go through all pairs of sequences and all intervals within each sequence on consecutive calls to model a first or best improvement heuristic.

Also note that an algorithm does not need to do all the work on a problem on its own, it can employ multiple other algorithms and combine them. A typical use would be an algorithm that first applies a construction heuristic and then an improvement heuristic, where those two sub-heuristics are independent building blocks on their own.

5.1.4 Implementation and Practical Application

The framework is implemented in Python, since the language allows to write compact, but readable code and is very flexible. However, the comparably slow speed is a weakness of pure Python. Therefore, the just-in-time compiler PyPy is used for execution, allowing to run Python code in speeds competitive with other programming languages.

The modularity of the presented framework allows the separation of different important components, which is very useful in practice to reuse building blocks for different problems and provide solutions quickly. Many properties and constraints can be reused for many different problems (e.g., a property that describes the amount of overlap between intervals of the same sequence can be reused for all problems where such overlap is not wanted), and even if new properties need to be introduced, they can be added quickly using existing templates for properties (e.g., many properties only depend on differences between consecutive intervals in a sequence, so each new property of this kind only needs to specify how to add each difference to the overall value). Properties are collected over time, making it more and more likely that needed properties for new problems are already implemented.

On the other hand, the algorithms are completely independent from the problem constraints (unless specialized algorithms are implemented to depend on specific constraints if needed), so different algorithms can easily be applied to a problem, or the problem specification can be changed without requiring any changes on the algorithm. General improvement methods can even be used on completely different problems, only requiring to pass an appropriate set of moves that match the structure of the problem, without any further adaptations.

Further, from a software engineering view the framework allows easier testing of individual components. Since individual properties are independent from each other, they can be tested extensively in unit tests, making sure that regular and edge cases work as intended. Overall, the framework showed to be very suited to quickly adapt to varying problem specifications and is frequently used by our industrial partner to solve different versions of problems for different customers and usage scenarios.

5.1.5 Mapping Problems into the Framework

The three main problems of this thesis can now be represented in the interval-based framework as follows:

Rotating Workforce Scheduling

The input data for RWS as given in Section 2.2.1 is stored in the additional data component of the instance. The solution representation consists of a single interval sequence that represents the solution array T, where each interval represents one particular shift. Each shift is additionally assigned an integer that maps to the type of shift within the set $\mathbf{A}$. A shift is assigned to a particular day by using the period attribute, while start and end

are not used for the constraints except with the extension regarding weekly rest times. The offset o can be fixed according to Equations (2.18) and (2.19) and therefore is part of the instance as well.

Constraints are then mainly defined to keep track of sequences of intervals that are assigned to consecutive periods (sequences of work), or on the number of periods between intervals (sequences of days off). Instead of counting the shifts on each day of the week, the correct number of shifts is already assigned in a construction heuristic (described later), and shifts are always moved only by whole weeks. This way, the fact that the demand is fulfilled transfers into an invariant rather then additional constraints.

Minimum Shift Design

For MSD the input data consists of a demand table and a set of shift types (see Section 3.2). The shift types are translated to interval types in the framework, which can directly define windows for the start times and lengths of the intervals. The demand is stored as part of the additional data. The result is represented as one interval sequence containing a set of newly defined intervals, each of them representing one shift. The differences to the demand are tracked using an aggregation in the framework, tracking the two properties for T_1 and T_2. The additional optimization goals are tracked with additional constraints on the same sequence.

Bus Driver Scheduling

Regarding BDS (see Section 4.2), the instance provides the set of bus legs $\mathbf{L}$ as an interval sequence where each interval corresponds to one of the bus legs, extending the basic interval definition by the additional attributes like start and end position, driving time, and tour. The distance matrix as well as the values for working time at the start and end of the shift are part of the additional data.

A solution now corresponds to a set of interval sequences where each sequence represents one particular shift, and each bus leg is assigned to exactly one shift. Instead of using constraints to enforce this, the construction heuristic makes sure to return a corresponding solution, and it is later again treated as an invariant when changing the solution using other heuristics. The large number of problem constraints is represented by constraints on these sequences of intervals. While the set of constraints is especially complex and interdependent for this problem, the structure of the framework still allows to separately track the corresponding values using individual properties, some of them based on separated evaluations, while more complex ones use common evaluation components for interlinked properties, e.g., for all properties dealing with working time regulations.

5.2 Hyper-Heuristics Setup

The interval-based framework introduced in the previous section allows to build different solution methods, both problem-specific and more general. In the remainder of this

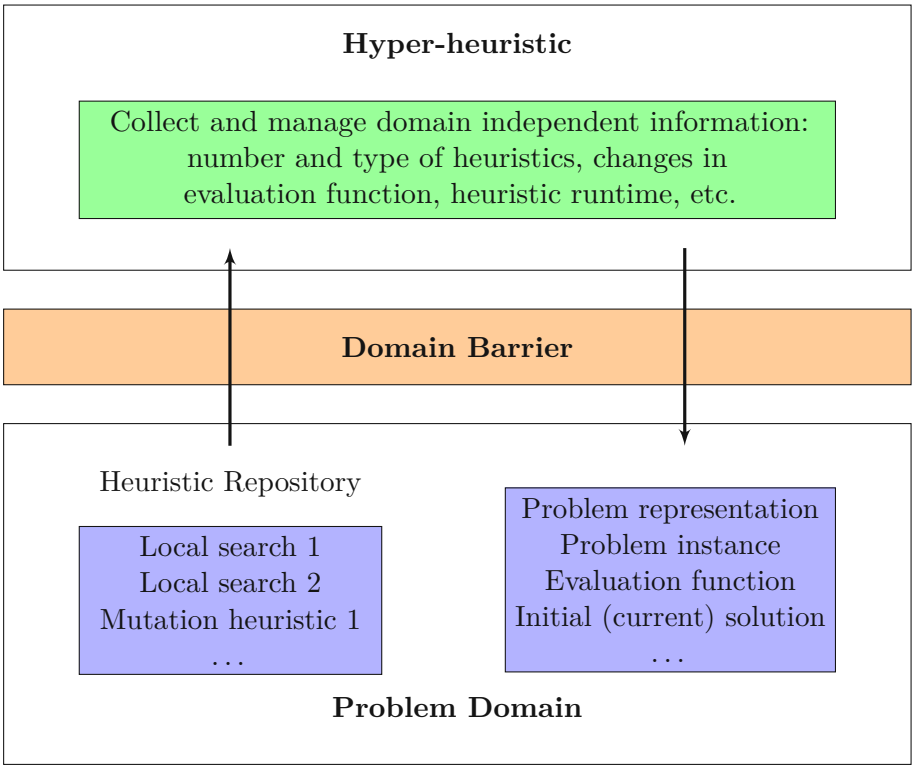


Figure 5.2: High-level concept of hyper-heuristics based on Burke et al. [BGH+11]

chapter, the focus will be on a specific type of very general heuristic optimization methods, which are hyper-heuristic techniques.

In contrast to problem-specific methods, hyper-heuristics are designed to work in a domain-independent way to provide efficient solutions across different domains, just requiring a set of Low-level Heuristics (LLHs) and comparably few or even no parameter inputs in order to create the appropriate heuristic for solving a particular problem instance on the fly based on the performance of the individual LLHs [BGH+13, BHK+19].

Figure 5.2 shows the high-level concept, which is based on the strict enforcement of the *domain barrier*, which separates the hyper-heuristic from the problem domain. The only information known to the hyper-heuristic are the values of a set of solutions, and the number and type of a set of problem-specific heuristics, the Low-level Heuristics. Until timeout, the hyper-heuristic chooses one of the available LLHs, sends this choice to the problem domain where the LLH is executed and the result (runtime of the execution and change in solution quality) are reported back to the hyper-heuristic. Based on the collected information about the performance of the individual LLHs the hyper-heuristic can build up knowledge which LLH to choose next. However, the strict domain barrier ensures that this process stays completely problem-independent, allowing the hyper-heuristic to be applied to any problem domain without adaptation. Instead, the adaptation for new

problem domains consists in the design of useful Low-level Heuristics for the domain that provide meaningful changes to a solution.

5.2.1 Related Work

The term hyper-heuristics for high-level problem solving methods that operate on a search space of Low-level Heuristics instead of directly on the solution [BHK+19] was introduced in 2000 [CKS01], even though the general ideas were used even earlier. The most common classification of hyper-heuristics [BHK+10] distinguishes selection hyper-heuristics, choosing from available LLHs, and generation hyper-heuristics that build heuristic operators from smaller building blocks. It further distinguishes construction hyper-heuristics that build a solution from scratch, while perturbation hyper-heuristics modify one or more solution candidates. Further the type of learning is distinguished. In this context this thesis investigates the use of perturbative selection hyper-heuristics, and further introduces the new hyper-heuristic LAST-RL in this category with reinforcement learning. The classification scheme was updated in 2019 [BHK+19].

A review of work in hyper-heuristics was published in 2013 [BGH+13]. Much work in the area is related to the Cross Domain Heuristic Search Challenge 2011 (CHeSC 2011) [BGH+11], where the framework HyFlex [OHC+12] was introduced for a uniform implementation of different hyper-heuristics. The competition featured six different domains, which are MaxSAT (SAT), Bin Packing (BP), Personnel Scheduling (PS), Flowshop (FS), Travelling Salesperson Problem (TSP), and Vehicle Routing Problem (VRP). These domains are implemented in the HyFlex framework, which provides an interface for the implementation of hyper-heuristics in Java and enforces the domain barrier, making sure that hyper-heuristics do not know specifics of the domain. The HyFlex framework uses Low-level Heuristics in four different categories:

- Local search: LLHs are guaranteed to return solutions that are not worse than how they started.
- Mutation: Any changes to the solution are allowed.
- Ruin-and-recreate (destroy-and-repair): These LLHs are supposed to destroy and rebuild a part of the solution, potentially leading to a larger change in the solution structure compared to mutations and local search.
- Crossover: Since HyFlex allows to work on a pool of solutions, crossover LLHs are applied to a pair of solutions.

The challenge was won by Mısır et al. [MVDCVB12, MVDCVB13] with a combination of adaptive mechanisms to manage a set of active Low-level Heuristics. A streamlined version of this algorithm was published by Adriaensen et al. [AN16], using Accidental Complexity Analysis to remove complexity from the algorithm that did not improve solution quality. The following places in the competition were taken by a self-adaptive

variable neighbourhood search [HCF12], and an algorithm using cycles of diversification and intensification [Lar11]. Extensions to the framework have been provided later [OWHC12], including several additional domains [AON15].

Reinforcement learning, which we use in our own hyper-heuristic, is based on the notion of executing actions depending on the current state, making the system transition to a different state and returning a reward. A great introduction is provided by Sutton et al. [SB18]. Many hyper-heuristics employ some kind of learning. However, the term reinforcement learning is usually limited to systems that use trial-and-error exploration of the state-action space leading to the discovery of good or promising actions, and delayed rewards that might be realized after several states have been explored [CWL18, SB18].

In the original competition, this setting is used by Larose [Lar11] based on work on multiple agents with individual learning [MKC10] in a diversification-intensification cycle. It has also been used by Di Gaspero et al. [DGU12], where an action is to select the Low-level Heuristic type. Different options are compared, however, the approach did not yield very good results, which the authors attribute to the limited state representation. The state-of-the-art approach uses Q-learning [CWL18] to select a combination of an LLH selection method and a move acceptance method, together with a simple state aggregation. They report very good results (only beaten by the winner of the original competition), closely followed by the new approach [MM22] using reinforcement learning with a discrete state representation based on four features to learn the application of individual LLHs and investigating the effect of several different components of the learning system. Apart from HyFlex, deep reinforcement learning has also recently been used for hyper-heuristics [ZBQ$^+$22]. In contrast to the existing work, our approach uses a rich state representation not previously used in literature, and an exploration policy with adaptive probabilities based on Iterated Local Search.

A recent survey [DKÖB20] provides an overview of current work, updated classification, and different application areas for hyper-heuristics. Newer approaches evaluated on the HyFlex domains that outperform the competition results were presented by some authors, including a population-based approach using Gene Expression Programming [SAKQ14], a hyper-heuristic based on Monte-Carlo tree search [SK15], a hyper-heuristic based on a hidden Markov model [KK15], and an iterated multi-stage hyper-heuristic [KÖ16]. Work using solution chains based on the Luby sequence [LSZ93] reported good results recently [Chu20]. The current state-of-the-art method [AOO21] is an Iterated Local Search method with a learning component based on Adaptive Thomson Sampling.

5.2.2 Using Hyper-Heuristics in the Framework

The existing HyFlex framework is a useful place for the implementation of new hyper-heuristics as well as the evaluation of existing ones since it has been in use for many years to benchmark different hyper-heuristics on the problem domains from CHeSC 2011, allowing continued comparison of different methods. However, HyFlex is written in Java,

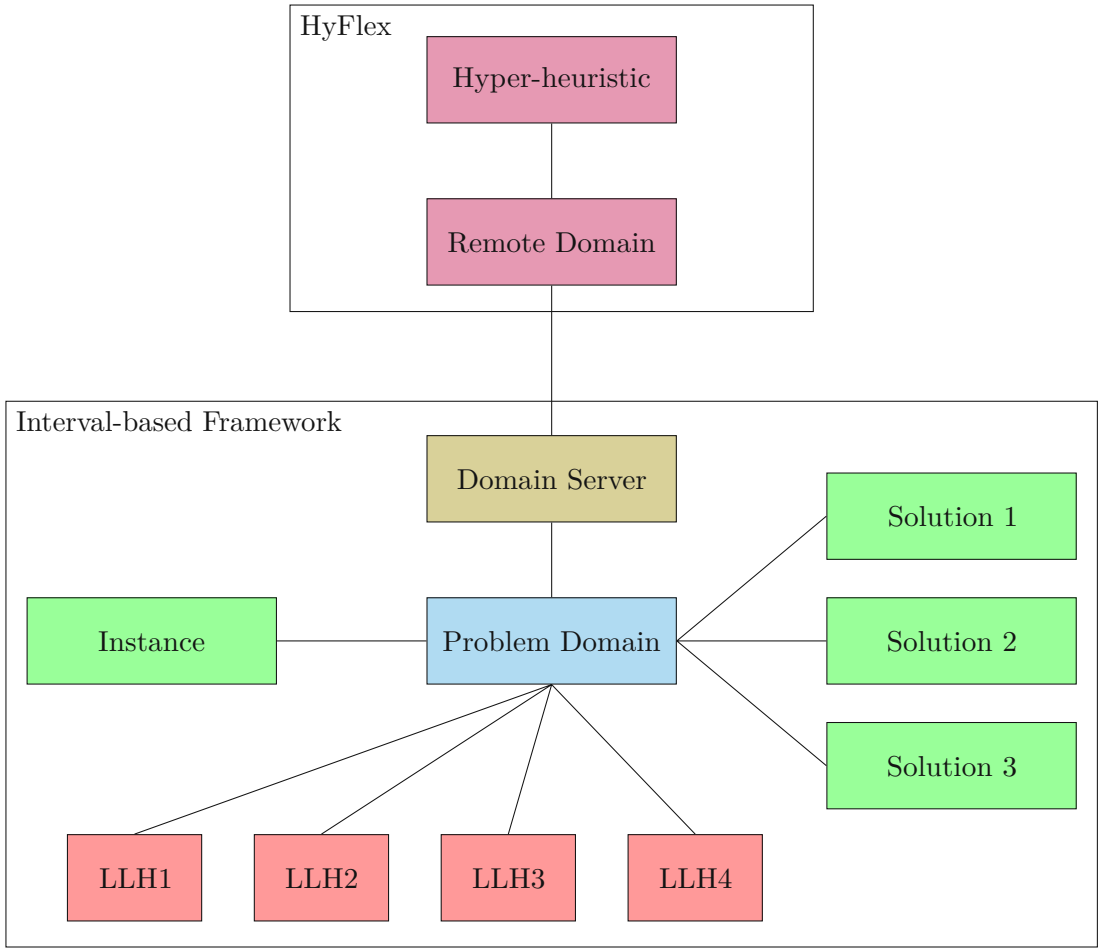


Figure 5.3: Connecting the framework with HyFlex

while the problem domains used for personnel scheduling in this thesis are implemented in the interval-based framework in Python.

Figure 5.3 shows the design of problem domains in the interval-based framework and the interface for the connection to HyFlex. In HyFlex a hyper-heuristic is applied to a problem domain. Instead of providing a specific domain, the *Remote Domain* allows to propagate each call from the hyper-heuristic to the domain to a remote implementation, in this case in the interval-based framework. Here, a *Domain Server* is started with the chosen problem domain, receives calls from the remote domain, passes them to the problem domain to handle the requested action, and returns the result. The problem domain manages the current instance to work on, the set of solutions for this instance, and the set of Low-level Heuristics that are available for this domain.

This remote connection is completely transparent to both the hyper-heuristic and the LLHs. A hyper-heuristic is applied the same way whether the domain is implemented

locally or remotely, additionally the transmission time for the remote connection is not counted towards the timeout. On the other side, the problem domain within the interval-based framework could be used by a locally implemented hyper-heuristic the same way it is used via the domain server. The remote interface for HyFlex is publicly available as part of a hyper-heuristics collection [MM21].

5.2.3 Low-Level Heuristics

For each of the problem domains we use three categories of Low-level Heuristics, which are improvement heuristics that are guaranteed to result in either better or same quality solutions, mutations which can change solution quality in any direction, and destroy-and-repair heuristics that are supposed to structurally change the solution. These categories are based on three of the four possible categories in the HyFlex framework. Since the first two of these are similarly implemented for all three domains, we describe the common heuristics first, and the used moves of the individual domains as well as the much more specific destroy-and-repair heuristics afterwards.

When LLHs are used in combination with some parameters, different versions of the LLHs are created and presented to the hyper-heuristics to allow them to choose those with the more suiting parameter settings. Some deviations to prevent LLHs that take too long to execute are mentioned later regarding the individual domains.

Improvement and Mutation Heuristics

Improvement heuristics are various versions of hill climbers. While each domain uses different moves, the algorithms in use are:

- First improvement with at most 1 or 10 steps to a better solution
- Best improvement with at most 1 or 10 steps to a better solution
- Random improvement with at most 1000 iterations (evaluations of a random neighbour) or 10 consecutive iterations without improvement

Mutations are based on two different sets of heuristics:

- Random improvement with metropolis acceptance criterion for mutations with a tendency to move towards improving or slightly worsening solutions, with at most 1000 iterations or 10 consecutive iterations without improvement and the following acceptance probability for worsening solutions (where Δ is the change in solution value):

$$p = \exp\left(\frac{-\Delta}{t}\right) \tag{5.1}$$

  Values for t are fixed to 1, 10, and 100.

- Random walk is used for mutations which typically significantly worsen the solution. Randomly chosen moves are applied 1 or 10 times in a row.

