TU WIEN Informatics

Optimizing Constraint Programming for Real World Scheduling of Test Laboratories

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Philipp Danzinger, BSc
Matrikelnummer 11775797

an der Fakultät für Informatik

der Technischen Universität Wien

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

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Wien, 8. Mai 2024

Philipp Danzinger

Nysret Musliu

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Optimizing Constraint Programming for Real World Scheduling of Test Laboratories

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Philipp Danzinger, BSc
Registration Number 11775797

to the Faculty of Informatics

at the TU Wien

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

Vienna, May 8, 2024

Philipp Danzinger

Nysret Musliu

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Philipp Danzinger, 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, 8. Mai 2024

Philipp Danzinger

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to express my gratitude for my supervisor, Associate Prof. Dr. Nysret Musliu, for his invaluable guidance and encouragement throughout this research. My heartfelt thanks go to Dr. Florian Mischek, who always provided valuable help and offered key insights. In addition, I want to thank my other colleagues at TU Wien for their helpful discussions and insights, and those at our industrial partner for providing the real-world requirements and feedback that guided this work.

Finally, I am deeply appreciative of the unwavering support of my family throughout this journey.

This work was financially supported by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association. Their support is gratefully acknowledged.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Das Test Laboratory Scheduling Problem (TLSP) ist ein kürzlich vorgeschlagenes Zeitplanungsproblem, das auf den realen Zeitplanungsanforderungen eines industriellen Testlabors basiert. TLSP ist mit dem bekannten Resource Constrained Project Scheduling Problem (RCPSP) verwandt.

Eine Erweiterung von TLSP im Vergleich zum RCPSP ist die Unterteilung von Ressourcen in drei Kategorien: Angestellte, Arbeitsbänke und Ausrüstung, die jeweils unterschiedliche Eigenschaften und Einschränkungen aufweisen. Im Laufe der Zeit stellten neue reale Zeitplanungsanforderungen diese Taxonomie in Frage.

Der erste Teil dieser Arbeit behandelt dieses Problem, indem eine neue Problemvariante vorgeschlagen wird, TLSP with Generalized Resources (TLSP-GR), die diese Ressourcentypen in ein einziges Konzept vereinheitlicht. Dies löst sofort einige der neuen Planungsanforderungen und bietet eine elegante Basis für die Implementierung weiterer. Zur Lösung des TLSP-GR wird ein Constraint-Programming-Modell entwickelt. Durch sorgfältige Generalisierung und Optimierung des neuen Modells bleibt dessen Leistung weiterhin konkurrenzfähig mit dem bestehenden State-of-the-Art-Modell für TLSP.

Der zweite Teil dieser Arbeit befasst sich mit der Erweiterung des CP-SAT-Solvers aus der OR-Tools-Suite von Google, um dessen Leistung bei komplexen Zeitplanungsproblemen wie TLSP zu verbessern. Obwohl CP-SAT für seine ausgezeichnete Leistung bekannt ist und regelmäßig Goldmedaillen bei der MiniZinc Challenge gewinnt, hinkte seine Leistung für TLSP-Instanzen bisher hinter dem akademischen Chuffed-Solver hinterher.

Um diese Lücke zu schließen, wird CP-SAT erweitert, um Priority Search und Suchstrategien mit zufälliger Variablenauswahl zu unterstützen, und es wird eine Hot-Start-Unterstützung zu seinem MiniZinc-Backend hinzugefügt. Obwohl die Verbesserungen durch Priority Search überraschenderweise durch eine Suchstrategie repliziert werden können, die diese Funktion nicht verwendet, machen die Erweiterungen insgesamt den Solver zu einer kompetitiven Option für TLSP, die mit den state-of-the-art Ergebnissen übereinstimmt, wenn der Solver in einen bestehenden VLNS-Algorithmus für TLSP eingebaut wird.

Mit den Methoden aus dieser Arbeit werden 4 neue Optimalitätsergebnisse für TLSP erzielt. Darüber hinaus werden neue Bestmarken für 17 von 33 Testinstanzen erreicht.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

The Test Laboratory Scheduling Problem (TLSP) is a recently proposed complex scheduling problem based on the real-world scheduling requirements of an industrial test laboratory. TLSP is related to the well-known Resource Constrained Project Scheduling Problem (RCPSP).

One extension in TLSP compared to the RCPSP is the division of resources into three categories: employees, workbenches, and equipment, each with different properties and constraints. Over time, new real-world scheduling requirements put this taxonomy into question.

The first part of this Thesis addresses this by proposing a new problem variant, TLSP with Generalized Resources (TLSP-GR), which unifies these resource types into a single concept. This immediately solves some of the new scheduling requirements and provides an elegant basis for implementing others. To solve TLSP-GR, a new Constraint Programming (CP) model is proposed. Through careful generalization and optimization of the new model, its performance on TLSP instances is still competitive with the existing state-of-the-art model for TLSP.

The second part of this Thesis concerns extending the CP-SAT solver from Google's OR-Tools suite to improve its performance on complex scheduling problems like TLSP. While CP-SAT is known for its excellent performance, routinely winning gold medals at the MiniZinc Challenge, its performance lagged behind the academic Chuffed solver on TLSP instances.

In an attempt to close this gap, CP-SAT is extended to support priority search and random variable selection, and hot-start support is added to its MiniZinc backend. While the improvements from priority search can surprisingly be replicated through a search strategy not using this feature, the overall extensions to the solver make it a viable option for TLSP, matching state-of-the-art results when included in an existing VLNS algorithm for TLSP.

Using the methods from this Thesis, 4 new optimality results are achieved for TLSP. In addition, new state-of-the-art penalties are achieved for 17 out of 33 test instances.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

The Test Laboratory Scheduling Problem (TLSP) [MM18b] is a recently introduced NP-complete scheduling problem that is based on the scheduling requirements of a real-world industrial test laboratory. TLSP is similar to the well-known Resource-Constrained Project Scheduling Problem (RCPSP) [BDM+99], albeit with several extensions. For instance, tasks need to be grouped into jobs before they can be scheduled. There are multiple projects to schedule, like in [WKS+14], jobs can be scheduled in different modes similarly to [TPM09], and there are heterogeneous resources like in [DPRL98]. Of particular relevance to this Thesis, resources are further split into employees, workbenches, and equipment, each group having different constraints and properties. To give one example, the number of required employees depends only on the chosen mode, while the required number of workbenches and equipment depends on the tasks themselves. TLSP solvers have been successfully deployed for daily scheduling operations in the laboratory of our industrial partner. Over time, new scheduling requirements emerged in the industrial partner's laboratory that went beyond the current TLSP formulation, and other test laboratories started showing interest. These requirements necessitate extending TLSP and adding new constraints.

Many of these new requirements are related to employees, workbenches, and equipment. Directly extending TLSP with new constraints to cover these requirements would have been quite cumbersome and involved duplicated effort for the different types of resources. It seemed more worthwhile to unify employees, workbenches, and equipment into a single concept called resources, which would allow the new requirements to be modeled more simply and with fewer added constraints. The first part of this Thesis concerns performing this generalization and solving it with a Constraint Programming (CP) model.

One state-of-the-art solution method for TLSP is a VLNS algorithm whose neighborhood can be explored by solving small subproblems with CP, as proposed in [DGMM20]. Currently by far the most efficient way to solve these subproblems is using the Chuffed solver [Chu11] with a custom search strategy based on priority search. Priority search is

a proposed extension to the search annotations in the MiniZinc constraint programming language that allows the solver to switch between different specified search strategies based on the bounds of arbitrary variables. The CP-SAT [PD21] solver (part of Google's OR-Tools software suite) has shown excellent performance in the MiniZinc Challenge [SFS+14] in recent years, routinely winning first place and beating both open-source and commercial solvers. However, in past experiments, CP-SAT has shown mediocre performance for TLSP. This might be due to its lack of support for priority search, which has been crucial for achieving high performance on TLSP using the Chuffed solver. The second part of this Thesis revolves around extending CP-SAT to improve its performance for TLSP and similar problems.

1.1 Aim of the Thesis

The aim of this Thesis is twofold:

1. To find a way to generalize TLSP such that the different types of resources are unified and that the problem can still be solved efficiently.

2. To investigate the impact of *priority search* on the performance of CP-SAT for TLSP and potentially find other improvements that could make the solver more competitive for TLSP.

To achieve this, I use the following methodology:

1. Generalizing resources in TLSP involves creating a new problem variant, TLSP with Generalized Resources (TLSP-GR), that merges employees, workbenches, and equipment into a single concept called resources, while still being able to model the original TLSP requirements. Then, a new constraint programming model for TLSP-GR needs to be created. To ascertain the performance in real-world use, an existing state-of-the-art VLNS algorithm for TLSP should be adapted to use the new CP model. Finally, the new CP model and the adapted VLNS algorithm need to be compared to existing state of the art for solving TLSP instances.

2. Extending CP-SAT with priority search: first, this involves analyzing the source code of CP-SAT to find out how user-defined search strategies are implemented and how priority search could be implemented. If possible, priority search needs to be implemented. To fully support the TLSP VLNS algorithm, randomized variable selection and hot start should be implemented as well. Then, the modified CP-SAT solver should be evaluated using both the CP model and the VLNS algorithm. The results should be compared to an unmodified version of CP-SAT, other solvers, and the state of the art solution methods for TLSP.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

1.2 Contributions

- This Thesis proposes *TLSP with Generalized Resources* (TLSP-GR), a new problem variant of TLSP that is more general and can adapt to more real-world scheduling requirements.
- The Thesis further proposes a Constraint Programming model for TLSP-GR and shows competitive performance with the state-of-the-art on existing TLSP instances.
- This Thesis presents an implementation of the priority search feature in Google's CP-SAT solver, showing the necessary changes to internal data structures and algorithms. Additional features, hot start, and random variable selection are also considered.
- Using the aforementioned methods, this Thesis presents some new state-of-the-art results for TLSP instances. To my knowledge, only 2 of the 33 publicly available TLSP instances have been solved to optimality. Using the solution approaches in this Thesis, I found optimal solutions for 4 additional instances, and, to my knowledge, new state-of-the-art penalties for 17 other instances.

1.3 Structure of the Thesis

This Thesis is structured as follows:

- Chapter 2 provides background information on Constraint Programming (CP), CP search strategies as implemented in the MiniZinc language, the Test Laboratory Scheduling Problem (TLSP), and the Very Large Neighborhood Search (VLNS) algorithm for TLSP.
- Chapter 3 introduces the new problem variant TLSP with Generalized Resources (TLSP-GR). This includes the problem description and a formal specification of the constraints. Then, a problem reduction from TLSP is described that shows how TLSP instances can be expressed as TLSP-GR instances. Then, a Constraint Programming model for TLSP-GR is proposed. Finally, some performance optimizations for the CP model are discussed.
- Chapter 4 describes how user-defined search strategies are implemented in Google's CP-SAT solver and what changes to its data structures and algorithms are necessary to implement priority search. It also covers the implementation of hot start and random variable selection features.
- Chapter 5 presents experimental results to evaluate the performance of the methods proposed in earlier chapters. First, the performance of the new CP model for TLSP-GR is compared to the existing state-of-the-art TLSP model, both on its own and as part of the VLNS algorithm. Next, the impact of priority search and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the other modifications to CP-SAT is analyzed, again using pure CP as well as the VLNS algorithm to solve TLSP instances. Finally, the results are compared to the state of the art results on TLSP.

- Chapter 6 concludes the Thesis. It summarizes the main results and contributions and discusses potential future work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Background and Related Work

This Chapter covers the background and existing research relating to this Thesis. Section 2.1 covers Constraint Programming (CP) in general, then Sections 2.2 and 2.3 go into detail about search strategies and their implementation in the constraint programming language MiniZinc. Next, Section 2.4 covers the existing research on the Test Laboratory Scheduling Problem (TLSP). Finally, Section 2.5 covers the Very Large Neighborhood Search (VLNS) algorithm that has been proposed for TLSP and represents a state-of-the-art solution approach.

2.1 Constraint Programming

Constraint Programming (CP) is a declarative programming paradigm based on the NP-complete *Constraint Satisfaction Problem* (CSP). A CSP instance consists of a set of variables with domains, and constraints. Historically, constraint programming has early roots in interdisciplinary applications in the 60s [Sut63], taking more shape throughout the 70s [Mon74] [Wal75]. Based on [Bar99], a constraint programming instance formally consists of:

- A set X of variables, $X = \{x_1, \ldots, x_n\}$.
- For each variable, a finite set D_i called its domain.
- A set of constraints, each restricting the possible assignments of one or more variables. Each constraint is described by a subset of the cartesian product of the domains of the variables it restricts.

A solution to a CSP instance is an assignment that assigns each variable some value from its domain, such that every constraint is satisfied.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Constraint programming is not limited to satisfiability and can also be used for optimization. A *Constraint Satisfaction Optimization Problem* (CSOP), or just *Constraint Optimization Problem* (COP), is a CSP with an additional objective function that maps every solution to a numerical objective value to minimize (or maximize). The solution to a CSOP is any valid solution to the underlying CSP whose objective value is minimal (or maximal).

Solvers for these problems usually work by combining propagation and search. Propagation means using the constraints to make logically valid inferences that restrict the possible domains of variables. Since propagation on its own is virtually never enough, the solver also has to search the space of possible solutions by branching on variables. Branching often happens on concrete variable assignments, but can, in principle, happen on any domain restrictions.

To increase performance, optimized data structures and heuristics like VSIDS [MMZ+01] can be employed. Many modern solvers, like Chuffed [Chu11] and CP-SAT [PD21] (the main CP solver included in Google's OR-Tools suite) also employ SAT-solving techniques, encoding some propositions about the variables as boolean variables. This allows them to use techniques like conflict-driven clause learning.

When the solver needs to make a decision that further constrains a variable, the decision can be purely based on heuristics, or guided by a user-specified search strategy. Search strategies that encode human intuitions about what a 'promising' or 'good' solution looks like can sometimes result in a big performance improvement. MiniZinc [NSB+07] is a constraint programming language that supports a variety of solvers and allows the programmer to specify basic rules about the variable and value selection strategies. Priority search [FGS+17] is a proposed extension to these rules allowing strategies that switch between different rules based on the current domain bounds of arbitrary variables. However, the support for priority search across solvers is quite limited. To my knowledge, it is currently only implemented in the academic Chuffed solver [Chu11] and not included in commercial or other open-source solvers.

2.2 MiniZinc Search Annotations

MiniZinc [NSB+07] is a solver-independent declarative programming language for Constraint Programming.

Of particular interest for this Thesis, MiniZinc includes support for user-specified search strategies through syntactic objects called annotations. Specifying a good search strategy can be critical for performance since a good choice of branching order can guide the solver into promising parts of the search space. For solvers that incorporate clause generation like Chuffed [Chu11] and CP-SAT [OrT], the branching order also has a strong impact on what clauses are generated.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
ann ::= seq_search(array of ann)

ann ::= int_search(array of var int, varsel, valsel)
ann ::= bool_search(array of var bool, varsel, valsel)
ann ::= set_search(array of var set of int, varsel, valsel)
ann ::= float_search(array of var float, <precision>, varsel,
   ↪ valsel)

varsel ::= [input_order, first_fail, smallest, dom_w_deg]
valsel ::= [indomain_min, indomain_median, indomain_random,
   ↪ indomain_split]
```

Listing 2.1: Search annotations supported by MiniZinc, as listed in the MiniZinc handbook version 2.8.3 [SMT24].

Listing 2.1 shows the syntax for the search annotations supported by MiniZinc according to [SMT24] in an EBNF-like notation. Each basic search annotation (`int_search`, `bool_search`, `set_search`, `float_search`) contains an array of decision variables (of type `var int`, `var bool`, `var set of int`, or `var float`), a variable selection strategy and a value selection strategy. When the solver needs to branch, it uses an annotation by sorting the variables according to the variable selection strategy, selecting the first non-fixed variable from the list, and branching according to the value selection strategy. `seq_search` can be used to chain multiple search annotations together. When all variables in the first annotation are already fixed, the next annotation is used, and so on. Of the different search annotations above, `int_search` and `seq_search` are most relevant to this thesis, since `set_search` and `float_search` are not widely supported by solvers and `bool_search` can be viewed as a special case of `int_search`.

The variable and value selection strategies include some typical heuristics used in Constraint Programming. For instance `input_order` just branches on the variables in the order in which they appear in the array, while `first_fail` selects the variable with the smallest remaining domain. The value selection strategies are quite self-explanatory. For instance, `indomain_min` chooses the numerically smallest possible value, while `indomain_random` selects randomly. Not every strategy and annotation is supported by every solver.

2.3 Priority Search

The default search strategies in MiniZinc have limitations: search annotations are always processed in the same order, and the priority of a variable inside of a search annotation can only depend on its own domain. For TLSP and TLSP-GR, this means it is not possible to schedule whole tasks or jobs at once unless the order in which the tasks or jobs are scheduled is fixed in advance.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Priority search [FGS+17] is a proposed extension to the MiniZinc search annotations that allows the solver to switch between different search annotations based on its current state.

