TU WIEN Informatics

Instance Space Analysis and Constraint Programming Models for Unrelated Parallel Machine Scheduling Problems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Matthias Moik, BSc.
Matrikelnummer 11810738

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dr. Nysret Musliu
Mitwirkung: Dr. Felix Winter

Wien, 2. Mai 2025

Matthias Moik	Nysret Musliu

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

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

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

TU WIEN Informatics

Instance Space Analysis and Constraint Programming Models for Unrelated Parallel Machine Scheduling Problems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Matthias Moik, BSc.
Registration Number 11810738

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dr. Nysret Musliu
Assistance: Dr. Felix Winter

Vienna, May 2, 2025

Matthias Moik	Nysret Musliu

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

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

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

Erklärung zur Verfassung der Arbeit

Matthias Moik, 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, 2. Mai 2025

Matthias Moik

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

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

Danksagung

Als Erstes möchte ich mich von ganzem Herzen bei meiner Familie, insbesondere bei meinen Eltern, bedanken, die mir das Studium ermöglicht und mich während der gesamten Zeit in jeder Hinsicht unterstützt haben. Ich weiß das sehr zu schätzen.

Ebenso möchte ich mich bei meinem Betreuer, Associate Prof. Dr. Nysret Musliu, und meinem Co-Betreuer, Dr. Felix Winter, bedanken. Ihr Feedback und die kontinuierliche Unterstützung haben mir sehr geholfen, motiviert zu bleiben und die Arbeit erfolgreich abzuschließen.

Diese Arbeit wurde im Rahmen des *Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling* durchgeführt. Die finanzielle Unterstützung wird mit großem Dank angenommen.

Zum Schluss ein riesiges Dankeschön an meine Freundin und alle meine Freunde, die mich durch das Studium begleitet haben. Eine bessere Lerngruppe oder bessere WG-Mitbewohner hätte ich mir nicht wünschen können.

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.

Kurzfassung

Terminplanung spielt eine wichtige Rolle für viele Firmen und Institutionen, zum Beispiel zur Verbesserung der Effizienz komplexer Produktionssysteme. Es gibt verschiedene Möglichkeiten, um problemspezifische Zeitpläne zu erstellen und zu optimieren. Die Wahl eines passenden Algorithmus für eine konkrete Probleminstanz ist dabei entscheidend, da die Qualität des resultierenden Zeitplans stark variieren kann. Diese Diplomarbeit fokussiert sich auf das Unrelated Parallel Machine Scheduling Problem, ein bekanntes Optimierungsproblem mit vielen verschiedenen Varianten. Für ein tieferes Verständnis des Instanzraums dieses Problems wurde jedoch bisher wenig unternommen. Das ist aber wichtig, da existierende Instanzen aus früheren Studien einen Bias aufweisen könnten und so zu falschen Schlüssen über die Performance von Algorithmen führen.

In dieser Diplomarbeit wird der Instanzraum des Unrelated Parallel Machine Scheduling Problems analysiert, um tiefere Einblicke in die Struktur des Optimierungsproblems zu gewinnen. Instanz-Eigenschaften, die unter anderem auf Graphen basieren, werden eingeführt, um die Probleminstanzen beschreiben und klassifizieren zu können. Sie ermöglichen auch die Visualisierung des Instanzraums. Es wird festgestellt, dass die generierten Instanzen aus bekannten Datensätzen sehr unterschiedliche Eigenschaften im Vergleich zu realen Instanzen aufweisen. Daher werden zusätzliche Instanzen mithilfe eines neuen Instanzengenerators erzeugt, die ähnlicher zu den realen Instanzen sind. Auf dieser erweiterten Menge an Instanzen werden vier exakte Methoden sowie vier heuristische Ansätze zur optimierung ausgewertet. Dabei wird festgestellt, dass die Anzahl an zugelassenen Maschinen pro Job einen großen Einfluss auf die Performance der untersuchten Methoden hat. Mit den gewonnenen Erkenntnissen aus diesen Experimenten können Klassifikationsmodelle trainiert werden, die versuchen, den am besten geeigneten Algorithmus für eine konkrete Instanz automatisiert auswählen. Die besten Modelle sind in der Lage, die optimale Lösung öfter zu finden als jeder einzelne Algorithmus separat betrachtet.

Zusätzlich wird eine zweite Variante dieses Optimisierungsproblems betrachtet. Basierend auf den Ergebnissen der vorherigen Analyse wird ein Constraint Programming Modell, das Intervallvariablen verwendet, formuliert. Die Implementierung dieses Modells ist in der Lage, die Optimalität von Lösungen zu kleinen Instanzen in signifikant kürzerer Zeit zu beweisen als bisher bekannte exakte Lösungsmethoden. Auf größeren Instanzen kann diese Methode sogar einen heuristischen Ansatz auf einigen Instanzen übertreffen.

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

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

Abstract

Scheduling plays a crucial role for many companies and institutions across various sectors. The efficiency of complex production systems can be increased, or personnel utilization can be optimized. There exist many approaches to generate and optimize such schedules. The choice of the correct algorithm for a specific problem is very important, as the quality of the schedule can vary a lot, which might have a direct impact on the expenses. This thesis focuses on the Unrelated Parallel Machine Scheduling Problem, which is a well-known problem with many variants utilizing different constraints. However, very little work has been done in the past to analyze the instance space of this problem. Existing instances used in past studies might be biased, which may lead to wrong conclusions about the performance of algorithms.

This thesis analyzes the instance space to gain deeper insights into the underlying structures of the Unrelated Parallel Machine Scheduling Problem. Instance features, including graph-based and probing features, are introduced to describe and classify problem instances. These features also allow a visualization of the instance space. It is found that the existing, randomly generated instances have very different characteristics compared to the available real-life instances. Therefore, an extension to an existing instance generator is proposed to create a novel set of instances resembling real-life ones. On the combined set of instances, four exact methods and four heuristic approaches are evaluated and analyzed. It is discovered that the number of eligible machines per job greatly influences the performance of the exact methods. A comparison of the heuristics shows a similar influence of machine eligibility. Simple classification models are trained on the gathered data for automated algorithm selection. The final models are able to find the best solution more often than each algorithm or solver on its own.

Additionally, another problem with an extended set of constraints is investigated. Based on the insights of the prior analysis, a Constraint Programming formulation utilizing interval variables is proposed. An implementation of this model is able to prove optimality for instances of small size significantly faster than existing exact models. Also, it finds new best solutions for larger problem instances, by outperforming a Simulated Annealing approach for some instances.

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

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

Contents

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

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

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

CHAPTER 1

Introduction

Scheduling plays a crucial role for many companies and institutions, as it can significantly increase efficiency, allowing them to process more orders simultaneously or utilize personnel better. While this scheduling can be done by hand for small cases, it is difficult to find an optimized schedule for a larger number of jobs that have to be scheduled, or a higher number of employees that have to be assigned to projects. Because such scheduling problems are NP-Hard in many cases (Lenstra et al. [1977], Du and Leung [1990]), it also becomes infeasible for computers to find the optimal solution within a reasonable runtime with increasing instance size, which makes the utilization of heuristics or approximation algorithms necessary for practical applications.

This thesis focuses on an Unrelated Parallel Machine Scheduling Problem (UPMSP) introduced by Moser et al. [2022]. In the most basic form of the Parallel Machine Scheduling Problem, one aims to schedule jobs with a given processing time onto available machines so that no two jobs run on the same machine at the same time and no job is interrupted. The goal is to find an arrangement so that an objective, such as the makespan (the finishing time of the last job), is minimized. A detailed definition of the UPMSP with all its additional constraints is given in Section 2.1. The UPMSP is a topic of great interest not only in research but also in the industry because it can model certain production processes quite well. This specific variation introduced by Moser et al. [2022] was developed in collaboration with partners from the industry in the Christian Doppler Labor for Artificial Intelligence and Optimization for Planning and Scheduling.

With growing businesses, the complexity and scale of such problems only keep increasing. Also, the problem instances get more and more specialized to different production systems, which might lead to various instance types within the problem domain itself. For example, the difference between two such subdomains could be the ratio of jobs to be scheduled to available machines, or the distribution of due dates of the jobs. Such a variety of problem instances comes with a caveat. Algorithms that perform well in one subdomain might struggle to find good solutions for others. For the same reason,

a fair comparison between various algorithms is often a challenging task. To compare algorithms, the usual approach is to evaluate them on a set of instances, followed by an analysis of their performance (values of the objective function). These results are interpreted using statistical tests to conclude which algorithms outperform others. To gain further insight into the performance of the algorithms, Smith-Miles et al. [2014] introduces a methodological approach called Instance Space Analysis (ISA), which yields novel insights that complement the benchmarking results. First, the ISA allows for an assessment of the bias of an instance set, and second, instances can be grouped by different types, which might reveal that there is no single best algorithm, but multiple best ones, depending on the instance type.

This method was already applied successfully to other scheduling problems, like the Job Shop Scheduling Problem by Strassl and Musliu [2022] or to a Personnel Scheduling Problem by Kletzander et al. [2021]. The aim of this thesis is to investigate the instance space for the UPMSP to identify and remove biases in existing instance sets and provide valuable insights about the influence of instance features on algorithmic performance. Also, the proposed instance features can be utilized by algorithm selection methods to predict the best algorithm for a given problem instance, which will also be explored in this thesis.

While the ISA can generate valuable insights into this version of the UPMSP, the problem specifications evolve continuously. Since the introduction of the UPMSP by Moser et al. [2022], new constraints were proposed and added to the problem. A recent version introduced by Horn et al. [2025] focuses on resource calendars and machine availabilities. Since these constraints were introduced very recently, only a few optimization methods are available for this problem. Another part of this thesis is to propose a novel Constraint Programming (CP) formulation to this problem, based on the insights gained from the previous work of this thesis. It is expected to outperform existing, comparable methods and find new best solutions for available instances.

