TU WIEN Informatics

Investigating Explanations of Infeasibility for Test Laboratory Scheduling Problems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Logic and Computation

eingereicht von

Aida Aliu, BSc
Matrikelnummer 11935975

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assoc. Prof. Dr. Nysret Musliu
Mitwirkung: Dr. Florian Mischek

Wien, 1. Dezember 2025

Aida Aliu

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 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.

Informatics

Investigating Explanations of Infeasibility for Test Laboratory Scheduling Problems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Logic and Computation

by

Aida Aliu, BSc
Registration Number 11935975

to the Faculty of Informatics

at the TU Wien

Advisor: Assoc. Prof. Dr. Nysret Musliu
Assistance: Dr. Florian Mischek

Vienna, December 1, 2025

Aida Aliu   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 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

Aida Aliu, 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.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 1. Dezember 2025

Aida Aliu

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.

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

Acknowledgements

First and foremost, I would like to thank my advisor, Assoc. Prof. Dr. Nysret Musliu, for his expert mentorship and constant support, which were pivotal in guiding this research. I am also profoundly grateful to Dr. Florian Mischek for his dedicated assistance and the valuable insights he shared throughout this journey.

I am deeply grateful to my family for their unwavering encouragement. In particular, I wish to thank my partner, Vigan, for his emotional support, and my daughter, Sira, whose arrival during this journey brought immense joy and motivation, inspiring me to persevere through every challenge.

This work was supported by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development through the Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling.

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

Die mathematische Unerfüllbarkeit von Planungsaufgaben stellt in vielen realen Anwendungsgebieten eine erhebliche Herausforderung für automatisierte Planungssysteme dar, da einander widersprechende Anforderungen häufig die Erstellung eines zulässigen Zeitplans verhindern. Diese Arbeit konzentriert sich auf die Unerfüllbarkeit von Instanzen des Test Laboratory Scheduling Problems (TLSP), einer Erweiterung des Resource-Constrained Project Scheduling Problems (RCPSP). Im Speziellen behandelt sie die Herausforderungen, denen industrielle Prüflabore gegenüberstehen, in denen eine große Anzahl von Tests durch qualifiziertes Personal mit spezialisierter Ausrüstung unter Einhaltung strikter zeitlicher Vorgaben, komplexer Abhängigkeiten zwischen Aufgaben und weiterer Nebenbedingungen durchgeführt werden muss. In diesem Kontext beeinträchtigen unerfüllbare Konflikte zwischen den Anforderungen bereits in der Planungsphase die operative Effizienz und erschweren fundierte Entscheidungen in industriellen Umgebungen.

Um diese Herausforderung zu bewältigen, stellen wir in dieser Arbeit ein Erklärungswerkzeug für mathematische Unerfüllbarkeiten vor, das zwei komplementäre Ansätze kombiniert. Der erste Ansatz basiert auf der Konflikterkennung: Durch Identifikation minimaler unerfüllbarer Teilmengen (Minimal Unsatisfiable Subsets) und maximaler erfüllbarer Teilmengen (Maximal Satisfiable Subsets) werden genau die Constraint-Gruppen aufgedeckt, die Unerfüllbarkeit verursachen. Die Wiederherstellung der Erfüllbarkeit erfolgt anschließend über die Berechnung minimaler Korrekturmengen (Minimal Correction Subsets). Der zweite Ansatz verwendet kontrafaktische Erklärungen, die minimale Modifikationen der Anforderungen aufzeigen, die notwendig sind, um Erfüllbarkeit wiederherzustellen. Hierfür nutzen wir einen mehrstufigen Relaxationsraum, der gleichzeitige Anpassungen mehrerer Anforderungen erlaubt und so eine flexiblere und differenziertere Problemlösung ermöglicht.

Unsere Methodik baut auf etablierten Verfahren aus der Constraint-Satisfaction-Problem (CSP)-Literatur auf und erweitert sie gezielt für den TLSP-Bereich. Der Unerfüllbarkeitserklärer ist mittels eines TLSP-spezifischen Constraint-Programming-Modells in OR-Tools implementiert, das auf Erklärbarkeit ausgelegt ist. Dieses Modell wird durch eine konfigurierbare Benutzeroberfläche ergänzt, die es den Anwendern ermöglicht, die Einstellungen des Erklärers anzupassen und mehrere Konfigurationen parallel auszuführen, wodurch die Abhängigkeit von manuellen Trial-and-Error-Methoden deutlich verringert wird.

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

Experimentelle Auswertungen an realen und synthetischen Benchmark-Instanzen zeigen, dass unser Ansatz den Zeit- und Arbeitsaufwand zur Identifikation und Behebung von Planungskonflikten erheblich reduziert. Obwohl die Lösung für mittelgroße Instanzen effektiv ist, bleibt die Skalierbarkeit bei sehr großen und komplexen Problemen eine Herausforderung und deutet auf weiteren Optimierungsbedarf hin.

Der in dieser Arbeit vorgestellte Unerfüllbarkeitserklärer ist bereits in einem industriellen Umfeld im Einsatz und unterstützt dort wirkungsvoll bei der Behebung realer Planungsunerfüllbarkeiten.

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

Abstract

Infeasibility in automated scheduling systems poses a significant challenge in many real-world applications, as conflicting constraints often prevent the generation of a feasible schedule. This thesis focuses on the infeasibility of instances of Test Laboratory Scheduling Problem (TLSP), an extension of the Resource-Constrained Project Scheduling Problem (RCPSP). Specifically, it addresses challenges faced by industrial test laboratories where a large number of tests has to be performed by qualified personnel using specialized equipment, while respecting strict temporal requirements, intricate task dependencies and various other constraints. In this context, infeasibility at the scheduling stage disrupts operational efficiency and hinders effective decision-making in industrial settings. To address this challenge, we propose an infeasibility explainer that integrates two complementary approaches. The first approach is based on conflict detection: by identifying minimal unsatisfiable subsets and maximal satisfiable subsets, the explainer pinpoints the precise groups of constraints that cause infeasibility. Feasibility can then be restored by computing minimal correction subsets. The second approach employs counterfactual explanations, which offer minimal modifications to the scheduling constraints that are necessary to restore feasibility. This method leverages a multi-point relaxation space, allowing for simultaneous adjustments across multiple constraints, thereby providing a more flexible and nuanced resolution. Our methodology builds on established techniques from Constraint Satisfaction Problem (CSP) literature and extends them specifically to the TLSP domain. The infeasibility explainer is implemented using a TLSP-specific constraint programming model in OR-Tools, designed for explainability. This model is enhanced by a configurable user interface that enables users to adjust explainer settings and execute multiple configurations in parallel, thereby significantly reducing reliance on manual trial-and-error methods.

Experimental evaluations, carried out on both industrial and benchmark instances, demonstrate that our approach substantially reduces the time and effort required to identify and resolve scheduling conflicts. Although the solution performs effectively on moderately-sized instances, scalability remains a challenge for very large and complex problems, suggesting a need for further optimization.

The infeasibility explainer described in this thesis is already deployed in an industrial setting, where it effectively aids in resolving real-world scheduling infeasibilities.

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.

Contents

Kurzfassung **ix**

Abstract **xi**

Contents **xiii**

1 Introduction **1**
1.1 Motivation 1
1.2 Aims of the thesis 2
1.3 Contributions 3
1.4 Structure of the thesis 4

2 Preliminaries and Related Work **5**
2.1 Preliminaries 5
2.2 Related Work 13

3 Explainability for TLSP **17**
3.1 Defining TLSP explanations 17
3.2 Defining background and foreground constraints 19
3.3 Conflict-based explanations 23
3.4 Counterfactual explanations 31

4 Implementation **35**
4.1 Adapting TLSP CP model for explanations 35
4.2 User Interface 43

5 Experimental evaluation **47**
5.1 Demonstration 47
5.2 Performance and scalability evaluation 64
5.3 User experience and practical insights 70

6 Conclusion **73**

Overview of Generative AI Tools Used **75**

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

List of Figures 77

List of Tables 79

List of Algorithms 81

Bibliography 83

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

CHAPTER 1

Introduction

1.1 Motivation

Automated scheduling systems are commonly used to assist human users in developing schedules and have become an indispensable tool in various real-world applications in several domains. The primary function of such systems is to provide a feasible solution that satisfies the constraints of the user. As these systems aid human decision-making, they need to provide interpretable and transparent information to human users regarding their decision-making processes.
Scheduling systems typically consist of numerous constraints and, due to their interactive nature, can rapidly result in over-constrained scenarios with conflicting user constraints. Consequently, these conflicting constraints may prevent the systems from generating feasible solutions, thus resulting in scheduling infeasibility. Scheduling infeasibility in automated scheduling systems refers to a situation in which the system cannot generate a valid schedule that satisfies all specified constraints and requirements. Several factors can contribute to scheduling infeasibility, including:

- Resource constraints: If there is a limited availability of resources and the scheduling algorithm cannot allocate these resources in a way that meets the demand for tasks.
- Temporal constraints: Some scheduling problems involve strict time-related constraints, such as deadlines or dependencies between tasks. If these constraints cannot be satisfied simultaneously, the scheduling system may fail to produce a feasible schedule.
- Complexity of constraints: In some cases, scheduling problems can be highly complex with numerous interrelated constraints. If the algorithm is not able to navigate through this complexity to find a valid solution, infeasibility may result.

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

- Incomplete or incorrect input data: If the input data provided to the scheduling system is incomplete or contains errors, it may lead to infeasibility. Incorrect information about resource availability, task durations, or other parameters can hinder the generation of a feasible schedule.

The absence of any explanation for the failure to generate feasible solutions undermines the primary purpose of automated scheduling systems that aid human users. In such cases, users are left with no solution and are compelled to manually check for reasons behind the infeasibility, which can be very time inefficient, thereby negating the utility of the scheduling system.
Therefore, a vital challenge of these systems is generating reasons that explain why the system cannot find a feasible solution under certain specifications.
Given the critical importance of this aspect, this thesis undertakes an investigation into the challenge of generating infeasibility explanations for a specific scheduling problem, namely the Test Laboratory Scheduling Problem (TLSP), introduced by [MM18a]. The TLSP is an NP-complete scheduling problem, a real-world scheduling problem inspired by the operational needs of industrial test laboratories. It extends the classical Resource-Constrained Project Scheduling Problem (RCPSP) [BDM+99] by introducing additional complexity through task grouping, diverse resource types, and real-world constraints like employee qualifications and equipment availability. Solvers for the Test Laboratory Scheduling Problem (TLSP) have already been successfully deployed in real-world industrial labs. However, in practice, users encountered challenges due to the lack of explainability when schedules turned out to be infeasible, making it difficult to understand and resolve conflicts in the input data. While existing research addresses explainability on broader Constraint Satisfaction Problems (CSPs), such efforts operate at a higher, more generalized level. This thesis, in contrast, targets the challenges of infeasibility explanations specifically within the domain of the TLSP. The rationale behind this specificity lies in the recognition that conflicts within problems manifest in unique ways, making problem-specific explanations more pertinent. Moreover, while high-level abstract explanations contribute to theoretical advancements, problem specific explanations enhance real-world usability by providing actionable insights tailored to practical scheduling challenges.

1.2 Aims of the thesis

The aim of this thesis is to address the critical challenge of generating infeasibility explanations for infeasible instances of TLSP. The primary objectives of this thesis are as follows:

1. **Develop the first infeasibility explainer for TLSP:** Design and implement an explainer specifically adapted to the Test Laboratory Scheduling Problem (TLSP), addressing its unique domain-specific characteristics. This involves adapting existing

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

explainability approaches and techniques on over-constrained systems to address the unique characteristics of the TLSP domain

2. **Generate human-readable explanations for infeasibility:** Create explanations that effectively identify the constraints causing infeasibility in TLSP instances, making them understandable to human users.

3. **Provide feasibility restoration insights:** Propose a set of constraint relaxations or modifications sufficient to restore feasibility in over-constrained TLSP scenarios.

4. **Incorporate user-centric configurability:** Enable users to customize the explainer by allowing them to configure the type of explanations.

5. **Evaluate the explainer's effectiveness:** Assess the practicality, scalability, and usefulness of the explainer through application on real-world TLSP instances, focusing on both technical performance and user feedback.

1.3 Contributions

The main contributions of this thesis are:

- We conduct a literature review on explanation approaches for overconstrained CSPs, with a specific focus on scheduling problems. Compare and evaluate existing approaches, by adapting them to the TLSP context.
- We propose a scheduling infeasibility explainer specifically tailored for TLSP, which is an adaptation of existing general approaches on over-constrained systems. The explainer provides a variety of explanation types to accommodate different user needs.
- We propose explanations which were collaboratively constructed with a real-world TLSP domain expert, incorporating their insights on common sources of conflicts.
- We provide users with the ability to configure the infeasibility explainer according to their preferences. This level of customization empowers users to tailor the explainer to their specific problem instances and constraints. This flexibility also allows users to choose the level of detail and granularity they require in their explanations. Whether it's pinpointing exact constraints, identifying problematic resources, or highlighting troublesome tasks/projects, our explainer can accommodate diverse user needs.
- We introduce a user-friendly interface for the infeasibility explainer, enabling interactive selection of explanation types, foreground constraints, and points of relaxation.
- We evaluate the explainer on real-world settings, assessing its efficacy in identifying common sources of infeasibility and evaluating its utility in practical settings.

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

1.4 Structure of the thesis

This thesis is organized into five main chapters, each contributing to the overall goal of understanding and addressing explainability in the Test Laboratory Scheduling Problem (TLSP).

The second chapter provides the theoretical foundations and contextual background for the research. It introduces fundamental concepts and terminology, including key notions such as Constraint Satisfaction Problems (CSPs) and flexible CSPs. Additionally, it reviews relevant work in the literature on explainability in constraint satisfaction, emphasizing both conflict-based and counterfactual explanation methods. The chapter situates the TLSP in the broader context of explainability research.

The third chapter focuses on adapting explainability techniques specifically to TLSP. It begins by formally defining TLSP and its representation within a constraint programming framework. The chapter then establishes the types of explanations addressed in this work, discussing the distinction between background (non-modifiable) and foreground (modifiable) constraints. This distinction serves as the basis for generating explanations in a controlled manner. The methodologies for generating explanations are described in detail, including those based on constraint conflicts and counterfactual reasoning.

The fourth chapter provides details of the implementation. It begins by introducing the tailored constraint programming model in OR-Tools, which serves as the foundation for the explanation generation process. Additionally, it presents the developed application, which enables users to interact with and utilize the implemented explainers.

The fifth chapter evaluates the proposed explainability framework. The chapter presents a detailed evaluation of the explainer's performance, focusing on its ability to generate meaningful, accurate, and interpretable explanations. The results are analyzed to assess the effectiveness of the proposed methods in identifying and resolving infeasibility in TLSP. The chapter concludes by summarizing the experimental findings and discussing their implications for the field.

Finally, the thesis concludes with a discussion of the contributions made, limitations of the current work, and potential directions for future research. This work aims to bridge the gap between constraint satisfaction explainability methods and real-world scheduling problems, providing a foundation for future advancements in explainability for TLSP and similar domains.

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 WIEN Bibliothek Your knowledge hub

CHAPTER 2

Preliminaries and Related Work

2.1 Preliminaries

We begin by outlining key concepts from the Constraint Programming literature related to explanations. These fundamental concepts are essential for framing the methodology we propose. Specifically, our approach relies on the following notions to produce explanations:

- Constraint Satisfaction Problems (CSPs), *Max-CSP*, and *Weighted CSP*
- Constraint sets
- Relaxation Spaces

2.1.1 Constraint Satisfaction Problems (CSPs) and *Max-CSP*

A *Constraint Satisfaction Problem (CSP)* is defined as a tuple (X, D, C) [RN10, Kum92], where:

- $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables.
- $D = \{D_1, D_2, \ldots, D_n\}$ is the set of finite domains, where each D_i contains the possible values for variable x_i.
- $C = \{c_1, c_2, \ldots, c_m\}$ is a set of constraints, where each constraint c_i specifies allowed combinations of values for a subset of variables $X_i \subseteq X$.

The objective of a CSP is to find an assignment of values to variables, $x_i \in D_i$, such that all constraints in C are satisfied. If such an assignment exists, the CSP is *satisfiable*; otherwise, it is *unsatisfiable*.

Traditional *CSPs* treat constraints as rigid or "hard" requirements. This means that every feasible solution must strictly satisfy all constraints without exception. However, in many real-world problems, this strictness is overly restrictive, as some flexibility or relaxation of constraints is often desirable. *Flexible CSPs* address this limitation by introducing mechanisms to handle violations of constraints in a controlled manner, enabling solutions that do not necessarily satisfy all constraints [RN10].

Flexible CSPs are particularly useful for over-constrained problems, where it is impossible to satisfy all constraints simultaneously. This relaxation of the classical CSP framework allows for prioritization or partial satisfaction of constraints, enabling the generation of solutions that balance feasibility and optimization. Below are some common types of flexible CSPs:

- **Max-CSP**: An optimization variant of CSPs. In *Max-CSP*, the goal is to maximize the number of satisfied constraints in a given problem instance. Formally, a *Max-CSP* instance is defined by (X, D, C) as in CSP, but instead of finding a solution that satisfies all constraints, we seek an assignment that satisfies the largest subset of constraints.

  Max-CSP is particularly useful in over-constrained scenarios where not all constraints can be simultaneously satisfied, providing a framework for identifying solutions that are as close to feasibility as possible.

- **Weighted CSP**: An extension of Max-CSP where each constraint is assigned a weight that represents its importance. The objective is to find a solution that maximizes the sum of the weights of satisfied constraints. This allows for more nuanced handling of constraint violations based on their relative importance [RN10].

2.1.2 Constraint sets

Definition: A minimal unsatisfiable set of constraints (**MUS**) of a constraint system C is any set S such that $S \subseteq C$, S is an unsatisfiable set, and every proper subset $S' \subset S$ is satisfiable [DGGO21].

The existence of a MUS in a constraint system C proves its infeasibility. Given that there exists only one MUS S in such system, removing any constraint $c \in S$ will restore the feasibility of system C. However, such systems can have multiple MUSes, in which case the feasibility cannot be guaranteed by removing constraints only from one MUS, rather we need to remove constraints from each MUS. When we want the set of removed constraints to be minimal, then we are looking for a hitting set of MUSes (as defined in Section 3.3.1), and each such hitting set is called a minimal conflict set (MCS).

Definition: A maximal satisfiable set of constraints (**MSS**) of a constraint system C is any set S such that $S \subseteq C$, S is a satisfiable set, and every proper superset S' of S $(S \subset S' \subset C)$ is unsatisfiable [DGGO21].

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

Definition: A minimal conflict set of constraints (**MCS**) of a constraint system C, is defined as the complement of some MSS of C. Any MCS can be understood as the minimal set of constraints that need to be removed from C to restore its feasibility [MSHJ+13].

2.1.3 Relaxation Spaces

Our approaches will work under two different relaxation spaces [FO07]:

- Two-point relaxation space: a relaxation of the system is understood as the removal of constraints. A two-point relaxation space either allows to have the constraint in the constraint set, or not.
- Multi-point relaxation space: constraints in multi-point relaxation can be relaxed at multiple discrete levels, which correspond to replacing a constraint with any weaker one.

2.1.4 TLSP Definition

This section describes the Test Laboratory Scheduling Problem (TLSP) and its properties, as first introduced by [MM18b]. TLSP is a scheduling problem that arises in industrial test laboratories, where a large number of tests must be performed by qualified personnel using specialized equipment. The problem is characterized by strict temporal requirements, intricate task dependencies, and various other constraints. In TLSP, a list of projects is given, each containing several tasks. For each project, the tasks must be partitioned into a set of jobs, with some restrictions on the feasible partitions. Then, those jobs must each be assigned a mode, time slots and resources. The properties and feasible assignments for each job are calculated from the tasks contained within.

For the implementation of the explainer, we use the constraint programming model of the latest TLSP variant, called *TLSP with Generalized Resources (TLSP-GR)*, introduced by Danzinger et al. [Dan24]. This variant generalizes the modeling features of TLSP that were previously specific to employees, workbenches, or equipment. Importantly, TLSP instances can still be represented within the TLSP-GR framework.