Specific Implementations for RWS

For the representation of this domain the shifts are already fixed to their columns in the solution matrix S corresponding to all workforce requirements already fulfilled, resulting in the following moves:

- `SimpleSwap`: Choose a column c and two rows r_1 and r_2, and exchange the corresponding two shifts. In the framework this corresponds to moving an interval a number of periods that is a whole multiple of the week length w since the whole schedule is represented as a one-dimensional sequence.
- `Swap`: Choose a column c, two rows r_1 and r_2, and a length $\ell \leq 7$, and perform a simple swap for ℓ consecutive cells starting from column c (moving to the next rows and first column when going past the last column). This exchanges up to a week of consecutive work assignments.

The moves are similar to those used previously [Mus06]. The initialization is more sophisticated than in previous heuristics for the problem. Instead of just randomly assigning shifts within each column, shifts are assigned according to the following algorithm: For each column, for each shift that needs to be assigned to the column, assign it to the row with the least increase in objective value. This already forms useful sequences of shifts during the construction process.

First and best improvement with the larger `Swap` neighbourhood are only used with a single step since they take longer to execute. For the mutations only the full `Swap` neighbourhood is used, since it includes the smaller `SimpleSwap` anyway.

Two new destroy-and-repair heuristics are used:

- `RemoveDay`: A random column c is chosen, all assignments for this day are removed and reassigned using the construction heuristic. The significance is that all rows are affected at once with this heuristic.
- `RemoveWeeks`: A random range of up to 10 consecutive weeks is chosen, all assignments from these weeks are removed and reassigned using the construction heuristic. This allows to reorganize larger consecutive blocks of shifts.

Specific Implementations for MSD

Three different new neighbourhoods are used for this domain:

- `NewShift`: A day d and shift type t are chosen. Then, a shift within the bounds set by the chosen shift type t is added to the schedule on day d.

- `RemoveShift`: A shift s is chosen and removed from the schedule.
- `ChangeShift`: A shift s is chosen and removed from the schedule. Then, a shift type t is chosen and a new shift within the bounds set by t is created and added to the schedule on day $d(s)$, the previously assigned day of shift s. This combines the previous two moves without the need to deal with a potential situation of low cover in between.

Compared to these moves, previous work [MSS04] uses a larger set of different moves more focused on shift classes, while we use fewer moves focused on individual shifts, to prevent dealing with an excessive number of LLHs.

Since still more moves than for the other domains are used, more options are available for the corresponding heuristics. `NewShift` and `RemoveShift` are used with 1 and 10 steps, the larger `ChangeShift` only with one step for first and best improvement. Additionally a best improvement heuristic with alternating `RemoveShift` and `NewShift` applications and at most 10 steps is used. It randomly starts with any of the two neighbourhoods. Further a version of this alternating search is included where only the combination of the two neighbourhoods needs to show an improvement in the solution value rather than each individual neighbourhood.

Due to the objective for reducing the number of different shift classes, it is often more beneficial to try to assign another shift to an already existing shift class instead of a new one. Therefore, when shifts are randomly generated for random improvement and random walk, the following procedure is applied: With 50% probability, an already existing shift class is randomly chosen and the corresponding shift returned, else any shift within the bounds of a randomly chosen shift type is generated. The different neighbourhoods are randomly selected with equal probabilities in each iteration of the randomized heuristics. The initialization heuristic uses random improvement with this setting for up to 10000 iterations or 100 iterations without improvement.

The new destroy-and-repair heuristics we propose for MSD combine two different remove operators with two different repair operators resulting in four heuristics. The remove operators are:

- `RemoveClass`: A shift class is randomly chosen and all shifts within this class are removed. This is supposed to help getting rid of potentially less useful shift classes.
- `RemoveDay`: All shifts on a particular day are removed, allowing to reschedule this day at once while leaving the other days untouched.

The repair operators are based on best improvement to focus on the best replacement shifts rather than introducing additional shift classes by using randomized repair operators:

- `RepairExisting`: Use only shifts from shift classes already in use to repair the schedule, filling the gaps as well as possible without introducing new shift classes.

- `RepairFull`: Use full best first search, setting the focus more on covering the demand.

Specific Implementations for BDS

The heuristic for the initial solution assigns bus legs to the shifts where they cause the least cost increase or to a new shift if cheaper. The following two moves based on the Simulated Annealing solution method are used for improvement and mutation heuristics:

- `Swap`: Choose one bus leg ℓ from one employee e_1 and a second employee e_2. Move ℓ from e_1 to e_2 and move back all bus legs in e_2 which now overlap with ℓ to e_1.
- `SequenceSwap`: Choose a consecutive sub-sequence $\ell_i, \ldots, \ell_j$ from one employee e_1 and a second employee e_2. Then move the whole sub-sequence from e_1 to e_2 and move back all bus legs in e_2 which now overlap with any bus leg $\ell_i, \ldots, \ell_j$ to e_1.

Since the `SequenceSwap` neighbourhood is rather large, first and best improvement are only used with a single step. The random improvement mutations are only used with `SequenceSwap` to prevent an excessive number of mutation heuristics.

The new destroy-and-repair heuristics focus on removing one or multiple employees from the solution and rebuilding it using the construction heuristic. They are particularly important as they are the only LLHs for this domain that can change the number of employees in the solution, which can be very important for solution quality:

- `RemoveEmployee`: An employee is chosen randomly and removed from the solution. All bus legs previously assigned to this employee are reassigned by the construction heuristic. An employee might be fully removed if it is possible to reassign all bus legs to other employees.
- `RemoveTour`: A tour is randomly chosen from the instance, all employees assigned to any part of this tour are removed from the solution. The tour might be arranged in a less fragmented way during reassignment.

5.2.4 Hyper-Heuristics Selection

We critically evaluate and compare ten state-of-the-art hyper-heuristics including approaches that provided very good results in the hyper-heuristic competition and available recent approaches that could be implemented by Mischek et al. [MM21]. All used hyper-heuristics are implemented in HyFlex. This ensures that they are applied under the same conditions.

- ALNS: Self-adaptive large neighbourhood search based on Laborie et al. [LG07] with alternating perturbation and reconstruction moves, learning weights for the LLHs based on their performance.

- BSW-ALNS: Bigram sliding window ALNS is a version of ALNS where the weights for the reconstruction moves are not learned independently, but based on the preceding perturbation move. The moving time windows based on Thomas et al. [TS18] are supposed to deal with the fact that different LLHs take different amounts of time by taking this into account when calculating the weights.
- SW-ALNS: Sliding window ALNS extends ALNS only with the time windows, but not the bigram extension.
- CH-BI (Bigram): Chuang [Chu20] describes several methods in his thesis based on solution chains. If any solution in the chain is better than the starting solution, it is accepted and the chain stopped, otherwise the whole chain is discarded. Chain lengths are chosen according to the Luby sequence [LSZ93]. The way to choose heuristics is different for the four variants in this comparison. For CH-BI the probability to select a heuristic depends on the previous heuristic in the chain and the number of times this pair of heuristics occurred in successful chains.
- CH-FR (Frequency): In this case the probability to choose a heuristic depends on the number of times it appeared in successful chains.
- CH-PR (Pruning): Heuristics with bad performance are pruned after a warm-up period, otherwise uniform random selection.
- CH-UN (Uniform): Here the heuristics are chosen according to a uniform random distribution.
- GIHH: This approach [MVDCVB12] was the winner of CHeSC 2011. It uses an adaptive dynamic heuristic set to monitor the performance of each heuristic to select heuristics in different phases, finds effective pairs of heuristics, and uses a threshold accepting method.
- L-GIHH (Lean GIHH): This hyper-heuristic was obtained by performing Accidental Complexity Analysis on GIHH [AN16], and was reported to provide similar or even slightly better performance while greatly reducing the complexity of GIHH.
- HAHA: Lehrbaum et al. [LM12] proposed a hyper-heuristic which switches between working on a single solution and a pool of solutions together with an adaptive strategy for selecting LLHs.

5.3 Evaluation of Hyper-Heuristics and Low-Level Heuristics

All problem domains and Low-level Heuristics were implemented in the interval-based framework in Python and run using PyPy 7.3.5 for adequate speed. The hyper-heuristics were implemented in the Java HyFlex framework and run using OpenJDK 8u292. All evaluations have been performed with one hour of runtime, which was also the maximum

Objective	ALNS	BSW-ALNS	SW-ALNS	CH-BI	CH-FR
F_1	62	50	67	82	79
F_2	64	59	67	78	83
F_3	68	57	68	82	80

Objective	CH-PR	CH-UN	GIHH	L-GIHH	HAHA
F_1	74	78	**98**	97	79
F_2	78	80	**99**	98	79
F_3	76	77	**100**	99	78

Table 5.1: Number of feasible solutions out of 100 per hyper-heuristic

runtime for compared methods except when stated otherwise. This includes both PyPy and Java execution, but not the transmission times of the interface to transparently be able to compare hyper-heuristics implemented in HyFlex or externally. The experiments were run on a computing cluster with Intel Xeon CPUs E5-2650 v4 (max. 2.90GHz, 12 physical cores, no hyperthreading), but each individual run was performed single-threaded. Each method was run on each instance 5 times to account for random variations. An extension of the result tables in this section containing the detailed results for all individual instances is available online[1]. The evaluation is done on the same publicly available benchmark instances that were used in the previous chapters to allow comparison with previous and future work.

5.3.1 Rotating Workforce Scheduling

This domain was evaluated on the original set of 20 real-life instances from the RWS benchmark set. While there are more randomly generated instances available, we compare to previous results for the optimization objectives presented in Section 2.4.5.

In contrast to the other domains this domain has a much stronger emphasis on feasibility. For several instances, it is already difficult to find a feasible solution. We used a constant weight of 100 for hard constraint violations and scaled the optimization objectives to be small enough to reliably receive feasible solutions. Table 5.1 shows the number of feasible results per method and objective. GIHH and L-GIHH obtain feasible results on all instances, with only few exceptions in some individual runs. The other methods, however, are not able to solve some of the more challenging instances. The versions of CH and HAHA perform in a similar way, ALNS can solve even fewer instances.

Looking at the optimization results, there is a strong division between easy and hard instances that was already visible in Chapter 2: For several instances, all methods reliably reach the optimum on every single run. E.g., for F_1, for 5 out of 20 instances every

[1] https://cdlab-artis.dbai.tuwien.ac.at/papers/hyper-heuristics-personnel/

Objective	Instance	MiniZinc	GIHH	L-GIHH	HAHA
F_1	9	34	**35**	**35**	34
F_1	11	4	**6**	**6**	5
F_1	12	**8**	7	**8**	7
F_1	15	-	**15**	13	10
F_1	18	**23**	21	20	22
F_1	19	24	**26**	23	18
F_1	20	-	32	26	**37**
F_2	9	**2**	3	3	3
F_2	10	**3**	4	4	5
F_2	11	**5**	6	6	13
F_2	12	**4**	5	5	5
F_2	13	**4**	5	5	6
F_2	15	-	**7**	8	-
F_2	16	**4**	5	6	6
F_2	17	**4**	5	5	6
F_2	18	**4**	6	7	8
F_2	19	28	**8**	11	23
F_2	20	-	**11**	15	25
F_3	9	**26522**	**26522**	26524	26524
F_3	10	**8772**	8784	8778	8778
F_3	11	21710	**20783**	**20783**	23596
F_3	12	**4836**	5243	**4836**	**4836**
F_3	15	-	**196806**	200887	209213
F_3	16	**16868**	16900	16888	16890
F_3	18	**84414**	87218	87218	87218
F_3	19	1456141	**1253063**	1267498	1339601
F_3	20	3855876	**3268396**	3295030	-

Table 5.2: Best result comparison for hard RWS instances

method obtained the optimum in each of the 5 runs, for 3 further instances at least one of the 5 runs reached the optimum for each method. On the other hand, for several of the hard instances previously no result was obtained in the timeout of one hour.

Table 5.2 shows a comparison of hyper-heuristic results with the MiniZinc solutions from Section 2.4.5 only on those instances where GIHH, L-GIHH and MiniZinc did not all find the optimum. Note that objective F_1 is a maximization objective, while F_2 and F_3 are minimization objectives. The results show that the hyper-heuristics can improve previous best known results for 12 out of 27 hard instances. In particular, solutions can be provided for all 5 instances which previously ran into timeout. On the other hand, for those instances where the optimum could not be reached, the distance to the optimum is

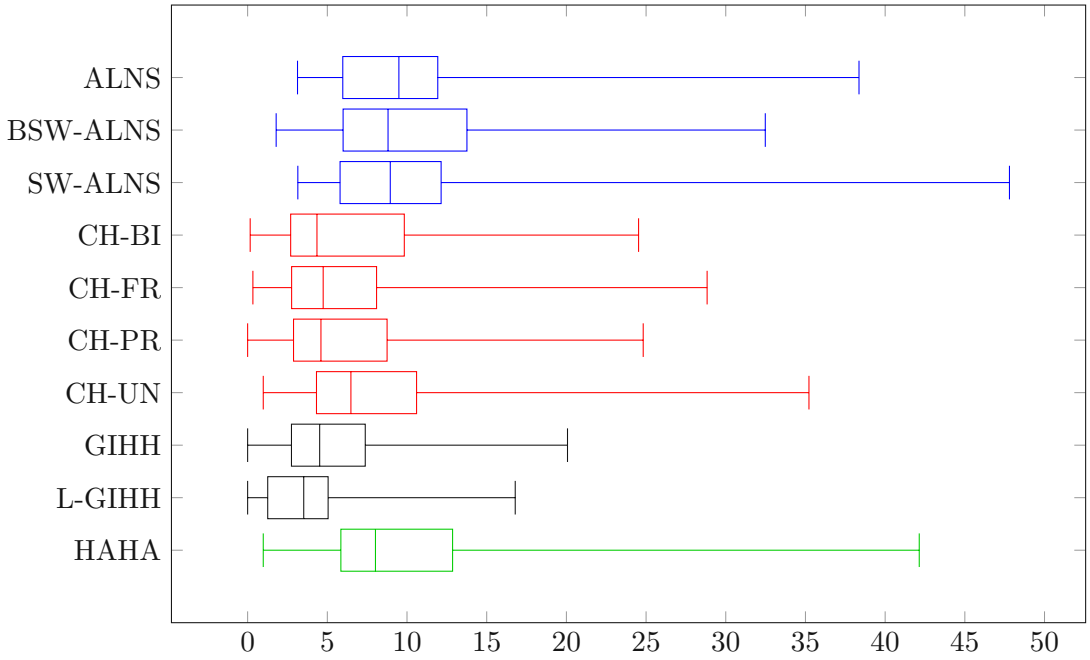


Figure 5.4: Best result deviations for reduced MSD

often very small.

In comparison regarding the optimization, GIHH is able to provide more best solutions than L-GIHH which is still the second best method. HAHA can provide one significantly better best known solution (F_1, 20), showing also for other instances that if it can provide feasible solutions, those are often of good quality, but those values are not reached reliably. While the other hyper-heuristics are less competitive, they can still improve some of the results compared to MiniZinc on the very difficult instances.

5.3.2 Minimum Shift Design

For this domain a large MSD benchmark set is available. In particular we use 93 instances in four data sets, but focus on sets 3 (30 realistic instances) and 4 (3 real-life or realistic instances), where demand and shifts will not align perfectly, since the first sets are artificially created to allow an exact cover which is unrealistic in practice.

However, the forth objective was only considered by one previous work [MSS04], but with evaluation only on the first data set. Furthermore, the weight of T_4 is set to 0 in the instance generator as soon as the objective interferes with the perfect cover. Therefore, either T_4 is not considered or does not have an effect anyway. In our evaluation, we use the bounds in the provided instances, but with a fixed weight of 1000 instead of the weight from the instances which is either 1000 or 0.

To have a baseline comparison, Figure 5.4 provides the deviation from the optimal results

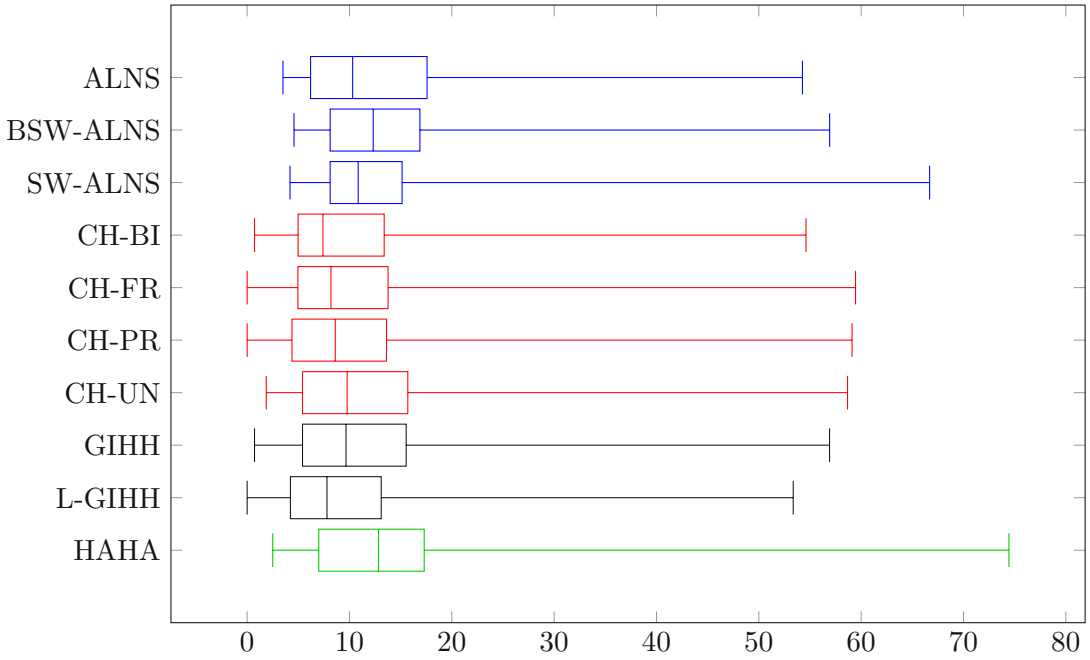


Figure 5.5: Best result deviations for full MSD

provided in Chapter 3 in percent for each hyper-heuristic on the realistic data sets 3 and 4 without using T_4. This provides two conclusions. First, it shows the differences in the hyper-heuristics. Here, L-GIHH clearly provides the best performance. In contrast to the other domains, however, GIHH shows a larger gap to L-GIHH. While regarding the worst deviation GIHH is still on the second place, the CH methods (except CH-UN) provide very good results in the average case and CH-BI outperforms GIHH in the median as well as upper and lower quartile. ALNS and HAHA perform significantly worse.

Second, the results show that especially L-GIHH can provide high quality results on those real-life instances, with the upper quartile at a gap of 5 %. While the exact method is clearly stronger on the reduced problem formulation, it would be much more difficult to adapt to the forth objective. For the hyper-heuristics setup, however, this is very easy to achieve and we can for the first time provide benchmark results on all instances with the forth objective enabled.