```
ann ::= priority_search(array of var int, varsel, array of ann)
```

Listing 2.2: Priority search MiniZinc annotation from [FGS+17], extending the annotations from Listing 2.1.

Listing 2.2 shows how priority search is defined. Like the search annotations before, it takes in an array of decision variables and a variable selection strategy. However, instead of a value selection strategy, there is an additional array of search annotations, which must have the same length as the variable array. During runtime, the solver ranks the variables according to the variable selection strategy, like before. However, when it chooses a variable, instead of branching on that variable, it processes its corresponding annotation instead. If that search strategy yields a branching decision, that decision is returned. Otherwise, the search strategy corresponding to the next best variable is considered, and so on.

The TLSP-GR model from Section 3.4 uses priority search to select a task based on its earliest possible start time (variable selection strategy `smallest` on $\dot{s}_a$) and then schedules that task completely by setting the job assignment, start time, resource assignments and mode assignment.

2.4 Test Laboratory Scheduling Problem (TLSP)

The *Test Laboratory Scheduling Problem (TLSP)* is an NP-complete scheduling problem first proposed by [MM18b]. The problem was modeled after the scheduling requirements of a real-world industrial test laboratory. TLSP is related to the classic Resource-Constrained Project Scheduling Problem (RCPSP), where tasks belonging to projects need to be scheduled and assigned resources. However, TLSP includes a substantial number of features that go beyond the standard RCPSP. For instance, TLSP allows tasks to be completed in different modes, like in the Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP) [TPM09]. There are multiple projects, similar to [WKS+14]. There are heterogeneous resource like in [DPRL98]. In addition, TLSP resources are further divided into different categories: employees, workbenches, and equipment, which each have slightly different properties and scheduling requirements. Perhaps most substantially, tasks in TLSP are not scheduled directly. Instead, they first need to be grouped into jobs, which become the unit of scheduling. The duration and required resources for a job are computed based on the task it contains. In addition to the sum of the tasks' durations, a job's duration contains an additional set-up time that encourages a solver to form large jobs. On the other hand, the resources assigned to a job need to cover all its tasks' requirements for the job's whole duration, encouraging smaller jobs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Multiple solution methods have been proposed for TLSP. Originally, these concentrated on a scheduling sub-problem, TLSP-S (also proposed in [MM18b]), where the grouping of tasks into jobs is fixed and given as part of the input. The TLSP-S literature covers Constraint Programming and a hybrid VLNS algorithm [GMM22], Simulated Annealing [MM21] and Constraint Answer-Set Programming [GMM21]. The full TLSP was first solved in [DGMM20] using Constraint Programming and a hybrid VLNS algorithm. Later approaches for the full TLSP also covered Simulated Annealing [MMS23] and hyper-heuristics [MM23].

2.5 Very Large Neighborhood Search (VLNS)

Very Large Neighborhood Search generally refers to local search techniques where the neighborhoods themselves have exponential size [AOEOP02]. Regarding TLSP, a VLNS-based algorithm was first proposed in [GMM22] for the scheduling sub-problem TLSP-S, and later generalized to the full TLSP by [DGMM20].

The algorithm is based on hill climbing and uses a neighborhood based on a destroy-and-repair heuristic. Its neighborhood is based on re-scheduling a small number of projects, as tasks in TLSP instances are grouped into projects, with roughly 5-100 projects in an instance. Classically, the algorithm starts from a feasible solution that is derived, for instance, by constraint programming. At a high level, the algorithm then repeatedly deletes the schedule for a small number of projects and re-schedules those projects with a constraint programming solver. The algorithm can use two different CP models: one for the full TLSP and one for TLSP-S, which cannot change the grouping of jobs into tasks but is faster. Whenever an improvement is obtained, the schedule is updated. The algorithm further employs a tabu list to avoid retrying unsuccessful project combinations.

This Thesis uses a variant of the VLNS algorithm where the neighborhood is also used to obtain the initial feasible solution, after starting from an infeasible solution generated by a greedy construction heuristic. In this case, when using the CP model to re-schedule a small number of projects, pre-processing ensures that conflicts in other projects are removed for the sub-instance. This can make instances feasible even if the CP solver cannot solve them on its own.

What follows is a more detailed description of how the algorithm works. Many aspects of the algorithm state, including the content of the tabu list and the number of projects to re-schedule, are duplicated for the TLSP and TLSP-S solvers, which is indicated using square brackets.

- At the beginning, the neighborhood sizes k are initialized to 1 (for both TLSP and TLSP-S). The solver timeouts t are initialized to *tlsp[s]MoveTimeout*.
- For each iteration:
  - Randomly choose between TLSP and TLSP-S solver using *tlspProb*.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

– Find a project combination for size k not part of the tabu list. When the solution is already feasible, the combination has k projects. If it is still infeasible, every increase of k beyond 1 means that for some random project, all overlapping projects are added $k-1$ times.

– Create a CP instance where these projects have their schedule deleted and tasks in other projects are replaced by dummy tasks to keep their resources occupied. With probability *tlsp[s]HotStartProb*, hot-start the solver with the existing schedule. If the solver is not hot-started, change the search strategy for resources to random to get more variation in the solutions. Solve the instance with the current timeout t.

– If successful, update the schedule accordingly, reset k to 1 and t to *tlsp[s]MoveTimeout*, and clean up the tabu list, removing all project combinations that overlap in time with the current project selection. If unsuccessful add the current combination to the tabu list, also storing whether the instance was infeasible or whether there was a timeout. If there are no new combinations for size k, increase it by 1 (or 2 with probability *tlsp[s]JumpProb*). If there are no new combinations of any size, increase t by an exponential factor, reset k to 0, and delete timeout-based entries from the tabu list.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

TLSP-GR (TLSP with Generalized Resources)

TLSP solvers have been deployed in a real-world industrial test laboratory, where they are being used to conduct long-time scheduling operations [DGJ+23].

Over time, additional scheduling requirements emerged that go beyond the current TLSP formulation and require extending instances with new parameters or adding additional constraints. At the same time, other test laboratories started showing interest in TLSP to manage their scheduling operations. Their scheduling requirements differed in some respects from those in the original laboratory, which would also necessitate extending TLSP. Many of these new requirements were related to employees, workbenches, and equipment.

For instance, TLSP contained a constraint that allows tasks to be linked, which denotes that they must be performed by the same employees. One requirement that emerged was that this constraint could be applied to workbenches and/or equipment as well. As another example, TLSP contained a soft constraint where tasks could specify preferred employees, which are assigned preferentially. Extending this to workbenches and equipment also turned out to be a useful feature. Finally, an additional requirement by the original laboratory was to have a constraint that would allow workbench assignment to restrict equipment assignment. This could be used to model equipment that is hard to move and should be used near its storage location.

A common thread with all of these additional requirements is that they are all made more complicated by the distinction between employees, workbenches, and resources. A large part of what sets employees apart from workbenches or equipment is the ability to link the employee assignment together for different tasks, and the ability to specify preferred employees. Yet, it turned out useful for practical scheduling to extend these properties to workbenches and equipment. The requirement about restricting workbench

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

and equipment assignment based on one another is a bit different but fits into a similar pattern. In talking with the industrial partner, it turned out that they also had a use-case for an equivalent feature relating other types of resources, for instance connecting employee assignment to equipment assignment. However, with the existing description of TLSP, this would require adding at least 3 new constraints. And even more, should the requirement arise to also connect, for example, equipment to other equipment, in this manner.

A more elegant solution might be to unify employees, workbenches, and equipment into a single concept, resources. Then the first two requirements, linking tasks for identical workbenches/equipment and preferred workbenches/equipment, could be satisfied without adding any new constraints at all. The third requirement, connecting workbench assignments to equipment assignments, could be generalized to all types of resources with a single constraint.

This Chapter proposes a new problem variant, TLSP with Generalized Resources (TLSP-GR), which can be considered a more general variant of TLSP. Instead of employees, workbenches and, equipment, TLSP-GR only contains resources. All modeling features from TLSP that were specific to either employees, workbenches, or equipment, are generalized in a manner that still allows TLSP instances to be expressed in TLSP-GR while adding new modeling possibilities. The Chapter also introduces a new constraint programming model for TLSP-GR, which is evaluated later in Chapter 5.

The rest of this Chapter is structured as follows:

- Section 3.1, the first Section in this Chapter, introduces the new problem variant TLSP with Generalized Resources (TLSP-GR) and presents its problem description. The Section describes how instances and solutions are specified, including an informal description of the intended semantics and constraints. Deviations from TLSP are highlighted by presenting the TLSP-GR specifications alongside the existing TLSP ones to provide a comparison, explain the reasoning behind the generalizations, and detail which modeling decisions had to be made during the generalization.
- Section 3.2 formally specifies the hard- and soft-constraints of the new TLSP-GR. Again, differences to the existing TLSP are highlighted.
- Section 3.3 explains how TLSP instances can be solved with a TLSP-GR solver and specifies the problem reduction from TLSP to TLSP-GR.
- Section 3.4 proposes a new Constraint Programming (CP) model for TLSP-GR. It encompasses both the grouping and scheduling stages and covers all hard and soft constraints.
- Section 3.5 explains various redundant constraints and other performance optimizations for the CP model. It places a strong focus on optimizations that are either

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

exclusive to TLSP-GR, or whose generalization from its TLSP counterpart proved particularly challenging.

3.1 Problem Description

Just like for TLSP, a TLSP-GR instance consists of a set of *projects* to schedule and an *environment*. Projects can contain any number of tasks together with their resource requirements and other information like precedences. The environment contains information about the length of the scheduling horizon, the available resources, as well as information about an existing schedule. A solution consists of a grouping of tasks into jobs, as well as time slot and resource assignments for these jobs.

Just like TLSP contains the scheduling sub-problem TLSP-S, TLSP-GR has a scheduling sub-problem TLSP-GR-S. TLSP-GR-S instances look and function just like TLSP-GR instances, except that the grouping from tasks into jobs is also given as part of the input, and not determined by the solver.

The rest of this section specifies the problem description for a TLSP-GR instance and contrasts it with TLSP. In this Section, the parts about TLSP are based on [MM18b], while the modifications for TLSP-GR are a novel contribution of this thesis.

3.1.1 Environment

In both TLSP and TLSP-GR, the scheduling horizon consists of a set of time slots $t \in T = \{0, \ldots, |T| - 1\}$. Since the grouping aspect of the problem allows for a more coarse-grained scheduling, one time slot typically corresponds to half a workday. However, this is not required.

Next, there is a set of modes $m \in M = \{1, \ldots, |M|\}$. In a solution, each job needs to be assigned exactly one mode. Modes allow for trade-offs between a job's duration and resource consumption. In TLSP, this only concerns employee requirements, but in TLSP-GR, it can affect other resource requirements as well. Each mode has an associated time factor $v_m \in \mathbb{R}$ that acts as a multiplier for the duration of tasks performed under that mode.

Additionally, there are the available resources.

TLSP only

TLSP splits resources into employees, workbenches, and equipment. Resources are heterogeneous, and individual units may be compatible with different tasks.

- Employees $e \in E = \{1, \ldots, |E|\}$ may be assigned to perform jobs.
- Workbenches $b \in B = \{1, \ldots, |B|\}$ serve as special locations to perform tasks. As such, each job may be performed on at most one workbench.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- Equipment is split into groups. The set of all groups is denoted $G^* = \{G_1, \ldots, G_{|G^*|}\}$. Each group $G_i = \{1, \ldots, |G_g|\}$ with index i contains $|G_i|$ pieces of equipment, and tasks can require different numbers of equipment pieces from each group.

Finally, in TLSP the number of employees required to perform a job depends only on the chosen mode m and is identical for all jobs. Therefore, the number of required employees for each mode m is part of the Environment and given by $e_m \in \mathbb{N}_0, m \in M$.

TLSP-GR only

TLSP-GR unifies employees, workbenches, and equipment into a single concept called *resources*. Like equipment in TLSP, the resources in TLSP-GR are partitioned into groups. As explained in detail in Section 3.3, when reducing TLSP instances to TLSP-GR, employees and workbenches each become a single resource group, alongside the existing equipment groups.

- Resources are split into groups. The set of all resource groups is denoted $R^* = \{R_1, \ldots, R_{|R^*|}\}$. Each resource group with index i consists of individual resources $R_i = \{1, \ldots, |R_i|\}$. Tasks may require any number of resources from any resource group.

Since TLSP-GR needs to account for all types of resources from TLSP, the number of resources required by a task may depend on the task as well as its chosen mode. Therefore, the required number of Employees for performing a task is no longer part of the Environment.

Furthermore, $L^{rg} \subseteq R^*$ represents the set of resource groups such that resource assignments for resources in this group must be identical for tasks that are linked (as defined later).

Finally, TLSP contained a soft constraint to minimize the number of different employees working on a project. To generalize it, TLSP-GR instances contain a set of resource groups $R^{min} \subseteq R^*$ for which the number of different units assigned to jobs of a project should be minimized.

3.1.2 Projects and Tasks

The basic anatomy of Projects and Tasks is identical for TLSP-GR and TLSP. Each instance contains a set of projects $p \in P$.

Each project p contains a set of tasks $a \in A_p$. For notational convenience, p_a refers to the project that the task a belongs to. The set of all tasks is $A^* = \bigcup_{p \in P} A_p$. Each task can only be part of one project, so $A_p \cap A_q = \emptyset$ for all $p, q \in P, p \neq q$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Furthermore, the tasks in each project p are partitioned into task families $F_{p,i} \subseteq A_p$ indexed by the project p and an additional index i. The family of a task a is denoted by f_a, the set of families for project p is F_p, and the set of all families across projects is F^*. The set of tasks belonging to family f is denoted A_f. Task families serve a two-fold purpose: firstly, only tasks from the same family may be grouped together into a job. Secondly, each family has a setup time $s_f \in \mathbb{R}_0^+$ associated with it. When a job is formed from tasks in a family f, the setup time s_f gets added to the sum of the durations of its tasks to calculate the total duration of the job. The setup time is also subject to the mode time factor v_m discussed earlier in 3.1.1.

Finally, each project p contains a set L_p of sets of linked tasks. Formally, L_p is an equivalence relation over the tasks in p. In TLSP, linked tasks need to be performed by the same employees. In TLSP-GR, this can be toggled for each resource group.

Each task a has various properties associated with it:

- Each task has a release date $\alpha_a \in T$, which is the earliest possible start date. It also has a soft due date $\bar{\omega}_a \in T$, whose violation incurs a penalty, and a hard deadline $\omega_a \in T$, by which it must be completed.