This section largely reproduces the original model definition, with adaptations to align with TLSP-GR. We follow the notation and definitions from Mischek et al. [MM18a], adapting the relevant parts to reflect the changes in TLSP-GR and adding the new constraints introduced in this variant. A more detailed description of the TLSP-GR model and its differences from the original can be found in [Dan24].

A solution of TLSP is a schedule consisting of the following parts:

- A list of jobs, composed of one or multiple similar tasks within the same project.

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

- For each job, an assigned mode, start and end time slots, the employees scheduled to work on the job, and an assignment to a workbench and equipment.

The quality of a schedule is judged according to an objective function that is the weighted sum of several soft constraints and should be minimized. Among others, these include the number of jobs and the total completion time (start of the first job until end of the last) of each project.

2.1.5 Input parameters

A TLSP instance can be split into three parts: The laboratory environment, including a list of resources, a list of projects containing the tasks that should be scheduled together with their properties and the current state of the existing schedule, which might be partially or completely empty.

Environment

In the laboratory, resources of different kinds are available that are required to perform tasks:

- 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 $R_i = \{1, ..., |R_i|\}$. Tasks may require any number of resources from any resource group

$L^{rg} \subseteq R^*$ represents the set of resource groups such that resource assignments for resources in this group must be identical for tasks that are linked

The scheduling period is composed of discrete *time slots* $t \in T = \{0, ..., |T| - 1\}$. Each time slot represents half a day of work. Tasks are performed in one of several *modes* labeled $m \in M = \{1, ..., |M|\}$. The chosen mode influences the following properties of tasks performed under it:

- The *speed factor* v_m, which will be applied to the task's original duration.
- Required resources.

Projects and Tasks

Given is a set P of projects labeled $p \in \{1, \ldots, |P|\}$. Each project contains tasks $pa \in A_p$, with $a \in \{1, \ldots, |A_p|\}$. The set of all tasks (over all projects) is $A^* = \bigcup_{p \in P} A_p$.

Each task pa has several properties:

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

- It has a release date α_{pa} and both a due date $\bar{\omega}_{pa}$ and a deadline ω_{pa}. The difference between the latter is that a due date violation only results in a penalty to the solution quality, while deadlines must be observed.
- $M_{pa} \subseteq M$ is the set of available modes for the task.
- The task's duration d_{pa} (in time slots, real-valued). Under any given mode $m \in M_{pa}$, this duration becomes $d_{pa}^m := d_{pa} \cdot v_m$.
- 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 $R_{a,g}^{Pr} \subseteq R_g$. Only available resources may be assigned to a job containing a, and assigning non-preferred resources incurs a penalty.
- A list of direct predecessors $P_{pa} \subseteq A_p$, which must be completed before the task can start. Note that precedence constraints can only exist between tasks in the same project.

Each project's tasks are partitioned into families $F_{pf} \subseteq A_p$, where f is the family's index. For a given task pa, f_{pa} gives the task's family. Only tasks from the same family can be grouped into a single job.

Additionally, each family f is associated with a certain setup time s_{pf}, which is added to the duration of each job containing tasks of that family.

Finally, it may be required that certain tasks are performed by the same employee(s). For this reason, each project p may define linked tasks, which must be assigned the same employee(s). Linked tasks are given by the equivalence relation $L_p \subseteq A_p \times A_p$, where two tasks pa and pb are linked if and only if $(pa, pb) \in L_p$.

Initial Schedule

All problem instances include an initial (or base) schedule, which may be completely or partially empty. This schedule can act both as an initial solution and as a baseline, placing limits on employees' and tasks' schedules, in particular by defining fixed assignments that must not be changed.

Provided is a set of jobs J_0, where each job $j \in J_0$ contains the following assignments:

- The tasks in the job: $\dot{A}_j$
  - A fixed subset of these tasks $\dot{A}_j^F \subseteq \dot{A}_j$. All fixed tasks of a job in the base schedule must also appear together in a single job in the solution.
- The mode assigned to the job: $\dot{m}_j$

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

- The start and completion times of the job: $\dot{t}^s_j$ resp. $\dot{t}^c_j$
- The resources assigned to the job: $\dot{R}_{j,g}$ for resource group g

Except for the tasks, each individual assignment may or may not be present in any given job. Fixed tasks are assumed to be empty if not given. In all other cases, missing assignments will be referred to using the value ε. Time slots and employees can only be assigned if also a mode assignment is given.

A subset of these jobs are the started jobs J^S_0. A started job $j_s \in J^S_0$ must fulfill the following conditions:

- It must contain at least one fixed task. It is assumed that the fixed tasks of a started job are currently being worked on.
- Its start time must be 0.
- It must contain resource assignments fulfilling all requirements.

A started job's duration does not include a setup time. In the solution, the job containing the fixed tasks of a started job must also start at time 0. Usually, the resources available to the fixed tasks of a started job are additionally restricted to those assigned to the job, to avoid interruptions of ongoing work in case of a rescheduling.

A project can be *fixed* if all of its tasks are included in the initial schedule. The concept of *fixed projects* was not part of the original TLSP model but is frequently used in practice as a preprocessing step to reduce problem complexity or to preserve parts of an existing schedule. In this work, we integrate fixed projects directly into the model. Fixing a project means that all of its tasks must be scheduled in exactly the same way as in the initial schedule. That is, the jobs of a fixed project must contain the same tasks, and the assignments of these jobs must be identical to those in the initial schedule.

2.1.6 Jobs and Grouping

For various operational reasons, tasks are not scheduled directly. Instead, they are first grouped into larger units called jobs.

A single job can only contain tasks from the same project and family. Jobs have many of the same properties as tasks, which are computed from the tasks that make up a job. The general principle is that within a job, tasks are not explicitly ordered or scheduled; therefore the job must fulfill all requirements of each associated task during its whole duration.

Let $J = \{1, \dots, |J|\}$ be the set of all jobs in a solution and $J_p \subseteq J$ be the set of jobs of a given project p. Then for a job $j \in J$, the set of tasks contained in j is $\dot{A}_j$. j has the following properties:

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 WIEN Your knowledge hub

- $\tilde{p}_j$ and $\tilde{f}_j$ are the project and family of j.
- $\tilde{\alpha}_j := \max_{pa \in \dot{A}_j} \alpha_{pa}$, $\tilde{\bar{\omega}}_j := \min_{pa \in \dot{A}_j} \bar{\omega}_{pa}$, $\tilde{\omega}_j := \min_{pa \in \dot{A}_j} \omega_{pa}$ are the release date, due date and deadline of j, respectively.
- $\tilde{M}_j := \bigcap_{pa \in \dot{A}_j} M_{pa}$ is the set of available modes.
- $\tilde{d}^m_j := \lceil (s_{pjf_j} + \sum_{pa \in \dot{A}_j} d_{pa}) \cdot v_m \rceil$ is the (integer) duration of the job under mode m.
- $\tilde{r}_{jg} := \max_{pa \in \dot{A}_j} r_{pa,g}$ are the required units of resource group g.
- $\tilde{R}^{\text{Pr}}_{j,g} := \bigcap_{pa \in \dot{A}_j} R^{\text{Pr}}_{pa,g}$ are the preferred resources from resource group g of j.
- $\tilde{R}_{j,g} := \bigcap_{pa \in \dot{A}_j} R_{pa,g}$ are the available resources from resource group g of j.
- $\tilde{P}_j := \{k \in J \setminus \{j\} : \exists pa \in \dot{A}_j, pb \in \dot{A}_k \text{ s.t. } pb \in P_{pa}\}$ is the set of predecessor jobs of j.
- $\tilde{L}_p := \{(j,k) \in J \times J : j \neq k \wedge \exists pa \in \dot{A}_j, pb \in \dot{A}_k \text{ s.t. } (pa,pb) \in L_p\}$ defines the linked jobs in project p.

In addition, a solution contains the following assignments for each job:

- $\dot{t}^s_j \in T$ the scheduled start time slot
- $\dot{t}^c_j \in T$ the scheduled completion time
- $\dot{m}_j \in M$ the mode in which the job should be performed
- $\dot{R}_{jg} \subseteq R_g$ the set of assigned resources from resource group g

2.1.7 Constraints

A solution is evaluated in terms of constraints that it should fulfill. Hard constraints must all be satisfied in any feasible schedule, while the number and degree of violations of soft constraints in a solution give a measure for its quality.

For the purpose of modeling, we introduce additional notation: The set of active jobs at time t is defined as $J_t := \{j \in J : \dot{t}^s_j \leq t \wedge \dot{t}^c_j > t\}$.

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

Hard Constraints

- **H1: Job assignment:** Each task must be assigned to exactly one job.

$$\forall p \in P, pa \in A_p : \exists! j \in J \text{ s.t. } pa \in \dot{A}_j$$

- **H2: Job grouping:** All tasks contained in a job must be from the same project and family.

$$\forall j \in J, pa \in \dot{A}_j : p = \tilde{p}_j \wedge f_{pa} = \tilde{f}_j$$

- **H3: Fixed tasks:** Each group of tasks assigned to a fixed job in the base schedule must also be assigned to a single job in the solution.

$$\forall j_0 \in J_0 : \exists j \in J \text{ s.t. } \dot{A}^F_{j_0} \subseteq \dot{A}_j$$

- **H4: Job duration:** The interval between start and completion of a job must match the job's duration.

$$\forall j \in J : \dot{t}^c_j - \dot{t}^s_j = \tilde{d}^{\dot{m}_j}_j$$

- **H5: Time Window:** Each job must lie completely within the time window from the release date to the deadline.

$$\forall j \in J : \dot{t}^s_j \geq \tilde{\alpha}_j \wedge \dot{t}^c_j \leq \tilde{\omega}_j$$

- **H6: Task precedence:** A job can start only after all prerequisite jobs have been completed.

$$\forall j \in J, k \in \tilde{P}_j : \dot{t}^c_k \leq \dot{t}^s_j$$

- **H7: Started jobs:** A job containing fixed tasks of a started job in the base schedule must start at time 0.

$$\forall j \in J, j_s \in J^S_0 : t^F_{j_s} = 1 \wedge \dot{A}^F_{j_s} \subseteq \dot{A}_j \implies \dot{t}^s_j = 0$$

- **H8-GR: Single assignment:** At any one time, each workbench, employee, and device can be assigned to at most one job.

$$\forall R_g \in R^*, r \in R_g, t \in T : |\{j \in J_t : r \in \dot{R}_{jg}\}| \leq 1$$

- **H9-GR: Resource group requirements:** Each job must have enough resources of each resource group assigned to cover the demand for that group.

$$\forall j \in J, g \in R^* : |\dot{R}_{jg}| = \tilde{r}_{jg}$$

- **H10-GR: Resource availability:** The resources assigned to a job must be taken from the set of available resources for each group.

$$\forall j \in J, g \in R^* : \dot{R}_{jg} \subseteq \tilde{R}_{jg}$$

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 WIEN Your knowledge hub

- **H11: Linked jobs:** Linked jobs must be assigned exactly the same resources if their resource groups is included in L^{rg}.

$$\forall R_g \in L^{rg}, p \in P, (j,k) \in \tilde{L}_p : \dot{R}_{jg} = \dot{R}_{kg}$$

- **Mode availability**[1]: Mode assigned to a job should be available to all of its tasks:

$$\forall j \in J, pa \in \dot{A}_j : \dot{m}_j \in M_{pa}$$

Soft Constraints

The TLSP contains few soft constraints, which are not required to be satisfied in a feasible solution. Therefore we are not going to discuss them, as we are interested only in constraints that can cause infeasibility.

2.2 Related Work

The explanation of infeasibility in CSPs is generally approached in two primary ways: using minimal conflict sets, and through counterfactual explanations. Below, we provide a detailed description of each category and highlight the corresponding existing work.

2.2.1 Minimal Conflict Sets

Explanation generation in CSPs has a long-standing history, primarily centered on the identification of minimal conflicts (or MUSes) and maximal satisfiable sets (MSSes). A minimal conflict set is defined as the smallest subset of constraints that make the problem infeasible, while maximal satisfiable sets identify the largest satisfiable subsets of constraints. These notions have been extensively studied in the literature [Jun01, Jun04, LS08b, DGGO21, MSHJ$^+$13].

The QuickXplain algorithm proposed by Junker [Jun04] is one of the foundational approaches to efficiently compute minimal conflicts in CSPs. QuickXplain uses divide-and-conquer strategies to identify minimal unsatisfiable subsets without testing all subsets explicitly. Later, Ferguson and O'Sullivan [FO07] generalized conflict-based explanations to the Quantified CSP (QCSP) framework. Their work extended QuickXplain to allow for constraint relaxations, making it suitable for generating explanations in more complex contexts.

Liffiton and Sakallah [LS08a] introduced CAMUS, an approach to compute all minimal unsatisfiable subsets of constraints by iteratively solving optimization problems to identify MSSes. This method has been widely adopted in various domains, including SAT and CSPs. We explore the approach with a slight modification as explained later on in Section 3.3.

[1]Note: The mode availability constraint was inadvertently left out of the formal model presentation in [MM18a] and only described informally. It is included here for completeness.

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

These approaches emphasize the importance of identifying conflicts or relaxations as the primary mechanism for explaining infeasibility in CSPs. However, they do not explicitly address the notion of counterfactual explanations, which has recently gained attention in the field of explainable artificial intelligence (XAI).

In terms of the similarity of the problem subject to explainability, the paper by Lauffer et al. [LT19] is closely related to our work. This paper investigates explainability of scheduling infeasibility in the agent resource-constrained project scheduling problem (ARCPSP), an extension of the resource-constrained project scheduling problem. Most importantly, it provides a framework for generating human-understandable explanations of infeasibility for instances of the ARCPSP, using MUSes and MCSes, which closely aligns with our explanations approach defined in Section 3.3. The difference lies in the specific domain and constraints of the ARCPSP compared to the TLSP, as well as the MUS/MSS enumeration technique.

Similarly, Bleukx et al. [BBG+25] also adapt techniques from explainable constraint solving to their specific industrial scheduling domain. In particular, they integrate MUS-based conflict identification into their system to explain why certain schedules become infeasible. For MUS extraction, they use a deletion-based minimization approach, the *(*Shrink) and *SMUS* algorithms. To restore feasibility, they rely on the MUS/MCS duality by computing minimal correction sets (MCSes), and use Max-CSP for computing them. Their work demonstrates how classical explainability tools (MUS/MCS) can be adapted to real-world scheduling scenarios, which closely aligns with the aims of this thesis: applying and extending existing explainability techniques to the TLSP.

Another closely related work is that of Senthooran et al. [SKB+23], who propose an interactive system for exploring the infeasibility of combinatorial optimisation problems. Their approach combines MUS enumeration, MCS computation, soft-constraint relaxation via slack variables, and interactive visualisation to help users understand and resolve infeasibility. A key difference compared to our work lies both in how constraint relaxation is employed and in the techniques used to compute MUSes. Senthooran et al. use slack-based relaxations as part of their interactive feasibility-restoration process, primarily to guide users in selecting which constraints to soften during MUS computation. We also adopt a concept similar to slack variables, but in our case these relaxations serve as the foundation of a dedicated counterfactual explainer: they directly produce actionable counterfactual explanations (Section 3.4), requiring no additional post-processing. Moreover, while their framework is general and problem-independent, our approach is tailored specifically to the TLSP, allowing us to exploit domain structure to provide domain-specific explanations.

2.2.2 Counterfactual Explanations

Counterfactual explanations are a more recent and complementary approach to explaining infeasibility in CSPs. Unlike conflict-based methods, counterfactual explanations focus on identifying minimal changes to the problem that would render it feasible. This approach

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

was first popularized by Wachter et al. [WMR17] in the context of machine learning, where counterfactuals provide the smallest modification to a model's input to change its output. Keane et al. [KKDS21] further explored counterfactuals in XAI, analyzing over 100 methods and proposing a roadmap for improvement.

In the domain of constraint programming, Korikov et al. [KSB21] extended the notion of counterfactual explanations to optimization-based decisions using inverse optimization. They addressed the problem of explaining why a solution to an optimization problem does not satisfy a set of additional user-defined constraints. Their approach aimed to find the nearest counterfactual explanation by identifying minimal changes to the solution that restore feasibility while remaining as close as possible to the original solution.

Korikov and Beck [KB21b] generalized their work to CSPs and introduced the concept of inverse constraint programming. They demonstrated how counterfactual explanations can be generated using inverse optimization techniques with a cost vector. However, they noted that the connection between conflict-detection mechanisms in CSPs and counterfactual explanations remains underexplored. The presented methodology inspired our work to use constraint programming to generate counterfactual explanations for infeasibility.

Gupta et al. [GGO22] propose a novel approach to generating counterfactual explanations through constraint relaxations. The authors demonstrate that by relaxing constraints and re-solving the scheduling problem, it is possible to identify the minimum set of constraints that need to be relaxed in order to generate a feasible solution. This approach can provide users with valuable insights into why a particular solution was not feasible and what changes could be made to the original problem formulation to achieve a feasible solution. We explore a similar approach, which is explained in detail in Section 3.4.

2.2.3 TLSP solution approaches

Various solution methods have been proposed for the Test Laboratory Scheduling Problem (TLSP) and its subproblem with fixed task grouping (TLSP-S). Mischek and Musliu [MM18a] introduced metaheuristic approaches using a combination of different neighborhood structures, with Simulated Annealing (SA) consistently achieving the best results across different instance sizes. These methods were initially developed for TLSP-S, were later extended to the full TLSP [MMS21, MMS22].

Danzinger et al. [DGMM20, DGJ+23] proposed an exact approach based on Constraint Programming (CP), which performs well on smaller instances but struggles with scalability. To address this, the mentioned papers introduced a hybrid Very Large Neighborhood Search (VLNS) method, where small subproblems consisting of few projects are solved repeatedly, while the rest of the schedule remains fixed. This approach has proven effective even for large TLSP instances and currently represents a state-of-the-art method for both TLSP and TLSP-S.

Geibinger et al. [GMM21] explored an alternative exact method using Constraint Answer-set Programming (CASP), which was also integrated into the VLNS framework

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

for TLSP-S. However, CASP has not yet been extended to the full TLSP. Both SA and VLNS-based solvers have been successfully deployed in an industrial test laboratory and are used in daily operations to generate high-quality schedules.

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

CHAPTER 3

Explainability for TLSP

This section provides a detailed exploration of the concept of explainability for infeasibility within the TLSP. We begin with a brief introduction to TLSP in section 2.1.4, drawing from existing literature to set the context. We then define the various types and categories of explanations that are examined throughout this thesis in section 3.1. In Section 3.2, we discuss the crucial process of partitioning constraints into "background" (non-modifiable, hard constraints) and "foreground" (modifiable constraints), which facilitates controlled generation of explanations. Next, we delve into the technical aspects of generating explanations, all of which rely on exact methods. Finally, in Sections 3.3 and 3.4, we present the specific methodologies employed for generating two types of explanations: those based on constraint conflicts and counterfactual explanations, respectively.

3.1 Defining TLSP explanations

This section introduces the foundational concepts related to infeasibility and explanations within the context of the TLSP. It defines the problem of infeasibility, defines the explanations of infeasibility and discusses their roles in identifying and resolving infeasibility. Given that TLSP can be formalized as a CSP, we adopt the notation $\mathcal{P} := (\mathcal{V}, \mathcal{D}, \mathcal{C})$, where $\mathcal{V}$ refers to the set of variables, with finite domains in $\mathcal{D}$, and constraints on them defined in $\mathcal{C}$. Further, we distinguish two groups of constraints in $\mathcal{C} := \mathcal{B} \cup \mathcal{F}$, where $\mathcal{B}$ represents the set of *background constraints*, and $\mathcal{F}$ represents the set of *foreground constraints*. We dedicate a whole section for the partition of the constraints set $\mathcal{C}$ into $\mathcal{B}$ and $\mathcal{F}$, but for now it is sufficient to mention that *background constraints* are logical constraints, which cannot be relaxed therefore cannot be part of any explanations. In contrast *foreground constraints* can be relaxed and will be used for explanations.

Definition: **Infeasibility** is defined as the situation when there is no way of assigning valid values from $\mathcal{D}$ to the decision variables in $\mathcal{V}$ such that all constraints in $\mathcal{B} \cup \mathcal{F}$ are

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

satisfied.