1.1 Aims of the Thesis

The main aim of this thesis is to conduct an ISA for the UPMSP. In addition to this, the gained insights will be utilized to perform algorithm selection for the UPMSP and to introduce a CP formulation for an extension of the problem. The main goals are:

- Identifying state-of-the-art exact and heuristic approaches for the UPMSP, and complementing them with novel approaches for a broader analysis.
- Proposing instance features, including graph-based and probing features to distinguish and classify problem instances. Analyzing existing instance sets from the literature for biases and removing them by adding newly generated instances, using a novel instance generator.

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

- Identifying features that influence the performance of the exact and heuristic methods included in this analysis.
- Training an algorithm selection model that outperforms the individual algorithms, by utilizing the proposed instance features as the model input.
- Building a CP model that outperforms existing exact methods for a recently introduced UPMSP with an extended set of constraints.

1.2 Main Contributions

The main contributions of this thesis are:

- Existing state-of-the-art approaches like Mixed Integer Programming (MIP) formulations and Simulated Annealing (SA) approaches are reimplemented and validated. A novel formulation of a CP model, utilizing interval variables, and a Large Neighborhood Search (LNS) for the UPMSP are also introduced.
- A set of 150 instance features for the UPMSP is introduced, including novel probing features and graph-based features derived from two different graphs. An existing instance generator is extended to create a novel instance set, containing 764 instances, that are able to reach areas in the instance space that are not well covered by the existing instances.
- The algorithmic performance for exact and heuristic methods is analyzed in the context of the instance space, which allows the identification of the most influential features, regarding the performance of each algorithm.
- An algorithm selection model that is able to outperform the individual algorithms on an unseen set of instances is trained, based on the introduced instance features.
- A general approach for handling resource calendars and machine downtimes by making use of interval variables is introduced. A CP model including this approach yields better results than existing exact methods for this problem and also finds better solutions than SA under comparable conditions for a few instances.

1.3 Structure

The thesis is split into multiple parts. Chapter 2 introduces the exact problem definition of the UPMSP, as well as a brief description of the methodology used in the ISA. Next, the utilized exact methods and heuristics will be described in Chapter 3, while Chapter 4 focuses on the various parts of the ISA, like utilized problem instances and instance features, as well as the experimental setup. In Chapter 5, all results of the experiments are reported, evaluated, and discussed. The CP formulation and the performance evaluation for the extended UPMSP are presented separately from the ISA in Chapter 6. Finally, 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

conclusion on the accomplished work is given in Chapter 7, with an outlook for further work.

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

CHAPTER 2

Background and Related Work

In this chapter, we want to provide a complete description of the UPMSP, with all its additional constraints and the notation used from now on. Further, we will briefly introduce the concept and principle behind the ISA and the framework for algorithm selection. Finally, the current state of the art and related work for these topics is summarized.

2.1 The UPMSP

The class of UPMSPs contains many different versions. The basis for all of them is that several jobs have to be scheduled onto machines, where the processing time of the jobs depends on the machine they are scheduled on. Additional constraints lead to different versions like Perez-Gonzalez et al. [2019], which uses due dates and machine eligibility constraints, or Santoro and Junqueira [2023], which utilizes machine availability constraints. In this thesis, we focus on the variant by Moser et al. [2022] that describes a challenging, practical version of the problem, as well as another variant by Horn et al. [2025], which extends these constraints even further. In the following, we give a detailed problem definition of the variant introduced by Moser et al. [2022] and introduce the notation utilized in the rest of this thesis.

2.1.1 Constraints

An instance of the UPMSP includes a set of machines $M, |M| = M_c$ and a set of jobs $J, |J| = J_c$. A job has a given processing time, dependent on the machine on which the job is scheduled. Because these processing times can differ from machine to machine, the machines are called *unrelated*. Also, not all jobs can be scheduled on every machine. There exist so-called *machine eligibility constraints*. A job $j \in J$ is eligible on machine $m \in M$, if and only if $m \in E_j \subseteq M$. The processing time for job $j \in J$ on machine $m \in E_j$ is denoted with p_{jm}.

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

Furthermore, there are *sequence- and machine-dependent setup times* when scheduling job j directly after job i (the *predecessor* of job j) on machine m, called s_{ijm}. In real life, this could be, for example, the time needed to change tools on a workbench between two different tasks. There are also initial and final setup times, s_{0jm} and s_{j0m} respectively, before the first and after the last job is scheduled on a machine.

Each job has to be assigned to an eligible machine for processing. If job j is scheduled on machine m with predecessor i, it occupies machine m for $s_{ijm} + p_{jm}$ time steps, where no other setup or processing is allowed on the same machine.

2.1.2 Objective Function

To calculate the objective function, we first introduce the *completion time* C_j of a job j recursively using its predecessor job i with $C_j = C_i + s_{ijm} + p_{jm}$. The completion time of the first job on a machine is given by the initial setup time s_{0jm}, plus its processing duration p_{jm}. The *machine makespan* for machine m is defined as the completion time of the last job scheduled on it, plus the corresponding final setup time, $O_m := C_j + s_{j0m}$. An often-used objective in the literature is to minimize the maximal machine makespan $C_{max} := \max_{m \in M} O_m$, as it is reported in the literature review Marko and Jakobovic [2023].

The problem instance also contains *due dates* d_j for all jobs $j \in J$ that should be considered in the schedule. In an optimal case, every job is finished before its due date. If this is not possible, we will try to minimize the cumulative delay. This objective is the *minimization of the cumulative tardiness* $\sum_{j \in J} T_j$, where the individual tardiness of job $j \in J$ is defined as $T_j := \max(0, C_j - d_j)$.

The objective function introduced by Moser et al. [2022], namely the lexicographical ordering of the cumulative tardiness and the makespan, is used to distinguish between two solutions with the same cumulative tardiness. This objective function completes the problem definition of the UPMSP that is utilized for the ISA in the next chapters. With the three field notation introduced by Graham et al. [1979], it can be characterized as $R_m|E_j, s_{ijm}|Lex(\sum_j T_j, C_{max})$.

To fully describe the problem, including all constraints and the objective, one could refer to it as the Unrelated Parallel Machine Scheduling Problem with machine eligibility constraints, sequence- and machine-dependent setup times, minimizing the cumulative tardiness and the makespan.

2.1.3 Example Instance

In the following, we describe a simple problem instance of the UPMSP to illustrate the problem's constraints and objectives. The input data for a small example problem with four jobs and two machines is given in 2.1. The information can be arranged in a vector and matrix format. The vector d contains the due dates, and the matrix p contains the processing times for each job-machine combination. A dot means the job is not eligible

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

$$d = \begin{pmatrix} 4 \\ 2 \\ 7 \\ 9 \end{pmatrix} \qquad p = \begin{pmatrix} 3 & 4 \\ 3 & \cdot \\ 1 & 2 \\ \cdot & 2 \end{pmatrix} \qquad s_1 = \begin{pmatrix} 0 & 3 & 2 & 4 & 3 \\ 2 & 0 & 1 & 4 & 0 \\ 3 & 3 & 0 & 3 & 3 \\ 2 & 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 4 & 0 \end{pmatrix} \qquad s_2 = \begin{pmatrix} 0 & 3 & 2 & 4 & 3 \\ 1 & 0 & 4 & 2 & 0 \\ 1 & 2 & 0 & 8 & 2 \\ 4 & 3 & 2 & 0 & 3 \\ 1 & 0 & 4 & 2 & 0 \end{pmatrix}$$

Table 2.1: Input representation of an example instance with two machines and four jobs

for scheduling on the machine. Matrices s_m for $m \in M$ store the setup times on machine m. The setup time matrices have one row and column more than the due date vector and the processing time matrix. This is because of the initial and final setup time. As the introduced notation suggests, the dummy jobs used for initial and final setup times are denoted with 0. So row and column zero store the initial and final setup times, respectively.

2.1.4 Solution Representation

A solution can be represented as an ordered list of jobs for each machine, like in Table 2.2. Jobs 2 and 3 are scheduled on machine 1, and jobs 1 and 4 are processed on machine 2. The solution representation contains all the information needed to construct a unique schedule. We only need the order of the jobs, since we start the setup of a job immediately after its predecessor is finished.

Machine	Ordered Jobs
1	2,3
2	1,4

Table 2.2: Solution representation for a schedule corresponding to the example instance

Using this representation format, only two criteria must be satisfied to produce a valid solution. A feasible solution is given if:

- All jobs are scheduled exactly once
- All jobs are scheduled on one of their eligible machines

A visualization of the schedule represented in Table 2.2 to the example instance from 2.1 can be seen in Figure 2.1. The blue bars show the setup times, while the red ones represent the processing time of the jobs. This is the optimal solution for this problem instance with an objective function value of $(8, 11)$.

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

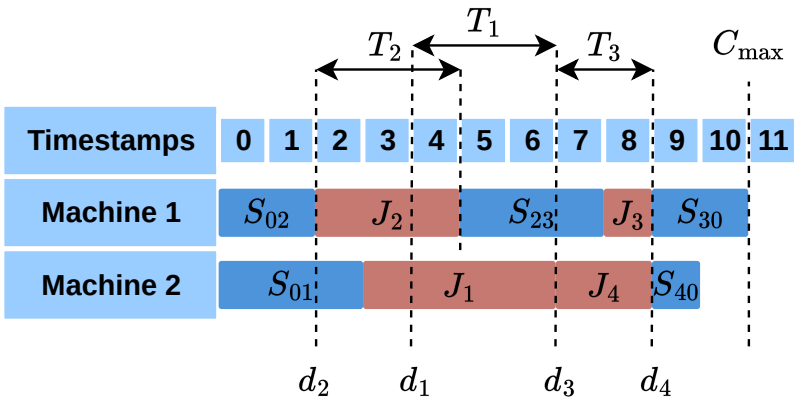


Figure 2.1: Visualization of the schedule representation in Table 2.2

2.2 Instance Space Analysis

Smith-Miles et al. [2014] proposed the ISA to enable a more detailed evaluation of algorithms than reporting aggregated metrics over a set of instances. Figure 2.2 depicts the methodological framework used for the ISA. It is an extension of the proposed algorithm selection framework of Rice [1976], which is visualized as the blue box in the graphic.