- Each task has a set $M_a \subseteq M$ of available modes.
- Each task has a duration $d_a \in \mathbb{R}_0^+$.
- Each task has a set of predecessor tasks from the same project $\mathcal{P}_a \subseteq A_p$ where $p = p_a$. Each task must be performed after every of its predecessor tasks, or grouped into the same job.

Next up, each task also has resource requirements. This is where TLSP-GR differs from TLSP.

TLSP only

In TLSP, each task also has the following properties.

- A set of qualified employees, $E_a \subseteq E$. In a solution, only employees from E_a may be assigned to a job containing a. The number of required employees only depends on the mode and therefore is not part of a task.
- A set of preferred employees $E_a^{Pr} \subseteq E_a$. Assigning a non-preferred employee to a job containing a incurs a penalty.
- A binary value $b_a \in \{0, 1\}$ that specifies whether this task needs to be performed on a workbench or not. It also has a set of available workbenches $B_a \subseteq B$, analogously to qualified employees.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

- For each equipment group $g \in G^*$, a task may require a number $r_{a,g} \in \mathbb{N}_{\not\vdash}$ of equipment pieces. Again, there is a set of available equipment pieces $G_{a,g} \subseteq G_g$ for each equipment group g.

In TLSP, employees, workbenches, and equipment groups are treated in a similar manner in many respects. In particular, each task has a set of available resources for employees, workbenches, and each equipment group. And for workbenches and equipment groups, each task specifies the number of units it requires.
This symmetry is broken in a few ways in TLSP: tasks can only require at most one workbench and, more importantly, preferred employees do not exist for other resources and the number of required employees only depends on the chosen mode, not the task.
TLSP-GR addresses the part about preferred employees by generalizing preferred employees to all resources. Regarding the different sources for the number of required resources, there are different reasonable ways to generalize this. One way would be that resource requirements in TLSP-GR can either depend on a job's mode or its tasks, but not both. Another option is that resource requirements can be a function of both tasks and modes, allowing for a more complicated interplay between tasks and modes. TLSP-GR uses the latter approach, which allows for more flexibility when specifying resource requirements. This potentially comes at the cost of drastically increasing the size of the search space and, for the CP model, the number of constraints. However, as discussed later in Section 3.5, this performance impact can be practically nullified in the CP model for TLSP-like instances by including additional constraint variants for special cases. Thus, this increased flexibility only comes at a performance cost where it is actually used.

TLSP-GR only

- From each resource group with index g in R^*, a task a requires a number $r_{a,g,m}$ of units that may also depend on the mode $m \in M$. It also has a set of available resources $R_{a,g} \subseteq R_g$, and a set of preferred resources $R^{Pr}_{a,g} \subseteq R_g$. Only available resources may be assigned to a job containing a, and assigning non-preferred resources incurs a penalty.

3.1.3 Initial Schedule

The original formulation of TLSP [MM18b] includes a way of specifying an existing schedule, including the grouping into jobs and assignments for time slots, modes, and resources. The specification states that this information can be used to jump-start the solver or to restrict solutions.

The problem description used for this thesis excludes most of this information, as it would complicate the model without adding much expressivity. Forcing some assignments

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

in solutions can also be achieved by preprocessing the instance to restrict release dates, due dates, and available resources and modes.

One part that is still of importance, however, is the set of base jobs J^0. Formally, each base job $j \in J^0$ is a set of tasks, so $j \subseteq A_f$ for some task family $f \in F^*$. All tasks from each base job must be part of the same job in a solution. The tasks $a \in j$ are called *fixed tasks*. Note that a solution may still add additional tasks to the job.

Additionally, as a subset of the base jobs, there are started jobs $J^{0S} \subseteq J^0$. Started jobs represent jobs that are already in progress. As such, in a solution, any job formed from a started job must have start time 0 and incurs no setup time. In practice, resource assignments for started jobs are usually restricted in preprocessing to prevent the solver from changing them.

For the scheduling-only subvariant TLSP-GR-S, the initial schedule also contains a grouping J of tasks into jobs. In this case, the grouping is fixed and given as part of the input, and not determined by the solver.

3.1.4 Solution Description

A solution to a TLSP-GR instance consists of:

- A grouping J of tasks into jobs. Each job $j \in J$ is a set of tasks. Only tasks from the same task family inside the same project may be grouped together. (For the scheduling sub-variant TLSP-GR-S, this part of the instance instead.)
- An assigned mode for each job. The mode chosen for a job must be available for all of its tasks.
- A start and completion time for all jobs. For every job, the start time must be bigger than or equal to the release time of all of its tasks and its completion time must be smaller than or equal to the deadline of its tasks. The difference between completion and start time must equal the job's duration. The duration of a job j containing tasks from family f and performed in mode m is given by $\lceil v_m \cdot (s_f + \sum_{a \in j} d_a) \rceil$. In the special case where $\exists j^0 \in J^{0S} \mid j \cap j^0 \neq \emptyset$, the job is considered a started job, and s_f in the formula is set to 0.
- Resource assignments for each job. For each resource group, the assigned resources must exactly cover the highest demand of any task in the job, and each assigned resource must be available for all tasks. Every individual resource unit may only be used by at most one job at the same time.

3.2 Solution Constraints

The previous Section defined the information a TLSP-GR instance contains and informally described what conditions a valid solution must meet. This Section formally specifies

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

both the hard constraints a solution must satisfy, as well as the soft constraints that represent various optimization objectives. Similar to the previous Section, the parts of this Section concerning TLSP are based on [MM18b] with adjustments to notation and some changes for consistency with the rest of this Thesis. The new constraints for TLSP-GR are a novel contribution of this Thesis.

3.2.1 Solution Variables

Formally, a solution consists of assignments for the following variables:

$J \subseteq 2^{A^*}$		... Set of jobs (given in TLSP-GR-S)
$\dot{s}_j \in T$	$j \in J$	... Assigned start time
$\dot{n}_j \in T$	$j \in J$	... Assigned completion time
$\dot{m}_j \in M$	$j \in J$	... Assigned mode

In addition, there are resource assignments. In TLSP, are three different types:

TLSP only

$\dot{a}_j^{Em} \subseteq E$	$j \in J$	... Employee Assignment
$\dot{a}_j^{Wb} \in B \cup \{\perp\}$	$j \in J$	... Workbench Assignment
$\dot{a}_{j,g}^{Eq} \subseteq G_g$	$j \in J, G_g \in G^*$	... Equipment Assignment

In TLSP-GR, there is just one type of resource assignment:

TLSP-GR only

$\dot{a}_{j,g} \subseteq R_g$	$j \in J, R_g \in R^*$	... Job Resource Assignment

For notational convenience, let j_a refer to the job j that task $a \in A^*$ is part of. The first constraint will ensure that this job always exists and is unique. Similarly, let J_p refer to the set of jobs that include tasks from project p.

3.2.2 Hard Constraints

$$\exists! j \in J : \ a \in J \qquad \forall a \in A^* \qquad \text{(H1)}$$

$$p_{a_1} = p_{a_2} \wedge f_{a_1} = f_{a_2} \qquad \forall j \in J, a_1 \in j, a_2 \in j \qquad \text{(H2)}$$

$$\exists j \in J : \ j^0 \subseteq j \qquad \forall j^0 \in J^0 \qquad \text{(H3)}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The **Job Assignment** constraint (H1) ensures every task is part of exactly one job.
The **Job Grouping** constraint (H2) enforces that all tasks within a job belong to the same project and family.
The **Fixed Tasks** constraint (H3) requires that tasks assigned to a fixed job must all be assigned to the same job in the solution.

$$\dot{n}_j - \dot{s}_j = \begin{cases} \left\lceil (\sum_{a \in j} d_a) \cdot v_{\dot{m}_j} \right\rceil & \text{if } \exists j^s \in J^{0S} : \ j^s \subseteq j \\ \left\lceil (s_{f_j} + \sum_{a \in j} d_a) \cdot v_{\dot{m}_j} \right\rceil & \text{otherwise} \end{cases} \qquad \forall j \in J \quad \text{(H4)}$$

$$\dot{s}_j \geq \max_{a \in j} \alpha_a \wedge \dot{n}_j \leq \min_{a \in j} \omega_a \qquad \forall j \in J \quad \text{(H5)}$$

$$j_{a'} \neq j_a \implies \dot{n}_{j_{a'}} \leq \dot{s}_{j_a} \qquad \forall a \in A^*, a' \in \mathcal{P}_a \quad \text{(H6)}$$

$$(\exists j^s \in J^{0S} : j^s \subseteq j) \implies \dot{s}_j = 0 \qquad \forall j \in J \quad \text{(H7)}$$

The **Job Duration** constraint (H4) requires the interval from start to finish of a job to align with the job's calculated duration, which is based on its task, chosen mode, and starting time (provided the job does not contain tasks from a started job).
The **Time Window** constraint (H5) specifies that each job must respect the release date and deadline of all of its tasks.
The **Task Precedence** constraint (H6) asserts that a job may only start after the completion of all its prerequisite jobs. Jobs inherit the precedence relation from their tasks. Tasks with a precedence relation may also be performed as part of the same job.
The **Started Jobs** constraint (H7) ensures that a job incorporating fixed tasks from a started job in the base schedule must begin at time slot 0.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

TLSP only

$$\left|\{j \in J:\ \dot{s}_j \leq t \wedge \dot{n}_j > t \wedge \dot{a}_j^{Wb} = b\}\right| \leq 1 \qquad \forall b \in B, t \in T \quad \text{(H8a-TLSP)}$$

$$\left|\{j \in J:\ \dot{s}_j \leq t \wedge \dot{n}_j > t \wedge e \in \dot{a}_j^{Em}\}\right| \leq 1 \qquad \forall e \in E, t \in T \quad \text{(H8b-TLSP)}$$

$$\left|\{j \in J:\ \dot{s}_j \leq t \wedge \dot{n}_j > t \wedge q \in \dot{a}_{j,g}^{Eq}\}\right| \leq 1 \qquad \forall G_g \in G^*, q \in G_g, t \in T \quad \text{(H8c-TLSP)}$$

$$(\dot{a}_j^{Wb} = \bot) \iff \forall_{a \in j}(b_a = 0) \qquad \forall j \in J \quad \text{(H9a-TLSP)}$$

$$|\dot{a}_j^{Em}| = e_{\dot{m}_j} \qquad \forall j \in J \quad \text{(H9b-TLSP)}$$

$$|\dot{a}_{j,g}^{Eq}| = \max_{a \in j}(r_{a,g}) \qquad \forall j \in J, G_g \in G^* \quad \text{(H9c-TLSP)}$$

$$(\dot{a}_j^{Wb} = \bot) \vee \forall_{a \in j}(\dot{a}_j^{Wb} \in B_a) \qquad \forall j \in J \text{ (H10a-TLSP)}$$

$$\dot{a}_j^{Em} \subseteq \bigcap_{a \in j}(E_a) \qquad \forall j \in J \text{ (H10b-TLSP)}$$

$$\dot{a}_{j,g}^{Eq} \subseteq \bigcap_{a \in j}(G_{a,g}) \qquad \forall j \in J, G_g \in G^* \text{ (H10c-TLSP)}$$

$$\dot{a}_{ja_1}^{Em} = \dot{a}_{ja_2}^{Em} \qquad \forall p \in P, L \in L_p, a_1, a_2 \in L \quad \text{(H11-TLSP)}$$

The **Single Assignment** constraints (H8a-TLSP–H8c-TLSP) prevent concurrent assignment of workbenches, employees, and equipment, respectively. In particular, (H8a-TLSP) ensures workbench exclusivity for every time slot $t \in T$, (H8b-TLSP) does the same for employees, and (H8c-TLSP) for equipment in all groups $G_g \in G^*$.

The **Resource Requirement** constraints (H9a-TLSP–H9c-TLSP) handle resource requirements. Workbench requirements are handled by (H9a-TLSP) and are binary, (H9b-TLSP) handles employee requirements depending on a job's mode, and equipment requirements in (H9c-TLSP) for each job j and equipment group are based on the task in j with the highest resource requirement for that group.

Similarly, the **Resource Suitability** constraints (H10a-TLSP–H10c-TLSP) that assigned workbenches, employees, and equipment are compatible with all tasks in a job. Like before, (H10a-TLSP) handles workbenches, (H10b-TLSP) ensures that employees are qualified, and the equipment compatibility is ensured by the constraint (H10c-TLSP) .

Lastly, the **Linked Tasks** constraint (H11-TLSP) ensures that linked tasks are performed by the same employees. Here, this takes the form of demanding that such tasks are either part of the same job or if they are part of different jobs, those jobs have the exact same employees assigned.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

TLSP-GR only

$$\left|\{j \in J:\ \dot{s}_j \leq t \wedge \dot{n}_j > t \wedge r \in \dot{a}_{j,g}\}\right| \leq 1 \qquad \forall R_g \in R^*, r \in R_g, t \in T \qquad \text{(H8-GR)}$$

$$|\dot{a}_{j,g}| = \max_{a \in j}(r_{a,g,\dot{m}_j}) \qquad \forall j \in J, R_g \in R^* \qquad \text{(H9-GR)}$$

$$\dot{a}_{j,g} \subseteq \bigcap_{a \in j}(R_{a,g}) \qquad \forall j \in J, R_g \in R^* \qquad \text{(H10-GR)}$$

$$\dot{a}_{j_{a_1},g} = \dot{a}_{j_{a_2},g} \qquad \forall R_g \in L^{rg}, p \in P, L \in L_p, a_1, a_2 \in L \qquad \text{(H11-GR)}$$

Compared to TLSP, TLSP-GR only needs one variant for each resource constraint.

The **Single Assignment** constraint (H8-GR) corresponds to (H8a-TLSP) through (H8c-TLSP) and ensures at any time slot t, each resource unit in every resource group R_g is assigned to at most one job. Extending this constraint is relatively straightforward since the original constraints in TLSP are very similar.

The **Resource Requirement** constraint (H9-GR) unifies (H9a-TLSP) through (H9c-TLSP) and specifies that for each job j and resource group R_g, the number of assigned resources exactly matches the highest requirement of any task in j, considering the job's mode.

Resource Suitability constraint (H10-GR) guarantees that resources assigned to a job from each group R_g are compatible with all tasks in the job.

Finally, the **Linked Tasks** constraint (H11-GR) ensures that linked tasks have the same resources assigned, for those resource groups included in L^{rg}.

3.2.3 Soft Constraints

Similarly to TLSP, TLSP-GR contains 5 soft constraints. Each one corresponds to one optimization objective and has an associated penalty $s_i, i \in \{1, 2, 3, 4, 5\}$.

$$s_1 = |J| \qquad \text{(S1)}$$

$$s_4 = \sum_{j \in J} \max(\dot{n}_j - \min_{a \in j}(\bar{\omega}_a), 0) \qquad \text{(S4)}$$

$$s_5 = \sum_{p \in P} \left(\max_{j \in J_p} \dot{n}_j - \min_{j \in J_p} \dot{s}_j \right) \qquad \text{(S5)}$$

The **Number of Jobs** constraint (S1) minimizes the total number of jobs.

The **Due Date** constraint (S4) penalizes jobs that end after their (soft) due date $\bar{\omega}_j$.

The **Project Completion Time** constraint (S5) seeks to minimize the total completion time for each project. That is, the time from the start of the first job, to the end of its last one.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

TLSP only

$$s_2 = \sum_{j \in J} |\dot{a}_j^{Em} \setminus \bigcap_{a \in j} (E_a^{Pr})| \tag{S2-TLSP}$$

$$s_3 = \sum_{p \in P} |\bigcup_{j \in J_p} \dot{a}_j^{Em}| \tag{S3-TLSP}$$

The **Preferred Employee** constraint (S2-TLSP) penalizes assigning an employee to a job when that employee is not preferred for all tasks of that job.

Finally, the **Number of Employees** constraint (S3-TLSP) minimizes the number of different employees assigned to complete jobs of a project.

TLSP-GR only

$$s_2 = \sum_{j \in J} \sum_{R_g \in R^*} |\dot{a}_{j,g} \setminus \bigcap_{a \in j} (R_{a,g}^{Pr})| \tag{S2-GR}$$

$$s_3 = \sum_{p \in P} \sum_{R_g \in R^{min}} |\bigcup_{j \in J_p} \dot{a}_{j,g}| \tag{S3-GR}$$

The **Preferred Resource** constraint (S2-GR) corresponds to (S2-TLSP) and penalizes assignments of resources to jobs that are not preferred by all of the job's tasks.

Lastly, **Number of Resources** constraint (S3-GR) replaces (S3-TLSP) and minimizes the number of different resources assigned to jobs in every project. It is applied to the resource groups specified to R^{min}.

3.3 Reducing TLSP to TLSP-GR

TLSP-GR is a generalization of TLSP. Accordingly, the TLSP-GR solvers I developed for this thesis are also designed to function as a drop-in replacement for a TLSP solver. To solve a TLSP instance, a TLSP-GR solver first converts the TLSP instance to a TLSP-GR instance. It then obtains a solution and converts the solution back to a TLSP schedule. The reduction is designed in such a way that solutions for TLSP and TLSP-GR have a one-to-one correspondence and have the same number of hard- and soft-constraint violations. In this Section, a line above an instance parameter or variable is used to denote data associated with a TLSP instance. For instance, $\overline{G^*}$ refers to the equipment groups of the TLSP instance, while R^* denotes the resource group of the TLSP-GR instance.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

3.3.1 Instance Conversion

A TLSP instance is given as input. It has time slots $\overline{T}$, employees $\overline{E}$, workbenches $\overline{B}$, equipment groups $\overline{G^*} = \{\overline{G_1}, \ldots, \overline{G_{|\overline{G^*}|}}\}$. Modes are given by $\overline{M}$. Each task requires $\overline{e_m}$ employees based on mode $m \in \overline{M}$ and the speed factor for modes is $\overline{v_m}$. The set of projects is $\overline{P}$.