Generally, infeasibility can be caused due to:

- **Background constraints**: in this case we say that the model is inconsistent. Explanations for this type of infeasibility would be useful for the developer to debug and correct the model, but are not relevant for this thesis.
- **Foreground constraints**: in this case we say that the user requirements are inconsistent. Explanations for this type of infeasibility are useful for the end user, and are relevant for this thesis.

We assume that the existing model for TLSP is correct and consistent, meaning that the set of background constraints is satisfiable, i.e. there exists an assignment of values from the domains of all variables that satisfies all background constraints.

As mentioned briefly in Section 1.1, TLSP draws from general factors causing infeasibility in RCPSPs. Moreover, TLSP presents additional unique sources of infeasibility shaped by its context. Some factors that can contribute to scheduling infeasibility in the context of TLSP are:

- Resource Constraints: In a test laboratory, resources such as testing equipment, specialized tools, and skilled personnel are often limited. If the scheduling algorithm cannot allocate these resources to meet the testing demand, infeasibility may arise. Different groups of resources may have different means of relaxation, while some may be easily replaceable, others may not be replaceable at all. That is why it is necessary to distinguish which resource is causing infeasibility.
- Temporal Constraints: schedules need to adhere to temporal constraints, such as setup times, task or project release dates and deadlines.
- Dependency Constraints: Some tasks may have dependencies on other tasks. In TLSP one such dependency is the concept of linked jobs, where assignments of resources units must be the same for all jobs in the set. Another constraint is that of precedence, where jobs cannot start before their predecessor.
- Interactivity: For automated scheduling problems with several constraints like TLSP, introducing new projects to an existing schedule can quickly lead to infeasibility, especially if we aim to preserve the existing schedule, which may have taken significant time to optimize.

Based on the above causes of infeasibility in TLSP, we can group explanations according to the faulty constraint involved:

1. **Resource requirement explanations**: Explanations point out which resources are causing the infeasibility.

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

2. **Temporal explanations**: Explanations point out which release date, or deadlines are causing the infeasibility.

3. **Single assignment explanations**: Explanations identify the resource requirements of a task that cannot be satisfied within the same time interval as all other requirements.

4. **Dependency explanations**: Explanations point out which dependencies cannot be satisfied, by giving the set of dependent jobs and the type of dependency, which can be *linked jobs*, *precedence*, or *mode availability*.

5. **Interactivity explanations**: Explanations point out which *fixed jobs/projects* are causing the infeasibility, where unfixing them would restore the feasibility. These explanations are especially useful for interactive systems, which need to answer whether a new task/project can be added to the existing schedule, or which combination of projects does not work.

Current work on this area suggests various definitions of explanations, however, our focus will be mainly on two of them:

1. Conflict-based explanations [LT19]. These explanations use MUSes and MSSes, to describe the source of infeasibility. The recovery of feasibility is given in terms of minimal correction sets (MCS), using two-point relaxations.

2. Counterfactual explanations [GGO22]. These explanations identify infeasibility as a minimal set of constraints that must be relaxed, modified, or removed to restore feasibility. The process of restoring feasibility is expressed in terms of multi-point relaxations, where simultaneous adjustments to multiple constraints are proposed to resolve conflicts effectively.

All of the mentioned constraints sets are subsets of the foreground constraints $\mathcal{F}$.

3.2 Defining background and foreground constraints

The first stage of generating infeasibility explanations is the process of partitioning the constraints into *background* and *foreground*. The former represents a group of logical constraints, which must be satisfied by every solution, and cannot be relaxed or modified. These constraints ensure that the schedule is logical. As such none of these constraints can be included in any explanations or relaxations. In contrast, the foreground constraints can be modified and relaxed, as they represent user requirements, therefore can also be included in explanations. This form of constraint partitioning is discussed in [GGO22, LT19, DGGO21].
This partition enables us to control the generation of explanations related to different aspects of the scheduling problem. Another important aspect of such partition is

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

that it allows us to control the size of the search space, as well as the quality of the explanations/relaxations. We will consider different partitions of background and foreground, and overview the trade-off between the quality of explanations, and the efficiency of finding such explanations. Different foreground constraints support different explanations, we can categorize them as shown in Table 3.1. All constraints not included in Table 3.1 will be considered background constraints, and will not be used for explanations.

Foreground	Explanations
Workbench requirements Employee requirements Equipment requirements	Resource requirement explanations
Release date Deadline	Temporal explanations
Single assignment	Single assignment explanations
Task precedence Mode requirements Linked jobs	Dependency explanations
Fixed jobs/projects	Interactivity explanations

Table 3.1: Explanations derived from corresponding foreground constraints

Depending on the methodology used for the infeasibility explainer, the foreground constraints shown in Table 3.1 will use different points of relaxations, thus we need to define them individually. Importantly, when we refer to removing, modifying, or relaxing a constraint, we mean doing so for a specific instance of that constraint type—for example, the deadline of a particular task or the resource requirement of a specific task—rather than altering the entire class of such constraints.

Resource requirement explanation For these explanations, we will consider explainers that utilize both two-point relaxation spaces and multi-point relaxation spaces. In the former, we only check whether the removal of a constraint restores feasibility, while in the latter, we must explicitly define the relaxation points. When these explanations are generated by an explainer that supports multi-point relaxations, we can define these points as follows. Initially the user can configure the bounds of relaxation for each type of resource: *workbench*, *employee*, or *equipment*, via a discrete variable X_{group} that represents the maximum amount of relaxation for the requirements of each resource group.

Given the task requirement for a resource r_{tg}, which must be allocated from the set of available resources $R_{t,g} \subseteq R_g$, we define the relaxation points as reduced values of the resource demand, achieved by decrementing the original value within a feasible range:

$$r_{tg}^R \in (max(r_{tg} - X_g, 0), r_{tg}),$$

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

where r^R_{tg} represents the relaxed requirements, and X_g represents the relaxation bound for the specific resource group g. X_g is a user configurable variable.

This relaxation ensures that the task's resource requirement is adjusted downward while staying within a feasible range defined by X_g. For example, if a task originally required 10 units of a resource group g and $X_g = 3$, the relaxed requirement could range from 7 to 10 units.

Temporal Explanation Similar to the resource requirement explanations discussed earlier, this section also considers explainers that support both two-point and multi-point relaxation spaces. While the general methodology remains consistent for the explainers that support a two-point relaxation space, we still need to explicitly define the relaxation points for the explainers that support multi-point relaxation spaces.

For a task t with a release date α_t and a deadline ω_t, the user can configure the bounds of relaxation through discrete variables X_α and X_ω, representing the allowable relaxation for the release date and deadline, respectively.

- **Release Date Relaxation:** The relaxation points for α_t are defined as decrements to the original release date, constrained by the lower bound X_α:

  $$\alpha^R_t \in (max(\alpha_t - X_\alpha, 0), \alpha_t).$$

  This allows the task to start earlier than originally scheduled, up to the maximum bound defined by X_R.

- **Deadline Relaxation:** Similarly, the relaxation points for ω_t are defined as increments to the original deadline, constrained by the upper bound X_ω:

  $$\omega^R_t \in (\omega_t, \omega_t + X_\omega).$$

  This adjustment pushes the deadline further into the future within a feasible range defined by the relaxation bound X_ω.

By defining these relaxation points, we create a structured framework to modify the release dates and deadlines of tasks to preserve feasibility. For example, if a task originally had a release date $\alpha_t = 10$ and a deadline $\omega_t = 50$, and the relaxation bounds are $X_\alpha = 3$ and $X_\omega = 5$, then the relaxed release date could range from 7 to 10, and the relaxed deadline could range from 50 to 55.

Single assignment explanations These explanations support only a two-point relaxation space. Relaxation is applied by removing any constraint that enforces a given resource's requirements. They differ from resource requirement explanations, as these explanations suggest changes only to the user's resource requirements. In contrast, single assignment explanations require additional modifications. If the explainer reports a

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

relaxation of a resource requirement for a task, this means that not only should the user's resource requirement be set to zero for the reported resource, but if the task belongs to any linked set, it must also be removed from that set. This is because linked sets restrict tasks to use the same resources. An example of why this is necessary is shown in Section 5.1.2.

These explanations are the only ones from the mentioned list that cannot be combined with all other explanations. This limitation arises from their corresponding foreground constraint, which is the only one in the list that functions as a background (logical) constraint. They cannot be combined with *Linked jobs* explanations, as the relaxation requires this constraint be removed either way.

As such, these explanations are not useful for directly restoring feasibility for the end user, as the relaxation requires removing constraints that may not be necessary. Despite this, their generation time is faster than that of other explanations, and they can still be highly effective for identifying resource conflicts, as they quickly produce results that clearly indicate the specific resource requirements causing the issue. They can be used in a two-phase process in conjunction with other explanations:

- First, the user runs the explainer with only *Single assignment* explanations enabled. This configuration is used to identify which resources are causing the infeasibility.
- Then, the user runs the explainer with *Resource requirement* explanations enabled only for the resource groups identified in the first phase. This configuration greatly reduces the search space, allowing the user to find solutions to restore feasibility more quickly.

Dependency Explanation Similar to the explanations discussed earlier, this section also considers explainers that support both two-point and multi-point relaxation spaces. For the *task precedence* and *linked jobs* explanations, which support only two-point relaxation space we skip this part, for the rest we define all relaxation points.

- **Mode Relaxation:** Relaxation of mode constraints allow the task to access additional modes from the global set of modes M. To enable this flexibility, we introduce a binary decision variable $X_{\text{mode},m}$, where $m \in M$, indicating whether a specific mode m can be added as a relaxation option for a given task. The user can configure these variables to restrict or permit relaxation to certain modes depending on the context.

  Given the original set of available modes for a task $\mathrm{M}_t \subseteq M$, the relaxation points are defined as follows:

  $$\mathrm{M}'_t = \mathrm{M}_t \cup \{m \in M \mid X_{\text{mode},m} = 1\}.$$

  This ensures that only the modes explicitly permitted by the user (through $X_{\text{mode},m}$) are considered for relaxation, allowing fine-grained control over how the flexibility

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

is applied. For example, if a task initially has $M_t = \{1, 3\}$ and the user sets $X_{\text{mode},2} = 1$ and $X_{\text{mode},4} = 0$, the relaxed set becomes $\{1, 2, 3\}$, enabling the task to utilize mode 2 as an additional option.

Interactivity Explanation Fixed Job/Project Explanations: We define two points of relaxation for these types of explanations:

-
  - **I:** We check if simply removing the fixed job/project constraint is sufficient to restore the satisfiability of the instance.
  - **II:** We check if additional relaxation of other foreground constraints is necessary to achieve the satisfiability of the instance.

If the explainer is configured to generate this type of explanation, then for a given job/project, other foreground constraints can only be relaxed if the job/project is either not fixed or initially fixed but unfixed in its relaxed version. The second point of relaxation, as listed above, is configurable by the user. The user can choose to disallow the relaxation of other constraints, which effectively results in a two-point relaxation space.

3.3 Conflict-based explanations

This section outlines the methodologies used to generate conflict-based explanations. A single infeasible instance of TLSP can have multiple MUSes, meaning that different minimal sets of constraints can independently conflict with one another. Each set represents a distinct explanation for the infeasibility, highlighting different aspects of the problem depending on the specific constraints involved. While any such minimal set can be used individually to explain the infeasibility, addressing all of them collectively is necessary to restore feasibility.
Taking this into consideration, we explore two different approaches for explainability:

1. Focus in finding all reasons for infeasibility, rather than restoring the feasibility. This translates to finding as many MUSes as possible. If we are able to find all of them, then we are able to restore the feasibility by computing MCSes as irreducible hitting sets of MUSes.

2. Focus in restoring the feasibility, rather than finding all conflicting sets of constraints. This translates to finding as many MCSes as possible, and then computing MUSes as irreducible hitting sets of MCSes.

We used the corresponding methodologies for the above approaches:

1. **findMUS**, a Minizinc solver. As findMUS only provides MUSes, additional post-processing is required to identify MCSes, this step includes computing the irreducible hitting sets of found MUSes.

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

2. **CAMUS**. An approach defined in [LS08a]. In this approach for generating all MUSes of a constraint system, all MCSes are first identified by iteratively finding the largest satisfiable subset that has not been discovered in previous iterations using *Max-SAT*. Once no new MCS is found, the irreducible hitting sets of the MCSes are computed, which represent all MUSes of the system.

In the following sections, we will thoroughly examine each of these approaches in detail. Furthermore, we will describe the algorithm employed for calculating irreducible hitting sets, as it plays a pivotal role in both of the aforementioned approaches.

3.3.1 Irreducible hitting sets

This section summarizes the algorithm for computing all irreducible hitting sets of a collection of sets, as we will use this algorithm in two contexts:

- To compute the MCSes from a given set of MUSes.
- To compute the MUSes from a given set of MCSes.

We use the method described in [LS08a] because it is straightforward to implement and achieves practical efficiency in most cases. The only difference from the original presentation is that, for clarity, we describe the algorithm in general terms, rather than specifically in terms of MCSes and MUSes as the referenced paper does.

Given a collection of sets $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ of constraints, a *hitting set* H is defined as a subset of the universal set of constraints $\mathcal{U} = \bigcup_{i=1}^{n} M_i$ such that $H \cap M_i \neq \emptyset$ for every $M_i \in \mathcal{M}$. A *minimal hitting set* is a hitting set for which no proper subset is also a hitting set. The goal of the algorithm is to enumerate all such minimal hitting sets for the given collection $\mathcal{M}$.

We explain the algorithm in two parts. First, the pseudo-code for the algorithm for finding a single hitting set is given in 3.1. The algorithm starts by selecting an arbitrary constraint c from an arbitrary set M_i, from the remaining collection of sets $\mathcal{M}'$. On line 6, this constraint is added to the current hitting set H, meaning that c must be an irredundant element of H. To ensure irredundancy, on line 7 of the 3.1, the subroutine **PropagateChoice**, detailed in 3.2, is called. This ensures that no other constraints from the same set M_i are included in H, as implemented on lines 1–7 of **PropagateChoice**.

On lines 8–12 of 3.2, the algorithm removes covered sets M from $\mathcal{M}'$, ensuring that only uncovered sets remain in the collection. Finally, line 7 of 3.1 invokes a subroutine that eliminates any set in $\mathcal{M}'$ that is a superset of another. The algorithm terminates when $\mathcal{M}'$ is empty, returning a single irreducible hitting set H of the collection of sets $\mathcal{M}$.

Lines 4 and 5 of Algorithm 3.1 will determine the computed hitting set. Thus making different choices at those two points will lead to different hitting sets. Therefore, the algorithm to compute all irreducible hittings sets described in [LS08a], is computed with

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 WIEN Your knowledge hub

Algorithm 3.1: SingleHittingSet($\mathcal{M}$)

Input: A collection of sets $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ over a universe of constraints $\mathcal{U}$

Output: A single hitting set $H \subseteq \mathcal{U}$

```
1  H ← ∅;
2  M' ← M;
3  while M' ≠ ∅ do
4  |   c ← SelectRemaingConstraint(M');
5  |   M_i ← SelectSetContaining(M', c);
6  |   H ← H ∪ {c} ;
7  |   M' ← PropagateChoice(M', c, M_i);
8  |   M' ← MaintainNoSupersets(M');
9  end
10 return H;
```

Algorithm 3.2: PropagateChoice($\mathcal{M}'$, c, M_i)

Input: The current collection of sets $\mathcal{M}'$, the current constraint c from the set M_i

Output: Modified $\mathcal{M}'$

```
1  foreach c' ∈ M_i do
2  |   foreach M_j ∈ M do
3  |   |   if c' ∈ M_j then
4  |   |   |   M_j ← M_j − {c'}
5  |   |   end
6  |   end
7  end
8  foreach M ∈ M' do
9  |   if c ∈ M then
10 |   |   M' ← M' − {M}
11 |   end
12 end
13 return M';
```

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

a recursive algorithm that branches at those two points and tries all possible choices for each.

Algorithm 3.3: AllHittingSets($\mathcal{M}, \mathcal{H}, H$)

Input: A collection of sets $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ over a universe of constraints $\mathcal{U}$, current set of irreducible hitting sets $\mathcal{H}$ of $\mathcal{M}$, and current hitting set H

Output: The complete set of irreducible hitting sets $\mathcal{H}$ of $\mathcal{M}$

```
1  if M = ∅ then
2  |   H ← H ∪ {H} return H;
3  end
4  foreach c ∈ RemaingConstraints(M) do
5  |   H′ ← H ∪ c ;
6  |   foreach M_i ∈ M such that c ∈ M_i do
7  |   |   M′ ← PropagateChoice(M, c, M_i) AllHittingSets(M′, H, H′)
8  |   end
9  end
10 return H;
```

The list of the optimizations presented in the paper, that we used in our algorithm contains:

- Pruning recursion branches using a hash table to store an intermediate state. This avoids duplicate hitting sets.
- Ordering of the clauses selected on line 5 to select first clauses from single-element sets.

The algorithm ensures completeness by systematically exploring all combinations of elements in $\mathcal{U}$, but its computational complexity is exponential in the worst case due to the combinatorial nature of the problem. However, by leveraging efficient pruning and minimality checks, the algorithm minimizes redundant computations, making it practical for moderately-sized collections $\mathcal{S}$.

3.3.2 First approach: using findMUS

The tool employed for this approach, known as findMUS, is a MiniZinc solver specifically designed to identify MUSes within an unsatisfiable constraint model. We utilize the modified model [GMM19] outlined in 4.1.1, wherein all foreground constraints are annotated. Annotating these constraints is essential because findMUS necessitates specifying, at runtime, which constraints should be treated as foreground constraints. This functionality facilitates the straightforward investigation of various sets of foreground constraints, thus giving us control over the search space.

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

Given any instance of the problem, where the set of foreground constraints F that we consider is actually the set of all foreground constraints $F = \mathcal{F}$, given arbitrary time the possible outcomes of the findMUS are:

- Instance is satisfiable: Therefore no MUSes can be found.
- Instance is unsatisfiable: A set of MUSes containing constraints from $\mathcal{F}$ is reported

Given any instance of the problem, and given a set of foreground constraints $F \subset \mathcal{F}$, given arbitrary time the possible outcomes of the findMUS are:

- Instance is satisfiable: Therefore no MUSes can be found.
- Instance is unsatisfiable:
  - A set of MUSes containing constraints from F is reported
  - No MUSes are reported, meaning the conflict is in a different set of foreground constraints

We can explore different smaller sets of foreground constraints when we are interested in finding any conflict faster. However, if we are also interested in regaining feasibility we need to consider the set of all foreground constraints, as only that way we can get all the MUSes, and then compute MCSes.
Therefore, in general we explore the utilization of findMUS in two settings:

1. As a sole explainer, where we use findMUS only to generate explanations via constraint conflicts, represented as MUSes.
2. As part of a system which is able to generate explanations of the conflicts, and in addition reports relaxations which can be used to regain feasibility. This explainer is composed of two parts, the findMUS which provides the MUSes (explanations), and the part that computes the MCSes (relaxations) from MUSes using procedure 3.3.1.

Given an existing MiniZinc model, leveraging it to construct an explainer using the findMUS approach has clear practical benefits. The process is relatively straightforward because it relies on using MiniZinc's built-in solver capabilities to find MUSes. Below we conclude the main advantages and drawbacks of such an approach:

- **Advantages**:
  - **Ease of implementation**: Since the MiniZinc model already represents the problem constraints, adding the necessary annotations for explanation purposes (tagging modifiable and non-modifiable constraints) is a relatively simple extension.

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 WIEN Your knowledge hub

  - **Alignment with existing models**: By using the MiniZinc framework, the approach avoids the need to rewrite or re-encode the problem in another format, making it practical for researchers or practitioners already working within the MiniZinc ecosystem.

- **Drawbacks**:

  - **Computational overhead**: While the approach is practical, it requires post-processing to compute MCSes from the identified MUSes. The MUSes themselves only indicate the infeasible combinations of constraints but do not directly tell us how to restore feasibility. Generating MCSes involves computing the irreducible hitting sets of MUSes, a task that adds a significant computational burden, especially for large or complex problem instances.
  - **Limitations in explanation quality**: The approach using findMUS works at the level of constraints, identifying sets that are responsible for infeasibility. However, it lacks the granularity to explain why a specific constraint fails. For example, it cannot identify which part of a constraint (e.g., a specific variable or term) is causing the conflict. This is due to the lack of constraint modification support in the model. MiniZinc models typically define constraints as atomic units, which limits their ability to pinpoint finer details such as variable contributions or parameter thresholds that lead to failure. Consequently, the explanations may feel generic or less actionable to users, as they fail to pinpoint the exact 'cause-and-effect' relationships behind infeasibility.