We start with a set of known instances I (from the literature or an instance generator), a subset of all possible instances P. The set of instance features F allows the mapping of the problem instances into a high-dimensional space. This can be used directly for the algorithm selection, which is what Rice [1976] proposed. In the ISA, which is more about identifying and analyzing influential features and exploring potential biases in the dataset, the distribution of the instances in the high-dimensional space is visualized. This is done by projecting it onto a two-dimensional plane using a similar approach to Principal Component Analysis (PCA). The adaptation utilized for the projection into the instance space is described in Muñoz et al. [2018]. Not only do we want to separate instances as much as possible from each other, but ideally, a separation of the instances into areas where a specific algorithm outperforms the others. Therefore, using algorithmic performance as an additional input makes sense in finding an optimal projection for this use case.

There still is the issue that the algorithms are only evaluated on a subset of the whole problem space, but we want to draw conclusions about new instances and areas of the instance space. The proposed footprints by Smith-Miles et al. [2013] aim to predict the algorithmic performance in between and around the area of its evaluation points based on statistical methods. The correctness of this extrapolation into new areas of the space relies on an unbiased instance set, as well as carefully designed instance features to ensure

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

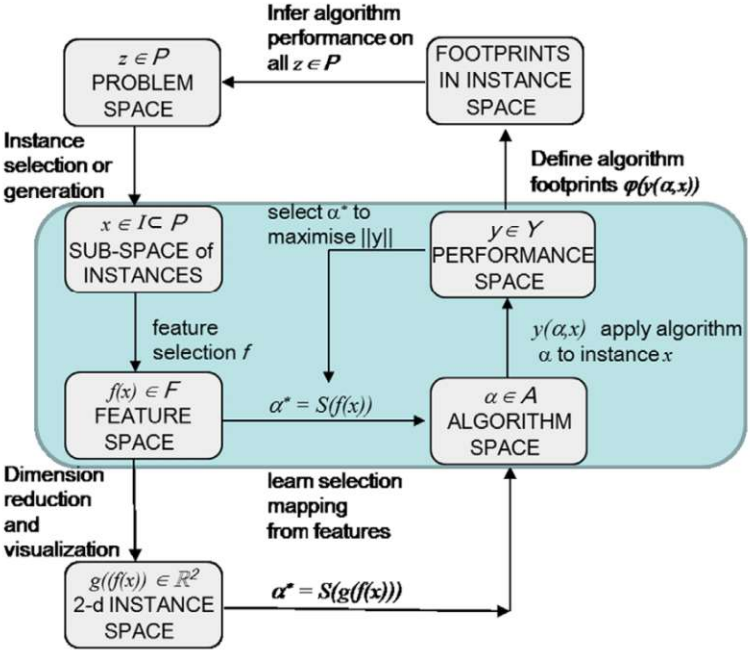


Figure 2.2: Visualization of the methodological framework of the ISA, taken from Smith-Miles et al. [2014]

that as much of the relevant information about an instance as possible is preserved. With this methodological approach, one can make a sophisticated prediction of whether an algorithm performs well or badly for instances from the whole problem space.

2.3 State of the Art and Related Work

The UPMSP is a widely known and studied problem with many variants that differ in some constraints and objectives. The problem in this thesis has machine eligibility constraints and sequence- and machine-dependent setup times, and we try to minimize the cumulative tardiness, with the makespan as a tie-breaker.

A formulation for MIP solvers for the UPMSP with setup times and minimizing the makespan was proposed by Avalos-Rosales et al. [2015], and Gedik et al. [2018] proposed a CP Model to solve the same variant. Vallada and Ruiz [2011] also considers the UPMSP with setup times and the makespan as an optimization criterion. They propose a Genetic Algorithm and also introduce a benchmark dataset. However, it cannot be utilized in our thesis because the dataset does not include the necessary due dates for the tardiness objective. Another related problem from the literature comes from Perez-Gonzalez et al. [2019]. They work with an objective function that minimizes cumulative tardiness, on a scheduling problem with machine eligibility constraints and sequence-

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

and machine-dependent setup times. They introduce new construction heuristics to the problem and a novel set of instances. Because of the presence of due dates in this dataset, we can utilize it for the ISA. Moser et al. [2022] proposes new versions of MIP solvers and SA approaches to the UPMSP investigated in our thesis. More recent publications focus on novel additional constraints rather than proposing improved algorithms for existing problems. For example, Saraç et al. [2023] introduces stochastic setup times to model real-world scenarios more accurately, and Dang et al. [2023] focuses on a machine scheduling problem considering unsupervised machines and cost-related tool switches. They propose an MIP formulation, as well as a Genetic Algorithm, with the objective of profit maximization.

The extended problem by Horn et al. [2025] we consider in the latter part of this thesis includes the already introduced constraints, precedence constraints, and machine and resource calendars. The objective function combines setup times, tardiness of jobs, and machine makespans. While Santoro and Junqueira [2023] focuses on machine availability constraints, they do not consider calendar-based resources. Yunusoglu and Topaloglu Yildiz [2022] introduces a CP approach for UPMSPs that includes multi-resource requirements, job precedences, and setup times. However, it cannot handle calendar-based resources and machine availabilities. The proposed combination of a Genetic Algorithm and an MIP approach by Dang et al. [2021] considers an objective function combined from tardiness and setup times, but uses different problem constraints. Horn et al. [2025] introduces a MiniZinc formulation (Nethercote et al. [2007]) that can be utilized by MIP and CP solvers. They also propose an SA approach to the problem. Because of the novelty and relevance of the constraints of this problem in practical applications, we aim to introduce a novel CP formulation. Based on the insights from the first problem, the focus lies on handling the resource and machine availability constraints with interval variables.

The ISA framework by Smith-Miles et al. [2014] was successfully applied to many related problems, like the Job Shop Scheduling Problem Strassl and Musliu [2022], the Personnel Scheduling Problem Kletzander et al. [2021], or the Curriculum-Based Course-Timetabling problem De Coster et al. [2022]. Also, it was used successfully to select optimal algorithm parameters depending on the test instance in Katial et al. [2024]. If the instance space shows some gaps, new instances must be generated. There exists an instance generator from Moser et al. [2022]. Alternatively, the approach introduced by Smith-Miles and Bowly [2015] could be utilized. They evolve new instances to fill the instance space if the available dataset does not represent the whole space adequately. However, the first option is unsuitable for generating instances with novel features because of its limited variability. The latter method contains a lot of randomness and does not allow for a precise and simple instance generation.

Algorithm selection was successfully applied for scheduling problems, like Strassl and Musliu [2022] for the Job Shop Scheduling Problem or Messelis and De Causmaecker [2014] for a Project Scheduling Problem. To get a broader overview, Kotthoff [2016] collected many algorithm selection applications for various problem types. The idea

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

of the algorithm selection in combination with the ISA is based on the framework of Rice [1976] and extended by Smith-Miles et al. [2014]. An algorithm selection utilizing a Support Vector Machine (SVM) is directly included in an online tool to conduct an ISA called *MATILDA*, proposed by Smith-Miles and Muñoz [2023]. However, these existing approaches cannot be utilized directly for the UPMSP without additional work. A more recent study by Wu et al. [2024] performs algorithm selection by utilizing algorithm characteristics extracted with the help of LLMs. However, this is out of the scope of this thesis. The thesis focuses on the instance features and the corresponding instance space, which leads to interpretable results and insights.

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

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

CHAPTER 3

Exact Solvers and Heuristics for the UPMSP

This chapter introduces the various methods used in the ISA. While some algorithms and models are taken from Moser et al. [2022], we also propose new strategies based on known concepts that must be adapted to our problem description.

3.1 Exact Solvers

We utilize two approaches for the exact solvers: Mixed Integer Programming (MIP) and Constraint Programming (CP). We look at two variations for each approach described in the following sections. These methods are classified as exact solution methods because, with enough memory and runtime, they could prove the optimality of a solution to a problem instance or its infeasibility.

3.1.1 Mixed Integer Programming Models

The MIP formulations were taken from Moser et al. [2022], where they were proposed for the first time. They are based on formulations from Perez-Gonzalez et al. [2019]. While Moser et al. [2022] evaluated six different MIP solvers, we only restricted ourselves to the two best from that analysis, namely $M4$ and $M6$. The formulation $M4$ uses constraint formulations proposed by Avalos-Rosales et al. [2015]. This is why we call this model from now on `MIP_A`. The transformation from $M4$ to $M6$ is done by swapping out a single constraint, where Helal et al. [2006] introduced the latter version. Therefore, we refer to this model as `MIP_H`. For the complete MIP formulation, we refer to the original work by Moser et al. [2022].

We implemented these models again because of the different frameworks used for this project. We cannot compare the performance of the newly implemented models to the

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

Variable	Description
$start_j$	Start time of job j
end_j	End time of job j
$duration_j$	Processing time of job j
T_j	Tardiness of job j
$start_{jm}$	Start time of optional job j on machine m
I_j	Interval variable representing job j
I_{jm}	Optional interval variable representing job j on machine m
$active_{jm}$	Boolean variable indicating if job j is scheduled on machine m
arc_{ijm}	Boolean variable indicating if job j is scheduled right after job i on machine m
C_j	End time of the final setup time after job j
T	Cumulative Tardiness
C_{max}	Makespan

Table 3.1: Variables used in CP formulation for the UPMSP

original work, due to varying versions of the Gurobi solver (Gurobi Optimization, LLC [2024]) utilized. We reimplemented the MIP formulation and reevaluated it with the newest Gurobi version. The results clearly improved compared to the older version in a few manually selected instances.