Figure 5.5 shows the deviations for MSD with the forth objective compared to the optimal solutions for reduced MSD. These are now only a lower bound that is not reachable for most instances, but it allows the same relative comparison which shows that the relative performance of the individual hyper-heuristics is very similar to before. However, this time the lowest median is actually achieved by CH-BI, while the best quartiles are still achieved by L-GIHH, but less distinct. It also shows that for a majority of instances, including this additional objective will raise the objective value by a few percent, while for a few instances it raises quite significantly.

Instance	SA	HC	BP	ALNS	BSW-ALNS	SW-ALNS
10	**14717.4**	14904.4	*14709.2*	14857.2	14814.2	14875.6
20	30860.6	30931.4	*30290.3*	30837.4	30775.4	30941.4
30	50947.4	51544.2	*49846.4*	51203.4	51018.4	51164.2
40	69119.8	69533.6	*67000.4*	69036.4	68825.2	69360.2
50	87013.2	**86718.6**	*84341.0*	87271.8	87326.8	87375.8
60	103967.6	103780.0	*99727.0*	103722.6	103429.6	103876.8
70	122753.6	122912.8	*118524.2*	122132.8	122349.0	122597.8
80	140482.4	139765.2	*134513.8*	139410.8	139477.0	139747.2
90	156385.0	156239.4	*150370.8*	155429.8	155115.8	155374.4
100	173524.0	172327.8	172582.2	171447.6	171609.2	172406.4

Instance	CH-BI	CH-FR	CH-PR	CH-UN	GIHH	L-GIHH	HAHA
10	14843.4	14838.8	14805.6	14742.0	14787.0	14773.6	14966.6
20	30773.8	30706.6	**30671.2**	30732.8	30731.6	30694.0	31222.4
30	50978.0	50946.6	50903.6	51002.2	**50765.8**	50854.2	51642.6
40	68909.2	**68583.4**	68847.6	68859.8	68639.6	68645.4	69866.8
50	87191.0	87091.2	87034.0	87019.8	86762.0	86729.8	88062.8
60	103444.6	103521.8	103464.8	103510.6	**103138.8**	103149.8	104434.4
70	122156.6	122247.2	122025.6	122092.2	121671.8	**121660.6**	122688.2
80	139902.2	139382.4	139209.2	139146.2	139123.0	**139041.6**	140504.4
90	155327.2	**154938.0**	154972.4	155282.8	155093.8	155113.2	156052.4
100	171350.2	171718.6	**171182.4**	171464.2	171278.2	171325.4	172318.0

Table 5.3: Best results for BSD with different solution methods

5.3.3 Bus Driver Scheduling

This domain was evaluated on the BDS benchmark set of 50 benchmark instances used in Chapter 4. They span 10 size categories from around 10 tours (70 legs) to around 100 tours (1000 legs) based on real-life demand distributions.

Table 5.3 shows the average of the best results per instance category for the hyper-heuristics in comparison to the previous best results (Simulated Annealing, Hill-climber, Branch and Price). The entry in bold highlights the best heuristic result (all methods except B&P), the value in italics the best overall result. All included hyper-heuristics can outperform previous heuristic results on at least 3 categories, with CH-FR, CH-PR, GIHH, and L-GIHH outperforming them in 8 categories.

Branch and Price provides near-optimal solutions for most instances except the largest ones in the data set. However, it has two major disadvantages. First, it has problems scaling to larger instances. Real-life instances are often even larger than the largest benchmark instances. However, even on the largest size of the given instances the hyper-heuristics provide better results than B&P, despite B&P using twice as much time for these instances. In fact, several methods improve the previous best known solutions for two out of the five instances in the largest category for this problem.

Method	BDS	F_1	F_2	F_3	MSD	MSD+T_4
ALNS	3	10	7	7	10	11
BSW-ALNS	4	9	6	5	16	16
SW-ALNS	5	11	6	7	16	18
CH-BI	4	12	11	11	28	32
CH-FR	9	13	13	8	25	30
CH-PR	11	12	13	8	25	38
CH-UN	5	12	12	9	18	19
GIHH	13	**17**	**20**	**16**	28	25
L-GIHH	**15**	16	15	15	**76**	**46**
HAHA	1	13	8	11	9	14

Table 5.4: Number of wins for each hyper-heuristic (best objective)

Method	BDS	F_1	F_2	F_3	MSD	MSD+T_4
ALNS	0	5	4	5	4	5
BSW-ALNS	3	5	3	4	5	5
SW-ALNS	4	6	4	4	5	5
CH-BI	6	10	5	6	8	25
CH-FR	5	10	5	6	12	15
CH-PR	2	10	6	6	12	30
CH-UN	4	11	6	6	12	12
GIHH	16	**18**	**19**	**15**	13	16
L-GIHH	**19**	13	13	10	**88**	**42**
HAHA	0	9	5	10	5	5

Table 5.5: Number of wins for each hyper-heuristic (average objective)

Therefore, hyper-heuristics can be considered the new state-of-the-art method for very large instances. Second, compared to the rule-specific implementation of the sub-problem in B&P, hyper-heuristics can be adapted to different rule-sets easily, which is often important in practice.

5.3.4 Comparison of Hyper-Heuristics

Table 5.4 shows the number of times a hyper-heuristic obtained the best result out of all 10 hyper-heuristics in comparison on an instance (ties are awarded to all hyper-heuristics involved in the tie). Table 5.5 repeats this competition for the averages over the five runs on each instance, to make sure that a hyper-heuristic is not overvalued based on some lucky best results, but poor average results.

This final comparison sums up the picture from the previous sections regarding the relative performances of the hyper-heuristics: L-GIHH is the best heuristic on two out

of the three domains, especially clearly in MSD, and only beaten by the original GIHH on the RWS domain. The CH versions (except CH-UN) perform well on all domains, sometimes even outperforming GIHH. HAHA shows reasonably good performance for RWS, but struggles with the other domains. The picture is fairly similar regarding the comparison of best or average values.

The versions of ALNS did not perform well. While several options were tried before the comparison for its parameters (5 minutes warmup, 1 minute window length), performance might increase with more careful tuning. However, no tuning was required for the other hyper-heuristics. Note that even though the LLHs have several parameters as well, different options of LLHs with different parameter settings were successfully used across all three domains to prevent the need for detailed tuning.

While the quality of the best results seems to depend on the set of available LLHs, the winning hyper-heuristics show to be very efficient at selecting a good mix relative to their competitors in different scenarios (even in filtering out ill-behaving LLHs when a bug occurred for one of the domains). We tracked the number of heuristic applications for each heuristic and each run and found some interesting patterns: First, different hyper-heuristics show several different LLH preferences. Second, the same hyper-heuristic usually favours similar LLHs for different instances of the same domain, but some heuristics are actually swapped depending on the instance.

Specifically looking at the destroy-and-repair heuristics shows that for most of the instances and hyper-heuristics in the different domains, the destroy-and-repair heuristics are frequently used. Further, some additional experiments showed that without any destroy-and-repair heuristics, e.g., on BDS the winning method L-GIHH looses its ability to outperform the results from literature on all but one instance, clearly showing that these are relevant to the performance. They seem similarly important for MSD, while less important for RWS.

Since high-quality results have been achieved on different domains, and there is a clear result regarding the winning hyper-heuristics on our scheduling domains, these will be deployed by our industrial partner to improve their current scheduling systems, especially in the context of new and changing requirements.

5.3.5 Conclusion

In this section, we provided a major study on using various hyper-heuristics to solve different domains in personnel scheduling which include complex constraints and combined optimization goals from practical applications. We provided new Low-level Heuristics for three different scheduling domains used by our industry partner, applying hyper-heuristics to them for the first time. We compared several versions of state-of-the-art hyper-heuristics and provided multiple comparisons for the relative performance of the hyper-heuristics on the individual domains. The results showed that especially Lean GIHH, but also the original GIHH are very well suited to efficiently solve the different scheduling domains.

With these methods we were able to provide several advances for the individual domains. For RWS, we could provide several new best known solutions for the different optimization goals including several solutions for instances previously running into timeouts without a solution. For MSD, for the first time we provided comprehensive results using the forth objective that has high practical relevance, hopefully fuelling more research into this more realistic version of the problem. For BDS, we were able to outperform previous heuristic approaches and find two new best known solutions for the benchmark data set.

In practice, these results are very useful in many ways. Besides the specific improved solutions, the results show that using hyper-heuristics for complex practical problems is a very useful approach to provide high-quality solutions without spending too much time for highly problem-specific solution methods. In real-life applications there are often different variations of these problems, frequently including customer-specific constraints or adapted optimization goals. However, the LLHs are independent from the additional constraints or different objectives due to the architecture of our interval-based framework, so they do not need to be adapted to different customers, allowing rapid adaptation for changing requirements.

5.4 A New Hyper-Heuristic Using Reinforcement Learning on a Large State

Often hyper-heuristics try to find certain combinations of Low-level Heuristics that are beneficial to apply, e.g., a combination of a mutation followed by a local search. However, a combination that is beneficial in the early part of the search might not work well in a later part of the search. E.g., in the early part it might be easy to find an improvement anyway, while in later parts of the search it might be difficult to escape local optima. On the other hand, the solution value and the execution time are the only information the hyper-heuristic gets back from the LLH. This is important to stay domain-independent.

Still, the trajectory of solution values can help to draw conclusions about the current state of the search. Therefore, we propose to use this information to learn efficient applications of Low-level Heuristics and introduce a new set of 15 features to characterize the current state of the search. Then we use these features for the novel hyper-heuristic LAST-RL (**La**rge **St**ate **R**einforcement **L**earning) based on reinforcement learning using the linear state-action value function approximation method tile coding, the learning method SARSA-lambda, and an epsilon-greedy policy that uses probability distributions based on Iterated Local Search (ILS) for increased performance. In order to keep the approach as general as possible, it is developed on the combined set of instances from six different domains from the Cross Domain Heuristic Search Challenge 2011 (CHeSC 2011), where it can provide good results compared to other recent approaches. Then the new hyper-heuristic is applied to the three real-life scheduling domains in this thesis, providing very good results and several further improvements compared to the evaluation provided earlier in this chapter.

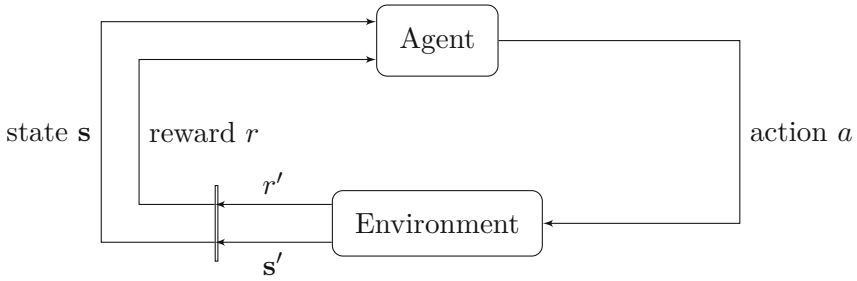


Figure 5.6: Core loop of reinforcement learning [SB18]

5.4.1 Large State Reinforcement Learning (LAST-RL)

Reinforcement Learning is an iterative process based on the concept of states, actions, and rewards. Figure 5.6 shows the core feedback loop. The system is in a current state $\mathbf{s}$, in our case the search state. The learning agent chooses an action $a \in \mathbf{A}$ based on a notion of value for the current state and the expected state after executing this action. In our case, each Low-level Heuristic corresponds to an action. Then the learning agent L receives a reward r for the action a it chose and the system transfers to the new state $\mathbf{s}'$. The agent wants to maximize the rewards it collects and updates its value estimations based on the reward it got.

While the set of actions is clear, many decisions need to be made regarding the other aspects of a reinforcement learning system. The remainder of this chapter will describe the novel hyper-heuristic LAST-RL (**La**rge **St**ate **R**einforcement **L**earning), with the core learning loop as shown in Algorithm 5.1.

As an overview, initialization is done in lines 1 to 4. The main loop in lines 5 to 24 is performed until timeout T. Line 6 defines the length c of the next episode (Subsection Solution Chains). The main part of reinforcement learning is executed in the inner loop in lines 7 to 19 (Section 5.4.2). The steps deal with computing the value function approximation x (line 8), choosing the action according to the policy π (line 9, Section 5.4.4), executing the LLH resulting in the change in solution value Δ and spent runtime t (line 10), calculating the reward r (line 11), updating the state to $\mathbf{s}'$ (line 12, Section 5.4.3), using the learning agent to calculate the new weight vector (line 13), and moving the current state to the new state (line 14). Lines 15 to 18 update different notions of a best solution (overall and best since the last reset) if the current solution is better, which ends the current episode. Line 20 sets the current solution to the best one since the last reset and lines 21 to 24 perform a reset if the conditions are met (Subsection Reset Behaviour).

Initialization

Going into more details, first, the problem domain is initialized for the given instance. The initial state is set up as well as the feature weights $\mathbf{w}$, which are initially set to 0.

Algorithm 5.1: LAST-RL learning loop

Input: A state-action feature function x, a policy π, a reward function R, a learning agent L, a chain length provider *chain*, and a restart criterion *restart*

Data: An instance I, an action set $\mathbf{A}$, and a timeout T

```
1  initialize(I);
2  s ← initialState;
3  w ← 0⃗;
4  S ← initializeSolutions();
5  while time < T do
6      c ← chain();
7      for i ← 1 to c do
8          x′ ← x(s, A);
9          a ← π(x′, A, w);
10         Δ, t ← executeLLH(a, S);
11         r ← R(Δ, t);
12         s′ ← updateState(s, a, Δ, t);
13         w ← L(s, a, r, s′, w);
14         s ← s′;
15         if newBest(S) then
16             updateBest(S);
17             break;
18         end
19     end
20     gotoBest(S);
21     if restart(S, time) then
22         reset(s, S);
23     end
24 end
```

Hyper-heuristics can work on a single solution or on multiple solutions. In our approach, we mainly work on a single solution, but still need to keep several solutions in $\mathbf{S}$:

- The global best solution found so far for this instance
- The best solution found so far since the previous reset
- The current working solution
- A set of solutions for use with the crossovers similar to GIHH [MVDCVB12]. The idea is to preserve at least some diversity for the application of crossover LLHs despite the use of a single-solution hyper-heuristic by keeping a set of 5 additional solutions, and replacing a random one each time a new best solution since the

previous reset is found. Each crossover is then applied on the current working solution and a random solution from this crossover pool.

At the start the construction heuristic provided by the domain is executed 10 times, and the best result is set for all solutions in $\mathbf{S}$.

Solution Chains

The main loop is executed until the timeout T is reached. In general a reinforcement learning problem is either episodic, or continuous. In our case, we use episodes which end when either a new best solution since the last reset is found, or a certain number of heuristic applications have passed. After each episode, the working solution is changed back to the best solution found since the last reset.

The decision how to choose the lengths of these chains, represented by the function `chain`, is a very critical one. If chains are too short, promising sequences of heuristics are cut off before reaching a new best solution. However, if chains are too long, too much time is wasted following sequences of heuristics that will not lead to any improvements.

The concept of solution chains was investigated by Chuang [Chu20], where the Luby sequence [LSZ93] was proven to be optimal regarding the expected number of operations until an improvement is found, given that the probability $q(\ell)$ of finding an improvement within at most ℓ heuristic applications is unknown. The Luby sequence $\mathcal{L}$ is defined as follows for $j = 1, 2, \ldots$:

$$\mathcal{L}(j) = \begin{cases} 2^{k-1} & \text{if } j = 2^k - 1 \\ \mathcal{L}(j - 2^{k-1} + 1) & \text{if } 2^{k-1} \leq j < 2^k - 1 \end{cases} \tag{5.2}$$

$$\mathcal{L} = 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, \ldots \tag{5.3}$$

Equation (5.2) shows the recursive definition of the Luby sequence, Equation (5.3) shows the first few values of the sequence. Most of the resulting numbers are very small, but with exponentially increasing distance to the start exponentially larger numbers occur in the sequences. In Algorithm 5.1, each call to `chain` in line 6 returns the next number from this sequence, starting with $j = 1$ at the first call.

The good results achieved using the Luby sequence in the work by Chuang [Chu20] were further confirmed recently [MM22], and indeed the sequence also showed very good results in our approach. However, there are two adaptations used in our work based on the fact that we sometimes do have some insight regarding the probability distribution $q(\ell)$.

The first novel adaptation is based on the fact that predominantly the hyper-heuristic will use cycles of diversification and intensification, as explained in the policy section later (Section 5.4.4). Therefore, the solution value is expected to first increase, followed by a decrease using local search heuristics. This lead to the adaptation of line 7, continuing the inner loop beyond c in the following cases:

- The previous heuristic was not a local search heuristic: In this case the diversification is not yet done, so the hyper-heuristic should at least try the intensification to see whether the chain might be successful.
- The previous heuristic lead to an objective improvement: In this case the intensification is currently progressing well, so it should be continued until either getting to a new best solution or not making progress any more.

In any case, however, the chain is aborted if the length reaches $2 \cdot c$ to prevent going too far beyond the expected length. This is necessary as the cycles of diversification and intensification are not guaranteed, the hyper-heuristic might also learn to just repeatedly use mutations until the end of the chain.

The second novel adaptation is to use information collected from previous chains, more precisely to use the average length $\bar{\ell}$ of successful chains (chains leading to a new best solution since the last reset). Therefore, c is actually obtained by multiplying the value returned from the Luby sequence with $\max\{\bar{\ell}/2;\ 1\}$. Successful chains end immediately when a new best solution is found, leading to them often being shorter than the current maximum chain length c, this together with using only half the average prevents a feedback loop forcing chains to become excessively long. However, this approach allows to use longer chains for problems where longer chains are successful more often, and therefore the frequent very short chains in the Luby sequence would be unlikely to succeed.

Reset Behaviour

At the end of each solution chain, the hyper-heuristic changes the active solution to the best solution found since the last reset in line 20 of Algorithm 5.1. However, sometimes the search might not find any improvements any more for a longer period of time. In a practical application this might be used to stop the search completely, allowing faster execution on smaller instances. For benchmarking of hyper-heuristics, however, a fixed runtime is typically used. Therefore, a reset of all solutions except the global best is performed if any of the two following criteria are met, as checked by `restart` in line 21:

- 1000 consecutive chains without improvement
- At least 100 consecutive chains without improvement and at least one tenth of the total runtime since the last reset

The reason there are two options is the vast difference in runtime of the individual Low-level Heuristics between different domains. Resetting too early might cut off good parts of the search space that are difficult to discover, while resetting too late means potentially spending too much time in local optima that are hard to escape. In any case, for the last 20 % of the runtime, the global best solution is set as the active solution in order to promote further intensification on the best attempt among the different resets.