A concrete example of the limitations in explanation quality Annotations in MiniZinc can be applied to constraints to provide more context or metadata for the generated explanations. As an example, in most cases it would be beneficial to know which task is failing to satisfy the constraint. For some constraints, identifying the task that is faulty is very straightforward using annotations like below:

```
constraint :: "JobPrecedence"
forall(t in Tasks, t2 in prerequisiteTasks[t])(
  (taskJobs[t] == taskJobs[t2] \/ endTime[taskJobs[t2]] <=
     startTime[taskJobs[t]]) :: "JobPrecedence:t1=\(t),t2=\(t2)"
);
```

Listing 3.1: JobPrecedence constraint from [GMM19] modified with annotations

With this annotation, *findMUS* will report that the *JobPrecedence* constraint cannot be satisfied for the corresponding tasks. However, in the case of the cumulative constraint, adding annotations to identify specific tasks contributing to infeasibility is more challenging compared to a simpler constraint like the *JobPrecedence* example.

```
constraint :: "SingleResourceAssignment"
forall(e in ResourceClasses)(
```


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 WIEN Your knowledge hub

```
    cumulative(startTime, [jobDuration(t) | t in Tasks],
        [resourceClassesJobs[e, t] | t in Tasks],
        resourceClassQuantities[e])::
        "SingleResourceAssignment:e=\(e)
  );
```

Listing 3.2: SingleResourceAssignment constraint from [GMM19] modified with annotations

The challenge stems from the fact that annotations in MiniZinc operate at the constraint level. While MiniZinc supports annotations on single constraints, it does not allow annotations on individual components within global constraints. To illustrate this more precisely, we investigate the case of *cumulative* in our model. While we can annotate the *forall* loop or the *cumulative* constraint itself with the faulty *resource*, you cannot directly attach annotations to individual tasks within the *cumulative*. Therefore, the *cumulative* constraint does not provide intermediate feedback about which specific tasks are causing the overload. For example, it does not output which combination of tasks exceeds the resource limit at a specific time point. If we want to track which tasks are causing the infeasibility in the *cumulative* constraint, we can reformulate the problem to explicitly check resource violations for individual tasks or time points. An alternative model that can support this approach is given in [GMM19], where we can track interval variables. However, this workaround sacrifices some of the compactness and efficiency of the cumulative constraint, as it essentially decomposes it into simpler checks. Consequently, the practicality advantage of this approach diminishes. While this trade-off between solver efficiency and explanations quality may not be noticeable for small instances, it becomes a significant issue for larger ones.

3.3.3 Second approach: based on CAMUS

From the inherent complexity disparity between problems in NP (e.g., SAT) and those categorized under Co-NP (e.g., UNSAT), it follows that finding satisfiable subsets of constraints is an easier task than identifying unsatisfiable subsets. In our context, this manifests as the comparative ease of finding MSSes (as complements of MCSes), rather than directly finding MUSes. This rationale is followed by CAMUS, which is an approach that first generates all MUSes of a constraint system and then finds the MUSes by computing the irreducible hitting sets of generated MCSes.

Similar to CAMUS, our approach is composed of two phases, with independent algorithms:

- Phase 1: Identifying all MCSes (lines 1-11 of Algorithm 3.4)
- Phase 2: Computing MUSes (line 12 of Algorithm 3.4)

The difference of our approach lies in the first phase, where instead of finding MCSes by successively solving an optimization problem as an iterated MaxSAT problem, we solve

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, WIEN Your knowledge hub

Algorithm 3.4: Two-Phase Approach for Computing MUSes

Input: ϕ: OR-Tools model, A: Set of foreground constraints assumption variables

Output: $\mathcal{MCS}$: Set of all MCSes, $\mathcal{MUS}$: Set of all MUSes

```
1  𝓜𝓒𝓢 ← ∅ ;
2  φ′ ← AddAssumpVars (φ,A) ;
3  while true do
4  │  (S, MCS) ← SolveMaxCSP(φ′);
5  │  if MCS = ∅ then
6  │  │  break // No more MCSes can be found
7  │  end
8  │  𝓜𝓒𝓢 ← 𝓜𝓒𝓢 ∪ {MCS};
9  │  φ′ ← φ′ ∧ BlockingClause (MCS);
10 │  φ′ ← AddHints (S);
11 end
12 𝓜𝓤𝓢 ← AllHittingSets (𝓜𝓒𝓢,∅,∅);
13 return 𝓜𝓒𝓢, 𝓜𝓤𝓢;
```

it as an iterated *Max-CSP* problem.
We use OR-Tools with the adapted model 4.1.2, to iteratively solve the instance as *Max-CSP* problem. The used OR-Tools model is explained in detail in Section 4.1.2. The key difference from the existing TLSP model is the usage of assumption variables for foreground constraints, which determine if a constraint is activated or not. These assumption variables act as clause selectors: each constraint is conditionally enforced using a Boolean variable—if the variable is set to false, the corresponding constraint is effectively deactivated (i.e. not enforced by the solver).

The algorithm shown in 3.4 expects as input along with the OR-Tools model the list of assumption variables that represent the set of foreground constraints $F \subseteq \mathcal{F}$ that will be explored. Then on line 2, via the $AddAssumpVars$, the given model is modified such that only the given assumptions variables can be deactivated.
On each iteration we keep track of the generated solution S, and the MCS from the execution of $SolveMaxCSP$ (line 4), which is a simple procedure that calls a *Max-CSP* solver with the current model that tries to maximise the number of satisfied assumption variables. By excluding previously found MCSes in each iteration (line 9), we guarantee that at some point no more MCSes can be found, therefore the complete set of MCSes is generated eventually. We use the current solution to provide heuristic guidance to the solver by suggesting possible values for decision variables for the next iteration (line 10). The *while* loop on line 3 breaks once we have exhausted the whole set of MCSes for the given set of foreground constraints. Once we break from the loop and we have the set of MCSes, we use it to compute the set of MUSes (line 12), as irreducible hittings sets of the reported MCSes, via the 3.3.1 algorithm.

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

As a result, this approach will generate explanations that explain the infeasibility via the conflicts that it found (MUSes), and provide solutions how to regain the feasibility via MCSes (by removing one of them).

3.4 Counterfactual explanations

We will use a definition of counterfactual explanations adapted from [WMR17], [GGO22], and [KB21a]. The goal of such explanations is to provide suggestions on minimal changes to the problem's constraints, enabling recovery from an infeasible state. More formally, given an infeasible TLSP instance denoted by $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{B} \cup \mathcal{F})$, a counterfactual explanation is a set of minimal changes to constraints in $\mathcal{F}$, such that the new problem $\mathcal{P}' = (\mathcal{V}, \mathcal{D}, \mathcal{B} \cup \mathcal{F}')$ now is feasible.

Definition: Let $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{B} \cup \mathcal{F})$ be an infeasible instance of TLSP, and $\mathcal{P}' = (\mathcal{V}, \mathcal{D}, \mathcal{B} \cup \mathcal{F}')$ a feasible one. A counterfactual explanation, $\mathcal{E}$, corresponds to the minimal set of changes to $\mathcal{F}$, such that $\mathcal{E} = \mathcal{F}' \setminus \mathcal{F}$. [GGO22]

We generate these explanations by re-solving the instance with relaxed constraints. We solve it with *Weighted CSP*, maximising the number of satisfied constraints. We use the OR-Tools solver with the model presented in Section 4.1.2 for re-solving the instance. *Weighted CSP* allows for the incorporation of preferences and priorities in the resolution of infeasibility, extending beyond traditional CSP methods. This approach uses a multi-point relaxation space, therefore we define the relaxation points for each foreground constraint in Section 4.1.2.

The relaxation space defines how foreground constraints can be modified. Weighting techniques determine what constitutes a "minimal change" in this framework. The explored weighting techniques are listed below:

- **Uniform relaxation cost:** Each relaxed constraint is assigned a unit weight. This assumes all constraints are equally important, focusing on minimizing the number of relaxed constraints.
- **Bound-based relaxation cost:** A constraint's weight represents the magnitude of its relaxation. Specifically, the weight is the absolute difference between the original bound (b_c) and the relaxed bound ($\tilde{b}_c$).
- **Preference-weighted relaxation cost:** Assign custom weights (w_c) to foreground constraints based on their importance or preference. These weights guide the optimization process to prioritize changes to less critical constraints.

Depending on the weighting technique, the objective function optimizes for the following:

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

- **Objective O1**: Maximising the number of satisfied constraints, meaning, minimizing the number of relaxed constraints:

$$\max \sum_{a \in \mathcal{A}} a$$

Here, $\mathcal{A}$ is the set of assumption variables, where each A represents whether a constraint was relaxed.

- **Objective O2**: Minimize the magnitude of relaxations for constraints that support multi-point relaxation space:

$$\min \left[\sum_{a \in \mathcal{A}} a \times |\tilde{b}_{c(a)} - b_{c(a)}| \right]$$

This penalizes relaxations based on the size of the bound changes.

- **Objective O3**: Optimize for preferred constraints:

$$\min \sum_{a_c \in \mathcal{A}} a_c \times w_c$$

Here, w_c is a custom weight assigned to each foreground constraint c, which uses its corresponding assumption variable a_c. Using this objective, the explainer prioritizes satisfying constraints with lower weights, which in turn leads to generating explanations that predominantly involve constraints with higher weights.

In our implementation we use a combination of the three objectives, using appropriate scaling and normalization. We then compare the sets of explanations generated by different combinations of the objectives.

Recent work by [KB21a] explored inverse constraint programming (ICP) for generating counterfactual explanations in optimization problems. This work specifically inspired the use of weighting techniques to produce minimal changes that align with user preferences, such as custom weights and penalties for constraint relaxation.
The methodology explained so far generates a single counterfactual explanation. To explore alternative explanations, we can utilize *blocking clauses* to iteratively exclude previous explanations. We can choose what we want to block for the next iteration:

- **Blocking B1**: Blocking relaxations containing previously reported constraints similar to [LS08a, LT19,]. This ensures that at least one constraint from the previous explanation is excluded in subsequent iterations. The clause added is:

$$\bigvee_{A \in \mathcal{A}_i} A$$

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

- **Blocking B2**: A novel technique proposed in this work that blocks specific relaxation values rather than entire constraints. The clause added is:

$$\bigvee_{A \in \mathcal{A}_i} A \longrightarrow (\tilde{b}_{c(A)} \neq \upsilon_i(\tilde{b}_{c(A)}))$$

  Here, $\upsilon_i(x)$ represents the relaxation value from iteration i. This allows constraints to be reused in subsequent explanations but with different relaxation values.

The multi-point relaxation space and weighted optimization allow tailoring explanations to specific user preferences or application requirements. By blocking specific constraints or relaxation values, the method ensures diversity in counterfactual explanations.

To speed up the performance in each iteration we use a warm start with the previous iteration's solution.

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.

CHAPTER 4

Implementation

In the implementation chapter, we begin by discussing the adaptation of an existing TLSP constraint programming model to support explanation generation. This process involves integrating features that facilitate the identification of infeasibility and the generation of explanations. In addition to the constraint programming model, we introduce a user-friendly GUI that allows users to configure and interact with the explanation generation process. The GUI enables users to customize and configure different types of explanations, input TLSP instances, and review the generated explanations, making the system more accessible and intuitive.

4.1 Adapting TLSP CP model for explanations

All of the approaches employed in this thesis utilize exact methods for generating explanations. Therefore, we adapt a pre-existing constraint programming (CP) model of the TLSP [GMM19] to facilitate the generation of explanations. As detailed in the previous sections, we use this model in two modeling languages: MiniZinc [NSB$^+$07] and OR-Tools [PF].

4.1.1 Adaptation of the model in MiniZinc

The adaptation of the TLSP CP model [GMM19] in MiniZinc to support the generation of explanations for the methodologies that we use is straightforward. The reason behind this practicality being that we use this model only with explainers offered by MiniZinc, therefore the approaches for explanations are built-in. The only modification that we need is to add expression annotations to the foreground constraints introduced in Table 3.1. We only need to specify the names of the constraints, and the variables that we want to track. The following listings (4.1 - 4.4) show how this is done using the model from [GMM19], modified with annotations.

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

```
constraint :: "ResourceGroupRequirements"
forall (g in ResourceGroups)(
  if g in modeDependantResourceGroups then
    forall(t in Tasks where isJob(t))(
      % Optimization: Requirement depends only on mode => ignore other
          tasks of the job, no need to compute a max
      (sum(e in ResourceClasses where resourceGrouping[e] ==
          g)(resourceClassesJobs[e,t]) == requiredResourceClasses[t,
          g, modeAssignment[t]])::
          "ResourceGroupRequirements:t=\(t),g=\(g) "
    )
  elseif g in taskDependantResourceGroups then
    forall(t in Tasks where isJob(t))(
      % Optimization: Requirement depends only on task => set
          modeAssignment[t] to 1
      (sum(e in ResourceClasses where resourceGrouping[e] ==
          g)(resourceClassesJobs[e,t]) == max(t2 in Tasks where
          familyAssignment[t] ==
          familyAssignment[t2])(requiredResourceClasses[t2, g, 1] *
          bool2int(taskJobs[t2] == t)))::
          "ResourceGroupRequirements:t=\(t),g=\(g) "
    )
  else
    forall(t in Tasks where isJob(t))(
      % Complex case: For a job t, assign the maximum requirement over
          all tasks t2 in the same family with job assignment t
      (sum(e in ResourceClasses where resourceGrouping[e] ==
          g)(resourceClassesJobs[e,t]) == max(t2 in Tasks where
          familyAssignment[t] ==
          familyAssignment[t2])(requiredResourceClasses[t2, g,
          modeAssignment[t]] * bool2int(taskJobs[t2] == t)))::
          "ResourceGroupRequirements:t=\(t),g=\(g) "
    )
  endif
);
```

Listing 4.1: Resource Explanations

```
constraint :: "JobDuration"
forall (j in Tasks where isJob(j)) (
  forall(m in availableModes[j])(
    (modeAssignment[j] == m -> (endTime[j] - startTime[j]) >=
        calculateJobDurationScaled(j, m)) ::
        "JobDuration:j=\(j),m=\(m)"
  )
);
```


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 WIEN Your knowledge hub

Listing 4.2: Temporal Explanations

```
constraint :: "SingleResourceAssignment"
forall(e in ResourceClasses)(
  cumulative(startTime, [jobDuration(t) | t in Tasks],
      [resourceClassesJobs[e, t] | t in Tasks],
      resourceClassQuantities[e]):: "SingleResourceAssignment:e=\(e) "
);
```

Listing 4.3: Single resource assignment explanations

```
constraint :: "ModeAvailability"
forall(t in Tasks)(
  (modeAssignment[taskJobs[t]] in availableModes[t]) ::
      "ModeAvailability:t=\(t)"
);


constraint :: "JobPrecedence"
forall(t in Tasks, t2 in prerequisiteTasks[t])(
  (taskJobs[t] == taskJobs[t2] \/ endTime[taskJobs[t2]] <=
      startTime[taskJobs[t]]) :: "JobPrecedence:t1=\(t),t2=\(t2)"
);

constraint :: "LinkedJobs"
forall(l in LinkedSets)(
  forall(t,t2 in linkedTasks[l] where t < t2)(
    forall(r in ResourceClasses where resourceGrouping[r] in {1})(
      resourceClassesJobs[r, taskJobs[t]] == resourceClassesJobs[r,
          taskJobs[t2]] :: "LinkedJobs:l=\(l),t1=\(t),t2=\(t2),r=\(r)"
    )
  )
);

constraint :: "FixedTasks"
forall(f in FixedSets) (
  all_equal([taskJobs[t] | t in fixedSets[f]]) :: "FixedTasks:f=\(f)"
);
```

Listing 4.4: Dependency Explanations

4.1.2 Adaptation of the model in OR-Tools

In this adapted model, we utilize Boolean variables to represent each foreground constraint. The value of each such variable indicates whether the constraint is enforced or not. These variables play a similar role to the *clause-selector variables* described in [LS08a]. In

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 WIEN Your knowledge hub

OR-Tools, these variables are referred to as *assumption* variables and for each foreground constraint f we add its assumption variable a_f. We will use notation $\mathcal{A}$ to denote the set of all assumption variables. When the explainer supports a multi-point relaxation space, for each foreground constraint $f \in \mathcal{F}$ mentioned in Table 3.1 we additionally add an integer variable $\tilde{b}_f$ that represents the relaxed constraint bounds, which would yield a satisfiable instance of the problem. When we add these integer variables, we also need to add the relation between these variables and the assumption variable a_f of the corresponding foreground constraint to the model:

$$\neg a_f \iff \tilde{b}_f < b_f \text{ where } \tilde{b}_f \in (max(b_f - X_f, 0), b_f)$$
$$\neg a_f \iff \tilde{b}_f > b_f \text{ where } \tilde{b}_f \in (b_f, b_f + X_f)$$

where b_f represents the original constraint bounds, and X_f represents the user-configurable variable mentioned in Section 3.2 which we use to control the maximum amount of relaxation for the constraint.
We will go through each of the explanation categories and their respective foreground constraint listed in Table 3.1.

Resource explanations

Resource requirements, Employee requirements, Equipment requirements
We add an assumption variable for each resource group $g \in R^*$ and each task $t \in A^*$, we denote this by rq_{tg}. In addition, for each such assumption variable, we also add the integer variable r^R_{tg} for controlling the relaxation of the bound of the constraint of the resource requirements for the task. We define these variable's values as:

$$rq_{tg} = \begin{cases} 1 & \text{the task's } t \text{ requirements for resource group } g \text{ can be, satisfied as given.} \\ 0 & \text{the task's } t \text{ requirements for resource group } g \text{ as given. cannot be satisfied. The instance can be satisfied if we reduce the requirements of the resource group to } value(r^R_{tg}) \end{cases}$$

Whereas, the values of the integer variable r^R_{tg} are explained in 3.2:

$$r^R_{tg} \in (max(r_{tg} - X_g, 0), r_{tg}),$$

We add the necessary relation between the assumption variable and the constraint:

$$\forall g \in R^*, \forall t \in A^* : rq_{tg} \iff r_{tg} = r^R_{tg}$$

where r_{tg} is the task resource requirements, which must be taken from the set of available resources $R_{t,g} \subseteq R_g$.

We use these assumption variables when we want our explanations to identify the resource requirements causing the infeasibility, as well as the amount of units that are missing.

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

Temporal explanations

Release Date We add an assumption variable for each task $t \in A^*$, denoting it by α'_t, and the integer variable α^R_t for controlling the relaxation of the constraint bounds (the release date). We define these variables as:

$$\alpha'_t = \begin{cases} 1 & \text{the task } t \text{ can be scheduled within the given release, date.} \\ 0 & \text{the task } t \text{ cannot be scheduled within its deadline. The instance can be satisfied if we modify its release date to } value(\alpha^R_t). \end{cases}$$

Whereas, the values of the integer variable α^R_t are explained in 3.2:

$$\alpha^R_t \in (max(\alpha_t - X_\alpha, 0), \alpha_t).$$

We add the necessary relation between the assumption variable and the constraint:

$$\forall t \in A^* : \alpha'_t \iff \alpha_t = \alpha^R_t$$

Deadline We add an assumption variable for each task $t \in A^*$, denoting it by ω'_t, and the integer variable ω^R_t for controlling the relaxation of the constraint bounds (the deadline). We define these variables:

$$\omega'_t = \begin{cases} 1 & \text{the task } t \text{ can be scheduled within the given deadline,} \\ 0 & \text{the task } t \text{ cannot be scheduled within its deadline. The instance can be satisfied if we postpone its deadline for } (value(\omega^R_t) \text{ - } \omega_t) \text{ units.} \end{cases}$$

Whereas, the values of the integer variable ω^R_t are explained in 3.2:

$$\omega^R_t \in (\omega_t, \omega_t + X_\omega).$$

We add the necessary relation between the assumption variable and the constraint:

$$\forall t \in A^* : \omega'_t \iff \omega_t = \omega^R_t$$

Single assignment explanations

We add an assumption variable for each resource group $g \in R^*$ and each task $t \in A^*$, we denote it with sa_{gt}. We define this variable:

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

$$sa_{gt} = \begin{cases} 1 & \text{the task's } t \text{ requirements for resource group } g \text{ can be, satisfied} \\ 0 & \text{the task's } t \text{ requirements for resource group } g \text{ cannot. be satisfied. The instance can be satisfied if we remove these requirements or temporal constraints.} \end{cases}$$

A false $sa_{gt} = 0$ indicates that no time interval within the task t's start and end times satisfies both the resource group g's requirements and the requirements of all other tasks for this resource group within the same interval.