Each project $p \in \overline{P}$ contains a set of tasks $\overline{A_p}$. Each task $a \in \overline{A^*}$ has different properties. $\overline{\alpha_a}$, $\overline{\bar{\omega}_a}$ and $\overline{\omega_a}$ refer to the release date, due date, and deadline, respectively. It has available modes $\overline{M_a}$ and duration $\overline{d_a}$. The set of predecessor tasks for a is given by $\overline{\mathcal{P}_a}$. Each task has available workbenches $\overline{B_a}$ and whether it requires a workbench is given by $\overline{b_a} \in \{0, 1\}$. It has a set $\overline{E_a}$ of qualified employees and a set $\overline{E_a^{Pr}}$ of preferred employees. For each equipment group in $\overline{G^*}$ with index g, a task requires $\overline{r_{a,g}}$ units of equipment from the set of available equipment for that group, $\overline{G_{a,g}}$.

For each project $p \in \overline{P}$, its linked tasks $\overline{L_p}$ form an equivalence relation where equivalent tasks need to be performed by the same employees. Task families for project $p \in \overline{P}$ are given by $\overline{F_{p,i}}$.The set of all families across projects is $\overline{F^*}$. Finally, $\overline{J^0}$ is the set of base jobs and $\overline{J^{0S}} \subseteq \overline{J^0}$ is the set of started jobs.

To convert this TLSP instance to TLSP-GR, some parts just have to be copied. The time slots of the new TLSP-GR instance are given by $T := \overline{T}$, its modes by $M := \overline{M}$ with speed factors $v_m := \overline{v_m}$ for $m \in M$. Projects $P := \overline{P}$ have tasks with the same indices $A_p := \overline{A_p}$. For each task a, release date $\alpha_a := \overline{\alpha_a}$, due dates $\bar{\omega}_a := \overline{\bar{\omega}_a}$ and deadlines $\omega_a := \overline{\omega_a}$ can also be copied, as well as available modes $M_a := \overline{M_a}$ and durations $d_a := \overline{d_a}$. Linked tasks are $L_p := \overline{L_p}$ for $p \in P$. Likewise, task families $F_{p,i} := \overline{F_{p,i}}$ and the set $F^* := \overline{F^*}$ can be copied. Furthermore, $J^0 := \overline{J^0}$ and $J^{0S} := \overline{J^{0S}}$.

Still missing are resources (made up of employees, workbenches, and equipment), together with the tasks' resource requirements and sets of available and preferred resources. Additionally, the TLSP-GR instance needs to specify the set of resource groups where assignments must be identical for linked tasks, L^{rg}, and the set of resource groups for which the number of distinct assigned units should be minimized for each project, R^{min}.

The reduction works by representing employees and workbenches as their own resource groups, alongside the existing equipment groups. Thus, the resource groups for the TLSP-GR instance are given by $R^* := \{R_1, R_2, \ldots, R_{|\overline{G^*}|}, R_{em}, R_{wb}\}$. The original group elements representing resources can be copied into the new groups: $R_i := \overline{G_i}, i = \{1, 2, \ldots, |\overline{G^*}|\}$, $R_{em} := \overline{E}$, $R_{wb} := \overline{B}$. Since both linked tasks and resource minimization only apply to employees in TLSP, $L^{rg} := \{R_{em}\}$ and $R^{min} := \{R_{em}\}$. Then the resource requirements, availabilities, and preferences are given by:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

$$r_{a,em,m} := \overline{e_m} \qquad \forall a \in A^* \quad \forall m \in M \tag{3.1}$$
$$R_{a,em} := \overline{E_a} \qquad \forall a \in A^* \tag{3.2}$$
$$R^{Pr}_{a,em} := \overline{E^{Pr}_a} \qquad \forall a \in A^* \tag{3.3}$$
$$r_{a,wb,m} := \overline{b_a} \qquad \forall a \in A^* \quad \forall m \in M \tag{3.4}$$
$$R_{a,wb} := \overline{B_a} \qquad \forall a \in A^* \tag{3.5}$$
$$R^{Pr}_{a,wb} := R_{wb} \qquad \forall a \in A^* \tag{3.6}$$
$$r_{a,g,m} := \overline{r_{a,g}} \qquad \forall a \in A^* \quad \forall G_g \in \overline{G^*} \quad \forall m \in M \tag{3.7}$$
$$R_{a,g} := \overline{G_{a,g}} \qquad \forall a \in A^* \quad \forall G_g \in \overline{G^*} \tag{3.8}$$
$$R^{Pr}_{a,g} := R_g \qquad \forall a \in A^* \quad \forall G_g \in \overline{G^*} \tag{3.9}$$

Formulas (3.1–3.3) describe the parameters for the employee resource group R_{em}, indexed by em. The number of required employees is the same for all tasks and depends only on the mode, like in TLSP. Available and preferred employees are copied from TLSP as well.

Formulas (3.4–3.6) define the parameters for the workbench group R_{wb}, indexed by wb. The required number depends on the task, and since $\overline{b_a}$ is a binary value with domain $\{0, 1\}$, it can also be used to describe the number of required resources. The available workbenches are copied from TLSP, but in TLSP-GR, every workbench is preferred, because TLSP has no constraint for preferred workbenches. Including all workbenches in the set of preferred workbenches ensures that TLSP-GR's preferred resources constraint effectively doesn't apply to workbenches.

Lastly, (3.7–3.9) specify the parameters relating to equipment. This time, the index is not a constant like em or wb, but a variable g used to quantify over all TLSP equipment groups. Like with workbenches, the number of required equipment pieces is copied from the TLSP instance, depending only on the task. Available equipment can be copied as well. And like with workbenches, all equipment pieces are preferred, since TLSP has no constraint for preferred equipment pieces. Again, this effectively disables the preferred resources constraint for equipment.

3.3.2 Solution Conversion

Once a solution has been obtained for the generated TLSP-GR instance, it can be converted back to a solution for the original TLSP instance.

Let J denote the set of jobs in the solution, and $\dot{s}_j$, $\dot{n}_j$ and $\dot{m}_j$ the start time, end time and mode assignment of jobs $j \in J$, respectively. Finally, $\dot{a}_{j,g}, j \in J, R_g \in R^*$ denotes the resource assignment.

Apart from resource assignments, the solution can be copied into the TLSP schedule:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

$$
\begin{aligned}
\overline{J} &:= J \\
\overline{\dot{s}_j} &:= \dot{s}_j && \forall j \in J \\
\overline{\dot{n}_j} &:= \dot{n}_j && \forall j \in J \\
\overline{\dot{m}_j} &:= \dot{m}_j && \forall j \in J
\end{aligned}
$$

The resource assignments need to be split back into employee, workbench, and equipment assignments:

$$
\begin{aligned}
\overline{\dot{a}_j^{Em}} &:= \dot{a}_{j,em} && \forall j \in J \\
\overline{\dot{a}_j^{Wb}} &:= \begin{cases} b & \text{if } \dot{a}_{j,wb} = \{b\} \\ \bot & \text{if } \dot{a}_{j,wb} = \{\} \end{cases} && \forall j \in J \\
\overline{\dot{a}_{j,g}^{Eq}} &:= \dot{a}_{j,g} && \forall j \in J, G_g \in \overline{G^*}
\end{aligned}
$$

The formulas above show how to recover the employee, workbench, and equipment assignment from the TLSP-GR solution. Again, em and wb refer to the indices of the TLSP-GR resource groups representing employees and workbenches, R_{em} and R_{wb}. A case distinction is required for handling the workbench assignment since it needs to be converted from a set that may contain zero or one entries from R_{wb} to the domain $B \cup \{\bot\}$. Note that the cases are exhaustive for any valid TLSP-GR solution because all requirements for resources in R_{wb} in the converted instance are either 0 or 1.

3.4 Constraint Programming Model

This Section presents a Constraint Programming (CP) model for TLSP-GR, which is a novel contribution of this Thesis. The hard and soft constraints that don't concern resources are based on an existing model for TLSP from [DGMM20].

In addition to the model for TLSP-GR, I also developed a model for the problem variant TLSP-GR-S, where the grouping of jobs into tasks is fixed and given as part of the input. For brevity, this model is not shown here. The model for TLSP-GR-S is similar to the TLSP-GR model presented here, with a few differences that amount to simplifications. In the TLSP-GR-S model, the decision variables and constraints for grouping are missing, the scheduling requirements for jobs are given as values instead of being calculated from tasks, and no rounding of durations is needed because the durations of jobs can be precomputed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The model was implemented in the solver-independent constraint programming language MiniZinc [NSB+07], and the following description was written with that implementation in mind. However, the formulation should easily carry over to other modeling languages or solvers. In fact, it only uses features that are widely supported across many solvers, staying clear of variables for sets and floating points.

The CP model for TLSP-GR covers the grouping of tasks into jobs as well as scheduling the jobs. This means that the parameters for scheduling are quite dynamic and change based on the selected grouping. In order to eliminate as many symmetries as possible and keep the model as solver-agnostic as possible, the grouping aspect is implemented as follows: there is an array of variables $\dot{\xi}(a) \in A^*$ that serve as pointers. Each task a points to the task that serves as a representative for the job that a belongs to. Tasks from the same job have to point at the same task. Thus a task that is pointed at must also point to itself. To break the symmetry in selecting the representative task, the representative task of a job has to be the one with the smallest id (according to some arbitrary total ordering of the tasks).

To enhance readability, the model description used the shorthand notation J for the set of tasks that have been chosen as representatives, i.e. $J = \{a \in A^* \mid \dot{\xi}(a) = a\}$. Note that the contents of this set change during the solving process, so a constraint like $P(a) \mid a \in J$ will be compiled as $(\dot{\xi}(a) = a \implies P(a)) \mid a \in A^*$.

Finally, the problem description includes real-valued numbers for task durations d_a, setup times s_f, and mode speed modifiers v_m. Since support for real (or floating-point) numbers is limited across solvers, the model slightly overapproximates the durations using integers. Overapproximating the durations cannot result in invalid schedules, but it could render some valid solutions infeasible. Task durations are scaled up to minimize the impact of rounding. Task durations in the synthetic and real-world instances available for TLSP are typically between 0.1 and 10 time slots. Past experience [DGMM20] has shown that a rounding factor of $\mathcal{M} = 10000$ still has a small impact on the model's runtime and that further increases usually don't improve the best achievable solution, at least for small instances, where it was possible to verify.

3.4.1 Decision Variables

To build a solution, the solver needs to decide on the following variables:

$\dot{\xi}(a) \in A^*$	$a \in A^*$	... Assigned representative task
$\dot{m}_a \in M$	$a \in A^*$	... Assigned mode
$\dot{s}_a \in T$	$a \in A^*$	... Assigned start time
$\dot{n}_a \in T$	$a \in A^*$	... Assigned completion time
$\dot{d}_a \in T$	$a \in A^*$	... Assigned duration (for convenience and performance)

In addition, there are variables for resource selection. In TLSP, there are three arrays:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TLSP only

$\dot{a}^{Em}_{a,e} \in \{0,1\}$ $a \in A^*, e \in E$ ... Job Employee Assignment
$\dot{a}^{Wb}_{a,b} \in \{0,1\}$ $a \in A^*, b \in B$ ... Job Workbench Assignment
$\dot{a}^{Eq}_{a,q} \in \{0,1\}$ $a \in A^*, G_g \in G^*, q \in G_g$ ... Job Equipment Assignment

In TLSP-GR, a single array handles all resources:

TLSP-GR only

$\dot{a}_{a,r} \in \{0,1\}$ $a \in A^*, R_g \in R^*, r \in R_g$ ... Job Resource Assignment

3.4.2 Hard Constraints

The first set of constraints concerns the task grouping.

$$\dot{\xi}(\dot{\xi}(a)) = \dot{\xi}(a) \qquad a \in A^* \tag{3.10}$$
$$p_a = p_{\dot{\xi}(a)} \wedge f_a = f_{\dot{\xi}(a)} \qquad a \in A^* \tag{3.11}$$
$$\texttt{all\_equal}(\{\dot{\xi}(a) \mid a \in j^0\}) \qquad j^0 \in J^0 \tag{3.12}$$
$$\dot{\xi}(a) \leq a \qquad a \in A^* \tag{3.13}$$

Constraint (3.10) forces representative tasks to point at themselves, which implies that $\dot{\xi}(a)$ partitions tasks into jobs. (3.11) enforces that tasks may only point at a representative task from the same project and family, which implies that only tasks from the same project and family can form a job. (3.12) enforces that tasks from fixed jobs are grouped together. Together, (3.10–3.12) already describe a legal grouping. (3.14) is technically redundant for correctness but breaks the symmetry of choosing a representative task.

$$\dot{s}_{\dot{\xi}(a)} \geq \alpha_a \wedge \dot{n}_{\dot{\xi}(a)} \leq \omega_a \qquad a \in A^* \tag{3.14}$$
$$\dot{d}_a = \dot{n}_a - \dot{s}_a \qquad a \in J \tag{3.15}$$
$$Setup(a) = \begin{cases} 0 & \text{for } \exists j^s \in J^{0S}, a_2 \in \dot{A}_{j^s} \mid \dot{\xi}(a_2) = a \\ s_{f_a} \cdot v_{\dot{m}_a} & \text{else} \end{cases} \tag{3.16}$$

$$\dot{d}_a \cdot \mathcal{M} \geq Setup(a) + \sum_{a_2 \in A^* \mid \dot{\xi}(a_2) = a} d_a \cdot v_{\dot{m}_a} \qquad a \in J \tag{3.17}$$
$$(\dot{d}_a - 1) \cdot \mathcal{M} < Setup(a) + \sum_{a_2 \in A^* \mid \dot{\xi}(a_2) = a} d_a \cdot v_{\dot{m}_a} \qquad a \in J \tag{3.18}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Constraints 3.14 through 3.18 concern job durations. (3.14) enforces that the start date and deadline of every task inside a job must be respected. Constraint (3.15) connects $\dot{d}_a$ to $\dot{s}_a$ and $\dot{n}_a$. Even though the duration variables $\dot{d}_a$ are redundant, they serve as a useful notational short-hand and including them in the model empirically improves performance. Formula (3.16) defines a shorthand notation for the setup time of a job. The formula accounts for the job's mode and for whether the job contains started tasks. Then, constraints (3.17–3.18) calculate and round up the duration of jobs. Note that the d_a and s_f parameters have already been scaled up by $\mathcal{M}$ in preprocessing.

$$\dot{\xi}(a) = \dot{\xi}(a_2) \vee \dot{n}_{\dot{\xi}(a_2)} \leq \dot{s}_{\dot{\xi}(a)} \qquad a \in A^*, a_2 \in \mathcal{P}_a \tag{3.19}$$

$$\dot{s}_{\dot{\xi}(a)} = 0 \qquad j \in J^{0S}, a \in A_j \tag{3.20}$$

$$\dot{m}_{\dot{\xi}(t)} \in M_a \qquad a \in A^* \tag{3.21}$$

Constraint (3.19) enforces task precedences and (3.20) forces jobs with started tasks to start at the beginning. Constraint (3.21) enforces that the mode of a job must be available to all of its tasks.

TLSP only

In TLSP, the resource related constraints are duplicated for employees, workbenches, and equipment.

$$\texttt{cumulative}((\dot{s}_a)_{a\in A^*}, (\dot{d}_a)_{a\in A^*}, (\dot{a}^{Em}_{a,e})_{a\in A^*}, 1) \qquad e \in E$$

$$\texttt{cumulative}((\dot{s}_a)_{a\in A^*}, (\dot{d}_a)_{a\in A^*}, (\dot{a}^{Wb}_{a,b})_{a\in A^*}, 1) \qquad b \in B$$

$$\texttt{cumulative}((\dot{s}_a)_{a\in A^*}, (\dot{d}_a)_{a\in A^*}, (\dot{a}^{Eq}_{a,e})_{a\in A^*}, 1) \qquad G_g \in G^*, e \in G_g$$

The three constraints above enforce that no employee, workbench, or equipment piece is used by two jobs at the same time.

TLSP-GR only

In TLSP-GR, the single usage constraints can be merged into one.

$$\texttt{cumulative}((\dot{s}_a)_{a\in A^*}, (\dot{d}_a)_{a\in A^*}, (\dot{a}_{a,r})_{a\in A^*}, 1) \qquad R_g \in R^*, r \in R_g \tag{3.22}$$

Constraint (3.22) suffices for TLSP-GR to ensure that no resource is used twice at the same time.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TLSP only

$$e_{\dot{m}_a} = \sum_{e \in E} \dot{a}^{Em}_{a,e} \qquad a \in J$$

$$1 = b_a \implies 1 = \sum_{b \in B} \dot{a}^{Wb}_{\dot{\xi}(a),b} \qquad a \in A^*$$

$$\max_{a_2 \in A^* \text{ s.t. } \dot{\xi}(a_2)=a} r_{a_2,g} = \sum_{e \in G_g} \dot{a}^{Eq}_{a,e} \qquad a \in J, G_g \in G^*$$

Although the three constraints above may look very different at first sight, they all serve a similar purpose in ensuring that the correct number of employees, workbenches, or equipment pieces is assigned to each job.

TLSP-GR only

$$\sum_{r \in R_g} \dot{a}_{a,r} = \max_{a_2 \in A^* | \dot{\xi}(a_2)=a} r_{a_2,g,\dot{m}_a} \qquad a \in J, R_g \in R^* \tag{3.23}$$

Constraint (3.23) ensures that each job is assigned exactly the correct number of resources from each resource group. Given the mode that a job is performed in, for each resource group, the assignment exactly covers the highest demand from any of its tasks for that group.

TLSP only

$$\dot{a}^{Em}_{\dot{\xi}(a),e} \implies e \in E_a \qquad a \in A^*, e \in E$$

$$\dot{a}^{Wb}_{\dot{\xi}(a),b} \implies b \in B_a \qquad a \in A^*, b \in B$$

$$\dot{a}^{Eq}_{\dot{\xi}(a),e} > 0 \implies e \in G_{a,g} \qquad a \in A^*, G_g \in G^*, e \in G_g$$

In TLSP, the above constraints enforce that only qualified employees, as well as available workbenches, and available equipment pieces are assigned. That is, they must be available for all tasks in the job.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TLSP-GR only

$$\dot{a}_{\dot{\xi}(a),r} > 0 \implies r \in R_{a,g} \qquad a \in A^*, R_g \in R^*, r \in R_g \tag{3.24}$$

Constraint (3.24) ensures that resources assigned to a job are available to all tasks that are part of the job.

Finally, there is the linked tasks constraint.

TLSP only

$$\dot{a}^{Em}_{\dot{\xi}(a),e} = \dot{a}^{Em}_{\dot{\xi}(a_2),e} \qquad e \in E, p \in P, (a, a_2) \in L_p$$

In TLSP, this constraint was called the *Linked Employees Constraint* and only applied to employees.

TLSP-GR only

$$\dot{a}_{\dot{\xi}(a),r} = \dot{a}_{\dot{\xi}(a_2),r} \qquad p \in P, (a, a_2) \in L_p, R_g \in L^{rg}, r \in R_g \tag{3.25}$$

In TLSP-GR, linked tasks can apply to different resource groups. Constraint (3.25) ensures that for all included resource groups (groups $R_g \in L^{rg}$), the jobs containing linked tasks have the same resource units assigned.

3.4.3 Soft Constraints

Like TLSP, TLSP-GR contains multiple soft constraints. Since MiniZinc only supports single-objective optimization, the objective is modeled as a weighted sum of the different constraint penalties. The test instances considered in this Thesis weigh all constraints equally.

The weights are given by $w_i \in \mathbb{R}_0^+, i \in \{1, 2, 3, 4, 5\}$. The objective function is given by $\sum_{i \in \{1,2,3,4,5\}} (s_i \cdot w_i)$, where s_i is given by:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