If there would be a regular reset within the last 30 % of the runtime, this change to the global best is started immediately since not much time would be available between the reset and the final phase anyway.

While the solutions are reset to a new initial solution by `reset` in line 22, the weights for reinforcement learning are kept, potentially allowing much faster improvements after the reset. The additional solutions for crossover heuristics are not kept since a strength of the resets is additional diversification of the solutions, while keeping solutions for crossover would reduce this benefit.

5.4.2 Reinforcement Learning Components

In each inner loop (lines 7 to 19) in Algorithm 5.1, first the chosen function approximation is calculated from the state $\mathbf{s}$ and the action set $\mathbf{A}$ by the function x in line 8. The resulting vector $\mathbf{x}'$ is then linearly weighted by the weight vector $\mathbf{w}$ and used to compute an approximate value for each combination of the current state $\mathbf{s}$ and an action a from the action set $\mathbf{A}$. The policy π uses these values to chose an action a to execute in line 9.

This execution results in a change Δ in the solution objective, and a used runtime t in line 10. These values are used to calculate a reward r by using the function R in line 11. Further the new state $\mathbf{s}'$ is calculated as the result of the action execution in line 12.

Finally the learning agent L uses the data obtained from the previous state $\mathbf{s}$, the action a, the reward r, and the new state $\mathbf{s}'$ to update the weight vector $\mathbf{w}$ in line 13. The system moves to the new state in line 14, updates the solutions in case a new best solution since the last reset was obtained in lines 15 to 18, and the next loop continues.

Value Function Approximations

One of the important questions is how the learning agent estimates a value for a given state, or as in our case, for a given combination of state and action. Since we want to use a rich state representation, we operate in an infinite feature space due to the real-valued features, and we are very unlikely to ever revisit exactly the same state again. Therefore, tabular methods cannot be used, and a value function approximation is needed. This way, observations for a state-action pair generalize to nearby states.

In principle, any regression method could be used for learning the value function. However, the learning process is repeated up to around a million times when running the hyper-heuristic, and the time used for learning is missing from the time available for Low-level Heuristic execution. Therefore, we focused on linear value function approximations to try to balance learning and runtime considerations.

The candidates we considered for value function approximations in line 8 of Algorithm 5.1 always follow the same overall procedure, starting from the feature vector $\mathbf{s}$ of dimension 15 and action $a \in \mathbf{A}$, and transforming it to a feature vector $\mathbf{x}'$ of dimension $d \cdot |\mathbf{A}|$, where d depends on the chosen approximation method:

- Normalize the feature vector $\mathbf{s}$ according to the bounds for each feature, resulting in $\hat{\mathbf{s}}$.
- Calculate $\mathbf{x}$ of dimension d from $\hat{\mathbf{s}}$ based on the chosen approximation method.
- Compute the state-action features from the state features as a cross-product of the state features $\mathbf{x}$ and the action set $\mathbf{A}$, resulting in $\mathbf{x}'$ of dimension $d \cdot |\mathbf{A}|$, which repeats the state features for each action.
- Use $\mathbf{x}'$ weighted by the vector $\mathbf{w}$ of the same dimension for the value approximation.

Four different options were considered for the transformation from $\hat{\mathbf{s}}$ to $\mathbf{x}$.

Linear. This is the most simple approximation, setting $\mathbf{x} = \hat{\mathbf{s}}$. It is very fast, however, due to its simple nature, it is not able to learn more complex interactions between the features, making it unsuitable for achieving good results.

Fourier Basis. This approximation is based on the well-known Fourier series [Tol12] and uses it to calculate the following features:

$$x_i(\hat{\mathbf{s}}) = \cos(\pi \hat{\mathbf{s}}^\top \mathbf{c}^i) \tag{5.4}$$

Here, $\mathbf{c}^i = (c_1^i, \ldots, c_k^i)^\top$, with $c_j^i \in \{0, \ldots, n\}$ for $j = 1, \ldots, k$ and $i = 1, \ldots, (n+1)^k$. $k = 15$ is the dimension of $\mathbf{s}$, and n is chosen according to the desired approximation quality. However, this produces $(n+1)^k$ possible features, which is much too large for our dimension k. Therefore, we tried using n only up to four, and limit the number of non-zero components z in each $\mathbf{c}^i$ by at most four as well, reducing the number of features to a manageable range, however, still without reaching good results.

Radial Basis Function. The idea of this approximation is to define a set of center states $\mathbf{c}_i \in C$, and then calculate feature i by the distance of state $\hat{\mathbf{s}}$ to $\mathbf{c}_i$ according to a Gaussian curve:

$$x_i(\hat{\mathbf{s}}) = \exp\left(-\left(\frac{||\hat{\mathbf{s}} - \mathbf{c}_i||}{\sigma}\right)^2\right) \tag{5.5}$$

The distance metric in use is the Euclidean distance, the parameter σ can be set to change how the strength of the features degrades with increasing distance to the center. The number of features can directly be chosen for this approximation, however, the high number of dimensions makes it difficult to find a useful balance between a suitably large number of points to cover the high-dimensional space well enough while still not loosing too much efficiency.

Tile Coding. Here, the state space is is partitioned into portions of equal size, called tiles. One such tiling would be just a case of state aggregation. However, this leads to rather arbitrary cuts on the edges of the tiles. Instead, tile coding uses multiple such tilings, each with a different offset. This allows to create a smooth representation of the state space. In our case, the offsets of the different tilings are created with a random jitter.

Tile coding produces one feature for each tile in each of the tilings. However, for each tiling, only one feature is active for any state (features are binary), and many tiles are never active during the whole learning process. This allows a sparse representation that only stores active features, and therefore to use many more features than with the dense representations above while still staying very efficient. Therefore, we ended up using this approximation method, setting the parameters to 10 tilings (n_t) with 3 tiles per dimension (resolution r_t) based on initial experiments.

Rewards

In any case, for the current state $\mathbf{s}$ a certain action a has been chosen. In the context of our hyper-heuristic, the corresponding Low-level Heuristic is executed in a certain runtime t and results in a change in objective Δ. Now the question is how to present a reward for this performance to the reinforcement learning algorithm using function R in line 11 of Algorithm 5.1. Since we use an episodic formulation, a reward is only created at the end of each solution chain.

A natural choice might be to present the change in solution value as a reward (negated since the objective for the hyper-heuristic is to minimize, while the learning agent wants to maximize). However, this does not seem to work too well. Magnitudes of these changes can be very different, and even for repeated resets on the same instance the different initial solutions might lead to very different margins to near-optimal areas of the search space. Further, typically larger improvements are made at the start of the search, where several methods can easily lead to large changes, while the more difficult part later in the search produces much smaller improvements. In fact, improvements later in the search should be valued higher since they are harder to find. Therefore, the reward for a new best solution is set to $\sqrt{maxFC}$, where $maxFC$ is the maximum length of consecutive unsuccessful chains since the last reset. This value serves as a measure of difficulty to escape the current local optimum.

Another natural idea would be to incorporate the runtime: Since the total runtime for the hyper-heuristic is limited, wasting runtime is a negative aspect and should be penalized. However, putting too much weight on this aspect can easily lead to problems since every heuristic execution takes time, generating a negative reward, while even good combinations of heuristics only provide a positive reward with some probability.

For the CHeSC 2011 domains that were used to design and test the hyper-heuristic, most LLHs on each domain show similar behaviour regarding runtime, therefore no advantage could be found using runtime as part of the reward function. However, this greatly

changed in the final evaluation on the real-life domains, therefore adding runtime to the reward function was the only adaptation that was done to fit the hyper-heuristic better for the application on our real-life domains. The details are explained in Section 5.6.

Based on experiments leading to the conclusions presented above, the following reward r was chosen to be applied at the end of each solution chain of length c' with total execution time t':

$$isNewBest = \begin{cases} 1 & \text{if new best solution found} \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

$$r = isNewBest \cdot \sqrt{maxFC} - \frac{100 \cdot t'}{T \cdot c'} \tag{5.7}$$

Equation (5.6) distinguishes whether the chain ended due to a new best solution or reverting to the previous best, Equation (5.7) combines the positive reward for new solutions with the penalty for runtime, which is relative to the total runtime T and the chain length c'.

At the end what matters is to reach new best solutions in short time, therefore the reward is always 0 until the end of the solution chain to allow arbitrary intermediate changes to the solution, while the evaluation using the reward is applied at the end of the chain.

Learning Agents

At the end of each inner loop, the question is, given the series of states, actions, rewards, and updated states, how to update the weight vector $\mathbf{w}$ to reflect the new knowledge in the value function approximation with the new weight vector $\mathbf{w}'$, using learning agent L in line 13 of Algorithm 5.1. Two different learning agents were investigated, both require the value function approximation to be differentiable as they are based on gradient descent. $\hat{q}(\mathbf{s}, a, \mathbf{w})$ denotes the value the learning agent estimates for performing action a in state $\mathbf{s}$. This value is calculated using the state-action feature function x to obtain the state-action feature vector $\mathbf{x}'$, and calculating $\mathbf{x}'^{\top} \cdot \mathbf{w}$. Both methods use a learning rate α, were high values result in recent rewards overwriting previous knowledge faster.

Gradient Descent Q-Learning. For this learning method the following update rules are used:

$$\delta = r + \gamma \cdot \max_{a' \in \mathbf{A}} \hat{q}(\mathbf{s}', a', \mathbf{w}) - \hat{q}(\mathbf{s}, a, \mathbf{w}) \tag{5.8}$$

$$\mathbf{w}' = \mathbf{w} + \alpha\delta \cdot \nabla\hat{q}(\mathbf{s}, a, \mathbf{w}) \tag{5.9}$$

Equation (5.8) calculates δ as the sum of the reward and the difference of the estimated value of the new state discounted by discount factor γ and the estimated value of the previous state. Equation (5.9) uses this value to update the weights using the learning rate α and the gradient of the $\hat{q}$ function, which is simply $\mathbf{x}'$ due to using linear function approximation.

Gradient Descent SARSA-Lambda. This method differs from Q-learning by using the policy to obtain the action to use for calculating the estimated value for the next state (SARSA stands for state, action, reward, state, action), and by using an eligibility vector $\mathbf{e}$ that accumulates the information of the previous steps of the learning agent for improved learning of not only actions that directly lead to rewards, but longer sequences of actions that lead to a reward. The updates are done as follows:

$$a' = \pi(x(\mathbf{s}', \mathbf{A}), \mathbf{A}, \mathbf{w}) \tag{5.10}$$

$$\delta = r + \gamma \cdot \hat{q}(\mathbf{s}', a', \mathbf{w}) - \hat{q}(\mathbf{s}, a, \mathbf{w}) \tag{5.11}$$

$$\mathbf{e}' = \lambda \cdot \mathbf{e} + \nabla \hat{q}(\mathbf{s}, a, \mathbf{w}) \tag{5.12}$$

$$\mathbf{w}' = \mathbf{w} + \alpha\delta \cdot \mathbf{e}' \tag{5.13}$$

Equation (5.11) calculates δ using the next action obtained by the current policy in Equation (5.10). Equation (5.12) updates the eligibility vector $\mathbf{e}$ using the decay factor λ, which is an additional parameter for this learning agent. This update uses the derivative that was directly applied to the weight vector in Q-learning. The update to $\mathbf{w}$ is instead done via $\mathbf{e}'$ in Equation (5.13).

This approach showed to be superior to Q-learning in preliminary experiments and was further used with the parameters $\alpha = 0.001$, $\gamma = 0.8$, and $\lambda = 0.8$.

5.4.3 Search-state Features

Still missing from the full picture for LAST-RL is the definition of the state features. The state is initialized in line 2 and updated in line 12 of Algorithm 5.1. The main issue of finding a suitable representation of the search state for hyper-heuristics is that not much information is available in the first place. The hyper-heuristic knows the number of Low-level Heuristics and the time limit. From each execution it gets the new solution value and the time the LLH took to execute. In previous literature, the approaches either used a very simple state representation with only few discrete states, or even just a single state. Mischek et al. [MM22] use four integer features, where the first two are the same that we use as well. Regarding the other two, while on different scales, tail length has similarities to our `chainProgress`, and recent improvement captures information similar to `lastImprMag`.

However, the limited information that is available still allows the hyper-heuristic to know where in the search progress it is by the time and the trajectory of the objective values, and it allows to build a picture of the current state of the search by looking at, e.g., the number of heuristic applications without any improvement and similar features. Therefore, in total we introduce the following novel set of 15 features to characterize the search state. We present the definition as well as lower and upper bounds, since the features are normalized to be used for our value function approximation. We also highlight values that take very different values for different domains, making it challenging to find a reasonable normalization without domain knowledge. Some of the following

expressions use the following definition of the magnitude of a real value x based on the natural logarithm that works both for positive and negative values and transitions gracefully at $x = 0$:

$$\text{mag}(x) = \text{sgn}(x) \cdot \ln(1 + |x|) \tag{5.14}$$

- `lastHeur`: The index of the last heuristic that was applied. Range: Integer from -1 (first application or reset to best known solution) to $numHeur - 1$ (where $numHeur$ is the total number of available heuristics).
- `lastType`: The type of the last heuristic. Range: Integer from -1 (first application or reset to best known solution) to 3, as the four categories ruin-recreate, crossover, mutation, and local search are available.
- `lastChangeSign`: The sign of the last objective change. Range: Integer in $\{-1, 0, 1\}$, for a coarse distinction of the current direction of the objective.
- `lastChangeMag`: The magnitude of the change in solution value from the last application $\text{mag}(\Delta)$. Range: $-\text{mag}(initial)$ to $\text{mag}(initial)$ (where $initial$ is the value of the initial solution). Ranges for solution values differ by orders of magnitude between different domains. Changes in relation to the initial value can still be very different between domains, but the focus on the magnitude of the change and the normalization based on the initial solution value can reduce these variations to a more manageable range.
- `chainProgress`: The percentage of the current solution chain that has been processed so far. Range: [0; 1]. A solution chain is a sequence of solutions that explores the search space from some initial solution. In our approach, either a new best solution is found until the end of the chain, or the solution is reset to the best solution found so far. The length of this chain is pre-determined by the hyper-heuristic, therefore the progress in the current chain can be used as a feature.
- `lastImprMag`: The magnitude $\text{mag}(hL)$ of the number of heuristic applications since the last improvement hL. Range: 0 to 5. This feature captures how long it took since the last improvement as a measure of whether the current state is in a local optimum that is difficult to escape. Note that the magnitude of the number of heuristic applications can vary greatly since heuristic applications can take very different times for different domains.
- `stepsMag`: The magnitude $\text{mag}(hT)$ of the total number of LLH applications since the last reset hT. Range: 0 to 5. Like for the previous feature, the actual values can also vary greatly between different domains.
- `time`: The total runtime since the last reset. Range: 0 to T. This feature measures time progress intended to help differentiate between early parts of the search where fast and easy improvements can be expected, and later parts where it is much more difficult to escape from local optima.

- `relativeImprMag`: The magnitude of the relation between the objectives of the current solution and the initial solution mag($cur/initial$). Range: 0 to 2. Usually the current solution is expected to be better than the initial one, but during diversification operations higher values might also be reached, leading to the larger maximum value.

- `relativeBestMag`: The magnitude of the relation between the objectives of the current solution and the best solution mag($cur/best$) where *best* is the objective value of the best solution since the last reset. Range: 0 to 2. Moving too far from the best solution might result in a state where there is no good chance to get back to the part of the search space with high-quality solutions, while sticking too close to the best known solution might limit exploration of the search space.

- `relImpr`: The relative number of improving heuristic applications hI/hT. Range: 0 to 1. This feature is intended to indicate whether there are many improving heuristic options or only few.

- `rel0`: The relative number of heuristic applications that did not change the objective ($h0/hT$). Range: 0 to 1. This feature might help to identify situations where heuristics have a hard time changing the solution (e.g., local search heuristics in a local optimum).

The previous features always capture either a current change from the last heuristic application or a collective value for the whole search since the last reset. In contrast, the last three features capture values only for a recent part of the search, allowing to put more focus on the recent history compared to the whole history. These features each use the horizon $H = 10$ for the number of heuristic applications that are taken into account.

- `avgChangeHMag`: The magnitude of the average change $\bar{\Delta}$ in the last H heuristic applications mag($\bar{\Delta}$). Range: $-\text{mag}(initial)$ to mag($initial$).

- `relImprH`: The relative number of improvements in the last H heuristic applications. Range: 0 to 1.

- `rel0H`: The relative number of heuristic applications that did not change the objective in the last H heuristic applications. Range: 0 to 1.

In contrast to the other features, the first three features are integer. Therefore, we do not apply tile coding to these dimensions of the feature vector, but keep the discrete values, as these are very important features for chains of LLHs where no approximation is needed. All other features are continuous and are normalized to the interval [0; 1] according to their lower and upper bounds. However, values above or below these bounds are not truncated, but result in normalized feature values outside of the interval [0; 1]. This should not occur often, but allows to distinguish such outliers from the regular values at the cost of introducing some additional tiles when using tile coding.

5.4.4 Combining Reinforcement Learning with Iterated Local Search

The final missing decision is how to arrive at selecting an action from the given state and the state-action value function approximation. This is done by the policy π in line 9 of Algorithm 5.1. A policy should favour actions with higher value for exploitation, but should also return all possible actions with a non-zero probability to allow exploration.

A classical policy frequently used in reinforcement learning is epsilon-greedy. The basic idea is simple, the action with the highest current state-action value is chosen with probability $1 - \varepsilon$, while a random action is chosen with probability ε for a selected value of ε. The parameter controls the balance between exploitation and exploration.

However, this did not show to be sufficient to reliably find good chains of heuristics. The reason is simple, there are too many possibilities how to build chains of heuristics. Given the set $\mathbf{A}$ of actions and a chain of length c, there are $|\mathbf{A}|^c$ possibilities for heuristic chains. The domains from CHeSC 2011 have 8 to 15 LLHs, the MSD domain even has 22. This results in 64 to 484 possibilities just to chain two heuristics, or 512 to 10648 possibilities to chain three heuristics. Further, due to the heuristic nature of the whole system, chains have to be applied repeatedly in order to really determine their utility in improving the solution. Overall, this leads to a pool of choices for heuristic sequences that is too large and too uncertain to reliably provide good solutions.

On the other hand, many of the best performing hyper-heuristics from literature including the most recent ones are based on the ideas of Iterated Local Search (ILS) [LMS03], where perturbation and intensification phases are repeated. This lead to the following novel improvement for reinforcement learning for hyper-heuristics: Instead of using uniform random selection with probability ε, the category of the next LLH is chosen according to probabilities based on ILS. Only within the chosen category the next LLH is chosen based on a uniform random distribution.