3.1.2 Constraint Programming Models Utilizing Interval Variables

While the following CP formulation can be utilized with any CP solver, we use CP-SAT. The CP-SAT solver is part of the Google OR-Tools package(Perron and Didier [2024]). It combines CP with SAT methods to find good solutions to combinatorial optimization problems. While the constraints can be formulated similarly to the MIP models, additional options like global constraints and interval variables can also be helpful for the formulation of the UPMSP. We introduce a CP formulation for the problem and propose a slight modification to create a second version.

The formulation of the constraints is based on Gedik et al. [2018]. Still, it differs in a few ways because they do not utilize machine eligibility constraints or due dates, and try to optimize the maximal machine makespan.

Table 3.1 describes the variables used to formulate the model, which is defined by Constraints (3.1) - (3.14). We also define $J_0 := J \cup \{0\}$, which is the set of all jobs, extended by the dummy job 0. This dummy job can be scheduled on all machines, so $E_0 = M$. The input data from the problem instance uses the notation introduced in Section 2.1.

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.

Before we start with the actual constraint formulation, we want to add a brief explanation of two concepts utilized, namely interval variables and the circuit global constraint.

In general, interval variables are defined by grouping three integer variables or integer values, indicating the start, size, and end of the interval. We simply use `interval` to indicate this type of variable and define it by passing the start, size, and end as parameters. Intervals can also have a *fixed size*, which takes only an integer variable as the start and an integer value as the duration. Also, an interval variable can be *optional*, meaning a Boolean indicates if the interval is actually active or if it is ignored by other constraints. In our formulation, we also make use of optional, fixed-size interval variables using the notation `opt_fixed_size_interval`. The parameters start, duration, and the Boolean, to indicate if it is active, define the optional fixed-size interval variable. Utilizing these special variable types, global constraints that are defined on a set of interval variables, like `no_overlap`, will be usable in the modeling process.

Secondly, we want to explain the `circuit` global constraint first introduced in Lauriere [1978] and later improved by Francis and Stuckey [2014]. This constraint is used to constrain a graph represented by a successor for each node, such that the resulting arcs form a circuit. Using self-loops, nodes can be excluded from the global circuit. In our case, this constraint allows us to efficiently capture the order of the jobs on each machine, starting and ending at the dummy job 0. The `circuit` global constraint takes a set of arcs with a corresponding Boolean variable as input. The Boolean indicates if the arc is present in the cycle.

Starting with Constraints (3.1) - (3.2), we define the global and machine-specific interval variables.

$$I_j = \texttt{interval}(start_j, duration_j, end_j) \quad \forall j \in J \tag{3.1}$$

$$I_{jm} = \texttt{opt\_fixed\_size\_interval}(start_{jm}, p_{jm}, active_{jm}) \quad \forall j \in J, m \in E_j \tag{3.2}$$

Constraint (3.3) simply enforces the tardiness variable to take on its correct value. If job j ends before d_j, the tardiness is 0. Otherwise, it is the difference between the end of the processing time and the due date, as it was defined in the problem description in Section 2.1.

$$T_j = \max(0, end_j - d_j) \quad \forall j \in J \tag{3.3}$$

Constraint (3.4) ensures that each job is only scheduled on one machine. The corresponding global constraint is called `exactly`, which ensures that exactly N variables of a collection of variables have assigned value v. In our case, $N = 1$ and $v = True$, so it is ensured that exactly one of the Boolean variables takes on the value $True$, and all others get assigned $False$. As already suggested by the notation in the constraint, we implemented this in our CP-SAT model using the CP-SAT-specific constraint called `exactly_one`.

$$\texttt{exactly\_one}(\{active_{jm} | m \in E_j\}) \quad \forall j \in J \tag{3.4}$$

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.

With Constraint (3.5), the model makes sure that no two jobs are being processed on the same machine at the same time. While the CP-SAT specific constraint is called `no_overlap`, it is usually called the `disjunctive` global constraint and was first proposed by Carlier [1982]. Because of the usage of optional intervals, this constraint only considers the jobs that are actually active on a given machine and ignores all the others, making interval overlaps still possible, if they are on different machines.

$$\texttt{no\_overlap}(\{I_{jm} | j \in J,\ m \in E_j\}) \quad \forall m \in M \tag{3.5}$$

Using Constraints (3.6) - (3.7), the model forces the global job intervals to take on the same start times and durations as the active, machine-dependent job intervals. Together with the preceding constraints, we have interval variables representing the jobs that avoid overlaps on each specific machine. Implications occurring in the constraints are enforced using the `only_enforce_if` method available for CP-SAT.

$$active_{jm} \Rightarrow start_j = start_{jm} \quad \forall j \in J, m \in E_j \tag{3.6}$$

$$active_{jm} \Rightarrow duration_j = p_{jm} \quad \forall j \in J, m \in E_j \tag{3.7}$$

The circuit Constraint (3.8) select arcs out of each machine-specific set of arcs to create cycles. To exclude jobs that are not active on the current machine, Constraint (3.9) is used, which forces self-loops on these jobs. The other jobs occur in the big cycle, including the dummy job 0, which acts as a start and end point for the schedule. Constraints (3.10) - (3.12) enforce the correct setup times between jobs, as well as in the beginning and the end. Note that C_j is only enforced if job j is actually the last on the machine. All other variables are still unconstrained, which avoids interference from them when enforcing C_{max}.

$$\texttt{circuit}(\{arc_{ijm} | i, j \in J_0,\ m \in E_i \cap E_j\}) \quad \forall m \in M \tag{3.8}$$

$$arc_{jjm} = \neg active_{jm} \quad \forall j \in J, m \in E_j \tag{3.9}$$

$$arc_{0jm} \Rightarrow start_j = s_{0jm} \quad \forall j \in J, m \in E_j \tag{3.10}$$

$$arc_{ijm} \Rightarrow start_j = end_i + s_{ijm} \quad \forall i, j \in J, i \neq j, m \in E_j \tag{3.11}$$

$$arc_{j0m} \Rightarrow C_j = end_j + s_{j0m} \quad \forall j \in J, m \in E_j \tag{3.12}$$

Finally, we set the total tardiness and makespan to their correct values with Constraints (3.13) - (3.14).

$$C_{max} = \max_{j \in J} C_j \tag{3.13}$$

$$T = \sum_{j \in J} T_j \tag{3.14}$$

We want to add a final note to this constraint formulation. For a working model, we could drop the interval variables by introducing another constraint that enforces the

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

correct end_j for all jobs. We chose to include the interval variables nevertheless because it enables us to utilize the `no_overlap` constraint. The inclusion of the interval variables showed promising results in preliminary experiments, where we also evaluated a version without them on a few handpicked instances.

Variations in Optimization

We included two variants of this CP formulation in the ISA to see if a small change brings any difference in performance for certain instances. Because we have a two-valued objective function for the UPMSP instead of one for most problems, there are two canonical ways to approach the optimization:

- **Combined**
  The first option is to use a large constant as a scalar multiplier for the total tardiness, and then add the makespan to it to get a single value as an objective function, where the total tardiness is lexicographically more important than the makespan. We did this by minimizing the objective function given in Equation (3.15).

  $$obj = c \cdot T + C_{max}, \quad \text{where} \quad c = 10^{\lceil log(horizon) \rceil} \tag{3.15}$$

  The constant c has the same number of digits as the *horizon*, which is an upper bound for the makespan of the instance. This upper bound is used to calculate the combined objective value by shifting the cumulative tardiness objective up so that it does not interfere with the makespan objective.

  The *horizon* is calculated by trying to maximize the makespan of a feasible schedule. This is done by selecting a single machine and scheduling all the eligible jobs on it. To avoid dealing with the job-dependent setup times, we simply pick the maximal setup time each job could possibly have on the current machine. By doing this for every machine and then selecting the maximal value among those, we have an upper bound of the makespan for the given problem instance. This version is referred to as `CP_C`.

- **Separate**
  The second strategy to optimize is to look at the objectives one by one. First, we optimize the total tardiness by setting the objective as in Equation (3.16). If the minimal tardiness, here denoted as T^*, of the problem instance is not found before the timeout, the search ends.

  $$obj = T \tag{3.16}$$

  But if T^* is found in time, we add Constraint (3.17) to the model, which fixes this optimal tardiness for the problem.

  $$T = T^* \tag{3.17}$$

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

Afterward, we continue with the optimization of the makespan by using the new objective in Equation (3.18). Now the optimization of the makespan is possible without changing the tardiness again, as it is already fixed by the constraint.

$$obj = C_{max} \tag{3.18}$$

This version, optimizing the objective separately, is referred to as `CP_S` in later chapters.

Making use of these two options might result in different outcomes for certain problem instances. For some, it could be beneficial to just focus on the tardiness first, and for others, it might prove to be better to have two measurements available that impact the objective function, allowing for more fine-grained solution steps.

3.2 Heuristics

The following methods are part of the class of heuristic approaches to optimization problems. Heuristic methods are usually used for larger instance sizes when exact solvers are not able to find a good or any solution at all. While the heuristics are fast in searching and finding new solutions, there is a trade-off, namely the loss of provability that a solution is optimal. Both search strategies utilized by us are based on Local Search. Local Search is an optimization strategy that checks the objective function for solutions that lie in a neighborhood of the currently best solution. The neighborhood of the best solution contains all solutions that can be achieved by applying a small change to it. In our case, this could be switching the positions of two jobs. The neighborhoods utilized for SA are described in detail in the original work by Moser et al. [2022], while the LNS approach is introduced by us and is described completely below.