$$s_1 = \sum_{j \in J} 1 \tag{3.26}$$

$$s_4 = \sum_{j \in J} \max(0, \dot{n}_j - \min_{a \in A^* \text{ s.t. } \dot{\xi}(a)=j} (\bar{\omega}_a)) \tag{3.27}$$

$$s_5 = \sum_{p \in P} (\max_{a \in A_p}(\dot{n}_a) - \min_{a \in A_p \text{ s.t. } \dot{\xi}(a)=a} (\dot{s}_a)) \tag{3.28}$$

Constraints (3.26) , (3.27) , and (3.28) are the same for TLSP and TLSP-GR. Soft constraint (3.26) minimizes the number of created jobs, while (3.27) minimizes the violations of tasks' due dates. Next, (3.28) minimizes the duration of projects. The other soft constraints relate to resources and differ between TLSP and TLSP-GR.

TLSP only

$$s_2 = \sum_{j \in J} \sum_{e \in (E \setminus (\bigcap_{a \in A^* | \dot{\xi}(a)=j} E_a^{Pr}))} \dot{a}_{j,e}^{Em}$$

$$s_3 = \sum_{p \in P} \sum_{e \in E} ((\sum_{a \in A_p} \dot{a}_{a,e}^{Em}) > 0)$$

In TLSP, these constraints minimize the assigned non-preferred employees and the number of different employees assigned to each project, respectively.

TLSP-GR only

$$s_2 = \sum_{j \in J} \sum_{R_g \in R^*} \sum_{r \in (R_g \setminus (\bigcap_{a \in A^* | \dot{\xi}(a)=j} R_{a,g}^{Pr}))} \dot{a}_{j,r} \tag{3.29}$$

$$s_3 = \sum_{p \in P} \sum_{R_g \in R^{min}} \sum_{r \in R_g} ((\sum_{a \in A_p} \dot{a}_{a,r}) > 0) \tag{3.30}$$

Soft constraint (3.29) minimizes the number of non-preferred assigned resources. This constraint is similar to TLSP, except it now applies to all resource groups. Finally, (3.30) minimizes the number of different resources used for each project. It can apply to any set of resource groups, as specified by R^{min}.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.5 Performance Optimizations

The implemented model incorporates various optimizations and redundant constraints to increase performance. Some of them are quite basic and similar to optimizations for TLSP from [Dan19], so this Thesis only briefly touches on them. More attention is paid to two specific optimizations regarding resource constraints, one of which is new and specific to TLSP-GR, and the other, which was not trivial to adapt to TLSP-GR.

Generally, there are significant performance improvements to be gained by limiting domains through redundant constraints. One simple but effective example is $\dot{\xi}(a) = a, \forall a \in A^* \mid a = (\min_{f_a} a)$, which enforces that for each family, the task with the smallest id points to itself. Some of the redundant constraints follow a simple schema: when a property of a task a limits the domain of a variable concerning task $\dot{\xi}(a)$, it usually follows that the same limit applies to the equivalent variable for task a itself. For instance, the resource availability constraint (3.24) implies that $\dot{a}_{r,a} > 0 \wedge \max_{m \in M} r_{a,g,m} > 0 \implies r \in R_{a,g}$ also holds (compared to (3.24) , $\dot{\xi}(a)$ has been replaced by a). These new constraints can be justified with a case distinction. Either $\dot{\xi}(a) = a$, in which case both constraints are identical. Or $\dot{\xi}(a) \neq a$, in which case the assignment for a is arbitrary, and adding new bounds does not hurt unless the new bounds are contradictory. Including such constraints can improve performance because these constraints are usually quite simple to apply and help the solver in restricting domains.

Aside from these simple cases, two optimizations are of particular interest for TLSP-GR. The first is exclusive to TLSP-GR and concerns the resource requirements constraint (3.23) . It is particularly useful when dealing with converted TLSP instances. The second one concerns merging equivalent resources into single entities. A similar idea already existed for TLSP, but generalizing it to TLSP-GR was not straightforward.

Resource Requirement Constraint

As TLSP-GR is a generalization of TLSP, resources in TLSP-GR must be able to model all possible requirements from TLSP. In TLSP, the number of required employees depends on a job's mode and the requirement for workbenches and equipment depends on which tasks are part of the job. In principle, there are two ways to cover these requirements with the combined concept of resources in TLSP-GR. One possibility is that the number of required resources can either depend on the mode or on the tasks, depending (for example) on the resource group in question. The other option is making the number of required resources a function of both the mode and tasks. *The first option* more closely resembles the structure of TLSP. *The second option* is more general and powerful, but asymptotically, it could cause (3.23) to generate $O(n^4)$ instantiated constraints, compared to $O(n^3)$ for the first option. The iterations contributing to n^4 are: iterating over resource groups, over tasks, over modes, and compiling the indirect reference introduced by $\dot{\xi}(a)$.

The solution I arrived at was to base the specification on the second option, where resource requirements can depend on modes and tasks. To keep the model performant in cases where resource requirements effectively depend only on the mode or tasks, I

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

introduce an optimized variant of (3.23) . During compile time, for each resource group, the model analyzes whether its requirements depend on the job's mode, tasks, or both. Depending on the case, a different optimized version of the constraint is used. This allows resource requirements in TLSP-GR to flexibly depend on a job's mode and tasks while minimizing the performance sacrifice for TLSP-like instances.

$$R^{\text{task-dependent}} := \{R_g \in R^* \mid \forall_{a \in A^*} \forall_{m_1, m_2 \in M_a} \quad r_{a,g,m_1} = r_{a,g,m_2}\}$$
$$R^{\text{mode-dependent}} := \{R_g \in R^* \mid \forall_{m \in M} \forall_{a,a' \in A^*} ((m \in M_a \wedge m \in M_{a'}) \implies r_{a,g,m} = r_{a',g,m})\}$$

$R^{\text{task-dependent}}$ describes the set of resource groups for which the required amount only depends on the task in question, which means they are not affected by the mode. In contrast, $R^{\text{mode-dependent}}$ describes the set of resource groups where the requirement may depend on the chosen mode, but not on the tasks that make up a job. The sets are computed during the model compilation and used to select which constraints are compiled.

$$\sum_{r \in R_g} \dot{a}_{a,r} = \begin{cases} r_{a,g,\dot{m}_a} & \text{if } R_g \in R^{\text{mode-dependent}} \\ \max_{a_2 \in A^* \mid \dot{\xi}(a_2) = a} r_{a_2,g,1} & \text{if } R_g \in R^{\text{task-dependent}} \\ \max_{a_2 \in A^* \mid \dot{\xi}(a_2) = a} r_{a_2,g,\dot{m}_a} & \text{otherwise} \end{cases} \qquad a \in J, R_g \in R^* \tag{3.31}$$

Constraint (3.31) is the optimized version of (3.23) . In the first case, the required number of resources only depends on the mode, which means that the formula can just use the requirement for task a instead of finding the maximum across all tasks in the same job. In the second case, the required number only depends on the task, which means that instead of $\dot{m}_a$, mode index 1 can be used as an arbitrary value.

Resource Classes

Just like TLSP, TLSP-GR deals with heterogeneous resources, where individual units from the same group may be suitable to perform different tasks. However, many benchmark and real-world instances contain some resources that are interchangeable. Interchangeable in the sense that swapping them for one another in a solution cannot introduce new constraint violations or change the penalty. Similarly to [Dan19], the model in this work takes advantage of this fact to improve performance. This is done by merging equivalent resources into *resource classes*, whose corresponding assignment variables in the CP model are converted from boolean to integer. In contrast to existing work, the optimization presented here is not limited to employees and addresses an issue that would result from a naive generalization.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The idea for TLSP was to consider equipment equivalent if they are available to and preferred by the same tasks and be required by the same tasks in the same situations. The latter is automatically satisfied by demanding that the resources are in the same group. Transferring this idea to TLSP-GR, two resources $r_1 \in R_i, r_2 \in R_i$ from the same resource group i could be considered equivalent if they are available and preferred for the exact same tasks.

The problem with this idea in TLSP-GR is that the identity of resources may be relevant for resources affected by the linked tasks hard constraint (3.25) or the resource minimization soft constraint (3.28) . Straightforwardly picking resource units in post-processing is no longer possible when linked tasks are involved, and the CP model can't calculate the number of different assigned resource units for a project if it doesn't yet know what those units are. One possible solution would be re-introducing variables for the individual resources into the CP model, and to link them to the resource class assignments, and using those for the resource minimization soft constraint. Enforcing that there is an assignment for individual resources that fulfills the linked tasks constraint would then correspond to a graph coloring sub-problem. A simpler solution would be to consider all resource units to be different for resource groups affected by one of the two constraints. For this Thesis, I chose the latter option to keep the CP model simpler. For existing TLSP instances, both artificial and real-world, this is likely also the more efficient solution overall. This is because only employees are affected by those constraints. When analyzing existing synthetic and real-world instances, employees are almost never equivalent according to the criteria above.

Thus we define the equivalence relation X_i for resource group $R_i \in R^*$ by:

$$\begin{aligned} r_1 X_i r_2 \iff & R_i \notin (L^{rg} \cup R^{min}) \wedge \\ & \forall_{a \in A^*}((r_1 \in R_{a,g} \iff r_2 \in R_{a,g}) \wedge (r_1 \in R^{Pr}_{a,g} \iff r_2 \in R^{Pr}_{a,g})) \\ & R_i \in R^*, r_1 \in R_i, r_2 \in R_i \end{aligned}$$

For each resource group R_i, the set $C_i := R_i/X$ is the set of resource equivalence classes, henceforth called resource classes. For some $c \in C_i$, $|c|$ is the number of resource units in resource class c.

We can now define available and preferred resources for tasks in terms of resource classes:

$$\begin{aligned} R^C_{a,g} &:= \{c \mid c \in C_g \wedge (\forall_{r \in c} r \in R_{a,g})\} && R_g \in R^*, a \in A^* \\ R^{Pr,C}_{a,g} &:= \{c \mid c \in C_g \wedge (\forall_{r \in c} r \in R^{Pr}_{a,g})\} && R_g \in R^*, a \in A^* \end{aligned}$$

The resource-related constraints from the CP model can then be reformulated using resource classes. The assignment of resource classes to jobs is now given by:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

$$\dot{a}_{a,c} \in \{0, 1, \ldots, |c|\} \mid a \in A^*, R_i \in R^*, c \in C_i \qquad \ldots \text{Job Resource Assignment}$$

The new hard constraints are:

$$\texttt{cumulative}((\dot{s}_a)_{a\in A^*}, (\dot{d}_{\dot{m}_a,a})_{a\in A^*}, (\dot{a}_{a,c})_{a\in A^*}, |c|) \qquad R_g \in R^*, c \in C_g \tag{3.32}$$

$$\sum_{c\in C_g} \dot{a}_{a,c} = \max_{a_2 \in A^* | \dot{\xi}(a_2)=a} r_{a_2,g,\dot{m}_a} \qquad a \in J, R_g \in R^* \tag{3.33}$$

$$\dot{a}_{\dot{\xi}(a),c} > 0 \wedge \max_{m\in M} r_{a,g,m} > 0 \implies c \in R^C_{a,g} \qquad a \in A^*, R_g \in R^*, c \in C_g \tag{3.34}$$

$$\dot{a}_{\dot{\xi}(a),c} = \dot{a}_{\dot{\xi}(a_2),c} \qquad p \in P, (a, a_2) \in L_p, c \in C_g \text{ where } R_g \in L^{rg} \tag{3.35}$$

Constraint (3.32) replaces (3.22) and enforces that resources are not used by multiple jobs at the same time. Because resources can now be somewhat homogenous, this amounts to enforcing that the number of units used from any resource class c never exceeds its size, $|c|$. Constraint (3.33) replaces (3.23) and enforces that the number of resources assigned to a job exactly covers the requirement of its most demanding task. It is very similar to (3.23) , except $\dot{a}_{a,c}$ is now an integer instead of a boolean. Constraint (3.34) enforces resource availability and replaces (3.24) . Finally, (3.35) replaces (3.25) and enforces that the jobs formed from linked tasks must have the same amount of resources assigned for each class. Since resources for groups $R_g \in L^{rg}$ are always considered distinct, the classes $c \in C_g$ have $|c| = 1$, and the assignment variables are effectively binary here.

$$s_2 = \sum_{j\in J} \sum_{R_g \in R^*} \sum_{c \in (C_g \setminus (\bigcap_{a\in A | \dot{\xi}(a)=j} R^{Pr,C}_{a,g}))} \dot{a}_{c,j} \tag{3.36}$$

$$s_3 = \sum_{p\in P} \sum_{R_g \in R^{min}} \sum_{c\in C_g} ((\sum_{a\in A_p} \dot{a}_{c,a}) > 0) \tag{3.37}$$

Soft constraint (3.36) replaces (3.29) and penalizes the number of non-preferred assigned resources. (3.37) replaces (3.30) and penalizes the number of different resources from groups in R^{min} assigned to each project. Like (3.35) , resource classes from affected groups always have size 1, which ensures that this constraint indeed counts distinct resource units.

Once a solution to this modified model has been obtained, the assignments for resource classes can be converted back to resource piece assignments by choosing arbitrary units from the assigned classes, while keeping track of which units from a class have been assigned and which are still free. Because resource units from the same class are fungible, this can never fail.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Extending CP-SAT with Priority Search

Google *OR-Tools* [OrT] is an open source software suite for solving combinatorial optimization problems. Alongside interfaces to external solvers and a custom MIP solver, it includes the *CP-SAT* solver [PD21]. CP-SAT is a portfolio solver for constraint programming. It combines a SAT-based lazy clause generation CP solver together with various other techniques like linear relaxations and local neighborhood search. In the past years, CP-SAT routinely won gold medals in the MiniZinc Challenge [SFS+14], which compares the performance of different solvers on CP instances from many different problem categories, each generated from MiniZinc code. For instance, in the 2023 challenge [min23], CP-SAT won first place in all relevant categories, outperforming open solvers like Chuffed, but also commercial ones like CPLEX or Gurobi.

However, when the original solver for the Test Laboratory Scheduling Problem (TLSP) was developed, the Chuffed solver [Chu11] was chosen instead. This is because Chuffed supports the priority search [FGS+17] annotation, which was crucial for achieving high performance on TLSP. Compared to traditional search annotations, priority search allows specifying that the solver should switch between different sub-annotations, depending on current variable bounds. In the case of TLSP, this allows for a search strategy that schedules tasks one by one, including time slots as well as resource assignments, and picks the task with the earliest possible start time each time. In contrast, the possible strategies with traditional annotations are more restricted: one could schedule all start times based on the earliest possible start (afterward followed by resource assignments). To schedule tasks as a unit with traditional annotations (including time slots, resource assignments, and mode assignments), the order would need to be predetermined. Initial experimentation with TLSP showed that Chuffed paired with priority search was usually more performant than other solvers like CP-SAT with simpler strategies.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Regarding TLSP and TLSP-GR, the lack of priority search in CP-SAT appears to be a significant drawback of an otherwise highly promising solver. In fact, in addition to the great performance at the MiniZinc Challenge, CP-SAT would hold another more practical advantage for our industrial partner: CP-SAT being a portfolio solver, it features a robust multi-threading mode. Given the abundance of multicore CPUs in today's world, this looks like a promising way to increase performance for their real-world scheduling operations.

This chapter addresses this challenge by introducing an implementation of priority search for the CP-SAT solver in Google OR-Tools. The implementation proved challenging and required several changes to internal data structures and algorithms since CP-SAT lacks support for nested search annotations. Additionally, other extensions (hot start and randomized variable selection) are considered.

The rest of this Chapter is structured as follows:

- Section 4.1 explains how search strategies work in CP-SAT, and how it uses and processes MiniZinc search annotations. It also explains the challenges to implementing priority search. This Section is based on my own analysis of the project's source code.
- Section 4.2 describes my implementation of priority search in CP-SAT, and which changes to the data structures and algorithms were necessary.
- Section 4.3 briefly describes two other adjustments I made to CP-SAT to support randomized variable selection and hot starts, both of which are useful for the VLNS algorithm for TLSP-GR.

4.1 Search in CP-SAT

The implementation of CP-SAT as a backend solver for MiniZinc is split into two parts. Firstly, the CP-SAT solver itself, which accepts instances in the form of custom data structures based on protocol buffers [Goo23], a data exchange format by Google. Secondly, there is a wrapper that converts input and output between MiniZinc's format and protocol buffers.

For search, the CP-SAT solver can use internal heuristics as well as a user-specified search strategy. When using CP-SAT as a MiniZinc backend, search annotations from MiniZinc are converted into `DecisionStrategyProto` objects that CP-SAT uses to receive user-specified decision strategies.