To enforce this, we define the necessary relationship between the assumption variable and the constraints using *optional intervals* and the *cumulative* feature from OR-Tools. The *cumulative* constraint enforces the single assignment as follows:

$$\forall g \in R^* : cumulative(|R_g|, [\forall t \in A^* : iv_{gt}])$$

where $|R_g|$ represents the total available units for resource group g, and iv_{gt} is defined as an optional time interval with: start time α_t, duration d_t, deadline ω_t, and its assumption variable sa_{gt}.

Dependency explanations

Mode Given the set of possible relaxed modes $M^R \subseteq M$, we add an assumption variable for each task $t \in A^*$,we denote it by m'_t, We also add the integer variable m_t^R which represent the mode that needs to be added to the set of available modes for task t. We define these variables as:

$$m'_t = \begin{cases} 1 & \text{the task } t \text{ can be scheduled using the available set of, modes.} \\ 0 & \text{the task } t \text{ cannot be scheduled using the available. set of modes. The instance can be satisfied if we can operate task } t \text{ under mode } m \text{ which is not in the given set of available modes for current task} \end{cases}$$

Whereas, the values of the integer variable m_t^R are explained in 3.2:

$$m_t^R \in M^R \backslash M_t.$$

We add the necessary relation between the assumption variable and the constraint:

$$\forall j \in J, \forall t \in \dot{A}_j, \forall m \in M^R \backslash M_t :$$

$$m'_t \iff \dot{m}_j = m_t^R$$

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

Task precedence We add an assumption variable for each set of precedence tasks (t_i, t_j), denoting it by $tp_{t_i t_j}$. We define these variables:

$$tp_{t_i t_j} = \begin{cases} 1 & \text{task } t_i \text{ can be scheduled before task } t_j. \\ 0 & \text{task } t_i \text{ cannot be scheduled before task } t_j. \text{ The instance can be satisfied if we remove this precedence constraint between these two tasks.} \end{cases},$$

We add the necessary relation between the assumption variable and the constraint:

$$\forall t_i \in A^*, \forall t_j \in P_{t_i} : tp_{t_i t_j} \iff \omega_{t_j} \leq \alpha_{t_i}$$

where P_{t_i} represents the list of direct predecessors of t_i, α_{t_i} and ω_{t_i} represent the release and the deadline.

Linked jobs We add an assumption variable for each set of linked tasks (L_p) for each project p, denoting it by lj_{L_p}. We define these variables as:

$$lj_{L_p} = \begin{cases} 1 & \text{all tasks } pa, pb, \text{ where } (pa, pb) \in L_p \text{ can all be scheduled with the same resources.} \\ 0 & \text{at least one of tasks } pa \text{ cannot be scheduled with the same resources as the other tasks } pb, \text{ where } (pa, pb) \in L_p. \text{ The instance can be satisfied if we remove this set of linked tasks from the instance.} \end{cases},$$

We add the necessary relation between the assumption variable, assigned jobs and the linked jobs constraint:

$$\forall p \in P, R_g \in L^{rg}, (t_1, t_2) \in L_p, t_1 \in \dot{A_{j_1}}, t_2 \in \dot{A_{j_2}}, : \\ lj_{L_p} \iff \dot{R_{j_1 g}} = \dot{R_{j_1 g}}$$

Fixed Jobs/Projects explanations

Fixed jobs We add an assumption variable for each fixed job $j \in J^0$ denoting it by fj_j. We define three relaxation points for this variable:

- **I**: Job stays fixed.
- **II**: Job is unfixed and nothing else changes. We check if simply removing the fixed job constraint is enough for restoring the instance's satisfiability
- **III**: Job is unfixed and additional changes to foreground constraints of unfixed jobs apply. We check if we also need to relax other foreground constraint of the unfixed jobs (including the current job) to achieve the satisfiability of the 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.
TU WIEN Bibliothek Your knowledge hub

Although there are three relaxation points, the explainer requires choosing between the second and third points before execution. As a result, this constraint effectively has two relaxation points at runtime. First let's define the set of all assumption variables of the foreground constraints that were mentioned up to this point as $\tilde{\mathcal{A}}$. For the second and third relaxation points we need to add the relations between fj_j and all of the assumption variables $(A^{t_1}_{fc_1}, ..., A^{t_n}_{fc_m})$, where $A^{t_i}_{fc_k} \in \tilde{\mathcal{A}}$, fc_k denotes the foreground constraint, and t_i corresponding task, $t_i \in \dot{A}_j$. Assumptions variables $A^{t_i}_{fc_k}$ can be false only if fj_j is false:

$$fj_j \Rightarrow n \times |\tilde{\mathcal{A}}| = \sum_{t \in \dot{A}_j} \sum_{c \in \tilde{\mathcal{A}}} A^t_c$$

Building upon this, the values of these variables are formally defined as follows:

$$fj_j = \begin{cases} 1^{(\mathbf{I})} & \text{job } j \text{ can be scheduled with its fixed constraints} \\ 0^{(\mathbf{II})} & \text{job } j \text{ cannot be scheduled with its fixed constraints. The instance can be satisfied if we unfix this job} \\ 0^{(\mathbf{III})} & \text{job } j \text{ cannot be scheduled with its fixed constraints. The instance. can be satisfied if we unfix this job, and in addition for each } A^{t_i}_{fc_k} \text{ if } value(A^{t_i}_{fc_k}) = 0 \text{, remove or relax its corresponding foreground constraints } fc_k \text{ for task } t_i \end{cases}$$

Fixed projects We use the same approach as in the fixed jobs constraint, but instead of jobs we use projects. We add an assumption variable for each fixed project p denoting it by fp_p. We define three relaxation points for this variable:

- **I**: Project stays fixed.
- **II**: Project is unfixed and nothing else changes. We check if simply removing the fixed project constraint is enough for restoring the instance's satisfiability
- **III**: Project is unfixed and additional changes to foreground constraints apply. We check if we also need to relax other foreground constraint of the unfixed projects (including current project) to achieve the satisfaction of the instance.

For the second relaxation point we need to add the relations between fp_p and all of the assumption variables $(A^{t_1}_{fc_1}, ..., A^{t_n}_{fc_m})$, where $A^{t_i}_{fc_k} \in \tilde{\mathcal{A}}$, fc_k denotes the foreground constraint, and t_i corresponding task, $t_i \in A_p$. Assumptions variables $A^{t_i}_{fc_k}$ can be false only if fp_p is false:

$$fp_p \Rightarrow n \times |\tilde{\mathcal{A}}| = \sum_{t \in A_p} \sum_{c \in \tilde{\mathcal{A}}} A^t_c$$

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

Building upon this, the values of these variables are formally defined as follows:

$$fp_p = \begin{cases} 1 & \text{project } p \text{ can be scheduled with its fixed constraints.} \\ 0^{(\mathbf{II})} & \text{project } p \text{ cannot be scheduled with its fixed constraints. The instance can be satisfied if we unfix this project} \\ 0^{(\mathbf{III})} & \text{project } p \text{ cannot be scheduled with its fixed constraints. The instance. can be satisfied if we unfix this project, and in addition for each } A^{t_i}_{fc_k} \text{ if } value(A^{t_i}_{fc_k}) = 0\text{, remove or relax its corresponding foreground constraints } fc_k \text{ for task } t_i \end{cases}$$

4.2 User Interface

The user interface (UI) was integrated into an existing TLSP solver [DGJ+23] to enhance its functionality by providing dynamic control over the infeasibility explainer. The UI functionality (without the explainer part) is thoroughly explained in [DGJ+23]. The UI is designed with user-friendliness and configurability in mind, enabling users to customize the explainer's settings to suit their specific scheduling requirements. Through the interface, users can adjust various parameters, such as:

- the type of explainer, conflict-based or counterfactual.
- the categories of explanations (foreground constraints) to be considered in the explanation generation process. Examples are shown in Fig. 4.3 and Fig. 4.4.
- for the counterfactual explainer, the relaxation bounds for resource and temporal constraints, as well as preference weights for all categories of explanations.

Furthermore, the UI supports the creation of multiple configurations, allowing users to experiment with different settings simultaneously (see Fig. 4.2). These configurations can be run all in parallel, thereby reducing the overall time required to obtain feasible scheduling solutions (see Fig. 4.1). They can also be saved and loaded for future use, providing a convenient way to store and reuse specific settings. Explanations are shown as soon as they are found in the bottom window of the UI. If multiple configurations are run, the explanations are displayed in separate tabs, making it easy to compare and analyze the results.

The integration of the UI not only simplifies the process of configuring the explainer but also enhances the user experience by streamlining the workflow within the TLSP solver. This integration ensures that the explainer is accessible to a wider audience, including users with limited experience in constraint programming or scheduling problems.

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

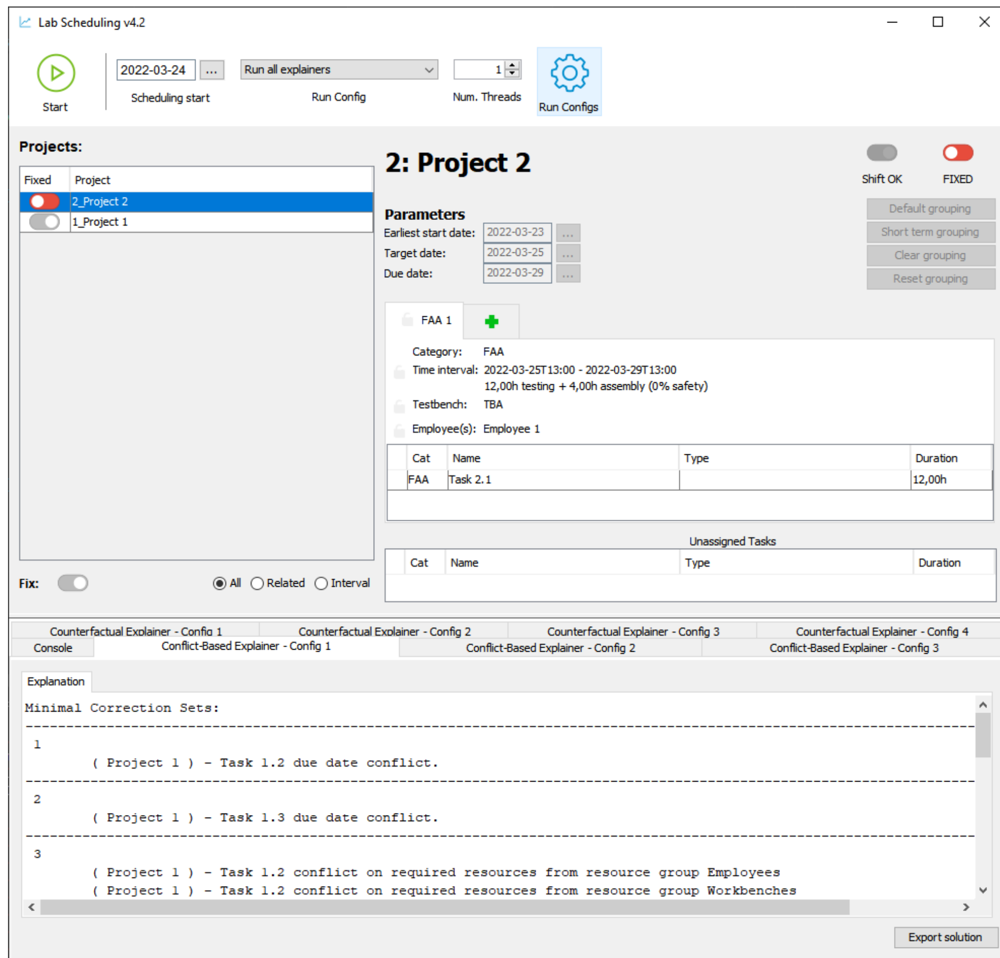


Figure 4.1: Configuring and running multiple TLSP explainer configurations.

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

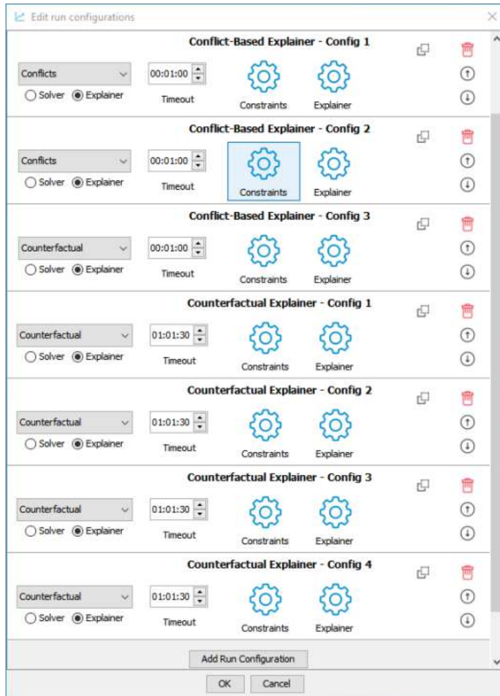


Figure 4.2: Multiple explainer configurations.

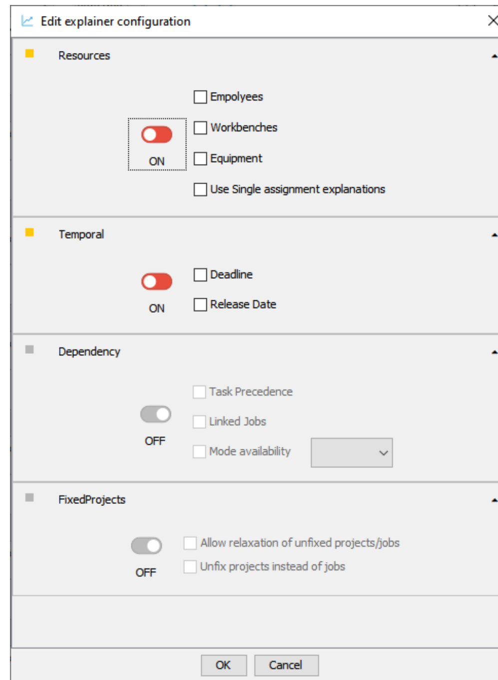


Figure 4.3: Conflict-based explainer configuration.

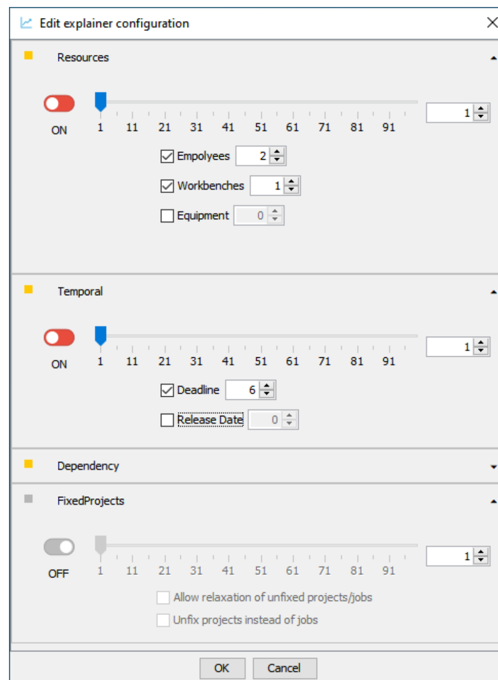


Figure 4.4: Counterfactual explainer configuration.

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.

CHAPTER 5

Experimental evaluation

In this chapter, we assess the performance of the proposed infeasibility explainer from multiple perspectives. The chapter is organized into three sections. First, we present qualitative results that illustrate the interpretability, clarity, and overall usability of the explanations generated by the explainer. Next, we evaluate the computational efficiency of the explainer using a series of benchmark TLSP instances as well as real-world instances. Finally, we review the performance of the explainer in a real-world industrial setting.

5.1 Demonstration

5.1.1 Problem Setup

We consider an example instance I of TLSP. It contains two projects, a total of four tasks, and tasks within each project belong to the same family. The following tables summarize the initial setup for a feasible instance of the TLSP:

- Table 5.1 shows the distribution of tasks across the two projects.
- Table 5.2 shows modes of operation, which affect task duration (modified to $task_duration * mode_duration_factor$). All tasks can operate under every mode shown in 5.2.
- Table 5.3 shows temporal requirements for each task.
- Table 5.4 lists resource requirements for each task for two groups of requirements, that of *employees*, and *workbenches*, as well as their respective available units.

As mentioned above, the shown instance is feasible, one simple solution is shown in Figure 5.1. The solution shows that each task is assigned to a separate job, resulting in four

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

Project ID	Tasks
Project 1	Task 1.1
	Task 1.2
	Task 1.3
Project 2	Task 2.1

Table 5.1: Projects and tasks

Mode ID	Name	Duration Factor
m_1	*Shift mode*	2/3
m_2	*Single employee mode*	1

Table 5.2: Mode requirements

Task ID	Duration	Release Date	Deadline	Family Setup Duration
Task 1.1	3	0	6	1
Task 1.2	3	0	8	1
Task 1.3	3	0	6	1
Task 2.1	3	0	10	1

Table 5.3: Temporal requirements

jobs each with one task. *Job* 1 and *Job* 3 do not share resources therefore are scheduled to be run from the start in parallel. *Job* 2 shares resources with *Job* 3, therefore it is scheduled to start after *Job* 3 is finished. *Job* 2 and *Job* 3 operate on mode m_1, therefore their total duration is 3 units of time (2 task duration + 1 job setup), whereas *Job* 1, and *Job* 4, operate on mode m_2, therefore its total duration is 4 units of time (3 task duration + 1 job setup). The solution is feasible, as all tasks are scheduled to be finished before their respective deadlines, and all resource constraints are satisfied.
We will use modified infeasible versions of this instance to demonstrate the capabilities of explainers, to show how the explainers can identify the constraints that cause the infeasibility. We will show two cases on how infeasibility can happen:

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

Task ID	Mode	Employees	Workbenches
Task 1.1	m_1	2	1
	m_2	1	1
	Available Resources	E_1, E_2, E_3	Wb_A, Wb_B, Wb_C
Task 1.2	m_1	2	1
	m_2	1	1
	Available Resources	E_1, E_2	Wb_A
Task 1.3	m_1	2	1
	m_2	1	1
	Available Resources	E_1, E_2	Wb_A
Task 2.1	m_1	2	1
	m_2	1	1
	Available Resources	E_1, E_2, E_3	Wb_A, Wb_B, Wb_C

Table 5.4: Resource requirements

- **Instance** I^L: We lose the feasibility of the instance I if we introduce a linked job constraint between *Task* 1.1 and *Task* 1.2, on resource group *Employees* ($L^{rg} = \{\text{Employees}\}$) :

$$L_1 = \{(Task\ 1.1,\ Task\ 1.2)\}$$

  This constraint enforces that *Task* 1.1 and *Task* 1.2 must use the same set of resources of group *Employees*. The issue arises because the only set of resources that are common for *Task* 1.1 and *Task* 1.2, are the ones that are also the only available resources for *Task* 1.3. Therefore, only two tasks out of three can be scheduled at the same time, leaving the third task to be done subsequently. The third task will not have enough time to be completed before the deadline, therefore the instance becomes infeasible.

- **Instance** I^F: We lose the feasibility of the instance I if we fix *Project* 2 to the schedule shown in Figure 5.2.