3.2.1 Construction Heuristic

SA and LNS both require an initial solution to start their optimization. We obtain this initial solution with the construction heuristic introduced by Moser et al. [2022]. Basically, it sorts the jobs by their due date and schedules them one after another on the machine where the job has the earliest finishing time. If two jobs have the same due date, or a job finishes on two machines at the same time, the ties are randomly broken. The detailed description, as well as the pseudo-code, can be found in Moser et al. [2022]. From now on, we refer to this construction heuristic as `Greedy`.

3.2.2 Simulated Annealing

The three SA Approaches that will be included for the ISA were taken from Moser et al. [2022]. We implemented them again in our framework and performed experiments to verify they behave in the same way and result in an equal performance as the original implementation in Section 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.

SA was first proposed by Kirkpatrick et al. [1983]. This approach is inspired by a cooling process that shows up in industrial production processes. The algorithm starts from a given solution and uses available neighborhoods to find similar solutions, and then checks if it improved over the current best solution or not. Usually, in Local Search, this neighborhood search is done from the best known solution. However, this has the disadvantage that it might get trapped in a local optimum. To escape such a local optimum, SA also allows the selection of a worse solution by chance, and lets the search continue from there.

The probability of selecting a worse solution is not constant but gets smaller with a higher number of iterations or longer runtime. The probability of accepting a worse solution depending on parameter t is given in Equation (3.19). Parameter t is called the temperature. A higher temperature leads to a higher acceptance probability.

$$\mathbb{P}(accept|t) = e^{-\frac{\delta}{t}}, \tag{3.19}$$

Here, δ is defined as the difference of the weighted objective functions from Equation (3.20) of the new candidate solution and the current solution.

$$f = 10\,000 \cdot T + C_{max} \tag{3.20}$$

The three variations proposed by Moser et al. [2022] differ only by their cooling schemes (change of the temperature t), which are briefly described below:

- **Reheating**
  The cooling is done with a constant cooling factor α after a fixed number of iterations per temperature. The temperature is reset to t_{max} if t_{min} is reached. We call this approach `SAR`.
- **Cooling**
  The algorithm tries to estimate the number of iterations left until the timeout, and adapts the cooling factor α, so that by the end of the time t_{min} is reached. The number of iterations per temperature stays fixed. From now on, this version is referred to as `SAC`.
- **Iterations**
  The heuristic considers the speed of the iterations from prior experiments and a constant cooling rate α to estimate how many iterations per temperature are possible, such that the minimal temperature t_{min} is reached at the timeout. This variant is called `SAI` from now on.

To get an even deeper insight into the exact workings of the neighborhoods and the algorithms themselves, as well as their tuning, we again refer to the original work by Moser et al. [2022].

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

3.2.3 Large Neighborhood Search

The other heuristic approach used for comparison in the ISA is an LNS. This approach utilizes the destroy-and-repair method, which is used to optimize parts of a solution, resulting in an improvement of the objective function of the whole solution over many iterations. Our LNS approach is a novel approach since existing LNS methods for PMSP like Ahuja et al. [2002] or Rolim et al. [2023] focus on slightly different constraints, which makes their proposed destroy and repair operators not applicable to our problem.

To control the size of the part of the solution that should be destroyed and repaired in each iteration, we introduce the parameter `job_limit`, which represents a soft limit to the maximal number of jobs that should be destroyed. We chose this approach over the regulation with a parameter that gives the percentage of jobs to destroy, because the exact solver, which is used as a repair operator, can only solve subproblems of small sizes efficiently. Giving a percentage would yield different sizes of subproblems, which might affect the performance of the utilized repair operators.

Because we use an exact solver as a repair operator, we focus on destroy operators, which lead to subproblems that can be optimized in a reasonable time. We aim to select a subset of jobs and a subset of machines that can then be passed as an independent problem to the solver. The resulting subschedule is inserted at the correct place in the full schedule. This is why we exclude the random destroy operator, because there is no real subproblem present. Even if we fix most jobs and only allow the positional change of a few, we would still have to load the whole problem instance into the model, which takes too much time for large instances, leading to a great reduction in iteration steps of the LNS algorithm.

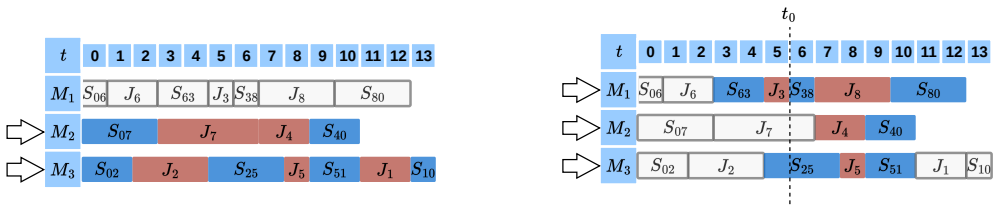
To create subproblems, we take a horizontal or vertical slice out of the currently best schedule. These slices can be adapted to reasonable subproblems in the way described below.

- `Machines`
  This destroy operator randomly selects a machine that is added to the subproblem with the currently scheduled jobs on it. More machines and jobs are added iteratively until `job_limit` is reached or crossed. Note that we actually allow more jobs than the limit suggests, because this way we have an easy and standardized way of handling schedules and do not have to make exceptions for cases where every machine has more jobs scheduled than the limit allows.

  We also introduce an option to weigh the machines for the random selection, by adding up the tardiness and earliness (defined as $\max(0, d_j - C_j),\ j \in J$) of all jobs on the machine. A bigger value suggests that there are more jobs that are not optimally timed (both too early or too late), and rescheduling might be helpful.

  Figure 3.1a visualizes the application of this destroy operator on an example schedule with `job_limit` $= 4$. The arrows indicate the selected machines. White boxes are used for jobs that stay fixed and are not part of the subproblem. We end

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.



(a) `Machines` (b) `Block`

Figure 3.1: Examples of the destroy operators

up with a subset of machines and jobs that can be treated like a normal instance by the repair operators.

- `Block`
  This destroy operator does not focus on single machines but rather on a timestamp. We start by randomly selecting a timestamp t_0 and a random number of machines. To get the jobs included in the subproblem, we order them by distance from their start time to t_0. Now, all, or the closest `job_limit` many of these jobs are selected for the subproblem. Ties are broken randomly. Other than in the `Machines` destroy operator, it might happen that the selected jobs have preceding or succeeding jobs that are not part of the subproblem. To consider that in the subproblem, the initial and final setup times will be adapted so that they include all preceding and succeeding jobs in the subschedule, respectively. To complement this description, the `Block` destroy operator is visualized in Figure 3.1b. Again, white boxes mark the jobs that are fixed and not part of the subproblem, arrows indicate selected machines, and `job_limit = 4`.

We propose three different repair operators to reach a feasible solution after the destruction step.

- `Random`
  This just randomly inserts the jobs on a feasible machine. Fixed jobs stay at their position, so this random insertion only happens in between fixed jobs in case of the `block` destroy.
- `Heuristic`
  This heuristic applies the `Greedy` construction heuristic, which is also used for SA and LNS in Section 3.2. This enables us to get a feasible solution to the subproblem in a short time.
- `CP_C`
  Uses the `CP_C` solver introduced in Subsection 3.1.2 to solve the subproblem. To avoid long runtimes per iteration, we run the solver for 0.5 seconds and return the

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

current best solution. If no solution is found in time, no solution is returned and LNS simply keeps the current best solution and starts the next iteration. To speed up the search, we pass the current schedule of the subproblem as a hint to the solver, so it can start its search from that point on.

The algorithm for LNS is given in 3.1, which takes the parameters described in the list below. The probabilities for the destroy and repair operators should sum to one, respectively.

Algorithm 3.1: Large Neighborhood Search (LNS)

```
1  new ← Greedy Solution;
2  best ← new;
3  while maximal runtime not exceeded do
4  |   destroy ← choice(machine, block);
5  |   subproblem ← destroy(best);
6  |   repair ← choice(random, heuristic, exact);
7  |   new ← repair(best, subproblem);
8  |   if new.cost < best.cost then
9  |   |   best ← new;
10 |   end
11 end
```

- `job_limit`
  Soft upper-bound for jobs used in the subproblem
- `prob_machine_weights`
  Probability that weights are used when randomly selecting machines to destroy
- `prob_machine_dest`
  Probability to use the `machine` destroy operator
- `prob_block_dest`
  Probability to use the `block` destroy operator
- `prob_random_rep`
  Probability to use the `random` repair operator
- `prob_heuristic_rep`
  Probability to use the `heuristic` repair operator
- `prob_exact_rep`
  Probability to use the `exact` repair operator

In the next chapters, we will refer to this optimization approach as `LNS`.

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

CHAPTER 4

Instance Space Analysis and Algorithm Selection for the UPMSP

The experimental setup for the ISA and the algorithm selection will be described in this chapter. We will start with the introduction of the utilized features for the ISA, followed by the instance sets and the benchmarking setup. Finally, the models utilized for algorithm selection are listed with their corresponding tuning parameters.

4.1 Features

To guarantee an instance space that is able to distinguish well between different instances, the proposal of many instance features is necessary. These features can be grouped into different categories. All features are described in the following sections.

4.1.1 Aggregation Functions

Parts of the information that can be extracted from a problem instance come in arrays with length M_c or J_c. For example, there is one due date for each job. So we need ways to aggregate them into a single value that can be used as an instance feature. We apply all the aggregation functions listed in Table 4.1 to these extracted features that consist of a set of values. These aggregation functions are basic statistical measurements, but were selected based on the feature construction by Strassl and Musliu [2022].