```
message DecisionStrategyProto {
   repeated int32 variables = 1;

   enum VariableSelectionStrategy {
      CHOOSE_FIRST = 0;
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
      CHOOSE_LOWEST_MIN = 1;
      CHOOSE_HIGHEST_MAX = 2;
      CHOOSE_MIN_DOMAIN_SIZE = 3;
      CHOOSE_MAX_DOMAIN_SIZE = 4;
   }
   VariableSelectionStrategy variable_selection_strategy = 2;

   enum DomainReductionStrategy {
      SELECT_MIN_VALUE = 0;
      SELECT_MAX_VALUE = 1;
      SELECT_LOWER_HALF = 2;
      SELECT_UPPER_HALF = 3;
      SELECT_MEDIAN_VALUE = 4;
   }
   DomainReductionStrategy domain_reduction_strategy = 3;

   message AffineTransformation {
      int32 index = 1;
      int64 offset = 2;
      int64 positive_coeff = 3;
   }
   repeated AffineTransformation transformations = 4;
}

[...]

message CpModelProto {
  [...]

  repeated DecisionStrategyProto search_strategy = 5;
}
```

Listing 4.1: Snippet from `ortools/date/cp_model_search.cc` [PF21a], with comments removed for brevity. Defines the data structure used for search annotations in CP-SAT.

Listing 4.1 presents the data format used for search strategies in CP-SAT. A CP instance (`CpModelProto`) can contain a list of decision strategies (`DecisionStrategyProto` objects). Each decision strategy contains a list of variables to branch on, a variable selection strategy, and a domain reduction strategy. The `transformations` field can be used to preserve search behavior when transformations are applied to variables but is not relevant to this Thesis.

One `DecisionStrategyProto` object roughly maps onto one `int_search` annotation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

from MiniZinc. In contrast, MiniZinc's `seq_search` annotations have no correlate in this format. This means that the tree structure of MiniZinc search annotations cannot be directly represented by the input format of CP-SAT. However, representing the tree is not necessary to support MiniZinc's `seq_search` and `int_search` annotations. Since the annotations in `seq_search` are always used in the same order, the CP-SAT preprocessor can flatten the tree with an in-order traversal, which results in a list of `int_search` annotations. This list is then converted into the `search_strategy` array in `CpModelProto`. Traversing this list always yields the same order of search strategies as traversing the original tree.

This behavior presents a problem for `priority_search`, since `priority_search` annotations require the solver to change the order of search annotations based on dynamic variable bounds. Therefore, implementing `priority_search` requires changes to the way search strategies are represented in CP-SAT.

Algorithm 1 Pseudocode for a simplified version of the CP-SAT implementation of search heuristics. The original C++ code is from [PF21b], while the pseudocode was created by me.

```
function FindNextVariableToFix(strategies, view)
    for strategy in strategies do
        candidate ← null
        candidateValue ← ∞
        for i ← 1 to strategy.variables.size() do
            var ← strategy.variables[i]
            if view.IsFixed(var) then
                continue
            value ← ApplyVariableSelectionStrategy (
                                strategy.variableSelectionStrategy, var)
            if value < candidateValue then
                candidate ← var
                candidateValue ← value
        if candidateValue = ∞ then
            continue
        return ApplyDomainReductionStrategy (
                                    strategy.domainReductionStrategy, candidate)
    return null
```

Algorithm 1 shows how CP-SAT uses the decision strategies during the search process. `FindNextVariableToFix` is called when the solver needs to branch on a new variable. The parameter `strategies` contains the `search_strategy` array from Listing 4.1, while `view` contains information about the current solver state. The aim of the function is to follow the order of the search strategy until it finds a non-fixed variable, and then return that variable combined with its new domain. The outermost loop iterates over

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

all decision strategy objects. Then an inner loop iterates over the variables included in the strategy. Each non-fixed variable from the strategy is first evaluated based on the variable selection strategy from the current `DecisionStrategyProto` object. The algorithm keeps track of the best variable it found (`candidate`) and a numerical rating (`candidateValue`) based on the heuristic. The implementation of the variable selection heuristic is abstracted away into a function `ApplyVariableSelectionStrategy` for simplicity. After the inner loop, if no non-fixed variables have been found, the outer loop continues with the next `DecisionStrategyProto` object. If the inner loop finds a non-fixed variable to branch on, the best such variable is returned, combined with its new restricted domain according to the domain reduction strategy.

Like with the data structures earlier, the structure of the algorithm needs to be changed to account for the dynamic order in which search annotations must be processed with `priority_search`.

4.2 Priority Search Implementation

As explained in the previous Section, implementing priority search in CP-SAT requires changing how decision strategies are represented internally.

Since protocol buffers support recursive data structures, the protocol buffers can be modified to allow for tree structures.

```
message DecisionStrategyProto {
   repeated int32 variables = 1;

   [...]

   enum DomainReductionStrategy {
      SELECT_MIN_VALUE = 0;
      SELECT_MAX_VALUE = 1;
      SELECT_LOWER_HALF = 2;
      SELECT_UPPER_HALF = 3;
      SELECT_MEDIAN_VALUE = 4;
      PRIORITY_SEARCH = 5;
   }
   DomainReductionStrategy domain_reduction_strategy = 3;

   [...]

   repeated DecisionStrategyProto searches = 5;
}
```

Listing 4.2: My modifications to the data structures from 4.1, allowing the protocol buffers to express nested search strategies.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Listing 4.2 shows my modifications to the protocol buffers. The intended semantics is as follows: when `DomainReductionStrategy` is set to a value other than `PRIORITY_SEARCH`, `searches` is ignored and the annotation is used to find a variable to branch on, like before. If `PRIORITY_SEARCH` is selected, `searches` must be populated and have the same length as `variables`. In this case, variables are still ranked based on the variable selection strategy. However, instead of restricting the domain of the non-fixed variable with the highest score, the solver instead looks for the highest-ranking variable whose corresponding search heuristic yields a branching decision.

Algorithm 2 shows my new implementation for using user-specified search strategies in CP-SAT. Instead of looping over a list of search strategies, the algorithm is now recursive and traverses a tree of search strategies. For compatibility, `FindNextVariableToFix` still matches the signature shown in Algorithm 1, taking in a list of search strategies and a `view` object containing the solver's current state. `FindNextVariableToFix` essentially contains the outer loop from Algorithm 1, iterating over its list of search strategies. Each search strategy is handled by the recursive function `SearchRecursively`.

`SearchRecursively` is where the heart of the algorithm is implemented. It is split into two main blocks, based on whether the given search strategy is `PRIORITY_SEARCH` or not. For annotations like `int_search`, the `else`-block matches the inner loop in Algorithm 1. The first block implements `PRIORITY_SEARCH`. Like for other search annotations, variables are ranked according to the variable selection strategy. However, unlike before, it is no longer trivial to check whether selecting a variable leads to a valid branching decision. Instead of a single `view.IsFixed(var)` check, this would require a recursive call to `SearchRecursively`. While replacing the check with a recursive call would be possible, I decided to rank and sort all variables first, reducing the number of recursive calls as much as possible. The performance impacts of this choice may depend on the individual instance. Concretely, the algorithm ranks all variables with the variable selection heuristic, saving the scores and variables to the `allCandidates` array. It then sorts the array based on the scores and attempts recursive calls based on that order. When a recursive call returns a branching decision, that decision is used.

4.3 Other Adjustments to CP-SAT

In addition to priority search, I also made two more adjustments to CP-SAT to improve performance with the CP model from Section 3.4 and the VLNS algorithm. Both were conceptually much simpler and far easier to implement.

- Supporting random variable selection. This is sometimes used for resource assignments by the VLNS algorithm. Implementing it simply involved extending the `VariableSelectionStrategy` enum in `DecisionStrategyProto` from Listing 4.1, and implementing the new strategy in Algorithm 2 using a pseudo-random number generator.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Algorithm 2 My new implementation of search heuristics for CP-SAT, now supporting `priority_search` annotations.

```
function FindNextVariableToFix(strategies, view)
    for strategy in strategies do
        result ← SearchRecursively(strategy, view)
        if result.HasValue() then
            return result
    return null
function SearchRecursively(strategy, view)
    if strategy.domainReductionStrategy = PRIORITY_SEARCH then
        allCandidates ← ∅
        for i ← 1 to strategy.variables.size() do
            var ← strategy.variables[i]
            value ← ApplyVariableSelectionStrategy (
                                strategy.variableSelectionStrategy, var)
            allCandidates ← allCandidates ∪ {(value, i)}
        Sort(allCandidates)
        for (candidateValue, candidate) in allCandidates do
            next ← strategy.searches[candidate]
            recursiveResult ← SearchRecursively(next, view)
            if recursiveResult.HasValue() then
                return recursiveResult
        return null
    else
        candidate ← null
        candidateValue ← ∞
        for i ← 1 to strategy.variables.size() do
            var ← strategy.variables[i]
            if view.IsFixed(var) then
                continue
            value ← ApplyVariableSelectionStrategy (
                                strategy.variableSelectionStrategy, var)
            if value < candidateValue then
                candidate ← var
                candidateValue ← value
        if candidateValue = ∞ then
            return null
        return ApplyDomainReductionStrategy (
                                strategy.domainReductionStrategy, candidate)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

- Supporting MiniZinc's `warm_start(array of var int, array of int)` annotation. The annotation allows the model to specify an initial variable assignment, for instance, to specify a known feasible solution. Since CP-SAT already implements a similar feature, this was a matter of translating the FlatZinc annotation to the appropriate protocol buffer. A complication was that the FlatZinc parser for CP-SAT seemed to strip the `array of int` part of the annotation. As a workaround, I used two annotations `warm_start_ortools(array of var int, array of warm_start_value)` and `warm_start_value(int)`. This allowed me to access the data without modifying the parser.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Experimental Results

This Chapter contains computational results for the approaches presented in earlier chapters. It is structured as follows:

- Section 5.1 described the synthetic and real-world benchmark instances used in this chapter.
- Section 5.2 evaluates the performance of the new TLSP-GR CP model to solve TLSP instances, both on its own and in conjunction with the VLNS algorithm.
- Section 5.3 shows computational results for the CP-SAT solver and my priority search implementation.
- Section 5.4 compares select results from the previous sections against the state of the art.

5.1 Benchmark Instances

The evaluations in this Chapter are based on instances from the literature [MM18a]. Overall, there are 33 instances. 30 of these instances have been randomly generated based on statistical patterns of real-world data. The other 3 instances have been taken directly from the day-to-day scheduling operations of the industrial partner. All 33 instances are available for download online at `https://www.dbai.tuwien.ac.at/staff/fmischek/TLSP/`. The instances and some of their properties are shown in table 5.1, with data from [MM18a].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

#	Data Set	ID	$\|P\|$	$\|A^*\|$	$\|T\|$	$\|E\|$	$\|B\|$	$\|G^*\|$
1	General	000	5	13	88	7	7	3
2	General	001	5	20	88	7	7	3
3	LabStructure	000	5	73	88	7	7	3
4	LabStructure	001	5	58	88	7	7	3
5	General	005	10	86	88	13	13	4
6	General	006	10	62	88	13	13	6
7	LabStructure	005	10	102	88	13	13	3
8	LabStructure	006	10	93	88	13	13	3
9	General	010	20	182	174	16	16	5
10	General	011	20	273	174	16	16	4
11	LabStructure	010	20	224	174	16	16	3
12	LabStructure	011	20	213	174	16	16	3
13	General	020	15	80	174	12	12	5
14	LabStructure	020	15	123	174	12	12	3
15	General	025	30	376	174	23	23	3
16	LabStructure	025	30	422	174	23	23	3
17	General	015	40	405	174	31	31	3
18	LabStructure	015	40	429	174	31	31	3
19	General	030	60	613	174	46	46	6
20	LabStructure	030	60	775	174	46	46	3
21	General	035	20	304	520	6	6	5
22	LabStructure	035	20	280	520	6	6	3
23	General	040	40	714	520	12	12	4
24	LabStructure	040	40	661	520	12	12	3
25	General	045	60	940	520	18	18	6
26	LabStructure	045	60	837	520	18	18	3
27	General	050	60	866	782	13	13	4
28	LabStructure	050	60	891	782	13	13	3
29	General	055	90	1282	782	19	19	5
30	LabStructure	055	90	1573	782	19	19	3
Lab1	Real-world	–	74	856	606	22	17	1
Lab2	Real-world	–	59	678	700	24	22	1
Lab3	Real-world	–	59	660	572	19	17	1

Table 5.1: A table of benchmark instances used for the evaluations in this chapter. The data for the table is taken from [Mis22]. The first 30 instances are randomly generated, while the last 3 are snapshots from the real-world scheduling system of the industrial partner. The randomly generated instances range from fairly small to very large and are further divided into *LabStructure* instances, whose statistical properties are designed to mirror real-world use cases, and *General* instances, which are designed to have more variation. For each instance, a few data points are given: the number of projects $|P|$, the number of tasks $|A^*|$, the number of time slots in the scheduling horizon, $|T|$. Finally, there are the number of employees, $|E|$, workbenches, $|B|$ and equipment groups, $|G^*|$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.2 TLSP with Generalized Resources

This Section presents benchmark results to measure the ability of the new TLSP-GR solution approaches to solve TLSP instances. To solve a TLSP instance with a TLSP-GR solver, the procedure from Section 3.3 is used to convert it. Benchmarks are performed both for the novel CP model presented in Section 3.4, as well as the augmented VLNS algorithm from Section 2.5, which is based on the aforementioned CP model. The results are compared to the existing state-of-the-art CP model from [DGMM20] and a variant of the VLNS algorithm from Section 2.5 that is identical except for the CP model used. The VLNS algorithm is a state-of-the-art solution approach for TLSP [Mis22], especially for longer runtimes like the ones used here.

Because I have access to the TLSP solvers, I decided to perform a like-for-like comparison by running both solvers on the same hardware, as opposed to comparing the literature and trying to normalize the runtimes for hardware differences. Each configuration was evaluated 3 times.

The experiments were performed on a machine with a 12 core *AMD Ryzen 9 5900X* desktop processor running at stock configuration and *64GB* of *DDR4-3200* RAM. Since Chuffed is single-threaded, 8 instances of the solver were executed in parallel to save time. Each solver was allotted a runtime of 30 minutes, which is roughly based on the 2 hours from the literature [DGMM20] and adjusted for hardware differences.

The aim here is not to show performance improvements but to demonstrate that TLSP instances can still be solved efficiently using the TLSP-GR solvers. The focus of TLSP-GR was not to increase performance, but to increase modeling power and make the problem more flexible to encompass a higher number of real-world problems. My expected result was a loss in performance, albeit hopefully manageable. This is based on the intuition that the new model might be more indirect for some constraints and uses more integer and fewer boolean variables. This could affect propagation in the CP solver.

To prevent performance regressions with the existing model, I used the same CP solver, *Chuffed* [Chu11], as in [DGMM20]. I had to use a slightly newer version, 0.10.4 (as opposed to 0.10.3 in [DGMM20]), because using the new TLSP-GR CP model with Chuffed 0.10.3 led to some internal crashes of the solver. This was due to a bug fixed in version 0.10.4. To ensure a fair comparison, both the existing TLSP model and the new TLSP-GR model were benchmarked using the 0.10.4 version.