The next heuristic category is therefore chosen according to the following procedure:

- The initial category for the very first heuristic of a chain is chosen according to uniform random probabilities (if crossover would have been chosen, but is not available, ruin-recreate is selected instead)
- If the previous heuristic was of type ruin-and-recreate, the next one is either of the same category with probability p_r, or in any of the others with equal probability $\frac{1-p_r}{3}$.
- If the previous heuristic was of type crossover, the next one is either of the same category with probability p_c, or in the categories mutation or local search with equal probability $\frac{1-p_c}{2}$.
- If the previous heuristic was of type mutation, the next one is either of the same category with probability p_m, or in the category local search with probability $1-p_m$.
- If the previous heuristic was of type local search, the next one will be as well.

This way the categories have a fixed order going through large-scale perturbations first (ruin-and-recreate, then crossover), followed by mutations and finally local search until the end of the chain, or until a new best solution is found. While the order is fixed, every category except local search can be skipped.

The probabilities to stay in the same category p_r, p_c, and p_m are defined based on previous experience collected during the execution of the hyper-heuristic so far, and the fact that if a sequence is stopped at each element with probability p, the expected length of the sequence is $\frac{1}{p}$:

$$p_r = 1 - \frac{1}{1 + \frac{a_r \cdot c}{a_r + a_c + a_m + a_l}} \tag{5.15}$$

$$p_c = 1 - \frac{1}{1 + \frac{a_c \cdot c}{a_c + a_m + a_l}} \tag{5.16}$$

$$p_m = 1 - \frac{1}{1 + \frac{a_m \cdot c}{a_m + a_l}} \tag{5.17}$$

Equations (5.15), (5.16), and (5.17) show the definition based on the average number of heuristics of type ruin-and-recreate, crossover, mutation, and local search in successful chains represented by a_r, a_c, a_m, and a_l respectively. The reason for this computation is that different domains might need vastly different ratios of heuristic types, e.g., one might benefit from using ruin-and-recreate followed by a lot of local search heuristics, while another might benefit from a large number of mutation calls. Therefore, the inner fraction calculates a target length for the number of heuristics of the same type in the current chain (+1 to prevent complete exclusion of unsuccessful types), the remainder transforms this target length into the appropriate probability value. Note that p_c and p_m do not use average chain lengths for types that are earlier in the sequence to increase their usage in light of the fact that each type has equal probability to directly be the starting type, making it more likely to skip earlier types.

Using this novel ILS-based selection for epsilon-greedy allows to greatly focus the hyper-heuristic on sequences of LLHs that are expected to provide good performance, since it greatly reduces the chance that a large amount of time is wasted on very unlikely combinations of LLHs. At first glance, this might look like going from a later type to an earlier type is completely prevented, maybe restricting the search space for good solution chains too much. However, this is not the case due to the rich state representation. LAST-RL might learn to use a mutation next for a specific part of the state space. If the state ends up in this part of the state space after applying a local search heuristic, and the policy does not land in the ε-case, then the mutation is executed after the local search without restrictions.

Regarding the value of ε, different values were investigated in preliminary experiments, including adaptive strategies where the value varies during the search. However, a fixed value of $\varepsilon = 0.1$ provided very good results.

5.5 Evaluating LAST-RL on the CHeSC Domains

The hyper-heuristic was implemented in the Java HyFlex framework and run using OpenJDK 8u292. The first evaluation was performed on the six domains from CHeSC 2011 [BGH+11] implemented in the HyFlex framework [OHC+12], which are MaxSAT (SAT), Bin Packing (BP), Personnel Scheduling (PS), Flowshop (FS), Travelling Salesperson Problem (TSP), and Vehicle Routing Problem (VRP). The implementation of reinforcement learning is done using the BURLAP library version 3.0.1 [Mac16]. While we ultimately want to apply LAST-RL to the complex-real life domains from this thesis, since hyper-heuristics are intended to be very broadly applicable, this step is necessary to ensure that LAST-RL is indeed general and not specifically tuned for our own domains.

The experiments were run on a computer with an Intel 12th Gen i9-12900K with 3.19 GHz and 32 GB of RAM, each individual run was performed single-threaded. The competition provided a benchmark script to normalize the runtime for a fair comparison, which allowed 240 seconds for each run on our setup.

For each of the domains, a set of ten to twelve instances is available. Out of these, five for each domain are used for the competition. In CHeSC 2011 the mode for comparison was as follows: Each hyper-heuristic is run 31 times on each of the five competition instances in each of the six domains. Then, the median result for each instance is taken and compared against the other hyper-heuristics using a formula-one-based system for distributing points: 10 points to the best hyper-heuristic, followed by 8, 6, 5, 4, 3, 2, 1 points for the following ranks. If different approaches provide the same result, the points for the affected ranks are summed up and shared between the equal competitors.

The values used for the required parameters for the learning agent and the epsilon-greedy policy were set as stated in the previous section. Note that we tried to use parameter tuning with SMAC 3 version 1.1.1 [LEF+22] on a preliminary version of our hyper-heuristic. Unfortunately, the configurations found performed worse than the manually tuned ones. This might be due to the variations between individual runs making it difficult to assess which setting works better without repeating each setting very often on many instances. However, the manually tuned configuration provided very good results, aiming for further improvements with a more sophisticated parameter tuning experiment is left for future work.

5.5.1 Comparison of Different Versions in the Original Competition

In the following evaluation we want to highlight the contributions of the most important aspects of LAST-RL, which are rich state reinforcement learning and the ILS-based epsilon-greedy policy, therefore we compare the following versions:

- ILS-only: Setting $\varepsilon = 1$, this version only uses the ILS heuristic type selection, ignoring reinforcement learning.

Version	Total		SAT		BP		PS		FS		TSP		VRP	
ILS-only	129.25	2	27.85	4	0	14	38.5	1	15.0	6	11.9	8	36.0	1
RL-only	137.40	2	0.00	15	17	5	36.0	1	30.5	3	17.9	4	36.0	1
Simple-state	142.10	2	28.20	4	14	5	26.0	3	20.0	5	17.9	4	36.0	1
No-time	169.75	1	27.85	4	17	5	30.0	2	32.5	3	27.9	3	34.5	1
LAST-RL	166.50	1	24.10	4	19	4	36.0	1	28.5	4	24.9	3	34.0	1

Table 5.6: Evaluation of LAST-RL versions in the original CHeSC 2011

- RL-only: Using the default epsilon-greedy procedure, this version lacks the influence of the ILS-based heuristic type selection.
- Simple-state: The state representation is reduced to just the previous heuristic in order to investigate the effect of the rich state representation. The previous heuristic still allows to learn good chains of heuristics, but without taking into account the current position in the search space.
- No-time: The original evaluation of the full version of the hyper-heuristic using the reward function without incorporation of the runtime component is given for comparison with the final version.
- LAST-RL: The full final version of the hyper-heuristic.

The following comparison is done evaluating each version individually against the 20 participants in the original CHeSC 2011.

Table 5.6 shows the points achieved in total and in the individual domains, and the rank among the 21 hyper-heuristics in the comparison. The results clearly show that the full versions, both with and without the additional time criterion, perform significantly better than the other versions in comparison. However, since several intelligent components contribute to the overall hyper-heuristic, even the versions lacking one of those components provided good results in the overall comparison with the other participants of the competition.

The detailed comparison on the individual domains provides several important insights regarding the contributions of the individual components of LAST-RL to the overall results. Reinforcement learning is especially important for Bin Packing (BP) where a score of 0 is reached using just the ILS selection, but also for Flowshop (FS) and Travelling Salesperson Problem (TSP), while top results can be achieved for MaxSAT (SAT), Personnel Scheduling (PS), and Vehicle Routing Problem (VRP) without reinforcement learning. On the other hand, the results for RL-only show that without the ILS component, SAT ends up with a score of 0, and TSP is also still at a low score.

Replacing the rich state representation with a simple state resulted in the highest score for any of the three version where a core component was disabled, but had a significant

Domain	Instance Median				
SAT	4.0	6.0	2.0	6.0	8.0
BP	0.015939896	0.0067153699	0.014561790	0.10839902	0.025000369
PS	21.0	9580.0	3217.0	1583.0	320.0
FS	6254.0	26780.0	6325.0	11388.0	26605.0
TSP	48194.92010	20932610.14216	6816.27341	66933.30437	53062.18846
VRP	60139.41330	13314.64004	145474.28532	20655.73734	146423.86789

Table 5.7: Median results of LAST-RL on the competition instances for comparison

Rank	Method	Reference	Total	SAT	BP	PS	FS	TSP	VRP
1	**LAST-RL**		166.5	24.1	19	36.0	28.5	24.9	34
2	GIHH	[MVDCVB12]	164.5	33.6	43	7.0	34.0	35.9	11
3	ML	[Lar11]	115.9	10.5	9	28.5	35.0	12.9	20
4	VNS-TW	[HCF12]	114.5	33.6	2	33.0	31.0	10.9	4

Table 5.8: Comparison of LAST-RL with the best competitors in the original CHeSC 2011

negative effect on four domains. Only SAT and VRP stay at the same levels as LAST-RL, while especially PS, FS, and TSP are affected.

Regarding the individual domains, all variants perform very well on the VRP domain, constantly achieving the first place in the competition. For most of the other domains, at least one of the versions lacking a core component performed at a similar level as LAST-RL, indicating that the disabled component is not so important for this domain. For TSP, however, only the combination of all core components was able to get good results, showing the benefit in the combination of the rich state representation and the use of reinforcement learning with the ILS exploration strategy.

While No-time actually was slightly ahead in the total score, the evaluation will still continue with LAST-RL, since this is the more general version that also provides good results on more of the real-life domains, besides the difference is very small. Table 5.7 shows the median values of LAST-RL on the competition domains to allow comparison in future work.

Table 5.8 shows the comparison of LAST-RL with the competitors of the original CHeSC 2011 participants, where LAST-RL would have taken the first place. Note that there is a fixed amount of points available for each instance, therefore, depending on the results for individual instances, adding LAST-RL as a new competitor can change the order among other participants compared to the original competition, as it happened for ML and VNS-TW in this case.

Our approach cannot only take the first place slightly before GIHH (called AdaptiveHH in the context of the competition), but also provides a very stable performance across

Rank	Method	Reference	Total	SAT	BP	PS	FS	TSP	VRP
1	TS-ILS	[AOO21]	184.50	33.25	34	32.0	30.00	37.25	18
2	GEP-HH	[SAKQ14]	124.59	20.33	22	0.0	50.00	13.25	19
3	FS-ILS	[ABN14]	107.99	33.25	1	2.0	24.83	26.91	20
4	MCTS-HH	[SK15]	99.83	13.58	18	29.0	5.00	17.25	17
5	GIHH	[MVDCVB12]	78.49	10.58	30	2.0	14.00	17.91	4
6	**LAST-RL**		77.49	3.58	9	22.5	14.40	7.91	20
7	QHH	[CWL18]	65.83	23.58	0	0.0	15.00	22.25	5
8	RL	[MM22]	64.08	9.08	29	0.0	10.00	8.00	8
9	ML	[Lar11]	52.99	1.25	0	23.0	12.83	2.91	13
10	VNS-TW	[HCF12]	47.24	15.33	0	24.0	7.00	0.91	0
12	FRAMAB	[FGP15]	35.25	1.25	0	16.0	0.00	0.00	18
16	CH-PR	[Chu20]	24.25	1.25	15	0.0	3.00	0.00	5

Table 5.9: Comparison of LAST-RL recent hyper-heuristics

all the domains with the lowest value of 19 for Bin Packing, while no other competitor has a value of at least 10 in each domain. This highlights that LAST-RL is indeed very general and can adapt to many very different domains without problems.

5.5.2 Comparison Against Recent Methods

Hyper-heuristics are a rapidly evolving field, and several new methods have been proposed since CHeSC 2011. To get a full picture of how our method compares against the most recent advances in the field, the competition setup is extended to include work that has since been able to outperform results from the competition, and where either the median values necessary for the competition or the implementation was available. These are FRAMAB [FGP15], GEP-HH [SAKQ14], MCTS-HH [SK15], QHH [CWL18], TS-ILS [AOO21], and the recent RL [MM22], which made their median results available for comparison. Further, FS-ILS [ABN14] was rerun, and CH-PR [Chu20] was evaluated according to the new implementation [MM21], where the pruning variant scored the highest.

Table 5.9 shows the results of the comparison of LAST-RL with the top three competitors from the original competition and the recent hyper-heuristics mentioned above. LAST-RL can achieve the sixth place (here very slightly behind GIHH due to the redistribution of the fixed number of points per instance). Clearly TS-ILS currently leads the results, not only in total but also most individual domains.

However, LAST-RL can reach the joint first place for the VRP domain (together with FS-ILS), and it can outperform previous approaches mainly based on reinforcement learning, not only ML from the original competition, but also the recent approaches QHH and RL. Therefore, in comparison with other recent hyper-heuristics, LAST-RL pushes the boundaries for methods based on reinforcement learning and shows that there is still unused potential in further investigating this approach for achieving high-quality results.

Further, LAST-RL can still gain at least some points in all domains, underlining its generality, since only TS-ILS (18) and MCT-HH (5) achieve a higher lowest domain score than LAST-RL (3.58) out of all competitors.

5.6 Evaluating LAST-RL on the Real-Life Scheduling Domains

After using CHeSC 2011 as the test bed, the final step is to evaluate LAST-RL on the real-life scheduling domains in this thesis, and investigate how well the reinforcement learning approach transfers from the smaller, more academic competition domains to the complex real-life domains, which have more constraints, longer runtime, and more Low-level Heuristics with more differences in their runtimes.

Indeed the first evaluation of the hyper-heuristic in the version No-time immediately showed very promising results for MSD, but less good results for the other two domains. A major factor in this regard seems to be the vast difference in runtimes for these domains. While MSD still has execution times varying between a few milliseconds and around half a second, for the largest BDS instances the larger end of the spectrum reaches more than 30 seconds, while for the largest RWS instances this difference jumps up to a gap from milliseconds to more than three minutes for some calls to best improvement heuristics, which is a difference of several orders of magnitude. Further, the longest calls now take a significant amount of the total runtime, meaning that even if such a call results in an improvement, it might be better to spend the time on other, much faster heuristics.

Therefore, the runtime aspect was included in the reward function after the first experiments on these domains, leading to the final version of LAST-RL as described in this chapter. As seen in the previous section, this additional criterion only had a very small impact regarding the competition domains, even leading to a small decrease in the overall number of points. However, in this section both variants will be added to the comparison of the hyper-heuristics for the real-life domains, showing the difference this criterion makes.

5.6.1 Rotating Workforce Scheduling

We now perform the same evaluations that have been done in Section 5.3.1, comparing the performance of the best previous methods GIHH, L-GIHH, and HAHA with LAST-RL and its variant No-time.

The results in Table 5.10 show that LAST-RL, like most of the other hyper-heuristics, also struggles with the feasibility aspect. There is a slight improvement using the time criterion for the reward, but still a large gap to the leading methods GIHH and L-GIHH. However, LAST-RL has the third best total number of feasible results (244), even though just very slightly ahead of CH-BI (242), CH-FR (242), HAHA (236), and CH-UN (235), which is on the same level as No-time (235).

Objective	GIHH	L-GIHH	HAHA	No-time	LAST-RL
F_1	**98**	97	79	78	80
F_2	**99**	98	79	77	82
F_3	**100**	99	78	80	82

Table 5.10: Number of feasible solutions out of 100 per hyper-heuristic

Obj.	Inst.	MiniZinc	GIHH	L-GIHH	HAHA	No-time	LAST-RL
F_1	9	34	**35**	**35**	34	32	32
F_1	11	4	**6**	**6**	5	5	4
F_1	12	**8**	7	**8**	7	7	7
F_1	15	-	**15**	13	10	-	-
F_1	18	**23**	21	20	22	20	20
F_1	19	24	**26**	23	18	-	20
F_1	20	-	32	26	**37**	-	-
F_2	9	**2**	3	3	3	3	3
F_2	10	**3**	4	4	5	4	4
F_2	11	**5**	6	6	13	10	9
F_2	12	**4**	5	5	5	5	5
F_2	13	**4**	5	5	6	6	5
F_2	15	-	**7**	8	-	-	-
F_2	16	**4**	5	6	6	6	6
F_2	17	**4**	5	5	6	7	6
F_2	18	**4**	6	7	8	10	8
F_2	19	28	**8**	11	23	-	-
F_2	20	-	**11**	15	25	-	24
F_3	9	**26522**	**26522**	26524	26524	30942	28734
F_3	10	**8772**	8784	8778	8778	8778	8796
F_3	11	21710	**20783**	**20783**	23596	21702	21706
F_3	12	**4836**	5243	**4836**	**4836**	5243	5243
F_3	15	-	**196806**	200887	209213	-	-
F_3	16	**16868**	16900	16888	16890	17052	16894
F_3	18	**84414**	87218	87218	87218	87238	87306
F_3	19	1456141	**1253063**	1267498	1339601	-	-
F_3	20	3855876	**3268396**	3295030	-	-	-

Table 5.11: Best result comparison for hard RWS instances

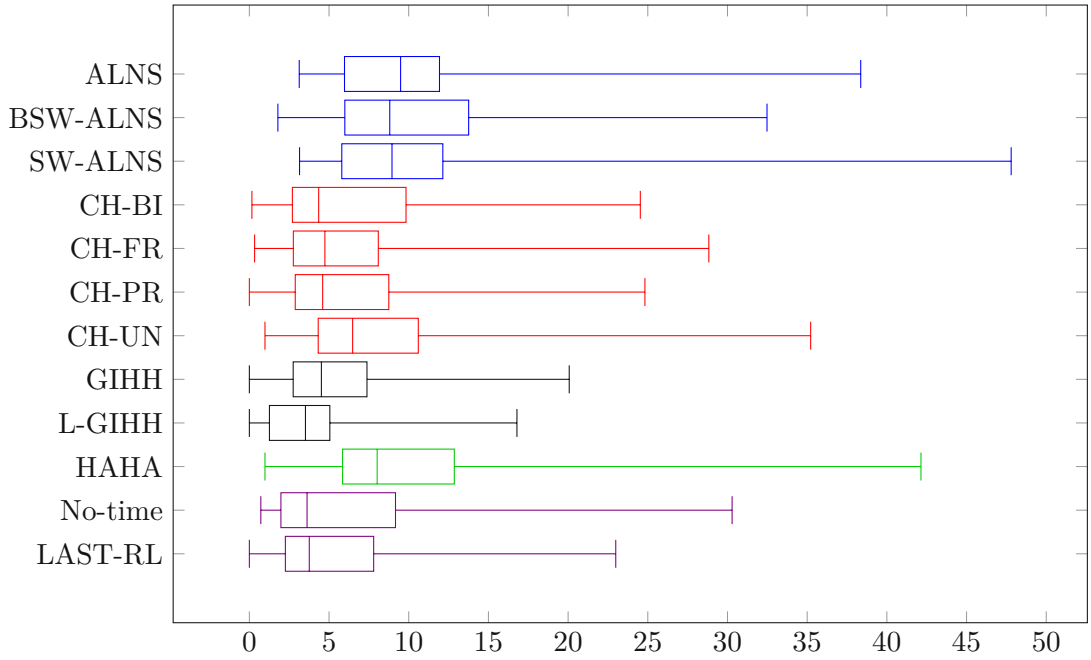


Figure 5.7: Best result deviations for reduced MSD

Table 5.11 shows the results on the hard instances from Secion 5.3.1. The conclusion is that also regarding solution quality, on this domain LAST-RL cannot keep up with the top methods GIHH and L-GIHH. Still, the results are often close, and the overall performance is similar to that of HAHA and the better variants of the CH hyper-heuristics. The difference between No-time and LAST-RL is small regarding the result values, there are several instances where LAST-RL is slightly better or can find a solution where No-time always stays infeasible, but also very few instances where No-time is actually very slightly better.