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

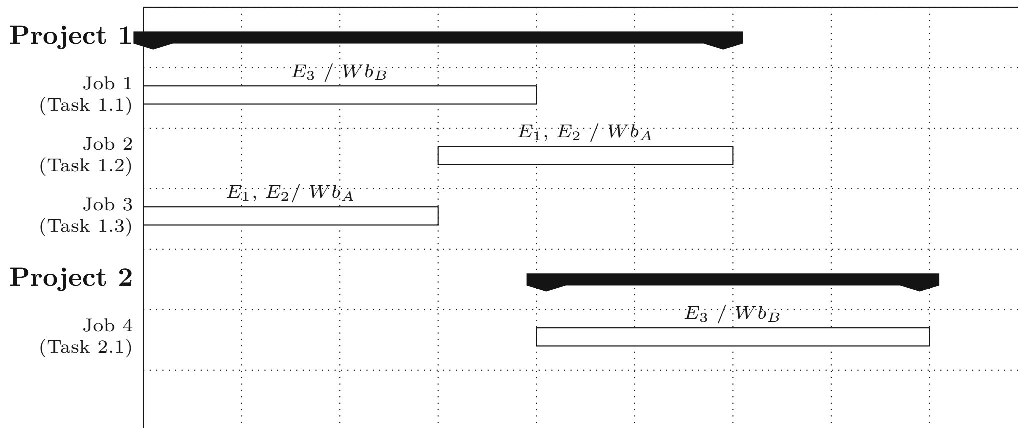


Figure 5.1: Solution to the TLSP instance

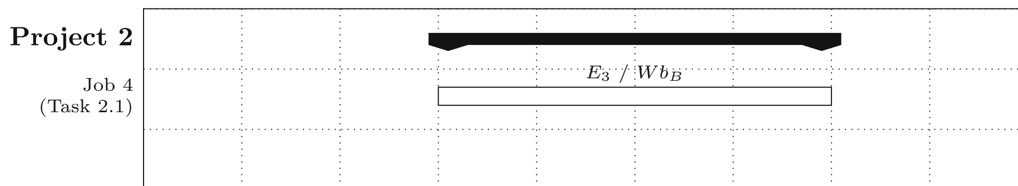


Figure 5.2: Fixed schedule for *Project* 2

We can explain the listed infeasibility in many ways, depending on what constraints we are looking for:

- We can explain the infeasibility as the result of **resource constraints**, that the resources are not available for all tasks to be scheduled at the same time.
- We can explain the infeasibility as the result of **linked tasks** constraint between the tasks *Task* 1.1 and *Task* 2.
- We can explain the infeasibility as the result of **temporal constraints**, that the tasks do not have enough time to be scheduled while satisfying all other constraints.
- We can explain the infeasibility as the result of **fixed projects**(*Project* 2), which restricts other tasks from using the same resources.

5.1.2 Results

Explainer using findMUS

The explainer defined in Section 3.3.2, generates the explanations shown in Table 5.5 when run on instance I^L. It identifies a single MUS, which highlights conflicts involving resources (1, 2), the duration of Task 1.1, and the resource group *Employees* for all tasks.

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

<table>
<tr><th>Sequence</th><th>Type</th><th>Explanation</th></tr>
<tr><td>1</td><td>MUS</td><td>
- SingleResourceAssignment:e=1
- SingleResourceAssignment:e=2
- JobDuration:j=1,m=1
- JobDuration:j=1,m=2
- ResourceGroupRequirements:t=1,g=1
- ResourceGroupRequirements:t=2,g=1
- ResourceGroupRequirements:t=3,g=1
Mapping:
<table>
<tr><th>Case</th><th>Type</th><th>Id</th></tr>
<tr><td>e=1</td><td>Resource</td><td>1</td></tr>
<tr><td>e=2</td><td>Resource</td><td>2</td></tr>
<tr><td>j=1</td><td>Task</td><td>1</td></tr>
<tr><td>t=1</td><td>Task</td><td>1</td></tr>
<tr><td>t=2</td><td>Task</td><td>2</td></tr>
<tr><td>t=3</td><td>Task</td><td>3</td></tr>
<tr><td>m=1</td><td>Mode</td><td>1</td></tr>
<tr><td>m=2</td><td>Mode</td><td>2</td></tr>
<tr><td>g=1</td><td>Resource Group</td><td>Employees</td></tr>
</table>
</td></tr>
</table>

Table 5.5: Conflicts generated using explainer defined in Section 3.3.2

While this MUS provides valuable insights into the sources of conflict, it fails to capture one of the primary causes of infeasibility: the *Linked jobs* constraint.

We run the explainer using the model defined in Section 4.1.1 with no filters on the foreground constraints, and with no limit on the number of the found MUSes (using the $-a$ findMUS flag).

We are not able to run it on instance I^F as the model does not support *Fixed Project* explanations.

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 WIEN Your knowledge hub

Explainer using CAMUS

We will consider three different configurations, evaluated on different infeasible instances, summarized as follows:

Config 1 on Instance I^L: Enable all possible foreground constraints, except those that are irrelevant to the current instance, such as release dates (all tasks start at time 0), mode availability (all modes are available for all tasks), and fixed projects (there are no fixed projects). No preference is applied to explanation types. (See Fig. 5.3)

Config 2 on Instance I^L: Similar to **Config 1**, but we switch the *Resource requirement* explanations with *Single assignment* explanations. (See Fig. 5.4)

Config 3 on Instance I^F: Enable only temporal explanations, and fixed projects. (See Fig. 5.5)

The explainer defined in Section 3.3.3 using **Config 1** on instance I^L generates the explanations shown in Table 5.6. As mentioned in the section, the explainer first generates all possible MCSes, and then computes all MUSes. In the table, we show the generated MCSes, labeled with sequence numbers 1 through 6.

The explainer identifies the *Linked jobs* conflict in the first MCS. In the second, fourth, and fifth MCSes, it detects temporal conflicts, whereas the remaining MCSes highlight resource requirement conflicts.

After identifying the MCSes, the explainer extracts MUSes from them, which we present with sequence numbers 1 and 2. From both MUSes, we can see that all tasks are in conflict with each other, and that the conflict is caused by the *Linked jobs* constraint on the resource group *Employees*, which cannot be satisfied due to deadline constraints.

In Fig. 5.6, we present the feasible solutions obtained by applying the MCSes listed in Table 5.6. The first row depicts the solution derived from applying MCS #2, the second row corresponds to MCS #3, and so forth. The solution obtained by applying MCS #1 is omitted, as it is identical to the one shown in Fig. 5.1.

Next, using **Config 2** we replace the *Resource requirement* explanations with *Single assignment* explanations and run the explainer again on instance I^L. The results are shown in Table 5.7. The explainer identifies all previously detected conflicts, modifies MCS 6, additionally finds MCSes 7, and 8.

Let's consider MCS 6. Resolving this conflict requires removing the resource requirements for $Task$ 1 from the resource group $Employees$. However, if we only remove the resource requirement, the instance would still be infeasible because the *Linked jobs* constraint between $Task$ 1 and $Task$ 2 would remain violated. To restore feasibility, we would also need to remove $Task$ 1 from the linked set.

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

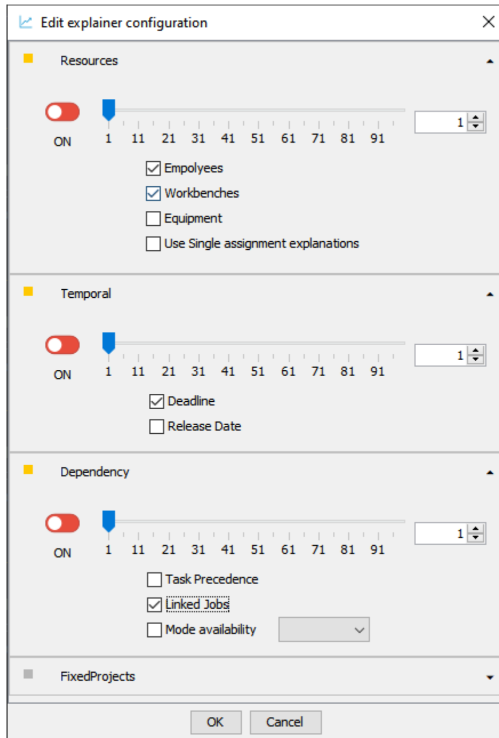


Figure 5.3: Explainer configuration: **Config 1**.

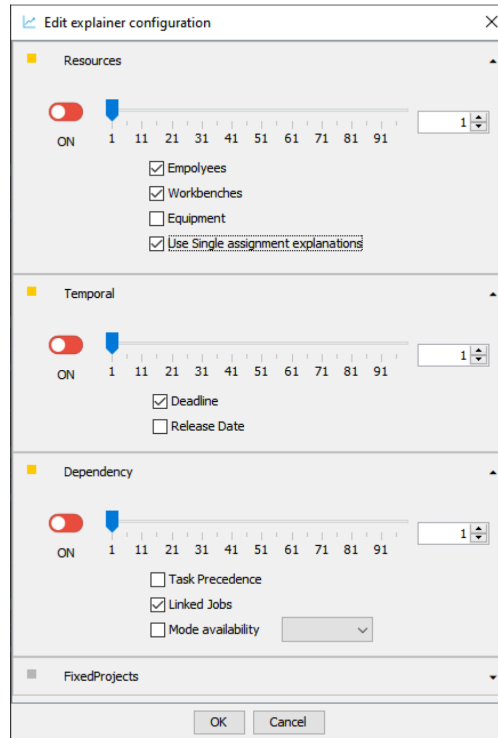


Figure 5.4: Explainer configuration: **Config 2**.

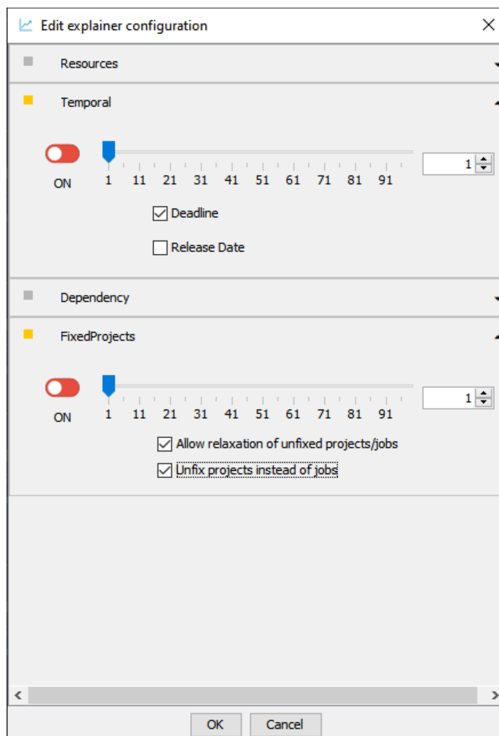


Figure 5.5: Explainer configuration: **Config 3**.

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

#	Type	Explanation
1	MCS	Remove LinkedTasks constraint on resource group Employees for the following tasks: Task 1.1, Task 1.2
2	MCS	Remove Task 1.3 deadline constraint
3	MCS	Remove Task 1.3 constraint on required resources from resource group Employees
4	MCS	Remove Task 1.1 deadline constraint
5	MCS	Remove Task 1.2 deadline constraint
6	MCS	- Remove Task 1.1 constraint on required resources from resource group Employees - Remove Task 1.2 constraint on required resources from resource group Employees
1	MUS	- LinkedTasks conflict on resource group Employees for the following tasks: Task 1.1, Task 1.2 - Task 1.2, 3 conflict on required resources from resource group Employees - Task 1.1, 2, 3 deadline conflict
2	MUS	- LinkedTasks conflict on resource group Employees for the following tasks: Task 1.1, Task 1.2 - Task 1.1, 3 conflict on required resources from resource group Employees - Task 1.1, 2, 3 deadline conflict

Table 5.6: Conflicts generated using explainer defined in Section 3.3.3 on instance I^L using configuration **Config 1**

The same applies to MCS 7 and MCS 8, which require removing the resource requirements for $Task$ 2 from the resource group *Employees*. If the explainer instead suggested removing only the resource requirements of $Task$ 2 for the resource group $Workbenches$, then removing it from the linked set would not be necessary. That is why the *Single assignment* explanations do not directly resolve the infeasibility; an additional check is needed to ensure they are sufficient to restore feasibility.

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

Lastly, we run the explainer using **Config 3** on instance I^F. This configuration is used to demonstrate how the *Fixed Project* explanations work. The results are shown in Table 5.8. The explainer identifies the *Fixed Project* conflict in the third MCS.

Counterfactual Explainer

For this experiment, the explainer uses an objective that combines all of the objectives mentioned in Section 3.4. This explainer uses blocking technique **Blocking B2**, as explained in the mentioned section.
We will run this explainer under different settings to highlight some of its key features. Specifically, we will consider four different configurations, summarized as follows:

> **Config 1:** Enable all possible foreground constraints, except those that are irrelevant to the current instance, such as release dates (all tasks start at time 0), mode availability (all modes are available for all tasks), and fixed projects (there are no fixed projects). No preference is applied to explanation types. Configuration will be applied to instance I^L. (See Fig. 5.7)
>
> **Config 2:** Similar to **Config 1** with additional changes. Prevent the $Workbenches$ group from being relaxed, allow the *deadline* to be relaxed by up to 4 slots, and prevent dependency constraints from being relaxed. No preference is applied to explanation types. Configuration will be applied to instance I^L. (See Fig. 5.8)
>
> **Config 3:** Apply a preference order for explanations: first $resource$, then $temporal$, and finally *dependency*. Configuration will be applied to instance I^L. (See Fig. 5.9)
>
> **Config 4:** Prioritize only $temporal$ explanations, while maintaining no preference for other types of explanations. Configuration will be applied to instance I^L. (See Fig. 5.10)
>
> **Config 5:** Enable only *temporal* and *fixed project* explanations, use it on instance I^F. (See Fig. 5.9)

For each configuration, we will present the generated explanations and briefly discuss the results:

> **Config 1:** Explanations can be seen in Table 5.10. The first explanation generated for this configuration is that of the **Linked jobs** constraint between *Task 1.1* and *Task 1.2*, which is understandable, as this explanation suggests the fewest changes (as per **Objective O2**). Next, we see an explanation related to resource requirements not being fulfilled for *Task 1.3*, suggesting a relaxation of 2 units. The number of modifications is the same as in the third explanation, which suggests postponing the deadline by 2 slots. Since no preference is applied, the order of explanations 2 and 3 is not fixed. The 4th and 5th explanations suggest the same number of modifications, and the order of these explanations is not fixed 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 WIEN Bibliothek Your knowledge hub

They are suggested before the 6th explanation, which suggests a total change of 3 from the original bounds. The latter changes are distributed over two tasks, meaning two constraints are modified, therefore it is suggested last.

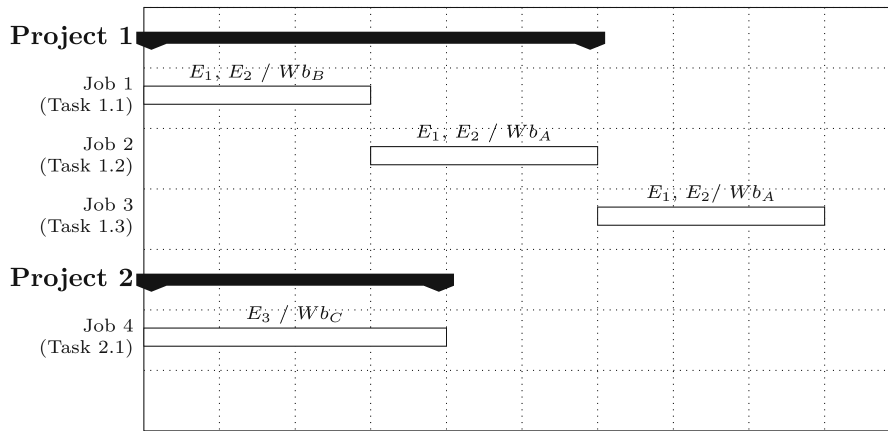
Config 2: Explanations can be seen in Table 5.11. This configuration removes several explanations that were generated in the previous configuration. We immediately see that the **Linked jobs** explanation is missing, as the explainer is now restricted from relaxing this constraint. Additionally, we no longer see explanations that relax the *Workbenches* resource group or allow deadlines to be relaxed by more than 4 slots (explanations 5 through 8 in Table 5.10). As mentioned in **Config 1**, the first and second explanations suggest the same number of modification. Therefore, their order has changed, and the second explanation from **Config 1** is now suggested first.

Config 3 & 4: Explanations can be seen in Table 5.11, where we show how the sequence of explanations changes when a preference is applied. In **Config 3**, following the applied preference, the explainer first suggests all **Resource** explanations, followed by **Temporal** explanations, and finally **Dependency** explanations. This order changes in **Config 4**, where the explainer first suggests **Temporal** explanations, as dictated by the applied preference. Since no preference is applied to the other two explanation types, the order of the last two explanations is determined by the amount of relaxation required.

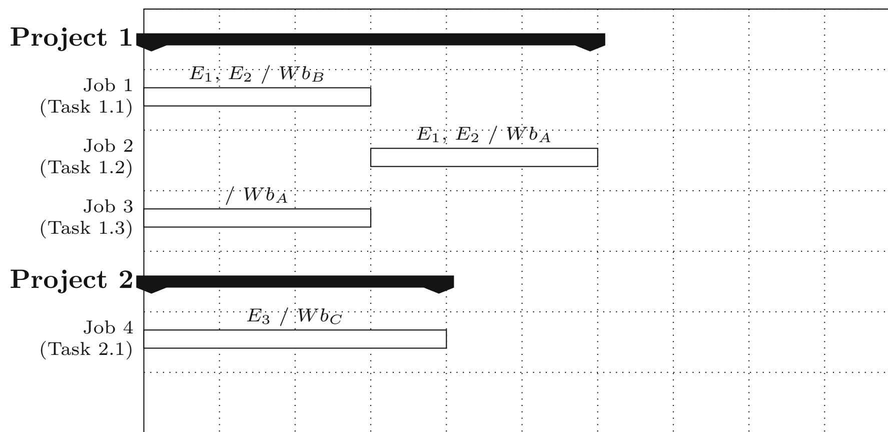
Config 5: Explanations can be seen in Table 5.13. We observe that the **Fixed projects** explanation is found and suggested first. This is expected, as the explainer is configured to prioritize this explanation type because it suggests only one change. Additionally, if the user does not want to unfix the project, then alternatives explanations/relaxations are suggested. The second explanation suggests relaxing the deadline for *Task 1.1* by 4 slots, and on the third explanation it is suggested to relax the deadline of *Task 1.3*.

In Fig. 5.11, we present the feasible solutions obtained by applying some of the MCSes listed in Table 5.10. The first row depicts the solution derived from applying MCS #3, the second row corresponds to MCS #5, and so forth. The solution obtained by applying MCS #1 is omitted, as it is identical to the one shown in Fig. 5.1.