The VLNS solver is based on the existing TLSP VLNS algorithm explained in Section 2.5, except it works with TLSP-GR instances and uses the new TLSP-GR CP model internally. The parameters for the VLNS algorithm have been taken from [DGMM20], except that *fixedMzTimeout* and *variableMzTimeout* have been reduced by a factor of 4 along with the total algorithm runtime to account for hardware differences. The new timeouts are now 5000ms and 10000ms, respectively. No extra parameter tuning was performed for the new TLSP-GR model.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

#	TLSP				TLSP-GR			
	CP		VLNS		CP		VLNS	
	Avg	Best	Avg	Best	Avg	Best	Avg	Best
1	57	**57**	57	**57**	57	**57**	57	**57**
2	71	**71**	71	**71**	71	**71**	71	**71**
3	142	142	141	**141**	142	142	141	**141**
4	120	120	103	103	119	119	102	**102**
5	244	244	248	**240**	287	287	240	**240**
6	180	180	140	**140**	188	188	140	**140**
7	359	359	284	284	415	411	283	**283**
8	311	311	284	284	311	311	283	**282**
9	689	689	423	**415**	729	729	428	428
10	956	956	507	**502**	1029	1029	507	503
11	1053	1053	817	**812**	1055	1055	820	815
12	737	737	646	644	804	804	648	**643**
13	336	336	316	316	364	364	315	**313**
14	453	453	411	**410**	492	492	413	413
15	1723	1723	926	901	1753	1753	903	**887**
16	1574	1574	1115	**1111**	1588	1588	1114	1113
17	1424	1424	1052	1045	1516	1516	1045	**1040**
18	1847	1847	1381	1369	1803	1803	1354	**1353**
19	2715	2715	1863	**1821**	2805	2805	2010	1964
20	3076	3076	2154	2142	2857	2857	2150	**2118**
21	949	949	570	**569**	1137	1137	571	571
22	980	980	731	**713**	1053	1053	729	721
23	T/O	T/O	1656	**1602**	T/O	T/O	1676	1644
24	T/O	T/O	1762	1747	T/O	T/O	1741	**1737**
25	4340	4340	1906	**1884**	3694	3694	1970	1940
26	3823	3823	2599	**2537**	3812	3812	2601	2579
27	3743	3743	1779	**1762**	3378	3378	1792	1781
28	3147	3147	2242	**2236**	2647	2647	2244	2239
29	T/O	T/O	2964	**2897**	T/O	T/O	3082	3062
30	6543	6543	4571	**4465**	6295	6295	4632	4585
Lab1	4990	4990	3470	**3449**	4709	4709	3774	3742
Lab2	3407	3407	2695	2653	3285	3285	2700	**2642**
Lab3	2972	2971	2608	2598	3013	3013	2587	**2580**

Table 5.2: Penalties for the solutions found after 30 minutes by the TLSP-GR solvers, compared to existing TLSP solvers. **CP** refers to using the respective CP model with Chuffed. **VLNS** uses the VLNS algorithm to first find a feasible solution, starting from a greedy (infeasible) schedule, and then iteratively improve upon it. Cells with **T/O** indicate that none of the runs were successful. Cells with numbers indicate that all 3 runs were successful, presenting the average and best penalties. No solver solved any instance only some of the time. For every instance, the best solution found by any solver is highlighted in bold.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.2 presents the results. Overall, the new TLSP-GR model is quite competitive with the existing one for TLSP. In both cases, the CP model solved 27 out of 30 instances. Overall, pure CP performance varies from instance to instance. Depending on the instance, either the TLSP-GR or TLSP model may have a significant lead. While the run-to-run variance is typically 0 for the pure CP model, likely due to the fixed search order, the differences in the models could cause the solver to enter different parts of the search space for TLSP and TLSP-GR. Depending on the instance, either solver could gain the advantage. The TLSP model seems to hold the advantage on many of the medium-sized instances (#5-#19), while the TLSP-GR model appears to pull ahead the largest instances (#25-#30 and the real-world instances).

When considering the VLNS algorithm, the TLSP-GR variant often lags behind a bit.

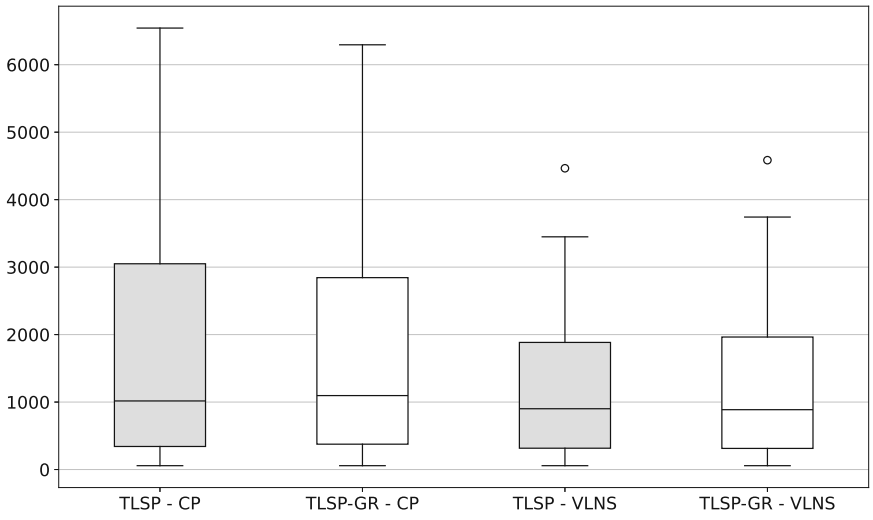

Figure 5.1: Boxplot representation of the results from Table 5.2. For the CP results, instances `23`, `24` and `29` were excluded due to infeasibility.

Figure 5.2 shows a condensed version of the results.

In summary, the new TLSP-GR CP model can achieve better or worse results than the existing TLSP model, depending on the instance. Both models could solve the same training instances. Further performance may be gained for VLNS with the TLSP-GR model by performing additional parameter tuning. Overall, the performance of both models appears to be quite similar.

5.3 Priority Search in CP-SAT

The evaluations in this Section investigate the impact of priority search on the performance of CP-SAT when solving TLSP-GR instances. I used the new TLSP-GR CP model

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

proposed in Chapter 3 together with the TLSP test instances from Section 5.1, converted to TLSP-GR.

As a CP solver, I used my modified version of CP-SAT 9.0 presented in Chapter 4 that introduces support for priority search. Each configuration was evaluated 3 times. Although CP-SAT supports multi-threading, the experiments in this Chapter only used the single-threaded mode to keep the results comparable to other solvers. The free search MiniZinc parameter was always enabled since it empirically seemed to noticeably improve performance in all situations. Free search, in principle, allows the solver to deviate from a specified search strategy. For CP-SAT in single-threaded mode, it also seems to enable an interleaved execution of part of the solver portfolio.

All evaluations were performed on a machine with a 12 core *AMD Ryzen 9 5900X* desktop processor running at stock configuration and *64GB* of *DDR4-3200* RAM. Since only single-threaded configurations were evaluated, 8 instances of the solver were executed in parallel to save time. Each solver was allotted a runtime of 30 minutes, which is roughly based on the 2 hours from the literature [DGMM20] and adjusted for hardware differences.

5.3.1 CP Model

For CP, I compare the results of different search strategies with CP-SAT when using the CP model alone. There are three configurations I consider: the default search strategy of the solver (with no MiniZinc annotations), a fixed search strategy that is similar to the priority search one except for scheduling tasks in a pre-determined order, and a priority search strategy that schedules tasks in a flexible order based on the lower bound for their starting time at runtime. The first two search strategies can run on an unmodified version of CP-SAT, while the latter variant only works with my modifications to the solver.

Table 5.3 shows the results for different search strategies. First, using only the solver's default search strategy, it solves 22 of 33 instances, 5 of which to optimality. This stands in contrast to the experience on TLSP [Dan19] with Chuffed, which could only solve the smallest instances with its default search strategy. Both custom search strategies significantly outperformed the default search, with fixed search and priority search solving 30 and 29 out of 33 instances, respectively. To my surprise, the difference between the fixed search and priority search is quite small, with these configurations trading blows depending on the instance. However, the fixed search strategy appears to have an edge on the largest instances.

5.3.2 VLNS

In addition to the pure CP solver, I also compared the different search strategies when incorporated into the VLNS algorithm described in Section 2.5. The VLNS algorithm repeatedly uses the CP model to solve small partial instances to reschedule a small

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

#	CP (CP-SAT)					
	Solver Search		**Fixed Search**		**Priority Search**	
	avg	**best**	**avg**	**best**	**avg**	**best**
1	57	**57***	57	**57***	57	**57***
2	71	**71***	71	**71***	71	**71***
3	141	**141***	141	**141***	141	**141***
4	101	**101***	101	**101***	101	**101***
5	242	242	240	**240***	240	**240***
6	140	**140***	140	**140***	140	**140***
7	295	295	285	**285**	292	292
8	289	289	287	**287**	288	288
9	461	**428**	438	438	470	470
10	754	754	772	772	701	**699**
11	841	**841**	855	855	855	853
12	776	774	694	693	675	**665**
13	314	**310**	313	313	311	311
14	422	**422**	422	**422**	431	424
15	1529	**1524**	1739	1739	1662	1580
16	1687	1674	1445	**1408**	1478	1454
17	1308	**1304**	1356	1355	1422	1406
18	2094	2042	1745	1701	1608	**1601**
19	2923	**2815**	3318	3257	3212	3187
20	t/o	t/o	3861	**3829**	4036	4008
21	745	742	716	692	678	**669**
22	865	**805**	943	925	873	873
23	t/o	t/o	4290	**4280**	4576	4562
24	4080	3960	3351	3345	3292	**3281**
25	t/o	t/o	t/o	t/o	t/o	t/o
26	t/o	t/o	4829	**4805**	4933	4908
27	t/o	t/o	4996	4864	4628	**4352**
28	t/o	t/o	4236	**4096**	4298	4298
29	t/o	t/o	t/o	t/o	t/o	t/o
30	t/o	t/o	t/o	t/o	t/o	t/o
Lab1	t/o	t/o	7030	**7008**	t/o	t/o
Lab2	t/o	t/o	3983	**3951**	4441	4439
Lab3	t/o	t/o	2835	**2819**	2927	2917

Table 5.3: CP penalties after 30 minutes. **Solver Search** uses the default search strategy of the solver, **Fixed Search** a search strategy that schedules tasks in a fixed order, and **Priority Search** the search strategy that takes advantage of my new implementation to dynamically decide in which order to schedule tasks. Out of 3 runs, average and best scores are reported. `t/o` indicates that all runs timed out. When only some of the runs were feasible, the number of feasible runs is shown instead of the average. **Bold** values indicate the best score for the instance in this table. * indicates that the solver proved optimality.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

number of projects. I applied the respective search strategy to all invocations of the CP solver within the algorithm.

Like in the CP experiments, I compare three search strategies: the solver's default strategy, a fixed search strategy that is similar to the priority search one except for scheduling tasks in a pre-determined order, and a priority search strategy that schedules tasks in a flexible order based on the lower bound for their starting time at runtime. In addition, to test the benefit of implementing hot start and random variable selection, I include the unmodified CP-SAT 9.0 solver as well.

The parameters for the VLNS algorithm are based on [DGMM20], except that *fixedMzTimeout* and *variableMzTimeout* have been reduced to 5000ms and 10000ms, respectively. This is the same reduction as with the overall solver timeout, to account for hardware differences. I performed some additional parameter tuning with *SMAC v3* [LEF+17], but after over 1000 runs, no better parameter configuration was returned and I had to stop due to time reasons.

Table 5.4 contains the benchmark results for the VLNS algorithm, where the inner CP model uses different search strategies. The addition of hot start, and random resource selection makes a big difference: the unmodified solver could only solve 17 out of 33 instances at least sometimes, while the modified solver with the same default search strategy could solve 26. As with CP, the fixed search strategy is roughly on par with priority search, both being able to solve 32 instances at least once.

5.3.3 Summary

Using the priority search search strateghy significantly increases performance compared to the solver's default search strategy, but very similar performance can be achieved via a fixed search strategy that does not use the priority search feature. This holds for CP and VLNS. However, the other extensions to CP-SAT, hot start and randomized variable selection, make a big difference for VLNS, making 6 additional instances solvable and improving penalties almost everywhere.

Figure 5.2 shows aggregate penalties for the two custom search strategies, fixed search, and priority search. The penalties are very similar overall.

5.4 Comparison to State of the Art

This Section aggregates data from the previous evaluation Sections to compare my results to the state of the art on TLSP.

The state-of-the-art results are taken from [DGJ+23], as that was the most up-to-date and comprehensive source I could find. The results are equivalent to [Mis22]. The state-of-the-art results are based on a dual *AMD Opteron 6272* system with a timeout of 2 hours. In my experiments, I used a timeout of 30 minutes that roughly adjusts for the increased performance of my *AMD Ryzen 9 5900X* system.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

#	VLNS							
	CP-SAT		Custom CP-SAT					
	Solver Search		Solver Search		Fixed Search		Priority Search	
	Avg	Best	Avg	Best	Avg	Best	Avg	Best
1	79	79	57	**57**	57	**57**	57	**57**
2	73	73	71	**71**	71	**71**	71	**71**
3	145	145	141	**141**	141	**141**	141	**141**
4	105	105	101	**101**	101	**101**	101	**101**
5	263	246	240	**240**	240	**240**	240	**240**
6	161	161	140	**140**	140	**140**	140	**140**
7	287	286	283	**283**	283	**283**	283	**283**
8	304	303	284	284	283	**283**	284	**283**
9	493	492	416	415	413	**413**	419	**413**
10	t/o	t/o	504	494	500	494	494	**493**
11	t/o	t/o	815	**810**	816	815	810	**810**
12	(2/3)	649	(1/3)	642	640	**640**	641	**640**
13	340	340	312	310	308	**308**	309	309
14	417	416	411	411	411	411	410	**410**
15	t/o	t/o	t/o	t/o	915	**862**	884	864
16	1163	1138	1108	1102	1100	**1095**	1100	1099
17	1167	1159	1041	1035	1037	**1032**	1035	1033
18	t/o	t/o	1349	1340	1332	1332	1335	**1329**
19	t/o	t/o	(1/3)	2222	(2/3)	**2131**	(1/3)	2470
20	t/o	t/o	2450	2301	2282	**2158**	2333	2192
21	679	676	543	539	542	**537**	548	547
22	760	756	712	**704**	712	709	712	707
23	t/o	t/o	(1/3)	1833	1555	**1495**	1548	1536
24	t/o	t/o	t/o	t/o	1718	1715	1716	**1713**
25	t/o	t/o	1966	1901	1928	1904	1931	**1893**
26	t/o	t/o	t/o	t/o	2633	**2530**	2690	2652
27	t/o	t/o	1815	1798	1746	**1693**	1712	1698
28	2307	2306	2237	2224	2227	**2216**	2228	2225
29	t/o	t/o	t/o	t/o	(1/3)	4027	(2/3)	**3585**
30	t/o	t/o	t/o	t/o	4599	4495	4572	**4467**
Lab1	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
Lab2	t/o	t/o	(1/3)	2800	2637	2625	(2/3)	**2591**
Lab3	t/o	t/o	t/o	t/o	2575	2568	2570	**2559**

Table 5.4: VLNS penalties after 30 minutes. **CP-SAT** refers to the unmodified version of CP-SAT, while **Custom CP-SAT** includes my modifications to support priority search, hot start, and random variable selection strategies. **Solver Search** uses the default search strategy of the solver, **Fixed Search** a search strategy that schedules tasks in a fixed order, and **Priority Search** the search strategy that takes advantage of my new implementation to dynamically decide in which order to schedule tasks. Out of 3 runs, average and best scores are reported. `t/o` indicates that all runs timed out. When only some of the runs were feasible, the number of feasible runs is shown instead of the average. **Bold** values indicate the best score for the instance in this table.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

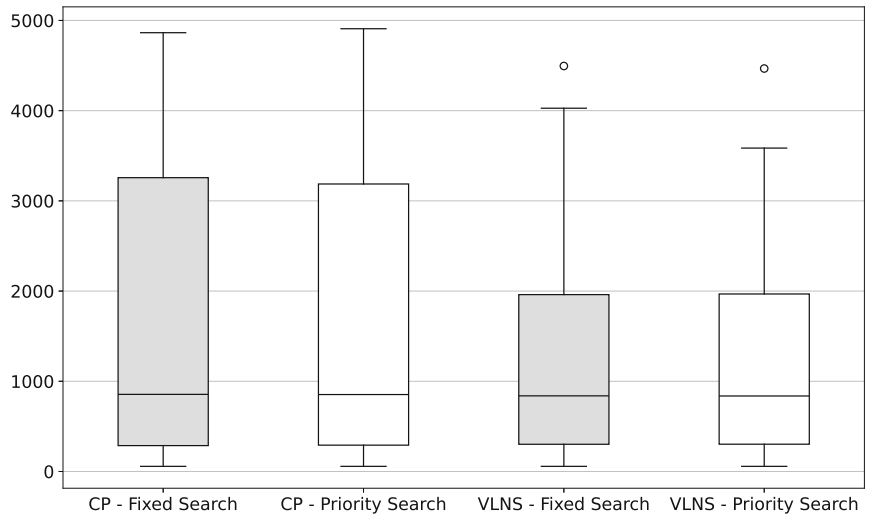

Figure 5.2: Boxplot representation of the results from Table 5.2, using the best scores for each configuration. For the CP results, instances `25, 29, 30` and `Lab1` were excluded due to infeasibility. For the VLNS results, only instance `Lab1` was excluded.