5.6.2 Minimum Shift Design

For this domain, we will again revisit the previous evaluation from Section 5.3.2.

Figure 5.7 shows the baseline comparison with the deviation from the optimal results provided in Chapter 3 in percent for each hyper-heuristic on the realistic data sets 3 and 4 without using T_4. In this case LAST-RL can provide a much better result: It can reach the second place after L-GIHH, slightly ahead regarding lower quartile and median compared to the next best results by GIHH and the better of the CH versions.

This is the real-life domain where the differences between runtimes of the individual Low-level Heuristics is the least pronounced, consequently it could be expected that the difference between No-time and LAST-RL should not be very large. In fact, No-time manages to achieve a slightly smaller lower quartile and median compared to LAST-RL, but a significantly higher upper quartile and maximum.

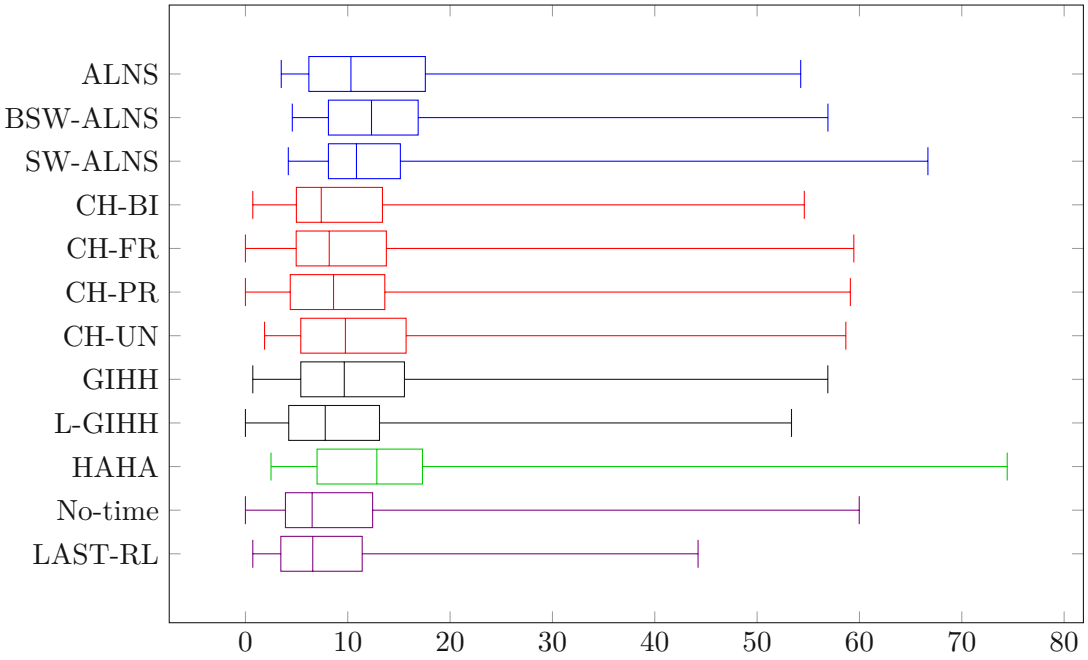


Figure 5.8: Best result deviations for full MSD

Figure 5.8 shows the deviations for MSD with T_4 compared to the optimal solutions for reduced MSD. In this comparison, L-GIHH looses the advantage it still had for the reduced version, and LAST-RL can clearly outperform all other hyper-heuristics in the comparison. The best values for lower quartile, median, upper quartile, and maximum are achieved by LAST-RL, and even No-time, which lies behind regarding lower quartile, upper quartile, and maximum, but has a slightly better median, can outperform the other hyper-heuristics in all these measures. Therefore LAST-RL is the superior hyper-heuristic to use for this domain.

5.6.3 Bus Driver Scheduling

Table 5.12 shows the average of the best results per instance category for Simulated Annealing, all hyper-heuristics that could achieve a best result in any size category in the previous comparison in Section 5.3.3, and the new hyper-heuristic versions No-time and LAST-RL.

Like for MSD, this comparison again shows very strong results of LAST-RL on this different real-life domain, with a clear majority of 5 out of the 10 size categories being won by LAST-RL, especially compared to the much more diverse picture in Table 5.3 where no method could win more than two categories.

Further, this domain provides the strongest support for the inclusion of the runtime criterion into the reward function, as without this criterion only a single category could have been won with the new hyper-heuristic due to wasting too much time with slow

Instance	SA	CH-FR	CH-PR	GIHH	L-GIHH	No-time	LAST-RL
10	**14717.4**	14838.8	14805.6	14787.0	14773.6	14824.4	14779.8
20	30860.6	30706.6	30671.2	30731.6	30694.0	30707.8	**30669.4**
30	50947.4	50946.6	50903.6	**50765.8**	50854.2	50936.6	50890.0
40	69119.8	68583.4	68847.6	68639.6	68645.4	68536.6	**68478.2**
50	87013.2	87091.2	87034.0	86762.0	86729.8	86963.6	**86681.8**
60	103967.6	103521.8	103464.8	103138.8	103149.8	103372.4	**102935.8**
70	122753.6	122247.2	122025.6	121671.8	**121660.6**	121991.4	121916.2
80	140482.4	139382.4	139209.2	139123.0	**139041.6**	139173.6	139250.2
90	156385.0	154938.0	154972.4	155093.8	155113.2	155117.8	**154915.0**
100	173524.0	171718.6	**171182.4**	171278.2	171325.4	171819.2	171589.4

Table 5.12: Best results for BSD with different solution methods

Method	BDS	F_1	F_2	F_3	MSD	MSD+T_4
ALNS	3	10	7	7	10	11
BSW-ALNS	4	9	6	5	16	15
SW-ALNS	5	11	6	7	16	17
CH-BI	3	12	11	11	26	24
CH-FR	7	13	13	8	23	25
CH-PR	9	12	13	8	23	28
CH-UN	5	12	12	9	18	18
GIHH	8	**17**	**20**	**16**	27	21
L-GIHH	**13**	16	15	15	**69**	37
HAHA	1	13	8	11	9	11
No-time	7	12	10	9	26	35
LAST-RL	11	12	12	10	34	**43**

Table 5.13: Number of wins for each hyper-heuristic (best objective)

improvement heuristics that do provide improvements, but at a rate too slow to justify their frequent usage.

5.6.4 Summary and Conclusion

Table 5.13 shows the number of times a hyper-heuristic obtained the best result out of all 12 hyper-heuristics in the extended comparison on an instance including No-time and LAST-RL (ties are awarded to all hyper-heuristics involved in the tie), Table 5.14 shows the same comparison for the average results.

These tables sum up the results already seen in the previous part of this evaluation. LAST-RL provides the new best result for the full MSD problem, outperforming the previous best L-GIHH, and it provides a new best average and close second place regarding the number of best wins on the BDS domain. The higher number of wins regarding

Method	BDS	F_1	F_2	F_3	MSD	MSD+T_4
ALNS	0	5	4	5	4	5
BSW-ALNS	2	5	3	4	5	5
SW-ALNS	3	6	4	4	5	5
CH-BI	5	10	5	6	8	12
CH-FR	4	10	5	6	12	10
CH-PR	2	10	6	6	11	17
CH-UN	3	11	6	6	12	11
GIHH	9	**19**	**19**	**15**	13	13
L-GIHH	14	13	13	10	**86**	35
HAHA	0	9	5	9	5	4
No-time	2	9	5	8	12	24
LAST-RL	**16**	9	6	7	14	**41**

Table 5.14: Number of wins for each hyper-heuristic (average objective)

averages together with the best results in 5 out of the 10 size categories in Table 5.12 show LAST-RL to be more reliable in providing high-quality results compared to L-GIHH. Only in the RWS domain the new hyper-heuristic falls behind, providing similar scores to the CH versions and HAHA.

Overall this provides the very promising result that LAST-RL was able to outperform the already high-quality results from previous hyper-heuristics on two out of the three complex real-life domains, indicating the strong potential for hyper-heuristics based on reinforcement learning for such practical applications. Just like the other hyper-heuristics in comparison, LAST-RL was developed using a completely different set of domains for evaluation, yet just required a small addition regarding runtime cost to directly be applied to the real-life domains, showing its generality and adaptability.

Therefore, the goal is to further investigate and extend this approach, identify reasons for the weakness on the RWS domain, and proceed to apply the new hyper-heuristic in the industrial practice of our industry partner.

5.7 Conclusion

In this chapter, we moved from the view of individual solution methods for the different real-life problem domains to consolidating them into a framework that can represent all of them, and then to use the general methodology of hyper-heuristics to provide high-quality solutions with a very general solution method.

First, the interval-based framework is designed to deal not only with the three example problems in this thesis, but with interval-based scheduling problems in general by using intervals, sequences of such intervals, and aggregations of values over time to then define properties and constraints that evaluate a solution according to the specification of a

problem. This allows to exchange and add individual constraints, making the approach very flexible for modelling and solving different problems, different variants of the same problem, or problems that have a frequently changing specification. These flexible solutions can be modified by various algorithms, making use of a general definition of moves and strategies to apply moves, to allow easy implementation of a wide range of algorithms in a modular way, where the algorithms are independent of the problem specification and existing moves can be reused for problems with similar structure independent of both algorithms and problem specification.

After transferring our problem domains into this framework, the next step was to design a new set of Low-level Heuristics for these domains to allow the application of hyper-heuristics to the problems. Several new LLHs have been proposed in the categories of local search, mutation, and ruin-and-recreate, which is especially important for structural changes in a solution. The flexible nature of the framework allows to implement these as algorithms and connect to the established HyFlex framework for the implementation of the hyper-heuristics.

Using the new LLHs, we could successfully evaluate a set of state-of-the-art hyper-heuristics, which provided high-quality results for all the domains under investigation, including new and improved results for hard RWS instances, solutions for MSD with the additional objective T_4 that is very important in practice, and new best solutions for some instances of BDS. Especially Lean GIHH, but also the original GIHH outperformed the other hyper-heuristics in comparison. This constitutes strong support for using hyper-heuristics on real-life scheduling domains. Therefore, the aim is to deploy the leading hyper-heuristics in the industrial software of our partner. In future work we plan to investigate the effects of the different LLHs in more detail, including the analysis of the chosen combinations of LLHs and the importance of specific sets of these heuristics for the solution quality.

Finally, we transitioned to our own hyper-heuristic, LAST-RL, which is based on reinforcement learning. While most hyper-heuristics include some aspect of learning, only a few of them use reinforcement learning as their core learning strategy. We introduced a novel rich state representation based on 15 features of the current search state instead of the previously used simple state models, in combination with the linear state-action value function approximation method tile coding and the learning agent SARSA-lambda. Solution chains are based on the Luby sequence, but dynamically adapted based on successful chains in the past. The policy is an epsilon-greedy policy, but with the novel mechanism of using ILS as the exploration strategy, combining this most successful approach regarding hyper-heuristic concepts with the reinforcement learning approach. The approach was first evaluated on the six domains from the Cross Domain Heuristic Search Challenge 2011, which provide a test bed outside the real-life domains to maximize generality of the new method. The evaluation of the core components showed that each of them contributes to the overall results, which allowed LAST-RL to outperform the original competitors of CHeSC 2011, and provide good results in comparison with the best recent hyper-heuristics. Specifically, LAST-RL outperforms previous approaches

mainly based on reinforcement learning, which shows the promise of using this method, especially when combined with ILS. Then we applied LAST-RL to our real-life scheduling domains, where it could outperform previously obtained results in two out of the three domains, showing the large potential for such general methods based on reinforcement learning in those real-world scenarios.

The interval-based framework, the development of new Low-level Heuristics, the evaluation of existing hyper-heuristics, and the design of LAST-RL all contribute to Research Goal 4 (Section 1.1.4). Further the application of reinforcement learning as well as other adaptive techniques in LAST-RL contribute to Research Goal 3 (Section 1.1.3).

In future work we want to further improve LAST-RL, including a more detailed investigation of the individual search state features, different learning agents like deep reinforcement learning, more sophisticated parameter tuning, and investigating what keeps LAST-RL from outperforming previous methods on RWS.

CHAPTER 6

Conclusion

Real-world personnel scheduling problems can come in many different forms and include frequent changes of constraints and objectives, requiring to develop general methods that are easy to adapt and apply to different domains. In this thesis a range of methods for several different example problems in the area of personnel scheduling have been investigated, starting with problem-specific methods to provide high-quality solutions including optimality guarantees, and then consolidating them in a general framework for interval-based problems for the application of general methods. We deeply explored hyper-heuristics, by providing new Low-level Heuristics for the different real-life problem domains, investigating state-of-the-art hyper-heuristics, and finally proposing the new hyper-heuristic LAST-RL based on reinforcement learning with a rich state representation and an exploration policy based on Iterated Local Search, leading to a very general methods that provides high-quality results both on academic benchmarks and real-world problem domains.

More specifically, in Chapter 2 we have presented a new exact model for Rotating Workforce Scheduling that is implemented using the modelling language MiniZinc and solved with the lazy clause generation solver Chuffed. First we added new model extensions that showed to be very efficient in detecting infeasible instances in very little computational time and used bounds for the number of blocks to define redundant constraints. Then we added new real-life requirements in the form of complex weekly rest time constraints and are still able to solve the majority of the benchmark instances in short computational time. We introduced new objectives to optimize the scheduling of free weekends and showed that for the majority of instances the optimum can be found and proven in short computational time. Moreover, the output of intermediate solutions allows the user to decide about the trade-off between runtime and quality while running the solver.

Then we used Instance Space Analysis to evaluate the performance of two constraint models and a metaheuristic on the RWS problem. We defined new sets of features

and evaluated them by using them for machine learning with random forests. With the results of the ISA using the original dataset, we were able to produce a larger, more diverse dataset to cover a larger part of the instance space including the real life instances and closing gaps in the generated instances. Using the extended dataset, we identified four features that provide a good projection for the analysis of the instance space, corresponding to the possibilities for block lengths and the distribution differences between different days. We found different strong and weak areas for the individual algorithms and showed that in combination the algorithms cover the overall instance space well. We also identified areas of feasible and infeasible instances and linked the hardness of instances with the transition between these areas. We also presented a portfolio that uses the most suitable algorithm based on the position of the instance in the instance space.

In Chapter 3 we presented three different constraint models for the Minimum Shift Design problem. Those models were implemented in MiniZinc and evaluated on the standard benchmark datasets using three different solvers, the lazy clause generation solver Chuffed and the MIP solvers Gurobi and CPLEX. The results showed that our new model `NETWORK`, using a network flow based formulation, clearly outperformed the other models. Further, while Chuffed could provide optimal solutions for several instances in the first two datasets, all optimal solutions could be found using Gurobi or CPLEX. Comparing Gurobi and CPLEX, most results are close, but Gurobi provided significantly shorter runtimes on several difficult instances. This was the first time in literature that all available instances could be solved to optimality within short runtime, using less than 1 minute for all instances but one, which could be solved in less than 8 minutes, highlighting the strength of our approach. Across different instances from all datasets, runtimes could be reduced significantly in comparison to previous work. Therefore, our approach constitutes the new state of the art for MSD.

Next, in Chapter 4 we presented the Bus Driver Scheduling problem using the complex set of rules in the Austrian collective agreement. We analysed characteristic features of real-life instances, in particular the demand distribution and waiting times and discussed their implications on scheduling. We provided new publicly available benchmark instances for further research and presented a solution method based on a construction heuristic and Simulated Annealing. We successfully applied these methods to the new benchmark instances and discussed the importance of a well-designed objective function. We provided insights in the successful application in real-life scenarios and compared with a problem from Brazil, where we could improve several benchmark instances.

We further presented a Branch and Price approach that includes a range of novel improvements for solving the high-dimensional subproblem that can be applied to other problems with similar characteristics. Experimental results showed that the approach provided strong lower bounds for SA and randomized hill-climbing, and improved or proved optimality for 48 out of 50 instances. The novel results had optimality gaps below 1% for small to medium size instances and represented high-quality solutions for larger instance sizes in reasonable time. Therefore the B&P approach may be considered the

new state-of-the-art exact method for solving small to medium sized instances for this complex scheduling problem.

Finally, in Chapter 5 the individual problems were brought together by the introduction of a framework based on the notion of intervals and sequences of intervals, where properties and constraints can be modelled and exchanged individually, allowing great flexibility in the definition of problems and their different variants. Further, a flexible and modular system of defining algorithms that can manipulate solutions based on moves allows to implement and reuse various solution methods with minimum adaptation effort.

This framework cannot only be used to directly apply solution methods like Simulated Annealing, but can also be used to define Low-level Heuristics for the use of hyper-heuristics on our real-life problem domains. We provided a major study on using various hyper-heuristics to solve these complex domains for the first time, including the proposition of several new LLHs. We compared several versions of state-of-the-art hyper-heuristics and provided multiple comparisons for the relative performance of the hyper-heuristics on the individual domains. The results showed that especially the approach Lean GIHH, but also the original GIHH are very well suited to efficiently solve the different scheduling domains. With these methods we were able to provide several advances for the individual domains. For BDS, we were able to outperform previous heuristic approaches and find two new best known solutions for the benchmark data set. For RWS, we could provide several new best known solutions for the different optimization goals including several solutions for instances previously running into timeouts without a solution. For MSD, for the first time we provided comprehensive results using the forth objective that has high practical relevance, hopefully fuelling more research into this more realistic version of the problem.

Next, we moved one step further to provide our own hyper-heuristic LAST-RL, which is based on reinforcement learning. In contrast to previous approaches, it uses a rich state representation for the first time based on 15 different features representing the current search state. It uses adaptive solution chains based on the Luby sequence, and an epsilon-greedy policy where the exploration part is based on an adaptive Iterated Local Search procedure. The new hyper-heuristic was evaluated on the six different domains from the Cross Domain Heuristic Search Challenge 2011, where it showed consistent good performance on all domains, outperforming the original participants of the competition and providing good results in comparison with recent advances in hyper-heuristics, especially by outperforming other approaches based on reinforcement learning. Further, the core components were evaluated individually, showing their contributions to the overall results. Finally, LAST-RL was applied to the real-life domains in this thesis, providing new best results compared to the other hyper-heuristics for two of the three domains, which shows that hyper-heuristics based on reinforcement learning have high potential to provide high-quality general solution methods for complex real-world problems.