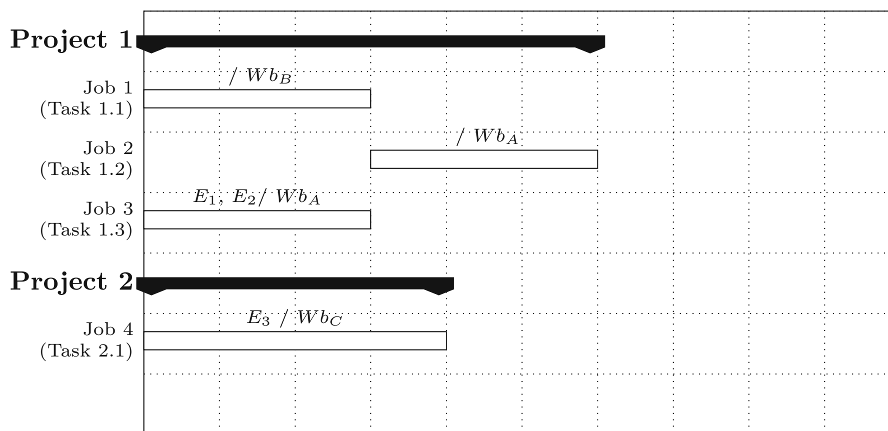
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) Feasibility through MCS #2



(b) Feasibility through MCS #3



(c) Feasibility through MCS #6

Figure 5.6: Feasible solutions to the TLSP instance by applying MCSes listed in Table 5.6

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

#	Type	Explanation
1	MCS	Remove Task 1.3 constraint on required resources from resource group *Employees*
2	MCS	Remove Task 1.3 deadline constraint
3	MCS	Remove Task 1.2 deadline constraint
4	MCS	Remove Task 1.1 deadline constraint
5	MCS	Remove LinkedTasks constraint on resource group *Employees* for the following tasks: Task 1.1, Task 1.2
6	MCS	Remove Task 1.1 constraint on required resources from resource group *Employees*
7	MCS	- Remove Task 1.2 constraint on required resources from resource group *Employees* - Remove Task 1.3 constraint on required resources from resource group *Workbenches*
8	MCS	- Remove Task 1.2 constraint on required resources from resource group *Employees* - Remove Task 1.2 constraint on required resources from resource group *Workbenches*
1	MUS	- LinkedTasks conflict on resource group *Employees* for the following tasks: Task 1.1, Task 1.2 - Task 1.1,2,3 conflict on required resources from resource group *Employees* - Task 1.2 conflict on required resources from resource group *Workbenches* - Task 1.1, 2, 3 deadline conflict
2	MUS	- Task 1.1, 3 conflict on required resources from resource group *Employees* - LinkedTasks conflict on resource group *Employees* for the following tasks: Task 1.1, Task 1.2 - Task 1.2, 3 conflict on required resources from resource group *Workbenches* - Task 1.1, 2, 3 deadline conflict

Table 5.7: Conflicts generated using explainer defined in Section 3.3.3 on instance I^L using configuration **Config 2**

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 WIEN Your knowledge hub

#	Type	Explanation
1	MCS	Remove Task 1.2 deadline constraint
2	MCS	Remove Task 1.3 deadline constraint
3	MCS	Unfix *Project 2*
1	MUS	- Remove Task 1.2 deadline constraint - Remove Task 1.3 deadline constraint - Unfix *Project 2*

Table 5.8: Conflicts generated using explainer defined in Section 3.3.3 on instance I^F using configuration **Config 3**

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

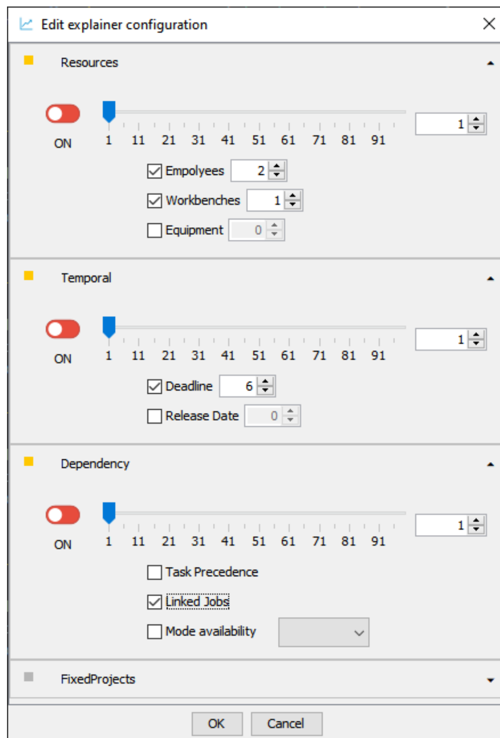


Figure 5.7: Explainer configuration: **Config 1**.

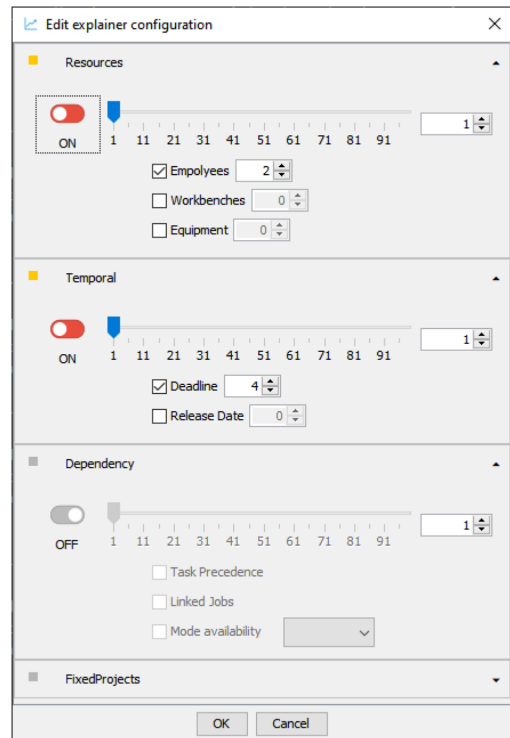


Figure 5.8: Explainer configuration: **Config 2**.

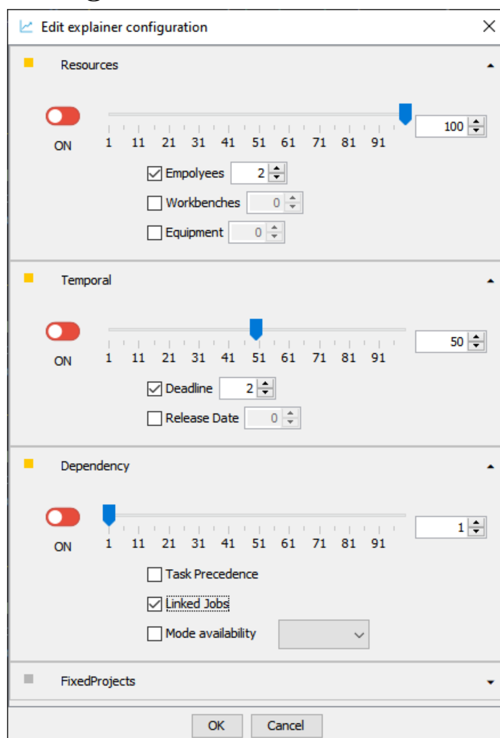


Figure 5.9: Explainer configuration: **Config 3**.

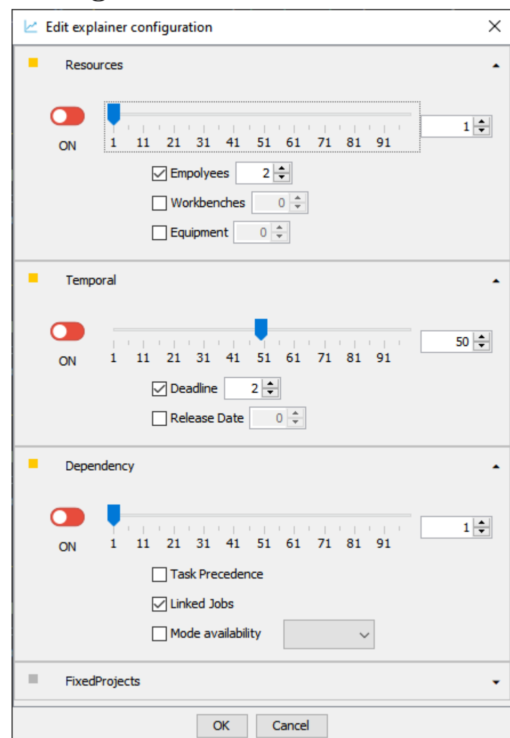


Figure 5.10: Explainer configuration: **Config 4**.

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 WIEN Your knowledge hub

Edit explainer configuration

Resources

Temporal

ON

1 11 21 31 41 51 61 71 81 91

1

Deadline 10

Release Date 0

Dependency

FixedProjects

ON

1 11 21 31 41 51 61 71 81 91

1

Allow relaxation of unfixed projects/jobs

Unfix projects instead of jobs

OK Cancel

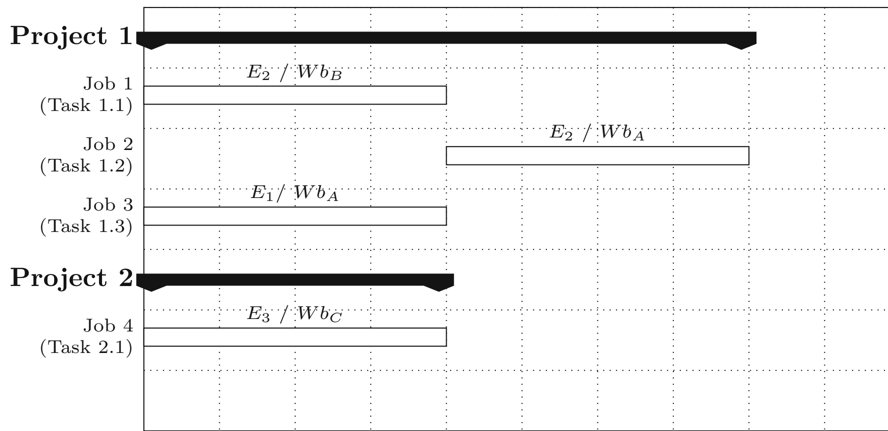
Table 5.9: Explainer configuration: **Config 5**.

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

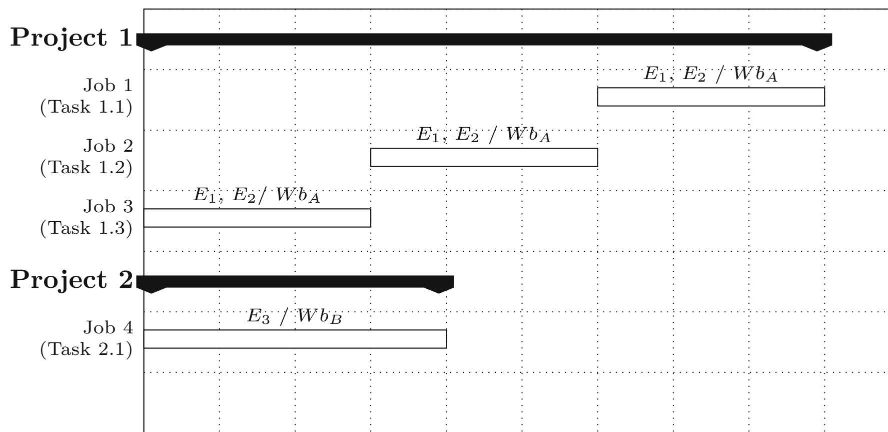
Sequence	Explanation
1	Remove LinkedTasks set on resource group Employees for the following tasks: Task 1.1, Task 1.2
2	Task 1.3: relax resource group requirements "Employees" for 2 units. (Mode m_1)
3	Task 1.2: postpone deadline for 2 slots
4	Task 1.3: postpone deadline for 5 slots
5	Task 1.1: postpone deadline for 5 slots
6	- Task 1.2: relax resource group requirements "Employees" for 2 unit. (Mode m_1) - Task 1.1: relax resource group requirements "Employees" for 1 unit. (Mode m_2)
7	- Task 1.1: postpone deadline for 4 slots - Task 1.2: relax resource group requirements "Workbenches" for 1 unit. (Mode m_2)
8	- Task 1.1: postpone deadline for 4 slots - Task 1.3: relax resource group requirements "Workbenches" for 1 unit. . (Mode m_2)
9	- Task 1.1: postpone deadline for 4 slots - Task 1.3: postpone deadline for 4 slots

Table 5.10: Counterfactual explanations generated using explainer defined in Section 3.4 and configuration **Config 1** shown in Fig.5.7

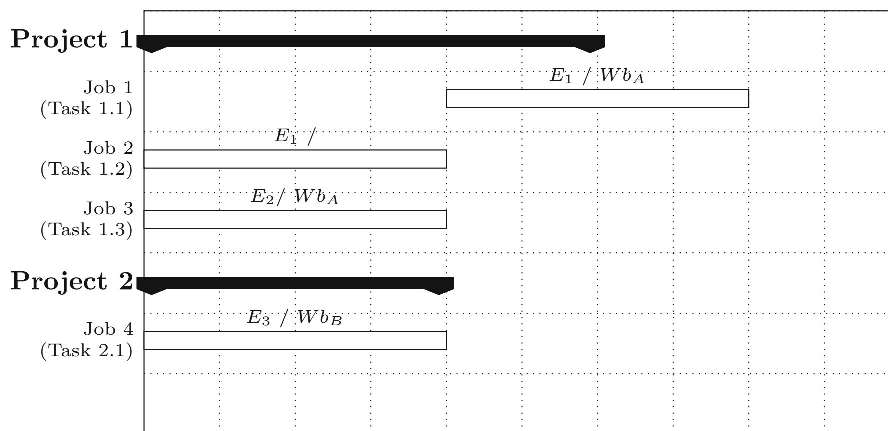
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) Feasibility through MCS #3



(b) Feasibility through MCS #5



(c) Feasibility through MCS #7

Figure 5.11: Feasible solutions to the TLSP instance by applying MCSes listed in Table 5.10

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

Sequence	Explanation
1	Task 1.2: postpone deadline for 2 slots
2	Task 1.3: relax resource group requirements "Employees" for 2 units
3	- Task 1.2: relax resource group requirements "Employees" for 2 unit. (Mode m_1) - Task 1.1: relax resource group requirements "Employees" for 1 unit. (Mode m_2)
4	- Task 1.1: postpone deadline for 4 slots - Task 1.3: postpone deadline for 4 slots

Table 5.11: Counterfactual explanations generated using explainer defined in Section 3.4 and configuration **Config 2** shown in Fig.5.8

Explanation	Config 3 Sequence	Config 4 Sequence
Task 1.3: relax resource group requirements "Employees" for 2 units	1	3
- Task 1.2: relax resource group requirements "Employees" for 2 unit. (Mode m_1) - Task 1.1: relax resource group requirements "Employees" for 1 unit. (Mode m_2)	2	4
Task 1.2: postpone deadline for 2 slots	3	1
Remove LinkedTasks set on resource group Employees for the following tasks: Task 1.1, Task 1.2	4	2

Table 5.12: Counterfactual explanations generated using explainer defined in Section 3.4 and configuration **Config 3 & 4** shown in Fig.5.9 and Fig.5.10

5.2 Performance and scalability evaluation

In this section, we evaluate the runtime efficiency as well as scalability of the proposed infeasibility explainer. Our evaluation is divided into two parts. The first part examines TLSP benchmark examples, a total of 10 randomly generated feasible instances of different sizes, where infeasibility is deliberately introduced by adding an unsatisfiable project. The second part assesses the performance on two real-world TLSP instances provided

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

Sequence	Explanation
1	Unfix *Project 2*
2	Task 1.2: postpone deadline for 4 slots
3	Task 1.3: postpone deadline for 6 slots

Table 5.13: Counterfactual explanations generated using explainer defined in Section 3.4 and configuration **Config 5** shown in Fig.5.9

by our industrial partner (in anonymized form). This analysis aims to highlight the scalability and practical applicability of our approach.

ID	Data Set	|P|	|A*|	|G|	Resources	h
000	General	5	16	5	20	88
005	General	10	89	6	126	88
010	General	20	185	7	10	88
015	General	40	408	5	16	174
021	General	15	151	8	165	174
025	General	30	379	5	64	174
030	General	60	616	5	311	174
035	General	20	307	7	54	520
048	General	60	735	5	241	520
051	General	60	867	8	272	782
Lab_1		66	904	11	582	240
Lab_2		75	919	13	519	240

Table 5.14: Instance details including instance ID, number of projects, number of tasks, resource groups, resource units, and timeslots

For each instance, all experiments were executed 10 times to account for runtime variability. We imposed a timeout of 24 hours for each run; if an explainer did not finish within this limit, the corresponding entry is marked with "–". We report the median and interquartile range (IQR) across these runs, as these statistics provide a more robust summary of performance than averages, especially in the presence of occasional outliers.

These controlled experiments allow us to measure the performance of the explainer under varying levels of complexity. Key metrics include:

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

- Time to first relaxation
  - Conflict-based explainer: time to the first MCS found.
  - Counterfactual explainer: time to the first counterfactual explanation.
- Total runtime
  - Conflict-based explainer: time to the conflict-based explanation.
  - Counterfactual explainer: time to explore all counterfactual explanations.
- Number of assumption variables introduced
- A comparative analysis of performance between different sets of explanation categories

We measure the total runtime for the explainer defined in Section 3.3.3 (denoted as *Conflicts-Based* explainer), as this explainer needs to find all MCSes before computing the MUSes, which are considered its actual explanations. Whereas, for the Counterfactual explainer, we measure the time to first relaxation, as this explainer generates explanations incrementally, and each relaxation is considered an independent counterfactual explanation. We ran our experiments on a machine with an *AMD Ryzen 9 6900HX CPU* (16 logical cores and 3.3GHz) and 16GB of RAM. As explained before, for the explainer implementation we used *OrTools* in *Java*.

5.2.1 Benchmark instances

In this subsection, we present an evaluation based on benchmark TLSP instances that have been modified to include an unsatisfiable project presented in Section 5.1.1, thereby introducing infeasibility into the problem. Used instances can be found at https://www.dbai.tuwien.ac.at/staff/fmischek/TLSP/. The instance details can be found on Table 5.14, row 1-10.

For all instances we use a configuration that includes resource requirement, temporal and dependency (Linked Jobs) explanations. We modify the relaxation bounds depending on the instance.

The results from the benchmark experiments (Table 5.15) indicate that our infeasibility explainer performs efficiently on moderately sized TLSP instances, with total execution times in the order of minutes when up to a few projects/tasks are involved. The performance behaviour observed in Table 5.15 is strongly influenced by the structural properties of the TLSP instances shown in Table 5.14. As the number of projects, tasks, resource groups, and the scheduling horizon increase, the number of generated assumption variables ($|\mathcal{A}|$) grows accordingly. This directly affects the cost of MUS/MCS extraction and therefore the runtime of both explainers. Instances with small task sets and short horizons (e.g., 000 or 005) result in relatively few assumption variables and consistently low runtimes. In contrast, larger instances (e.g., 030, 048, 051), which contain hundreds

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

ID	Explainer	$\lvert\mathcal{A}\rvert$	E	Median(s)	IQR	Min	T-Median	T-IQR
000	CB	46	✓	0.2340	0.030	0.194	1.376	0.031
	CF	46	✓	0.198	0.029	0.183	1.516	0.137
005	CB	270	✓	1.808	0.313	1.336	102.818	5.789
	CF	270	✓	3.054	0.450	2.695	75.413	0.660
010	CB	555	✓	94.846	4.733	84.461	693.260	59.187
	CF	555	✓	20.450	4.828	15.012	148.718	25.074
015	CB	1243	✓	37.825	5.638	34.235	379.124	91.033
	CF	1243	✓	66.973	8.166	59.957	1095.472	706.553
021	CB	461	✓	31.538	3.641	29.141	443.950	18.768
	CF	461	✓	14.656	2.427	12.569	91.722	22.408
025	CB	1145	✓	84.295	14.721	70.541	766.666	42.368
	CF	1145	✓	512.687	89.166	346.503	2333.250	553.164
030	CB	1866	✓	379.166	121.835	257.331	9738.707	-
	CF	1866	×	3687.764	379.274	3308.490	-	-
035	CB	933	✓	21.812	2.991	16.340	341.456	10.750
	CF	933	✓	49.538	22.976	26.810	812.261	281.557
048	CB	2242	×	479.150	23.151	456.001	-	-
	CF	2242	×	20673.569	303.189	20370.380	-	-
051	CB	2635	×	889.374	97.512	803.266	-	-
	CF	2635	×	90842.337	6834.552	80714.992	-	-

Table 5.15: Benchmark instances evaluation. Explainer: CB = Conflict-based, CF = Counterfactual. E = explanations exhausted, $|\mathcal{A}|$ = Number of assumption variables, $Median(s)$ = and IQR correspond to first relaxation runtime, $T - Median(s)$ and $T - IQR$ correspond to total time.

of tasks, multiple resource groups, and long horizons, lead to substantially higher $|\mathcal{A}|$ values and consequently to very large runtimes or even timeouts, in contrast to the smaller instances discussed above.