To confirm that the runtimes are roughly equivalent, I performed a short comparison between the state-of-the-art CP values achieved with Chuffed and my results with the existing TLSP model with Chuffed from Section 5.2 (picking the best run of 3 for my results), trying to replicate the results on my local hardware with the adjusted time out. Both models could solve the same instances. My replication achieved a better result for 14 out of 33 instances, a worse result for 11 instances, and performed identically on the remaining 8 instances.

Included in this comparison are the state-of-the-art results for CP (achieved with the Chuffed solver), VLNS (again using Chuffed internally), and Simulated Annealing (with a custom implementation in Java), all developed directly for TLSP. These are compared to the results from this Thesis, which are all based on TLSP-GR solvers that work on converted TLSP instances. From this Thesis, I included the results with Chuffed from Section 5.2 and the results achieved with my modified CP-SAT version and priority search from Section 5.3.

Table 5.5 shows the results from this Thesis compared to the state of the art.

Firstly, comparing CP results, the new TLSP-GR results in the *Chuffed* column are quite similar to the state of the art, except they are noticeably better for a few large instances. This reflects the results from Section 5.2. The results with CP-SAT are overall very different between solvers. CP-SAT manages to solve 6 instances to optimality, compared to only 2 for the other configurations. To my knowledge, this is the first time instances 3 through 6 were solved to optimality. For bigger instances, the are sometimes stark differences in both directions, with CP-SAT falling behind on the largest instances.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

#	CP					VLNS						SA	
	SOTA	This Thesis				SOTA		This Thesis				SOTA	
	Chuffed	Chuffed		CP-SAT		Chuffed		Chuffed		CP-SAT			
	best	avg	best	avg	best	avg	best	avg	best	avg	best	avg	best
1	**57***	57	**57***	57	**57***	57.0	**57**	57	**57**	57	**57**	58.0	58
2	**71***	71	**71***	71	**71***	71.0	**71**	71	**71**	71	**71**	72.0	72
3	142	142	142	141	**141***	141.0	**141**	141	**141**	141	**141**	149.4	147
4	119	119	119	101	**101***	101.0	**101**	102	102	101	**101**	106.2	105
5	244	287	287	240	**240***	240.0	**240**	240	**240**	240	**240**	284.8	263
6	180	188	188	140	**140***	140.0	**140**	140	**140**	140	**140**	157.6	157
7	355	415	411	292	292	283.0	**283**	283	**283**	283	**283**	296.8	291
8	310	311	311	288	288	283.6	283	283	**282**	284	283	298.2	293
9	713	729	729	470	470	419.0	415	428	428	419	**413**	461.2	444
10	1010	1029	1029	701	699	512.2	499	507	503	494	**493**	557.8	535
11	1011	1055	1055	855	853	822.8	816	820	815	810	**810**	915.2	905
12	764	804	804	675	665	646.8	643	648	643	641	**640**	667.6	663
13	337	364	364	311	311	308.4	**307**	315	313	309	309	334.4	331
14	447	492	492	431	424	410.4	**410**	413	413	410	**410**	419.8	417
15	1819	1753	1753	1662	1580	883.0	867	903	887	884	**864**	992.0	961
16	1599	1588	1588	1478	1454	1111.4	1109	1114	1113	1100	**1099**	(1/5)	1218
17	1416	1516	1516	1422	1406	1075.8	1038	1045	1040	1035	**1033**	1159.2	1137
18	1841	1803	1803	1608	1601	1341.0	**1328**	1354	1353	1335	1329	1475.2	1450
19	2751	2805	2805	3212	3187	1860.0	**1824**	2010	1964	(1/3)	2470	1956.8	1869
20	3146	2857	2857	4036	4008	2266.8	2193	2150	**2118**	2333	2192	(3/5)	2304
21	922	1137	1137	678	669	547.0	**542**	571	571	548	547	629.2	602
22	1062	1053	1053	873	873	744.8	742	729	721	712	**707**	769.0	761
23	t/o	t/o	t/o	4576	4562	t/o	t/o	1676	1644	1548	**1536**	1747.6	1613
24	t/o	t/o	t/o	3292	3281	t/o	t/o	1741	1737	1716	**1713**	1801.4	1780
25	4174	3694	3694	t/o	t/o	2217.6	2135	1970	1940	1931	**1893**	2280.0	2213
26	3861	3812	3812	4933	4908	2589.6	**2558**	2601	2579	2690	2652	2713.0	2667
27	3874	3378	3378	4628	4352	1769.6	1723	1792	1781	1712	**1698**	1999.2	1965
28	3180	2647	2647	4298	4298	2258.4	2235	2244	2239	2228	**2225**	2470.4	2439
29	t/o	t/o	t/o	t/o	t/o	t/o	t/o	3082	**3062**	(2/3)	3585	3645.2	3562
30	6508	6295	6295	t/o	t/o	4822.6	4714	4632	4585	4572	**4467**	(4/5)	4532
Lab1	4991	4709	4709	t/o	t/o	3377.0	**3296**	(2/3)	3742	t/o	t/o	(4/5)	3389
Lab2	3339	3285	3285	4441	4439	2669.2	2595	2700	2642	(2/3)	2591	2643.0	**2539**
Lab3	2979	3013	3013	2927	2917	2599.0	2590	2587	2580	2570	**2559**	2609.6	2592

Table 5.5: Penalties from this Thesis compared to the state of the art. **SOTA** columns refer to existing state-of-the-art results for TLSP, while **This Thesis** columns refer to the results with the TLSP-GR solution approaches developed in this Thesis. **CP-SAT** refers to the modified version of CP-SAT which adds support for priority search, hot start, and random variable selection strategies. **Chuffed** refers to the Chuffed solver. **SOTA** evaluations were usually performed with 5 runs each, except for **CP**, which only included one run. **This Thesis** results included 3 runs. `t/o` indicates that all runs timed out. When only some of the runs were feasible, the number of feasible runs is shown instead of the average. **Bold** values indicate the best score for the instance in this table, and * indicates that the instance was solved to optimality. **SOTA** values taken from [DGJ+23].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Compared to the state of the art, CP-SAT solves one fewer instance overall, but it fails on different instances, being able to solve 2 that the states of the art could not.

Regarding VLNS, the Chuffed TLSP-GR model again fares quite similarly to the state of the art. In addition to the difference in the CP model, the VLNS algorithm used in this Thesis is based on the variant also using the neighborhood to achieve feasibility, whereas the implementation in the state-of-the-art result generates a feasible solution with the CP solver first. This is the main reason why the VLNS results from this Thesis have more feasible instances. The VLNS solver based on CP-SAT struggles more with feasibility than the Chuffed version, leaving 1 instance unsolved. However, on most instances, VLNS with CP-SAT achieves the best results overall. Overall, VLNS with CP-SAT is comparable to the state of the art, achieving better penalties than the state of the art on 17 out of 33 instances.

In summary: the new TLSP-GR model introduced in this Thesis offers competitive performance with the existing TLSP state-of-the-art model. CP-SAT with priority search tends to fall behind on larger instances but is able to solve 4 new instances to optimality, bringing the total to 6. When TLSP-GR model is coupled with the new modified CP-SAT version that includes priority search, it is competitive with the state of the art, improving upon the state-of-the-art solution for 17 out of 33 instances.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Conclusion

This Thesis addressed two main topics: generalizing resources in the Test Laboratory Scheduling Problem (TLSP) to cover additional real-world scheduling requirements, as well as implementing priority search (and other smaller features) in Google's CP-SAT solver to improve its performance on scheduling problems like TLSP.

To generalize resources, I proposed a new problem variant of TLSP, TLSP with Generalized Resources (TLSP-GR). TLSP-GR unifies employees, workbenches, and equipment into a single concept called resources. This allows users to use constraints with all resource types that were previously restricted to specific resource types. It also serves as a basis for a straightforward implementation of new constraints like implications between assignments of resources. I further developed a Constraint Programming model for TLSP-GR. By including some targeted optimizations for TLSP-GR and carefully generalizing existing optimizations from TLSP where possible, the model is competitive with the state of the art on existing TLSP instances, both on its own and when included in the VLNS algorithm.

Concerning the CP-SAT solver, I analyzed the source code and existing implementation of user-defined search strategies as passed through the MiniZinc interface. It turned out that implementing priority search was realistic, albeit with some structural changes to the underlying data structures and search algorithm. I successfully implemented priority search as well as some smaller features helpful for the VLNS algorithm, randomized variable selection, and hot start. The modified CP-SAT solver significantly outperformed the existing version on VLNS, but to my surprise, this was mostly a result of the smaller feature additions, randomized variable selection, and hot start. While the performance with priority search far exceeded that of the solver's default search strategy, I found a search strategy without priority search that achieves similar results – something that did not seem to be possible in earlier research with the Chuffed solver.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Summarizing the computational results, the new TLSP-GR CP model achieves similar performance to the state-of-the-art model for TLSP. Utilizing this model together with the modified CP-SAT variant yielded 4 new optimality results in addition to the 2 already present in the literature, bringing the total up to 6 out of 33 publicly available instances. For larger instances, the best results could be obtained by using a variant of the existing TLSP VLNS algorithm that is augmented with the new TLSP-GR CP model and also uses its neighborhood to find the first feasible solution, internally using the modified CP-SAT solver (which includes priority search, hot start, and randomized variable selection). Together, these methods found penalties better than the state of the art for 17 of 33 test instances, indicating that with the modifications, CP-SAT becomes a competitive solver for TLSP.

The contributions of this Thesis are relevant both to academic research and industrial applications. Regarding research, the work with TLSP-GR shows how a scheduling problem and its corresponding CP model can be made more general, enabling new applications, without a notable performance sacrifice. The work on CP-SAT shows how priority search can be implemented in CP-SAT, which results in good performance on TLSP. However, I found a search strategy without priority search that achieved similar performance. Overall, the results highlighted a diminished importance of user-defined search strategies with CP-SAT compared to the academic Chuffed solver, at least relating to TLSP. The results also demonstrate the significant impact that other solver features – hot start and search strategies with randomized variable selection – can have on the overall performance of a VLNS algorithm. Concerning industrial applications, TLSP-GR is applicable to more real-world use cases than TLSP was, and the modified version of CP-SAT offers a promising state-of-the-art solution approach when combined with VLNS.

Future work could further extend TLSP-GR to additional requirements by adding new constraints. For instance, assignment restrictions concerning different types of resources can now be modeled in a simple and general way with TLSP-GR. The modifications to CP-SAT could be investigated with newer versions of the solver. Newer versions include additional scheduling search heuristics, which could make the solver's default search strategy more competitive. Scaling analysis could be performed when running CP-SAT in its multi-threaded mode. A multi-threaded version of the VLNS algorithm could also be considered, analyzing the trade-off between trying out different moves in parallel and increasing the thread count of the CP solvers.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

List of Algorithms

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[AOEOP02] Ravindra K. Ahuja, Özlem Ergun, James B. Orlin, and Abraham P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1):75–102, 2002.

[Bar99] Roman Barták. Constraint programming: In pursuit of the holy grail. *Proceedings of WDS99 (invited lecture)*, 01 1999.

[BDM+99] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[Chu11] Geoffrey Chu. *Improving combinatorial optimization*. PhD thesis, University of Melbourne, Australia, 2011.

[Dan19] Philipp Danzinger. Real world industrial test laboratory scheduling: Investigating constraint programming for task grouping in tlsp. Bachelor's thesis, TU Wien, Vienna, Austria, 2019.

[DGJ+23] Philipp Danzinger, Tobias Geibinger, David Janneau, Florian Mischek, Nysret Musliu, and Christian Poschalko. A system for automated industrial test laboratory scheduling. *ACM Trans. Intell. Syst. Technol.*, 14(1), mar 2023.

[DGMM20] Philipp Danzinger, Tobias Geibinger, Florian Mischek, and Nysret Musliu. Solving the test laboratory scheduling problem with variable task grouping. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30(1):357–365, Jun. 2020.

[DPRL98] S. Dauzère-Pérès, W. Roux, and J.B. Lasserre. Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research*, 107(2):289–305, 1998.

[FGS+17] Thibaut Feydy, Adrian Goldwaser, Andreas Schutt, Peter J. Stuckey, and Kenneth D. Young. Priority search with minizinc. In *ModRef 2017: The Sixteenth International Workshop on Constraint Modelling and Reformulation at CP2017*, 2017.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[GMM21] Tobias Geibinger, Florian Mischek, and Nysret Musliu. Constraint logic programming for real-world test laboratory scheduling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7):6358–6366, May 2021.

[GMM22] Tobias Geibinger, Florian Mischek, and Nysret Musliu. Investigating constraint programming and hybrid methods for real world industrial test laboratory scheduling, 2022. Original version on 12 Nov 2019.

[Goo23] Google. Protocol buffers - google's data interchange format. `https://github.com/protocolbuffers/protobuf`, 2023. Available online: `https://github.com/protocolbuffers/protobuf`.

[LEF+17] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, Stefan Falkner, André Biedenkapp, and Frank Hutter. Smac v3: Algorithm configuration in python. `https://github.com/automl/SMAC3`, 2017.

[min23] Minizinc challenge 2023. `https://www.minizinc.org/challenge/2023/results/`, 2023. Accessed: 2024-05-04.

[Mis22] Florian Mischek. *Automated project scheduling in real-world test laboratories*. Dissertation, Technische Universität Wien, Vienna, Austria, 2022.

[MM18a] Florian Mischek and Nysret Musliu. A local search framework for industrial test laboratory scheduling. In *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018), Vienna, Austria, August 28–31, 2018*, pages 465–467, 2018.

[MM18b] Florian Mischek and Nysret Musliu. The test laboratory scheduling problem. Technical report, Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, TU Wien, CD-TR 2018/1, 2018.

[MM21] Florian Mischek and Nysret Musliu. A local search framework for industrial test laboratory scheduling. *Annals of Operations Research*, 302(2):533–562, Jul 2021.

[MM23] Florian Mischek and Nysret Musliu. Leveraging problem-independent hyper-heuristics for real-world test laboratory scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '23, page 321–329, New York, NY, USA, 2023. Association for Computing Machinery.

[MMS23] Florian Mischek, Nysret Musliu, and Andrea Schaerf. Local search approaches for the test laboratory scheduling problem with variable task grouping. *Journal of Scheduling*, 26(5):457–477, October 2023. First online: 2021.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 530–535, 2001.

[Mon74] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[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 *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, pages 529–543, 2007.

[OrT] Or-tools - google optimization tools. `https://github.com/google/or-tools`.

[PD21] Laurent Perron and Frédéric Didier. CP-SAT Solver. `https://developers.google.com/optimization/cp/cp_solver/`, 2021. Accessed: 2024-05-04.

[PF21a] Laurent Perron and Vincent Furnon. Or-tools. `https://github.com/google/or-tools/blob/058618a9c44ebab22f634998e64aedba6da1b8e2/ortools/sat/cp_model.proto`, 2021. Accessed: 2023-04-15.

[PF21b] Laurent Perron and Vincent Furnon. Or-tools. `https://github.com/google/or-tools/blob/1ea133254a7b70c9d59da25ee7c12fb2f8085710/ortools/sat/cp_model_search.cc`, 2021. Accessed: 2023-04-15.

[SFS+14] Peter J. Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The minizinc challenge 2008–2013. *AI Magazine*, 35(2):55–60, Jun. 2014.

[SMT24] Peter J. Stuckey, Kim Marriott, and Guido Tack. The minizinc handbook - 2.6 search. `https://www.minizinc.org/doc-2.8.3/en/mzn_search.html`, 2024. Accessed: 2024-04-15.

[Sut63] Ivan E. Sutherland. Sketchpad: a man-machine graphical communication system. In *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*, AFIPS '63 (Spring), page 329–346, New York, NY, USA, 1963. Association for Computing Machinery.

[TPM09] Vikram Tiwari, James H. Patterson, and Vincent A. Mabert. Scheduling projects with heterogeneous resources to meet time and quality objectives. *European Journal of Operational Research*, 193(3):780–790, 2009.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Wal75] David L. Waltz. Understanding line drawings of scenes with shadows. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.

[WKS+14] Tony Wauters, Joris Kinable, Pieter Smet, Wim Vancroonenburg, Greet Vanden Berghe, and Jannes Verstichel. The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling*, 19:1–13, 11 2014.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.