6.1 Comparing Different Methodologies

Throughout the chapters of this thesis, a range of different methods has been investigated. First several different problem-specific methods have been investigated, then a general approach using hyper-heuristics has been introduced and applied in the context of our interval-based framework. This also raises the question whether such general methods can fully replace the problem-specific methods, or in case they cannot, what can we learn about the selection of an appropriate method for a new optimization task at hand.

We would argue that in order to be prepared for different problems in different application contexts, having a spectrum of different methods to apply is the best option, and all the different methods can show strengths that apply in different scenarios. If good solutions are required for new problems, new variants of existing problems, or problems that change their requirements very fast or are still in the process of having their requirements formalized, it pays off very well to have general approaches that can easily be adapted and produce good solutions very fast. The work in this thesis has shown that hyper-heuristics like our LAST-RL can provide high-quality solutions to different problems with minimum adaptation effort and are therefore very well suited for such a task.

On the other hand, such heuristic solutions cannot give optimality guarantees for the solutions. In some situations it might be beneficial or even necessary to have proven optimal solutions or at least a measure of the remaining gap to the optimum. In this thesis several different exact methods have been investigated that are suited for this task. Even in such situations where a problem needs to be modelled for exact solution methods, some generality can often be achieved. For example, we used the solver independent modelling language MiniZinc, which allows to flexibly choose the best-performing solver for the problem at hand. Further, also constraint modelling allows some modularity, as we have seen for RWS were different extensions can be defined. Those could easily be mixed together based on a given problem specification.

For RWS we have seen a strong preference for CP solvers, which based on our experience is related to the strong focus on feasibility for the problem. As often the highest difficulty comes from finding a feasible solution at all, Constraint Programming solvers can perform very well in excluding infeasible parts of the search space using constraint propagation. On the other hand, for MSD MIP solvers were much more successful, which we attribute to the very large feasible space (e.g., even an empty solution is feasible, even though very bad), where the focus is on finding the best solution in regard to several optimization criteria in this large space.

Finally, for BDS finding a exact solution method proved even more difficult due to the complexity of the problem. In such cases, decomposition techniques like Branch and Price can be the key to solving such problems with quality guarantees. Indeed, the solutions provided by our approach are of high quality even for medium to large size instances. However, scaling to even larger instances that can still occur in practice, e.g., when optimizing medium-sized Austrian cities at once, is very challenging. Further, the solution approach is very problem-specific, and even adapting it to a different version

of BDS would require extensive changes in the subproblem. However, even from such a problem-specific solution method there were several general ideas to learn how to deal with a high-dimensional Resource Constrained Shortest Path Problem that can be generally applied whenever such a large RCSPP occurs, whether as a subproblem for B&P or in a different context.

As it is usually the case, no method is superior in all scenarios, and an informed decision is often necessary. As seen in this thesis, different strengths and weaknesses of different methods can also help to learn more about the characteristics of the problem and make an informed decision which method to use based on the instance, as presented using Instance Space Analysis. Overall, we believe that especially the general methods presented in Chapter 5 provide a large benefit due to their flexibility and adaptability, but nevertheless it always pays off to have a good portfolio of different solution approaches to make an informed decision on which method is the best to apply in a particular scenario.

6.2 Real-Life Application

Since we are working with our industry partner XIMES GmbH, we do not only obtain real-life problem specifications and instances, but can also see our solution methods being put into practice for different products and customers. The constraint models presented in Chapter 2 have since been successfully integrated into the newest version of their software Shift Plan Assistant, which is in use by many customers for planning their rotating schedules.

Further, all three problem domains presented in this thesis are implemented in the interval-based framework which is frequently in use to deal with customer-specific problem formulations. The modular design and separation of solution methods and problem specifications allows to easily deal with a range of very customer-specific constraints. For example, the MSD problem can be used with preferred start and end times, various different configurations for adding breaks to the shifts, a hard minimum and maximum demand around the given preferred demand, and other similar constraints.

The BDS formulation can also be used with a range of different options, using various settings for the distinction of paid and unpaid time, restrictions of places suitable for breaks or vehicle changes, or other options for break splits. It can even be applied for problems outside of public transport as long as there are some predetermined stretches of work that need to be done by employees at specific places.

While the work in this thesis provides the optimization methods, a connection to a web-based interface developed by our industry partner allows to upload demand data and specify usage and weights of constraints using a range of input tables, and it allows to use the results in tabular form for further processing or an included visualization. The flexibility of the interval-based framework is critical in the rapid adaptation to new customers or changing requirements of the customers, and the interface allows to add

additional constraints by extending the input tables, directly providing access to the new constraints for the user.

Further, the results obtained using hyper-heuristics are very useful in many ways. Besides the specific improved solutions, the results show that using hyper-heuristics for complex practical problems is a very useful approach to provide high-quality solutions without spending too much time for highly problem-specific solution methods. Even with the flexible implementation of the framework, e.g., when using Simulated Annealing, it is often necessary to adapt the starting temperature and the cooling rate to the specific problem. However, the LLHs used for hyper-heuristics are independent from any additional constraints or different objectives, so they do not need to be adapted to different customers, and the hyper-heuristics to not need manual tuning to different problems, allowing rapid adaptation to changing requirements. Therefore, deployment of the leading hyper-heuristics in our comparison, especially LAST-RL, in the industrial software of our partner will be targeted next.

6.3 Future Work

In the previous chapters we have identified several options for future work. This includes to extend explanations for failure in RWS further than simple cases like individual constraints detecting infeasibility or the output of intermediate variables, e.g., by using minimal unsatisfiable subsets. ISA could be extended to further dimensions of the problem like larger numbers of shifts or to extended versions of the problem like optimization variants.

Regarding RWS and MSD, the biggest focus in our future work will be to look at interactions between these two problems and ultimately solving them together. The reason is that often these two problems need to be solved in sequence, with MSD first finding a set of shifts to cover a given demand, and then RWS assigning the generated shifts to specific employees on specific days. However, depending on the constraints for both of these problems, the solution from MSD might not fit for RWS. For example, consider a scenario where MSD has a large range of possible shift lengths to cover the demand. Using predominantly short shifts results in an employee with a fixed weekly working time having to work more of those short shifts per week, potentially too many, while using longer shifts in MSD requires fewer shifts per week in RWS. In fact this is the reason why objective T_4 in MSD is so important in practice, but by integrating these two problems, there is great potential to improve solutions even further.

When dealing with BDS, an area of interest would certainly be to hybridize the Branch and Price approach, making it applicable for even larger instances by moving from an exact solution of the Resource Constrained Shortest Path Problem to a heuristic one. However, it seems to be very difficult to find an approximation that still provides the good new columns while not taking such a long time. Another interesting possibility for future work is further investigating how to deal with uncertainties in the schedules, e.g, regarding delays due to traffic. One option used in practice is to use break buffers,

extending the minimum break time to have some room for delay. However, there might be more efficient ways than static buffer times, e.g., based on previous data regarding punctuality, and dealing with stochastic scheduling problems.

Regarding the general methods using hyper-heuristics, there are several directions that we want to explore in the future. First, a deeper investigation of the effects of individual Low-level Heuristics will be done. While we know that the mixture of LLHs of different types is necessary to achieve good results, and found ruin-and-recreate heuristics to be important for solution quality especially when they perform structural changes in the solutions that other heuristics can hardly do, there is much more to learn regarding the application of LLHs on our real-life domains, including the optimal number of heuristics, the effect of different types of heuristics as well as individual heuristics, the potential to include crossovers, and the inclusion of a parameter that the hyper-heuristic can set for the LLH, which is possible using HyFlex, but was not yet explored in our LLHs.

We also further want to improve our hyper-heuristic LAST-RL, which includes several research directions. First, an in-depth investigation of the individual search state features should be done to gain more data on the ideal composition of this state. A more sophisticated tuning run should be performed to achieve optimal values for the parameters set in preliminary experiments. Again, the setting of LLH parameters should also be done by the hyper-heuristic, potentially by some form of learning as well. Instead of using just linear state-action value function approximation methods, investigating deep reinforcement learning would be an interesting direction. Investigating why the results on RWS stayed behind those on other domains might give valuable insight how to further improve the hyper-heuristic. Further work might also leave the constraints of CHeSC 2011 and look at learning over multiple runs, e.g., to provide much better starting weights by learning them over multiple instances of the same domain. This would be interesting in practice since typically a solution method will be executed many times on a similar set of problems. Therefore, being able to both learn from scratch or reuse experience learned from previous instances might provide very useful.

List of Figures

List of Tables

List of Algorithms

Glossary

BDS benchmark set Benchmark set containing 50 instances for BDS based on real-life demand distributions. https://cdlab-artis.dbai.tuwien.ac.at/papers/sa-bds/. 77, 126

Chuffed Lazy clause generation solver [CSS+18]. https://github.com/chuffed/chuffed. 9, 12, 18, 26–28, 30, 32, 36, 39, 46, 47, 49, 52, 62, 63, 65, 68, 69, 155, 156

CP Optimizer IBM ILOG CP Optimizer is a commercial scheduling software. https://www.ibm.com/analytics/cplex-cp-optimizer. 89, 107

CPLEX IBM ILOG CPLEX Optimizer is a commercial MIP solver [CPL09]. https://www.ibm.com/analytics/cplex-optimizer. 52, 62, 63, 65, 68, 69, 100, 156

Gurobi Commercial MIP solver [GO18]. https://www.gurobi.com/. 12, 52, 62, 63, 65, 68, 69, 156

HyFlex Hyper-heuristic framework implemented in Java [OHC+12].. 114–117, 120–122, 143, 153, 161

MATILDA A toolkit for the assessment of algorithmic power using Instance Space Analysis. https://github.com/andremun/InstanceSpace. 10, 40

MiniZinc Free and open source constraint modelling language [NSB+07]. https://www.minizinc.org/. 9, 12, 26, 27, 32, 36, 39, 51, 59, 62, 63, 69, 106, 123, 124, 155, 156, 158

MSD benchmark set Benchmark set containing four data sets for MSD. https://www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html. 52, 61, 124

OpenJDK Open source implementation of Java. https://openjdk.java.net/. 100, 121, 143

PyPy Fast just-in-time compiler for Python. https://www.pypy.org/. 111, 121, 122

Python Programming language used for the interval-based framework. https://www.python.org/. 107, 111, 116, 121

RWS benchmark set Benchmark set containing 2000 instances for RWS. https://www.dbai.tuwien.ac.at/staff/musliu/benchmarks/. 27, 35, 122

RWS extended benchmark set Extended benchmark set containing 6000 instances for RWS resulting from the application of ISA. https://cdlab-artis.dbai.tuwien.ac.at/papers/isa-rws/. 36

Weka Machine Learning Software in Java. https://www.cs.waikato.ac.nz/~ml/weka/index.html. 39

XIMES GmbH Our industry partner. https://www.ximes.com/. 5, 9, 32, 51, 107, 159

Acronyms

B&P Branch and Price. 4, 7, 72, 73, 89, 90, 100–103, 106, 126, 127, 156, 158–160

BDS Bus Driver Scheduling. 6–8, 73, 89, 92, 94, 103, 104, 106, 109, 110, 112, 128, 129, 147, 151, 153, 156–160

BP Bin Packing. 114, 143, 144, 146

CHeSC 2011 Cross Domain Heuristic Search Challenge 2011. 8, 106, 114, 115, 129, 136, 141, 143–147, 153, 157, 161

CP Constraint Programming. 4, 6, 50, 62, 68, 158

FS Flowshop. 114, 143–145

GES General Employee Scheduling. 10, 107

ILS Iterated Local Search. 8, 106, 115, 129, 141–145, 153–155, 157

ISA Instance Space Analysis. 5, 10, 33, 35, 36, 39, 40, 50, 104, 155, 156, 159, 160

LLH Low-level Heuristic. 5, 7, 8, 106, 107, 113–117, 119–121, 128–131, 133, 134, 136, 138–142, 147, 149, 153–155, 157, 160, 161

MIP Mixed Integer Programming. 4, 6, 12, 52, 62, 63, 68, 69, 90, 156, 158

MSD Minimum Shift Design. 6, 51–53, 58, 69, 106, 109, 112, 119, 125, 128, 129, 141, 147, 150, 151, 153, 156–160

PS Personnel Scheduling. 114, 143–145

RCSPP Resource Constrained Shortest Path Problem. 7, 72, 73, 89, 92, 97, 99, 103, 106, 159, 160

RMP Restricted Master Problem. 90, 97, 102

RWS Rotating Workforce Scheduling. 5, 6, 9–14, 32, 33, 35, 36, 39, 49, 50, 53, 54, 69, 106, 109, 111, 128, 129, 147, 152–155, 157, 158, 160, 161

SA Simulated Annealing. 71, 72, 80, 82–84, 86, 88, 89, 92, 100, 103, 120, 126, 150, 156, 157, 160

SAT MaxSAT. 114, 143–145

SPA Shift Plan Assistant. 9, 23, 32, 159

TSP Travelling Salesperson Problem. 114, 143–145

VRP Vehicle Routing Problem. 114, 143–146

Bibliography

[ABIG+19] Anna Arlinghaus, Philip Bohle, Irena Iskra-Golec, Nicole Jansen, Sarah Jay, and Lucia Rotenberg. Working time society consensus statements: Evidence-based effects of shift work and non-standard working hours on workers, family and community. *Industrial Health*, 57(2):184–200, 2019.

[ABN14] Steven Adriaensen, Tim Brys, and Ann Nowé. Fair-share ILS: A simple state-of-the-art iterated local search hyperheuristic. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 1303–1310, New York, NY, USA, 2014. Association for Computing Machinery.

[AGM+16] Michael Abseher, Martin Gebser, Nysret Musliu, Torsten Schaub, and Stefan Woltran. Shift Design with Answer Set Programming. *Fundamenta Informaticae*, 147(1):1–25, November 2016.

[Alf04] Hesham K. Alfares. Survey, Categorization, and Comparison of Recent Tour Scheduling Literature. *Annals of Operations Research*, 127:145–175, March 2004.

[AN16] Steven Adriaensen and Ann Nowé. Case study: An analysis of accidental complexity in a state-of-the-art hyper-heuristic for HyFlex. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 1485–1492. IEEE, 2016.

[AON15] Steven Adriaensen, Gabriela Ochoa, and Ann Nowé. A benchmark set extension and comparative study for the HyFlex framework. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 784–791. IEEE, 2015.

[AOO21] Stephen A. Adubi, Olufunke O. Oladipupo, and Oludayo O. Olugbara. Configuring the perturbation operations of an iterated local search algorithm for cross-domain search: A probabilistic learning approach. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1372–1379. IEEE, 2021.

[APU18] Arjan Akkermans, Gerhard Post, and Marc Uetz. Solving the shifts and breaks design problem using integer linear programming. In *PATAT 2018: Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling*, pages 137–152, 2018.

[Ayk00] Turgut Aykin. A comparative evaluation of modeling approaches to the labor shift scheduling problem. *European Journal of Operational Research*, 125(2):381–397, 2000.

[Bak76] Kenneth R. Baker. Workforce allocation in cyclical scheduling problems: A survey. *Journal of the Operational Research Society*, 27(1):155–167, 1976.

[BDCVBVL04] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The State of the Art of Nurse Rostering. *Journal of Scheduling*, 7(6):441–499, November 2004.

[Ben80] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[BGH+11] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Barry McCollum, Gabriela Ochoa, Andrew J. Parkes, and Sanja Petrovic. The cross-domain heuristic search challenge – An international research competition. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 631–634, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BGH+13] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[BGM+08] Andreas Beer, Johannes Gaertner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. Scheduling Breaks in Shift Plans for Call Centers. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[BGM+10] Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. An AI-Based Break-Scheduling System for Supervisory Personnel. *IEEE Intelligent Systems*, 25(2):60–73, March 2010.

[BHK+10] Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*, pages 449–468. Springer, 2010.

[BHK+19] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. A classification of hyper-heuristic approaches: Revisited. In *Handbook of Metaheuristics*, pages 453–477. Springer, 2019.

[BJ90] Stephen E. Bechtold and Larry W. Jacobs. Implicit modelling of flexible break assignments in optimal shift scheduling. *Management Science*, 36(11):1339–1351, 1990.

[BJN+98] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W.P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

[BKP08] Sandjai Bhulai, Ger Koole, and Auke Pot. Simple methods for shift scheduling in multiskill call centers. *Manufacturing & Service Operations Management*, 10(3):411–420, 2008.

[BP76] Egon Balas and Manfred W. Padberg. Set partitioning: A survey. *SIAM Review*, 18(4):710–760, 1976.

[BQB11] Peter Brucker, Rong Qu, and Edmund Burke. Personnel scheduling: Models and complexity. *European Journal of Operational Research*, 210(3):467–473, May 2011.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[BSW22] Tristan Becker, Maximilian Schiffer, and Grit Walther. A general branch-and-cut framework for rotating workforce scheduling. *INFORMS Journal on Computing*, 34(3):1548–1564, 2022.

[BW90] Nagraj Balakrishnan and Richard T. Wong. A network model for the rotating workforce scheduling problem. *Networks*, 20(1):25–42, 1990.

[CdMNdA+17] Ademir Aparecido Constantino, Candido Ferreira Xavier de Mendonça Neto, Silvio Alexandre de Araujo, Dario Landa-Silva, Rogério Calvi, and Allainclair Flausino dos Santos. Solving a large real-world bus driver scheduling problem with a multi-assignment based heuristic algorithm. *Journal of Universal Computer Science*, 23(5):479–504, 2017.

[CGQR11] Marie-Claude Côté, Bernard Gendron, Claude-Guy Quimper, and Louis-Martin Rousseau. Formal languages for integer programming modeling of shift scheduling problems. *Constraints*, 16(1):55–76, 2011.

[CGR11] Marie-Claude Côté, Bernard Gendron, and Louis-Martin Rousseau. Grammar-based integer programming models for multiactivity shift scheduling. *Management Science*, 57(1):151–163, 2011.

[CGR13] Marie-Claude Côté, Bernard Gendron, and Louis-Martin Rousseau. Grammar-Based Column Generation for Personalized Multi-Activity Shift Scheduling. *INFORMS Journal on Computing*, 25(3):461–474, August 2013.

[CHT12] Wei-Mei Chen, Hsien-Kuei Hwang, and Tsung-Hsi Tsai. Maxima-finding algorithms for multidimensional samples: A two-phase approach. *Computational Geometry*, 45(1-2):33–53, 2012.

[Chu20] Chung-Yao Chuang. *Combining Multiple Heuristics: Studies on Neighborhood-base Heuristics and Sampling-based Heuristics*. PhD thesis, Carnegie Mellon University, 2020.