Notably, while both the conflict-based and counterfactual explainer approaches exhibit comparable total runtimes, the counterfactual explainer only requires the time to first

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

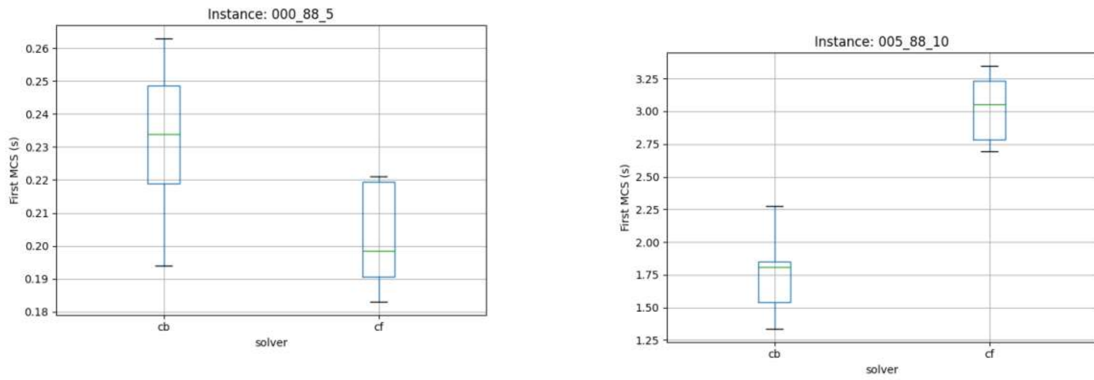


Figure 5.12

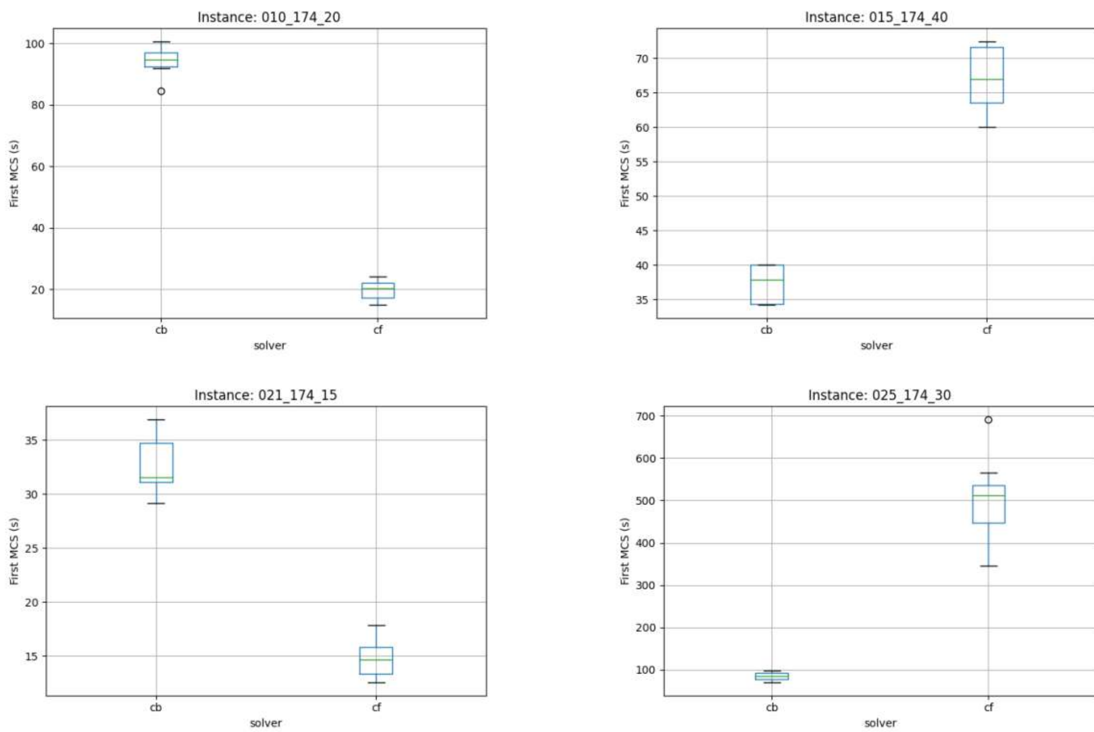


Table 5.16: IQR plots for time to first relaxation(MCS) for some benchmark instances 000, 005, 010, 015, 021, and 025.

relaxation to produce an explanation. This incremental nature allows it to deliver explanations earlier than the conflict-based explainer (see Fig. 5.13), which can be advantageous in high-complexity scenarios where computing all explanations is infeasible.

In summary, our evaluation confirms that the explainer is capable of providing timely and actionable insights for moderate TLSP instances. However, for very large and complex instances, further optimisation—such as reducing the number of generated assumption

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

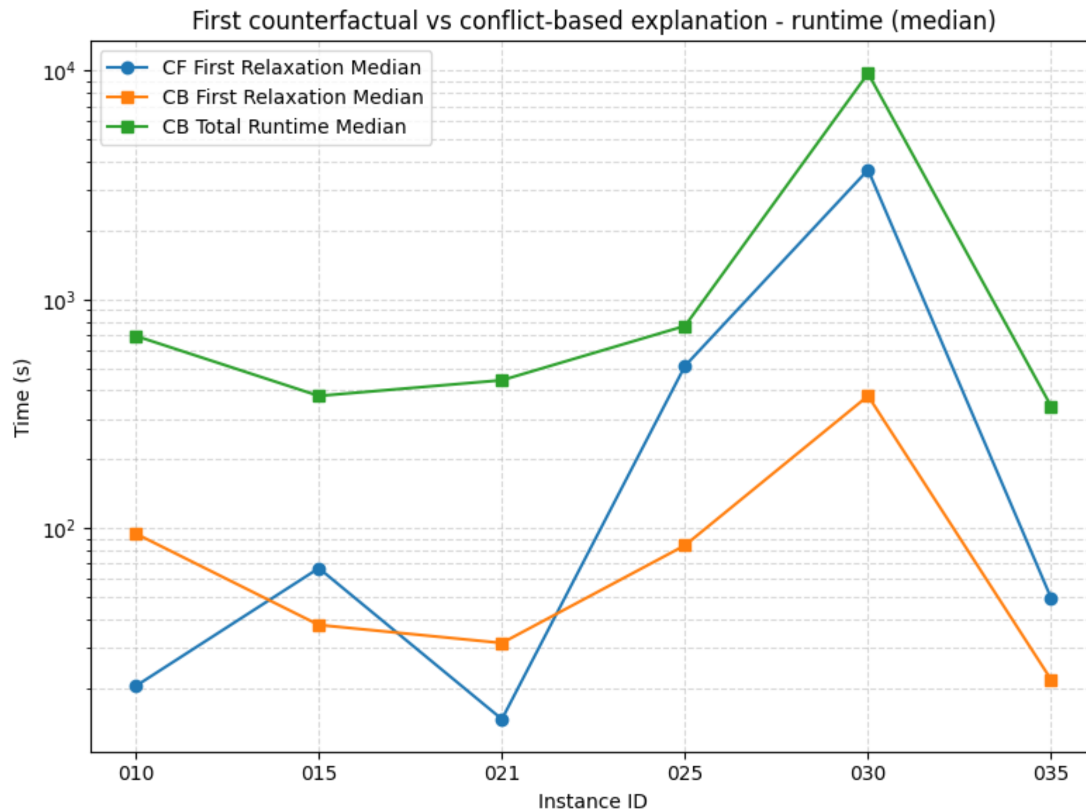


Figure 5.13: Comparison of time to first explanations between conflict-based and counterfactual explainers across benchmark instances.

variables, improving constraint encodings, or integrating heuristic pruning—will be necessary to maintain computational efficiency.

5.2.2 Real-world instance evaluation

This subsection evaluates the runtime efficiency of the explainer on actual TLSP instances, taken from an industrial setting. These real-world instances present practical challenges such as complex constraint interdependencies and high resource demands. Both instances had a feasible schedule initially, the infeasibility was encountered when a new project was introduced to the schedule. It should be noted that the infeasibility happens only when some of the initial projects were fixed to their initial schedule.

Its important to note that for these instances, the explainer is evaluated exclusively under objective O1. The evaluation on real-world TLSP instances demonstrates that the infeasibility explainer performs effectively in practical scenarios. When infeasibility is introduced by adding a new project to an existing schedule, the explainer reliably identifies the fixed projects that need to be unfixed, in a matter of minutes to hours.

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

ID	fp	Explainer	R	T	FP	E	$\|\mathcal{A}\|$	F. r.	Total
Lab_1	65	CB	×	×	✓	×	65	1.57	-
	65	CB	×	✓	×	✓	192	0.06	2.78
	65	CF	×	×	✓	×	65	1.67	-
	65	CF	×	✓	×	✓	192	0.11	11.07
Lab_2	74	CB	×	×	✓	✓	74	318.52	-
	74	CB	×	✓	×	✓	98	0.06	5.38
	74	CF	×	×	✓	✓	74	109.80	-
	74	CF	×	✓	×	✓	98	0.07	5.44

Table 5.17: Real-world instances evaluation. Explainer: CB = Conflict-based , CF = Counterfactual. fp = number of fixed projects, R = Resource explanations, T = Temporal explanations, FP = Fixed Project explanations, E = explanations exhausted, $|\mathcal{A}|$ = Number of assumption variables, F. r. = Median of time to first relaxation(minutes) , Total = Median of total runtime(minutes)

Subsequently, it provides resource and temporal explanations with minimal adjustments to the new project. The experiments indicate that the explainers can diagnose infeasibility and suggest relaxations within a reasonable runtime. Although larger and more complex instances still demand considerable computational resources, the overall performance represents a significant improvement over traditional trial-and-error methods. These findings confirm the practical utility of our approach in industrial settings, while also highlighting opportunities for further optimization.

5.3 User experience and practical insights

The practical impact of the explainer developed in this thesis was assessed through its deployment at an industrial partner. After deploying the system and introducing the schedulers to the functionality of the explainer and its GUI, they integrated it into their daily scheduling routines over several weeks and provided feedback based on their experiences. The feedback provided valuable insights into its effectiveness in real-world scheduling scenarios. Prior to using the explainer, resolving infeasibility relied on trial-and-error strategies, which could take anywhere from half an hour to several days. Common approaches included extending project due dates, unfixing related projects, or scheduling projects to the earliest available slot followed by a global optimization run. With the introduction of the explainer, resolving infeasibility became significantly more efficient. In cases where a feasible solution existed, the explainer provided a list of projects that needed to be unfixed within minutes, dramatically reducing the manual effort

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

required for scheduling. While the tool performs well for most instances, larger and more complex scheduling problems, particularly those in short-term planning, still demand considerable computational resources. The feedback also highlighted the importance of the explainer's user interface. The ability to clearly present the explanations and the resulting solutions was crucial for understanding the changes required to restore feasibility. The interface allowed users to interact with the explanations, enabling them to explore different solutions and understand the trade-offs involved. The feedback also emphasized the importance of the explanations' interpretability, as users needed to understand the reasons behind the suggested changes to make informed decisions.

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.

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 WIEN Your knowledge hub

CHAPTER 6

Conclusion

This thesis addresses the critical challenge of explaining infeasibility in automated scheduling systems, with a particular focus on the Test Laboratory Scheduling Problem (TLSP). After introducing the TLSP and its constraints, we identified the potential causes of its infeasibility. This was done in collaboration with our industrial partner, whose insights from real-world scenarios were invaluable. Based on the causes identified, we defined several categories of explanations, some of which are specific to the TLSP.

We explored existing approaches in the CSP literature to assess whether they could be adapted to meet our needs. We found that the most suitable approaches were those defined in [LS08a], [LT17], and [GGO22]. By adapting these approaches, we developed an infeasibility explainer that integrates two distinct explainers, each supporting different types of explanations. To the best of our knowledge, this is the first explainer specifically designed for the TLSP. These explainers were designed to generate explanations that are human-readable, and effectively identify the constraints causing infeasibility while also providing valuable insights for restoring feasibility. The first explainer is based on conflict detection, utilizing minimal unsatisfiable sets (MUS) of constraints and maximal satisfiable sets (MSS) to describe the source of infeasibility. Restoring feasibility is achieved through minimal correction sets (MCS). The second explainer supports counterfactual explanations, suggesting minimal changes required to restore feasibility, and, building on an existing approach, introduces novel weighting and blocking techniques.

Our approach was implemented using a tailored TLSP constraint programming model in OR-Tools. Additionally, we developed a configurable interface that improves usability by allowing users to configure the explainer's settings and run multiple configurations. This practical solution simplifies the diagnosis of infeasibility and provides suggestions for restoring feasibility, significantly reducing the reliance on time-consuming trial-and-error methods. The infeasibility explainer described in this thesis has been successfully deployed in an industrial setting, where it effectively aids in resolving real-world scheduling infeasibilities.

The experiments conducted included both qualitative analysis and efficiency assessments. We demonstrated various use cases, presenting the types of explanations generated and evaluating their effectiveness and computational performance.

Future work will focus on optimizing computational performance. Overall, the contributions of this thesis provide a solid foundation for integrating explainability into real-world constraint-based scheduling systems.

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

Overview of Generative AI Tools Used

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.

List of Figures

4.1 Configuring and running multiple TLSP explainer configurations. 44
4.2 Multiple explainer configurations. 45
4.3 Conflict-based explainer configuration. 45
4.4 Counterfactual explainer configuration. 45

5.1 Solution to the TLSP instance 50
5.2 Fixed schedule for *Project* 2 50
5.3 Explainer configuration: **Config 1**. 53
5.4 Explainer configuration: **Config 2**. 53
5.5 Explainer configuration: **Config 3**. 53
5.6 Feasible solutions to the TLSP instance by applying MCSes listed in Table 5.6 57
5.7 Explainer configuration: **Config 1**. 60
5.8 Explainer configuration: **Config 2**. 60
5.9 Explainer configuration: **Config 3**. 60
5.10 Explainer configuration: **Config 4**. 60
5.11 Feasible solutions to the TLSP instance by applying MCSes listed in Table 5.10 63
5.12 68
5.13 Comparison of time to first explanations between conflict-based and counterfactual explainers across benchmark instances. 69

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.

List of Tables

3.1 Explanations derived from corresponding foreground constraints 20

5.1 Projects and tasks 48
5.2 Mode requirements 48
5.3 Temporal requirements 48
5.4 Resource requirements 49
5.5 Conflicts generated using explainer defined in Section 3.3.2 51
5.6 Conflicts generated using explainer defined in Section 3.3.3 on instance I^L using configuration **Config 1** 54
5.7 Conflicts generated using explainer defined in Section 3.3.3 on instance I^L using configuration **Config 2** 58
5.8 Conflicts generated using explainer defined in Section 3.3.3 on instance I^F using configuration **Config 3** 59
5.9 Explainer configuration: **Config 5**. 61
5.10 Counterfactual explanations generated using explainer defined in Section 3.4 62
5.11 Counterfactual explanations generated using explainer defined in Section 3.4 64
5.12 Counterfactual explanations generated using explainer defined in Section 3.4 64
5.13 Counterfactual explanations generated using explainer defined in Section 3.4 65
5.14 Instance details including instance ID, number of projects, number of tasks, resource groups, resource units, and timeslots 65
5.15 Benchmark instances evaluation. Explainer: CB = Conflict-based, CF = Counterfactual. E = explanations exhausted, $|\mathcal{A}|$ = Number of assumption variables, $Median(s)$ = and IQR correspond to first relaxation runtime, $T - Median(s)$ and $T - IQR$ correspond to total time. 67
5.16 IQR plots for time to first relaxation(MCS) for some benchmark instances 000, 005, 010, 015, 021, and 025. 68
5.17 Real-world instances evaluation. Explainer: CB = Conflict-based , CF = Counterfactual. fp = number of fixed projects, R = Resource explanations, T = Temporal explanations, FP = Fixed Project explanations, E = explanations exhausted, $|\mathcal{A}|$ = Number of assumption variables, F. r. = Median of time to first relaxation(minutes) , Total = Median of total runtime(minutes) . . . 70

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.

List of Algorithms

3.1 SingleHittingSet($\mathcal{M}$) 25
3.2 PropagateChoice($\mathcal{M}'$, c, M_i) 25
3.3 AllHittingSets($\mathcal{M}, \mathcal{H}, H$) 26
3.4 Two-Phase Approach for Computing MUSes 30

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.

Bibliography

[BBG+25] Ignace Bleukx, Ryma Boumazouza, Tias Guns, Nadine Laage, and Guillaume Poveda. Modeling and Explaining an Industrial Workforce Allocation and Scheduling Problem. In Maria Garcia de la Banda, editor, *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, volume 340 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:24, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[BDM+99] Peter Brucker, Andreas Drexl, Rolf H. Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *Eur. J. Oper. Res.*, 112(1):3–41, 1999.

[Dan24] Philipp Danzinger. Optimizing constraint programming for real world scheduling of test laboratories. Master's thesis, Technische Universität Wien, Wien, 2024.

[DGGO21] Sharmi Dev Gupta, Begum Genc, and Barry O'Sullivan. Explanation in constraint satisfaction: A survey. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4400–4407. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Survey Track.

[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), March 2023.

[DGMM20] Philipp Danzinger, Tobias Geibinger, Florian Mischek, and Nysret Musliu. Solving the test laboratory scheduling problem with variable task grouping. In *Proceedings of the International Conference on Planning and Scheduling (ICAPS) 2020*, 2020.

[FO07] Alex Ferguson and Barry O'Sullivan. Quantified constraint satisfaction problems: From relaxations to explanations. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on*

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

Artificial Intelligence, Hyderabad, India, January 6-12, 2007, pages 74–79, 2007.

[GGO22] Sharmi Dev Gupta, Begum Genc, and Barry O'Sullivan. Finding counterfactual explanations through constraint relaxations. *CoRR*, abs/2204.03429, 2022.

[GMM19] Tobias Geibinger, Florian Mischek, and Nysret Musliu. Investigating constraint programming for real world industrial test laboratory scheduling. In Louis-Martin Rousseau and Kostas Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings*, volume 11494 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 2019.

[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.

[Jun01] Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving Problems with Constraints*, pages 1–7. Citeseer, 2001.

[Jun04] Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 167–172. AAAI Press / The MIT Press, 2004.

[KB21a] Alexander Korikov and J Christopher Beck. Counterfactual explanations via inverse constraint programming. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, pages 41:1–41:17. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[KB21b] Anton Korikov and J. Christopher Beck. Counterfactual explanations via inverse constraint programming. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 35:1–35:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[KKDS21] Mark T. Keane, Eoin M. Kenny, Eoin Delaney, and Barry Smyth. If only we had better counterfactual explanations: Five key deficits to rectify in the evaluation of counterfactual XAI techniques. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial*

Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021, pages 4466–4474. ijcai.org, 2021.

[KSB21] Anton Korikov, Alexander Shleyfman, and J. Christopher Beck. Counterfactual explanations for optimization-based decisions in the context of the gdpr. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4097–4103. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[Kum92] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Mag.*, 13(1):32–44, 1992.

[LS08a] Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reason.*, 40(1):1–33, 2008.

[LS08b] Mark H Liffiton and Karem A Sakallah. An overview of sat-based approaches to minimal unsatisfiable subsets. In *Proceedings of the 2008 IEEE Design Automation and Test in Europe Conference*, pages 423–428. IEEE, 2008.

[LT17] Kevin Leo and Guido Tack. Debugging unsatisfiable constraint models. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 2017.

[LT19] Niklas Lauffer and Ufuk Topcu. Human-understandable explanations of infeasibility for resource-constrained scheduling problems. In *ICAPS 2019 Workshop XAIP*, 2019.

[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)*, pages 465–467, 2018.

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

[MMS21] Florian Mischek, Nysret Musliu, and Andrea Schaerf. Local search approaches for the test laboratory scheduling problem with variable task grouping. *Journal of Scheduling*, 2021.

[MMS22] Florian Mischek, Nysret Musliu, and Andrea Schaerf. Local search neighborhoods for industrial test laboratory scheduling with flexible grouping. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2022)*, 2022.

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.

[MSHJ+13] Joao Marques-Silva, Federico Heras, Mikoláš Janota, Alessandro Previti, and Anton Belov. On computing minimal correction subsets. *IJCAI International Joint Conference on Artificial Intelligence*, pages 615–622, 08 2013.

[NSB+07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007.

[PF] Laurent Perron and Vincent Furnon. Or-tools v9.11 https://developers.google.com/optimization/.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition.* Pearson Education, 2010.

[SKB+23] Ilankaikone Senthooran, Matthias Klapperstück, Gleb Belov, Tobias Czauderna, Kevin Leo, Mark Wallace, Michael Wybrow, and Maria Garcia de la Banda. Human-centred feasibility restoration in practice. *Constraints An Int. J.*, 28(2):203–243, 2023.

[WMR17] Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *CoRR*, abs/1711.00399, 2017.

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.