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

Abbreviation	Description
mean	Mean value
median	Median value
std	Standard Deviation
min	Minimal value
max	Maximal value
range	Maximal minus minimal value
q1	First quartile
q3	Third quartile

Table 4.1: Aggregation function used for multi-valued features

Abbreviation	Description
job_ct	Number of jobs
mach_ct	Number of machines
mat_ct	Number of materials
u_job_ct	Number of unique jobs
u_mach_ct	Number of unique machines
job_mach_rat	Number of jobs, divided by the number of machines
job_mat_rat	Number of jobs, divided by the number of materials

Table 4.2: Single-valued general features

4.1.2 General Features

We will list features that are not probing or graph features but can be extracted from the instance input directly. They are referred to as general features. They can be split further into single-valued features in Table 4.2 and multi-valued features in Table 4.3.

Table 4.2 lists the general single-valued features. They include very basic features, like the number of jobs or machines, and the ratio between them. The *material* is an abstraction layer introduced by Moser et al. [2022] in their instance generator. Using this concept, setup times are not directly dependent on the job, but rather on the material a job utilizes. Each job has an assigned material, and the material itself has a machine- and sequence-dependent setup time. Note that an equivalent problem instance can be generated without the usage of materials, but it can save a lot of storage space because less redundant information has to be stored. Also, it can be argued that the problem structure is different, with many jobs having the same setup times, rather than completely randomly generated, unrelated ones. While this material count could be calculated from any given instance, we assume that instances generated without this specific feature use different materials for all jobs, since it is very unlikely that all randomly generated setup times of two jobs match.

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

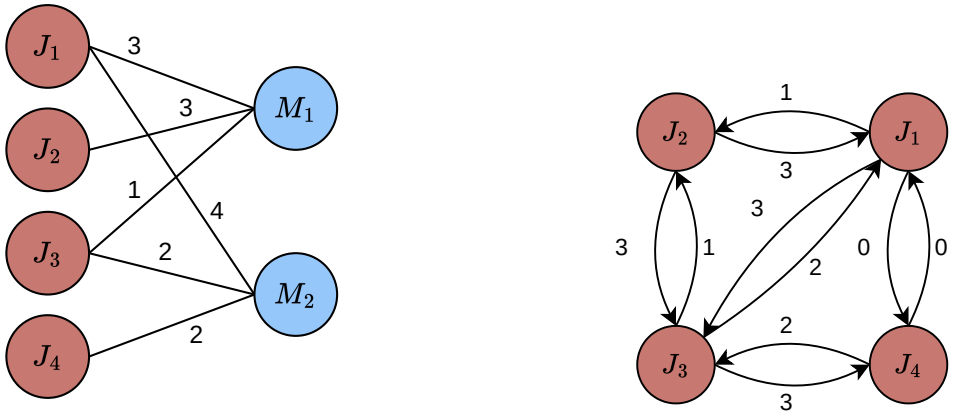
Abbreviation	Description
min_proc_m	Minimal processing time on a fixed machine, varying jobs
max_proc_m	Maximal processing time on a fixed machine, varying jobs
min_proc_j	Minimal processing time of a fixed job, varying machines
max_proc_j	Maximal processing time of a fixed job, varying machines
min_setup_pre	List of minimal preceding setup time before a fixed job
max_setup_pre	List of maximal preceding setup time before a fixed job
min_setup_suc	List of minimal succeeding setup time after a fixed job
max_setup_suc	List of maximal succeeding setup time after a fixed job
min_spare_m	Minimal spare time on a fixed machine, varying jobs
max_spare_m	Maximal spare time on a fixed machine, varying jobs
min_spare_j	Minimal spare time of a fixed job, varying machines
max_spare_j	Maximal spare time of a fixed job, varying machines
elig_m_ct	Number of eligible machines per job

Table 4.3: Multi-valued general features

We also want to explain how the counting of unique jobs works. A job is a duplicate if it has the same processing and setup time on all machines as another job. In u_job_ct, we only count unique jobs, so all duplicates only count once. The same applies to unique machines. Two machines are duplicates if the processing and setup times of all jobs scheduled on them stay the same.

The multi-valued general features can be found in Table 4.3. Because of the two- or three-dimensional data for processing and setup times, we need to aggregate multiple times to get a single-valued feature in the end. To make things clearer, we look at the first multi-valued feature, min_proc_m, and calculate it for the example problem instance from Subsection 2.1.3. The minimal processing time on a fixed machine, varying jobs would mean that we fix the machine M_1, and look at the processing times of all jobs on that machine. The minimal processing time of a job on M_1 is 1. For M_2 it is 2. We end up with the multi-valued feature $(1, 2)$, because the problem has two machines. This

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



(a) `jm_graph` (b) `jj_graph`

Figure 4.1: Extracted graphs from the example problem input 2.1

array is then aggregated further with the functions from Table 4.1. The *spare time* used in some features is defined as $d_j - p_{jm}$. The spare time indicates how many timesteps are available before the job has to be started, so it is still able to finish before its due date.

4.1.3 Graph Features

We also propose graph features for the UPMSP. First, we introduce two graphs derived from the problem input. The first one is a bipartite graph, where one side has jobs as nodes, while the other side consists of machines. The edge weight of edge (j, m) is the processing time p_{jm}. Figure 4.1a shows this for the example problem instance from Subsection 2.1.3. To distinguish this graph from the second one we will introduce, we call it `jm_graph`, since its edges only go from jobs to machines.

The second graph only has jobs as nodes. The arcs are aggregated values of the setup times between them. Because, in general, $s_{ijm} \neq s_{jim}$, the graph is bidirectional. Since the setup time between two jobs is dependent on the machine, there would be multiple values assigned to the same edge. To condense this into a single value, the weight of edge (i, j) is defined as the mean of the setup times, so $\frac{1}{|E_i \cap E_j|} \sum_{m \in E_i \cap E_j} s_{ijm}$. If two jobs have no machine where both jobs can be scheduled, there exists no edge. This graph is called `jj_graph` and Figure 4.1b shows the corresponding graph to the example problem from Subsection 2.1.3.

The same differentiation between single- and multi-valued features also plays a role for graph features. Table 4.4 shows the single-valued graph features. Note that the

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.

Abbreviation	Description
jm_clust	Average clustering of jm-graph
jm_dens	Average density of jm-graph
jj_clust	Average clustering of jj-graph
jj_dens	Average density of jj-graph

Table 4.4: Single-valued graph features

Abbreviation	Description
jm_deg	Node degrees of the jm-graph
jm_wght	Edge weights of the jm-graph
jj_deg	Node degrees of the jj-graph
jj_wght	Edge weights of the jj-graph

Table 4.5: Multi-valued graph features

density and average clustering of the jm_graph use the corresponding definitions for bipartite graphs, which differ from the conventional ones used for general graphs. The implementation we used comes from a framework introduced by Hagberg et al. [2008].

The multi-valued graph features utilized are given in Table 4.5. These are only very basic measurements. This choice was made to avoid extensive time being spent on feature calculation for larger instances, since many graph metrics require global knowledge of the whole graph and have a high computational complexity.

4.1.4 Probing Features

Probing features are a different class of features. They are based on the obtained schedule of an algorithm or solver, within a short amount of time. Our focus lies on probing features that rely on the solution obtained by Greedy. We chose to exclude probing features based on exact methods, since initial experiments showed that for the majority of the utilized instances, no valid solution could be found by any of them in a short amount of time, or they were trivial and optimized immediately. Table 4.6 shows the probing features we include in our analysis.

The first two features measure how well the construction heuristic is able to balance the jobs on the machines, which might be an indicator of the flexibility that an instance allows, regarding the placement of jobs. The naive lower bound mentioned in the third probing feature is calculated by assuming that every job can be scheduled, so that it utilizes its minimal processing and preceding setup time, and a perfect distribution over all machines. So we sum up the minimal processing and setup times for each job and divide that by the number of machines.

In total, we propose 150 features that can be extracted from an instance. Due to the aggregation functions, some of the features make little sense at first glance, and others

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

Abbreviation	Description
`j_bal_ct`	Mean derivation from the average number of jobs per machine
`j_bal_mksp`	Mean derivation from the average machine makespan
`rat_lb`	Ratio of greedy solution makespan to a naive lower bound

Table 4.6: Probing features

might yield duplicate values. However, we do not sort them out by hand, but rather let MATILDA, the online tool for ISAs, take care of this issue by selecting the most useful features for the separation of the instances and algorithmic performance. The few selected features are then further projected down to two dimensions for visualization purposes.

The feature name used in the experiments and results later is either just the feature name as it is in the table for single-valued features, or it is a combination of the aggregation function and the feature from the table for multi-valued features in the format `[feature]_agg_[aggregation function]`.

4.2 Utilized Instances

The ISA is conducted using instances from different sources to get a complete picture of the current state of the available benchmark instances. This set is extended with novel instances generated with an adaptation of an existing instance generator. The following sections will give an overview of the existing instance sets, as well as an explanation of the novel instance generator and the resulting instances.

4.2.1 Existing Instances

During the literature research, we identified two instance sets that can be used. The few other publicly available instance sets for UPMSPs are unusable for our problem description because they lack due dates or other necessary data for the version we focus on.

We have been provided with instances from Perez-Gonzalez et al. [2019] short `pez` in the following, which were initially used for a slightly different problem description, but contain all the necessary data to use them for our problem. These instances are grouped into *small*, *medium*, and *big*, depending on their number of jobs and machines.

Another instance set comes from Moser et al. [2022], referred to as `max` from now on, which includes generated instances, as well as a few real-life instances. The exact description of the implementation of the instance generator can be found in Moser et al.

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.

Name	Description	Number of Instances
pez_s	Small instances from pez	3 840
pez_m	Medium instances from pez	5 760
pez_b	Big instances from pez	6 000
max_gen	Instances from instance generator from max	560
max_rl	Real-life instances from max	28

Table 4.7: Existing instance sets

[2022] if more detailed information is needed. This instance generator is also the basis for the adaptation we propose in Subsection 4.2.2.