[CKS01] Peter Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In Edmund Burke and Wilhelm Erben, editors, *Practice and Theory of Automated Timetabling III*, pages 176–190, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[CL96] Hoong Chuin Lau. On the complexity of manpower shift scheduling. *Computers & Operations Research*, 23(1):93–102, 1996.

[CPL09] IBM ILOG CPLEX. V12.1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

[CSS+18] Geoffrey Chu, Peter J. Stuckey, Andreas Schutt, Thorsten Ehlers, Graeme Gange, and Kathryn Francis. Chuffed, a lazy clause generation solver. `https://github.com/chuffed/chuffed`, 2018.

[CSSC13] Shijun Chen, Yindong Shen, Xuan Su, and Heming Chen. A Crew Scheduling with Chinese Meal Break Rules. *Journal of Transportation Systems Engineering and Information Technology*, 13(2):90–95, April 2013.

[CWL18] Shin Siang Choong, Li-Pei Wong, and Chee Peng Lim. Automatic design of hyper-heuristic based on reinforcement learning. *Information Sciences*, 436:89–107, 2018.

[Dan54] George B. Dantzig. Letter to the editor–a comment on Edie's "Traffic delays at toll booths". *Journal of the Operations Research Society of America*, 2(3):339–341, 1954.

[DBVdBBD15] Philippe De Bruecker, Jorne Van den Bergh, Jeroen Beliën, and Erik Demeulemeester. Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243(1):1–16, May 2015.

[DGGK+07] Luca Di Gaspero, Johannes Gärtner, Guy Kortsarz, Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. The minimum shift design problem. *Annals of Operations Research*, 155(1):79–105, August 2007.

[DGGM+10] Luca Di Gaspero, Johannes Gärtner, Nysret Musliu, Andrea Schaerf, Werner Schafhauser, and Wolfgang Slany. A Hybrid LS-CP Solver for the Shifts and Breaks Design Problem. In María J. Blesa, Christian Blum, Günther Raidl, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics*, volume 6373, pages 46–61. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[DGGM+13] Luca Di Gaspero, Johannes Gärtner, Nysret Musliu, Andrea Schaerf, Werner Schafhauser, and Wolfgang Slany. Automated Shift Design and Break Scheduling. In A. Sima Uyar, Ender Ozcan, and Neil Urquhart, editors, *Automated Scheduling and Planning*, volume 505, pages 109–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[DGU12] Luca Di Gaspero and Tommaso Urli. Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In *International Conference on Learning and Intelligent Optimization*, pages 384–389. Springer, 2012.

[DKÖB20] John H. Drake, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020.

[DLFM11] Renato De Leone, Paola Festa, and Emilia Marchitto. A Bus Driver Scheduling Problem: A new mathematical model and a GRASP approximate solution. *Journal of Heuristics*, 17(4):441–466, August 2011.

[DS89] Martin Desrochers and François Soumis. A Column Generation Approach to the Urban Transit Crew Scheduling Problem. *Transportation Science*, 23(1):1–13, February 1989.

[EJK+04] Andreas T. Ernst, Houyuan Jiang, Mohan Krishnamoorthy, Bowie Owens, and David Sier. An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127(1):21–144, 2004.

[EJKS04] A.T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, February 2004.

[EM17] Christoph Erkinger and Nysret Musliu. Personnel scheduling as satisfiability modulo theories. In *International Joint Conference on Artificial Intelligence – IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 614–621, 2017.

[Eur] Eurostat. Employees working shifts as a percentage of the total of employees, by sex and age (%) - Products Datasets - Eurostat. `https://ec.europa.eu/eurostat/web/products-datasets/-/lfsa_ewpshi`. Accessed on 2022-02-11.

[FBCL16] Raúl Falcón, Eva Barrena, David Canca, and Gilbert Laporte. Counting and enumerating feasible rotating schedules by means of Gröbner bases. *Mathematics and Computers in Simulation*, 125:139–151, 2016.

[FGP15] Alexandre Silvestre Ferreira, Richard Aderbal Goncalves, and Aurora Trinidad Ramirez Pozo. A multi-armed bandit hyper-heuristic. In *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, pages 13–18. IEEE, 2015.

[GJ05] Balaji Gopalakrishnan and Ellis L. Johnson. Airline Crew Scheduling: State-of-the-Art. *Annals of Operations Research*, 140(1):305–337, November 2005.

[GKK+21] Tobias Geibinger, Lucas Kletzander, Matthias Krainz, Florian Mischek, Nysret Musliu, and Felix Winter. Physician scheduling during a pandemic. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 456–465. Springer, 2021.

[GM86] Fred Glover and Claude McMillan. The general employee scheduling problem. An integration of MS and AI. *Computers & Operations Research*, 13(5):563–573, January 1986.

[GMS01] Johannes Gärtner, Nysret Musliu, and Wolfgang Slany. Rota: A research project on algorithms for workforce scheduling and shift design optimization. *AI Communications*, 14(2):83–92, 2001.

[GMSS11] Johannes Gärtner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. TEMPLE – A domain specific language for modeling and solving staff scheduling problems. In *2011 IEEE Symposium on Computational Intelligence in Scheduling (SCIS)*, pages 58–64. IEEE, 2011.

[GO18] LLC Gurobi Optimization. Gurobi Optimizer reference manual. `http://www.gurobi.com`, 2018.

[HCF12] Ping-Che Hsiao, Tsung-Che Chiang, and Li-Chen Fu. A VNS-based hyper-heuristic with adaptive computational budget of local search. In *2012 IEEE congress on evolutionary computation*. IEEE, 2012.

[HLBSAR19] Noberto A. Hernández-Leandro, Vincent Boyer, M. Angélica Salazar-Aguilar, and Louis-Martin Rousseau. A matheuristic based on Lagrangian relaxation for the multi-activity shift scheduling problem. *European Journal of Operational Research*, 272(3):859–867, February 2019.

[ID05] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

[IRDGM15] O.J. Ibarra-Rojas, F. Delgado, R. Giesen, and J.C. Muñoz. Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological*, 77:38–75, July 2015.

[Irn08] Stefan Irnich. Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, 30(1):113–148, 2008.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

[KHSM17] Yanfei Kang, Rob J. Hyndman, and Kate Smith-Miles. Visualising forecasting algorithm performance using time series instance spaces. *International Journal of Forecasting*, 33(2):345–358, 2017.

[KK15] Ahmed Kheiri and Ed Keedwell. A sequence-based selection hyper-heuristic utilising a hidden Markov model. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 417–424, New York, NY, USA, 2015. Association for Computing Machinery.

[KM19] Lucas Kletzander and Nysret Musliu. Modelling and solving the minimum shift design problem. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 391–408. Springer, 2019.

[KM20a] Lucas Kletzander and Nysret Musliu. Scheduling bus drivers in real-life multi-objective scenarios with break constraints. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT 2021*, volume 1, pages 34–40, 2020.

[KM20b] Lucas Kletzander and Nysret Musliu. Solving large real-life bus driver scheduling problems with complex break constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 421–429, 2020.

[KM20c] Lucas Kletzander and Nysret Musliu. Solving the general employee scheduling problem. *Computers & Operations Research*, 113:104794, 2020.

[KM22] Lucas Kletzander and Nysret Musliu. Hyper-heuristics for personnel scheduling domains. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pages 462–470, 2022.

[KMG+19] Lucas Kletzander, Nysret Musliu, Johannes Gärtner, Thomas Krennwallner, and Werner Schafhauser. Exact methods for extended rotating workforce scheduling problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 519–527, 2019.

[KMM22] Lucas Kletzander, Tommaso Mannelli Mazzoli, and Nysret Musliu. Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 232–240, 2022.

[KMSM21] Lucas Kletzander, Nysret Musliu, and Kate Smith-Miles. Instance space analysis for a personnel scheduling problem. *Annals of Mathematics and Artificial Intelligence*, 89(7):617–637, 2021.

[KMVH21] Lucas Kletzander, Nysret Musliu, and Pascal Van Hentenryck. Branch and price for bus driver scheduling with complex break constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11853–11861, 2021.

[KÖ16] Ahmed Kheiri and Ender Özcan. An iterated multi-stage selection hyperheuristic. *European Journal of Operational Research*, 250(1):77–90, 2016.

[Koc15] Deniz Kocabas. Exact Methods for Shift Design and Break Scheduling. Master's thesis, Technische Universität Wien, 2015.

[KW07] Deborah L. Kellogg and Steven Walczak. Nurse scheduling: From academia to implementation or not? *Interfaces*, 37(4):355–369, 2007.

[Lap99] G. Laporte. The art and science of designing rotating schedules. *Journal of the Operational Research Society*, 50:1011–1017, September 1999.

[Lar11] Mathieu Larose. A hyper-heuristic for the CHeSC 2011. *CHeSC2011 Competition*, 2011.

[LBD+12] Quentin Lequy, Mathieu Bouchard, Guy Desaulniers, François Soumis, and Beyime Tachefine. Assigning multiple activities to work shifts. *Journal of Scheduling*, 15(2):239–251, April 2012.

[LEF+22] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[LG07] Philippe Laborie and Daniel Godard. Self-adapting large neighborhood search: Application to single-mode scheduling problems. In *Multidisciplinary International Scheduling Conference MISTA-07, Paris*, 2007.

[LH16] Dung-Ying Lin and Ching-Lan Hsu. A column generation algorithm for the bus driver scheduling problem. *Journal of Advanced Transportation*, 50(8):1598–1615, December 2016.

[LK03] Jingpeng Li and Raymond S.K. Kwan. A fuzzy genetic algorithm for driver scheduling. *European Journal of Operational Research*, 147(2):334–344, June 2003.

[LM12] Andreas Lehrbaum and Nysret Musliu. A new hyperheuristic algorithm for cross-domain search problems. In *International Conference on Learning and Intelligent Optimization*, pages 437–442. Springer, 2012.

[LMS03] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated Local Search. In *Handbook of Metaheuristics*, pages 320–353. Springer, 2003.

[LNB80] Gilbert Laporte, Yves Nobert, and Jean Biron. Rotating schedules. *European Journal of Operational Research*, 4(1):24–30, 1980.

[LP04] Gilbert Laporte and Gilles Pesant. A general multi-shift scheduling system. *Journal of the Operational Research Society*, 55(11):1208–1217, 2004.

[LPP01] Helena R. Lourenço, José P. Paixão, and Rita Portugal. Multiobjective Metaheuristics for the Bus Driver Scheduling Problem. *Transportation Science*, 35(3):331–343, August 2001.

[LSZ93] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

[Mac16] James MacGlashan. Burlap: Brown-UMBC reinforcement learning and planning. `http://burlap.cs.brown.edu/index.html`, 2016.

[MGS02] Nysret Musliu, Johannes Gärtner, and Wolfgang Slany. Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2):85–98, 2002.

[MJS21] Adrien Maillard, Marijke Jorritsma, and Steve Schaffer. Sailing towards an expressive scheduling language for europa clipper. In *Workshop on Knowledge Engineering for Planning and Scheduling at ICAPS 2021*, 2021.

[MKC10] David Meignan, Abderrafiaa Koukam, and Jean-Charles Créput. Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, 2010.

[MM21] Florian Mischek and Nysret Musliu. A collection of hyper-heuristics for the HyFlex framework. Technical Report CD-TR 2021/1, TU Wien, 2021.

[MM22] Florian Mischek and Nysret Musliu. Reinforcement learning for cross-domain hyper-heuristics. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 4793–4799. International Joint Conferences on Artificial Intelligence Organization, July 2022. Main Track.

[MMS+19] Claudia R.C. Moreno, Elaine C. Marqueze, Charli Sargent, Kenneth P. Wright Jr, Sally A. Ferguson, and Philip Tucker. Working time society consensus statements: Evidence-based effects of shift work on physical and mental health. *Industrial Health*, 57(2):139–157, 2019.

[MSM17] Mario A. Muñoz and Kate A. Smith-Miles. Performance analysis of continuous black-box optimization algorithms via footprints in instance space. *Evolutionary Computation*, 25(4):529–554, 2017.

[MSS04] Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. Local search for shift design. *European Journal of Operational Research*, 153(1):51–64, February 2004.

[MSS18] Nysret Musliu, Andreas Schutt, and Peter J Stuckey. Solver independent rotating workforce scheduling. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 429–445. Springer, 2018.

[MT86] Silvano Martello and Paolo Toth. A heuristic approach to the bus driver scheduling problem. *European Journal of Operational Research*, 24(1):106–117, January 1986.

[Mus05] Nysret Musliu. Combination of local search strategies for rotating workforce scheduling problem. In *International Joint Conference on Artificial Intelligence – IJCAI 2005, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 1529–1530, 2005.

[Mus06] Nysret Musliu. Heuristic methods for automatic rotating workforce scheduling. *International Journal of Computational Intelligence Research*, 2(4):309–326, 2006.

[MVBSM18] Mario A. Muñoz, Laura Villanova, Davaatseren Baatar, and Kate Smith-Miles. Instance spaces for machine learning classification. *Machine Learning*, 107(1):109–147, 2018.

[MVDCVB12] Mustafa Mısır, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. An intelligent hyper-heuristic framework for CHeSC 2011. In *International Conference on Learning and Intelligent Optimization*, pages 461–466. Springer, 2012.

[MVDCVB13] Mustafa Mısır, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. A new hyper-heuristic as a general problem solver: An implementation in HyFlex. *Journal of Scheduling*, 16(3):291–311, 2013.

[NSB+07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer Berlin Heidelberg, 2007.

[OAGSM18] Carlos Oliveira, Aldeida Aleti, Lars Grunske, and Kate Smith-Miles. Mapping the effectiveness of automated test suite generation techniques. *IEEE Transactions on Reliability*, 67(3):771–785, 2018.

[OHC+12] Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A. Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J. Parkes, Sanja Petrovic, et al. HyFlex: A benchmark framework for cross-domain heuristic search. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 136–147. Springer, 2012.

[OWHC12] Gabriela Ochoa, James Walker, Matthew Hyde, and Tim Curtois. Adaptive evolutionary algorithms and extensions to the HyFlex hyper-heuristic framework. In *International Conference on Parallel Problem Solving from Nature*, pages 418–427. Springer, 2012.

[Pet19] Sanja Petrovic. "You have to get wet to learn how to swim" applied to bridging the gap between research into personnel scheduling and its implementation in practice. *Annals of Operations Research*, 275(1):161–179, 2019.

[PLP09] Rita Portugal, Helena R. Lourenço, and José P. Paixão. Driver scheduling problem modelling. *Public Transport*, 1(2):103–120, June 2009.

[PVB12] Sanja Petrovic and Greet Vanden Berghe. A comparison of two approaches to nurse rostering problems. *Annals of Operations Research*, 194(1):365–384, 2012.

[QR10] Claude-Guy Quimper and Louis-Martin Rousseau. A large neighbourhood search approach to the multi-activity shift scheduling problem. *Journal of Heuristics*, 16(3):373–391, 2010.

[RGR16] María I. Restrepo, Bernard Gendron, and Louis-Martin Rousseau. Branch-and-price for personalized multiactivity tour scheduling. *INFORMS Journal on Computing*, 28(2):334–350, 2016.

[Ric76] John R. Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. Elsevier, 1976.

[RLM12] María I. Restrepo, Leonardo Lozano, and Andrés L. Medaglia. Constrained network-based column generation for the multi-activity shift scheduling problem. *International Journal of Production Economics*, 140(1):466–472, November 2012.

[RMVP13] Ana Respício, Margarida Moz, and Margarida Vaz Pato. Enhanced genetic algorithms for a bi-objective bus driver rostering problem: a computational study. *International Transactions in Operational Research*, 20(4):443–470, 2013.

[SAKQ14] Nasser R. Sabar, Masri Ayob, Graham Kendall, and Rong Qu. Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(3):309–325, 2014.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SK01] Yindong Shen and Raymond S.K. Kwan. Tabu Search for Driver Scheduling. In G. Fandel, W. Trockel, C. D. Aliprantis, Dan Kovenock, Stefan Voß, and Joachim R. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505, pages 121–135. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[SK15] Nasser R. Sabar and Graham Kendall. Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems. *Information Sciences*, 314:225–239, 2015.

[SM09] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1), January 2009.

[SMB15] Kate Smith-Miles and Simon Bowly. Generating new test instances by evolving in instance space. *Computers & Operations Research*, 63:102–113, 2015.

[SMBWL14] Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, 2014.

[SML12] Kate Smith-Miles and Leo Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research*, 39(5):875–889, 2012.

[SMM21] Kate Smith-Miles and Mario Andrés Muñoz. Instance space analysis for algorithm testing: Methodology and software tools. http://dx.doi.org/10.13140/RG.2.2.33951.48805, May 2021.

[SN19] Julian Schulte and Volker Nissen. Holistic workforce planning: Integrating staffing, shift design and scheduling using evolutionary bilevel optimization. In *Multidisciplinary International Scheduling Conference MISTA 2019*, 2019.

[SW88] Barbara M. Smith and Anthony Wren. A bus crew scheduling system using a set covering formulation. *Transportation Research Part A: General*, 22(2):97–108, March 1988.

[Tho95] Gary M. Thompson. Improved implicit optimal modeling of the labor shift scheduling problem. *Management Science*, 41(4):595–607, 1995.

[TK13] Attila Tóth and Miklós Krész. An efficient solution approach for real-world driver scheduling problems in urban bus transportation. *Central European Journal of Operations Research*, 21(S1):75–94, June 2013.

[TM11] Markus Triska and Nysret Musliu. A constraint programming application for rotating workforce scheduling. In *Developing Concepts in Applied Intelligence*, volume 363 of *Studies in Computational Intelligence*, pages 83–88. Springer Berlin / Heidelberg, 2011.

[Tol12] Georgi P. Tolstov. *Fourier series*. Courier Corporation, 2012.

[TS18] Charles Thomas and Pierre Schaus. Revisiting the self-adaptive large neighborhood search. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 557–566. Springer, 2018.

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

[WKO19] WKO.at. Kollektivvertrag für Dienstnehmer in privaten Autobusbetrieben gültig ab 1.1.2019. https://www.wko.at/service/kollektivvertrag/kv-private-autobusbetriebe-2019.html, 2019. Accessed 8 Jul. 2019.

[WM14] Magdalena Widl and Nysret Musliu. The break scheduling problem: Complexity results and practical algorithms. *Memetic Computing*, 6(2):97–112, June 2014.

[WR95] Anthony Wren and Jean-Marc Rousseau. Bus Driver Scheduling — An Overview. In G. Fandel, W. Trockel, Joachim R. Daduna, Isabel Branco, and José M. Pinto Paixão, editors, *Computer-Aided Transit Scheduling*, volume 430, pages 173–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[ZBQ+22] Yuchang Zhang, Ruibin Bai, Rong Qu, Chaofan Tu, and Jiahuan Jin. A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. *European Journal of Operational Research*, 300(2):418–427, 2022.