In Table 4.7, the different sources, as well as the number of instances, are listed. We will not be able to utilize all of the 16 188 instances for the ISA, because the time to evaluate all of them with all algorithms would exceed our available time. Therefore, we sampled a subset of 400 instances of each group, plus the 28 real-life instances.

4.2.2 Newly Generated Instances

To fill gaps in the instance space and also create more instances that resemble the available real-life instances, we introduce an extension to the instance generator from Moser et al. [2022]. In the first step, we simply generate instances with the available method. Below, we introduce a post-processing routine that is applied to these instances to generate our instance sets.

From manual inspection of the real-life instances provided by Moser et al. [2022], we found that there are only a few setup times different from zero. Also, these non-zero setup times are mostly identical, and when they differ, they remain of the same order of magnitude. This behavior is not reproducible with the existing instance generator. Therefore, we propose a post-processing procedure, where setup times s_{ijm} are randomly set to zero, with probability prob_zero_setup, which is a parameter that can be set manually upon generation.

While the available real-life instances only have one eligible machine per job, we also generate novel instances with multiple eligible machines in combination with the special structure of the setup times. Applying this post-processing to instances that use similar value ranges for job count, machine count, processing times, due dates, and setup times to the real-life instances yields a novel instance set. We also create another set of instances with this method, with value ranges more similar to the other randomly generated instances in the existing instance sets, with the goal of filling gaps in the instance space between them.

Table 4.8 contains the ranges for parameter settings of the original instance generator for the instance set called moik_rll (real-life-like) and moik_gap. For each instance generation, random values are picked from these ranges. Parameters that are not listed 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 Bibliothek Your knowledge hub WIEN

Parameter	Moik_rll	Moik_gap
Number of machines	{3 . . . 20}	{1 . . . 20}
Number of materials	{1 . . . 10}	machine_ct
Number of jobs	{100 . . . 1000}	{20 . . . 320}
Minimal processing time	{100 . . . 200}	{0 . . . 6}
Maximal processing time	{200 . . . 90 200}	{25 . . . 150}
Minimal setup time	{1 000 . . . 50 000}	{0 . . . 60}
Maximal setup time	min_setup + {0 . . . 1 000}	{60 . . . 200}
Multiple eligible machines	{0, 1}	{0, 1}

Table 4.8: Parameters different from default, utilizing the instance generator from Moser et al. [2022] to generate both new instance sets

set to their default value. For more information regarding the initial instance generator, we refer to the original work by Moser et al. [2022]. The post-processing parameter prob_zero_setup is also chosen randomly per instance, ranging between zero and one.

The naming scheme of our instances is [# of machines]_[# of materials]_[# of jobs]_[multiple eligible machines].max.

In total, moik_gen contains 798 instances and moik_gap 764, so combined 1562 novel instances for the UPMSP. In combination with the sampled instances from the existing instance set, we have 3190 instances that will be part of the ISA. For the algorithm selection, we split the instances into a training and a test set. The size of the test set is 20%, so 638 instances. The feature selection, projection into the instance space, and the training of the algorithm selection models will be performed just using the training set with the other 2552 instances. The test set is only used for the evaluation of the algorithm selection models.

4.3 Performance Measurements

To project the instances into a two-dimensional space, where instances are separated by their hardness, we need an objective measurement of how well an algorithm performs. For this, we cannot simply use the objective function. For example, if we take two instances that differ only by the scaling of each due date, processing time, and setup time by some factor, the objective function would also scale by that factor for the same schedule. Because the algorithms are not influenced by the scaling of the input data in an appropriate range, this would lead to the wrong conclusion about the goodness of an algorithm or the hardness of an instance. In the following, we discuss performance measurements considered by us.

To combine the two separate objectives into a single value, we utilize the same strategy that is used in Subsection 3.2.2, which is the weighted objective function $P = 10\,000 \cdot T + C_{max}$.

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.

While this weighted value does not correspond to a perfect lexicographical ordering, it leads to correct results in the vast majority of cases. This choice was mainly made because the usage of a multiplying constant, which ensures that the objectives do not interfere with each other, leads to large numbers, which resulted in difficulties for MATILDA in the instance space projection.

There are a few ways to approach the problem of the scale invariance we want to achieve. The optimal one would be to normalize it by the weighted objective value of the optimal solution P^* as in Equation (4.1). But this is not possible for all instances, because we would need to know the optimal solution beforehand, rendering this algorithm comparison as a whole useless. Below are other ways that try to approximate this optimal measurement of goodness of an algorithm that were considered for further experiments.

$$RO = \frac{P}{P^*} \tag{4.1}$$

- **Ratio to Best (RB)** The performance measurement is given by Equation (4.2), which is calculated by taking the best objective value reached by one of the utilized methods and normalizing all others by that performance. This yields performance values in the range $[1, \infty)$.

$$RB = \frac{P}{P_{Best}} \tag{4.2}$$

- **Ratio to Greedy (RG)** The second measurement in Equation (4.3) works in the same way as the first one but takes the objective value of the solution found by the `Greedy`. This gives performance values in $(0, \infty)$ because the exact methods might yield worse solutions than the construction heuristic.

$$RG = \frac{P}{P_{Greedy}} \tag{4.3}$$

- **Ratio to Upper Bound (RU)** Finally, we propose a metric that does not rely on the objective values of solutions by other algorithms. We normalize the objective values of the various algorithms by the upper bound, which was introduced as the *horizon* in connection with the modeling of the objective function of the CP models in Subsection 3.1.2. Because the tardiness of a job cannot be more than the makespan upper bound, we get an upper bound for the total tardiness with $J_c \cdot horizon$. Combining these two upper bounds to a single value allows us to calculate the ratio as is done in Equation (4.4). This metric results in values in the range (0, 1].

$$RU = \frac{P}{P_{UB}} \tag{4.4}$$

While we implemented and tested these different performance measurements, we, unfortunately, have no appropriate instances with known optimal solutions to find out

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

experimentally, which one correlates best with the optimal measurement P^*. To conduct meaningful experiments, we would require non-trivial instances with known optima that did not initially come from our own methods. If we only utilize instances where we already know that there is at least one of our methods that finds the optimal solution, it would always yield $RB = \frac{P}{P_{best}} = \frac{P}{P^*} = RO$. Only the cases where none of our methods are able to find the optimal solution, but can be compared to the known optimum, would reveal the real strengths of each performance metric.

We decided to use RB as the performance metric for the ISA. This decision is based on the analysis of different performance metrics for utilization in the ISA for the Job Shop Scheduling Problem by Strassl and Musliu [2022]. There, the metric P_6, which is RB in our case, outperformed all other proposed measurements easily. The Job Shop Scheduling Problem is, as the name suggests, from the same problem domain as the UPMSP, which supports the choice of this performance metric for our problem.

4.4 Algorithm Tuning

We will briefly go into the tuning and parameter choices made for the algorithms described in Chapter 3.

For the exact solvers, we did not change any of the default parameters, but the number of search workers. The default parameters of these models are very reasonable for most use cases. A higher number of workers, meaning parallel searches, usually leads to better results and is therefore set to eight.

For the SA variants, we can confidently use the same parameters as the tuning from Moser et al. [2022] yielded because our experiments run on the same machine setup as the original experiments.

Finally, the tuning of `LNS` was done by manually adjusting the tuning parameters and evaluating instances. The largest impact on performance was shown by `job_limit` and `prob_machine_weights`. Too many jobs in the subproblem would yield no improved results by the exact solver in the given time, while too few did not allow for an improvement over the current schedule, since it was already optimal most of the time. For the probabilities of destroy and repair operators, we simply used equal distribution, since our manual testing yielded no clear improvement for any other setting. We simply balance their probabilities. Note that the probabilities for destroy and repair operators must sum up to one, respectively, to get a valid configuration. The chosen parameters are listed in Table 4.9. Also, the parameter spaces utilized for tuning are listed there for completeness.

4.5 Algorithm Evaluation

The experiments to get the performances for each algorithm on all instances are performed on a computing cluster with 13 nodes, each featuring two Intel Xeon E5-2650 v4 CPUs

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

Parameter	Range	Tuning Result
job_limit	$\{5 \dots 30\}$	10
prob_machine_weights	$[0, 1]$	0.5
prob_machine_dest	$[0, 1]$	0.5
prob_block_dest	$[0, 1]$	0.5
prob_random_rep	$[0, 1]$	0.33
prob_heuristic_rep	$[0, 1]$	0.33
prob_exact_rep	$[0, 1]$	0.33

Table 4.9: Tuning parameters of LNS and the utilized settings

(12 cores @ 2.20GHz) with 40GB of RAM. The MIP formulations are implemented and executed with Gurobi 12.0.0 (Gurobi Optimization, LLC [2024]), and the CP models are implemented for CP-SAT from OR-Tools 9.11 (Perron and Didier [2024]).

The runtime of all algorithms is capped at 15 minutes to allow a fair comparison. We also evaluate each of the SA algorithms eight times on each instance and select the best one, to make it comparable to the exact solvers, which utilize eight cores in parallel.

It is possible that no solutions will be found in the given time. In this case, we replace the missing values with naive upper bounds. The *horizon* introduced in Subsection 3.1.2 is already an upper bound for the makespan, and by multiplying it by the number of jobs, it yields an upper bound for the cumulative tardiness as well.

For the SA approaches, we choose the best of the eight performances for further analysis, to account for the fact that the exact solvers utilized eight cores in parallel. Note that LNS utilizes an exact solver and therefore also eight threads in parallel, and is only evaluated once.

4.6 Feature Selection and Projection

We decided to conduct two separate analyses for the exact and heuristic methods. This choice, after initial experiments, showed that on the majority of instances, the heuristic methods outperformed or reached the same result as the exact methods, making a comparison obsolete. However, one might argue that the exact methods are able to prove optimality, unlike the heuristics. So, their application yields a different value than just the solution alone, which is also beneficial in some contexts.

The performance metric *RB* described in Section 4.3 is applied to these results. This performance metric, together with the extracted features of each instance, is the input needed for the ISA tool MATILDA. We utilized the web interface of MATILDA for our experiments, rather than running it locally. Regarding the exact solvers, we decided to exclude instances where none of them could find any solution within the given time. These instances do not allow for a good performance measurement and would only distort

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

the constructed instance space. In the case of the heuristic methods, this is not necessary since they always yield a valid solution, starting with the greedy construction heuristic.

For the ISAs, we mostly use the default values from MATILDA. We set the *Performance Threshold* to 5% for both of them. The number of features that should be selected for the projection is also set to six for both ISAs.

4.7 Algorithm Selection

The goal of the algorithm selection is to choose the most promising algorithm out of a set of algorithms for a given problem instance. In our case, this is done by using the instance features as input for the model.

Because this is a basic classification task, we expect to get good results using simple models and therefore refrain from using more complicated models, such as Deep Neural Networks. Also, the available data would not suffice to train larger models, limiting the options to choose from to the simpler ones.

4.7.1 Models

Below is a brief description of all the utilized models for Algorithm Selection on the UPMSP. To find optimal parameters, a grid search with 5-fold cross-validation was performed for each model. The tuning parameters are listed in separate tables. If a parameter is not mentioned, it means that the default value was used. The models were implemented and fitted using `scikit-learn 1.6.1` (Pedregosa et al. [2011]).

Most Frequent Classifier
The baseline for the other algorithm selection models is the classifier, which always predicts the class that appears most often in the training set. This classifier has no relevant parameters. From now on, we refer to it as `MF`.

K Nearest Neighbors
The K Nearest Neighbors classifier, short `KNN`, makes predictions based on the k closest instances measured by some metric. Because the distance metrics might depend on scaling, the input is normalized before fitting the algorithm. Table 4.10 shows the parameters that will be tuned and the corresponding options available in the grid search.

The parameter p is a part of the Minkowski distance. For $p = 1$, we get the Manhattan metric, $p = 2$, the Euclidean metric, and with $p \to \infty$, it approaches the Maximum metric, also called Chebyshev distance.

Random Forrest
The random forest, called `RF` from now on, is an ensemble of many decision trees and makes predictions by the majority of the results of the decision trees. This can improve the accuracy and reduce the overfitting compared to a single decision tree. Parameters for this model are listed in Table 4.11.

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

Parameter	Range
n_neighbors	$\{3, 5, 7, 9\}$
weights	uniform, distance
p	$\{1, 2, \infty\}$

Table 4.10: Tuning parameters for KNN

Parameter	Range
n_estimators	$\{100, 200, 500\}$
criterion	gini, entropy, log_loss
min_samples_leaf	$\{1, 2, 5\}$

Table 4.11: Tuning parameters for RF

Parameter	Range
C	$\{1, 2, 5\}$
kernel	linear, poly, rbf, sigmoid
degree	$\{2, 3, 4\}$ (only relevant if kernel=poly)
gamma	scale, auto

Table 4.12: Tuning parameters for SVM

Support Vector Machine
A support vector machine, short SVM, finds a hyperplane that maximizes the margin between different data classes. Utilizing kernel functions, they can also handle non-linear separation problems. Table 4.12 lists the considered parameters in the tuning process.

4.7.2 Evaluation

The training labels are multi-labels because, in some cases, two or more algorithms reach the same solution. The model predicts multiple classes, which will be scored using the subset accuracy. A prediction is seen as correct if it matches the true labels completely. But because we want a single algorithm selected per instance in the end, we have to break ties in some way. For this, we decided to sort the algorithms by the number of times they performed best on the instances from the training set and break ties by favoring the first method occurring in this order.

In addition to the subset accuracy measurements on the test data, we also report more detailed insights into the results by including other classification measurements, confusion matrices, and comparing the models to each other by the number of correct predictions they made after using our tie-breaking approach. The best among these models is used for the final evaluation and comparison to the performance of individual algorithms.

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

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

CHAPTER 5

Experimental Evaluation

In this chapter, we discuss the results obtained from the experiments using the experimental setup described in Chapter 4.

5.1 Algorithm Performance

Before we analyze the instance space and evaluate the algorithm selection models, we benchmark the exact methods utilized and compare them to each other on a small set of instances, which was also used by Moser et al. [2022] for this purpose. These experiments are executed on a machine with an Intel i7-1355U CPU with 10 cores and 32GB of RAM. The runtime is set to 30 minutes, as described in Moser et al. [2022]. A comparison of the performance of our reimplemented MIP models to the reported results in the original work does not make sense, since the utilized versions of Gurobi differ, as well as the hardware specifications.

However, for the SA approaches, we are able to compare our reimplementation directly to the original work, since we can run the algorithms for the same number of iterations, making it hardware independent. This benchmarking should be seen as a validation of the correct reimplementation of the algorithms and not as a direct comparison of their performance with each other, since this will be done in the context of the instance space in the following sections.

5.1.1 Exact Methods

We report results from our implementation of the MIP formulations for Gurobi and the two CP formulations implemented for CP-SAT. Table 5.1 contains the results for a set of 25 instances that were also utilized by Moser et al. [2022].

The first line of the result shows the cumulative tardiness of the best solution found, with the best bound in brackets behind it. The second line contains the makespan and its

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

best bound as well. Due to the nature of the multidimensional objective function, we do not have a valid bound for the makespan if the tardiness is not already optimal. Because of this, many bounds are actually not available, which is represented by a dash (-). The best performing solver is highlighted by using a bold font, while objective values that are proven to be optimal are marked with a star (*).

By looking at the objective values obtained by the exact methods, we can see that for most instances, `CP_S` finds the best solution among them. Instances where `MIP_A` or `MIP_H` outperform the CP approaches seem to have more machines and jobs than the others, while instances with many jobs in combination with very few machines favor CP-SAT. However, no clear pattern can be found only using 25 instances, so this hypothesis will be discussed again in combination with the instance space in the following sections.

There exist two instances where all models were able to find and prove the optimality of a solution. In addition to that, the CP models were able to find a third optimal solution. Also, for most instances, the best bounds found by `CP_C` and `CP_S` are better than the bounds for `MIP_A` and `MIP_H`. At least on these 25 instances, the CP formulations proposed by us seem to perform better than the existing MIP formulations utilized by Gurobi.

5.1.2 Simulated Annealing

We perform similar validation benchmarks for the SA approaches that were reimplemented on the description by Moser et al. [2022]. Although the randomness prevents the results from matching exactly, our versions are able find solutions of similar quality most of the time. In Table 5.2, these results are reported for all three SA variants. Again, the best result is highlighted by a bold font for each of the three versions separately, and the first value represents the cumulative tardiness of the found solution, while the second value is the makespan. In the case of SA, we do not have any lower bounds to the solution compared to the exact solvers above. The 25 instances were drawn randomly from the set of instances where solutions were provided for the original implementations.

For `SAC`, our method yields better results for 10 of the 25 instances and is generally not too far off on the others. The same applies to `SAR`, where our implementation yields 15 better and one equally good result as the originally reported results. However, with the re-implementation of `SAI`, we only improve results over the reported ones in four cases and tie for one. This suggests that our implementation performs worse than the original one. However, this might be due to the parameter tuning, because these benchmarks were run on a different machine. `SAI` is especially affected by this because of the fixed iteration budget $\mathcal{I}$ that was chosen based on previous experiments in the original work. While this first benchmark was performed on the setup described at the beginning of this section, the final experiments, yielding the results used for the instance space construction and analysis, will be conducted on the same setup as Moser et al. [2022] utilized, so we still include our implementation of `SAI` without further adaptation.

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

Instance	MIP_A		MIP_H		CP_C		CP_S	
p-13-80-80-1	1189	(117)	1430	(159)	494	(363)	**363** *	(363)
	488	(-)	520	(-)	352	(-)	**361**	(33)
p-15-60-60-1	712	(225)	674	(206)	**538**	(458)	567	(475)
	350	(-)	394	(-)	**334**	(-)	367	(-)
p-15-63-80-1	841	(29)	1218	(35)	583	(283)	**515**	(294)
	428	(-)	492	(-)	456	(-)	**390**	(-)
p-16-100-100-1	2499	(2)	1181	(2)	615	(111)	**598**	(117)
	500	(-)	516	(-)	387	(-)	**481**	(-)
p-16-180-180-1	3355920	(0)	**21682**	(3)	77456	(133)	394695	(156)
	18644	(-)	**1010**	(-)	3613	(-)	9982	(-)
p-17-100-100-1	4714	(1)	3183	(68)	1597	(661)	**1357**	(666)
	485	(-)	442	(-)	392	(-)	**454**	(-)
p-18-80-80-2	1090	(23)	1153	(81)	874	(465)	**684**	(483)
	390	(-)	405	(-)	402	(-)	**327**	(-)
p-20-180-180-1	**16762**	(1)	21009	(1)	158148	(174)	120640	(225)
	870	(-)	971	(-)	6215	(-)	5409	(-)
p-22-140-140-1	8448	(1)	8090	(1)	3253	(194)	**2939**	(64)
	605	(-)	651	(-)	584	(-)	**593**	(-)
p-29-140-140-1	5446	(1)	5746	(2)	**4698**	(100)	5073	(207)
	379	(-)	458	(-)	**478**	(-)	479	(-)
p-3-17-20-1	**937**	(303)	**937**	(318)	945	(632)	945	(609)
	525	(-)	**525**	(-)	568	(-)	568	(-)
p-7-19-40-1	1929	(175)	1672	(149)	1859	(685)	**1618**	(708)
	553	(-)	535	(-)	455	(-)	**458**	(-)
p-9-180-180-1	42396	(1)	38253	(48)	**8815**	(291)	11923	(333)
	1816	(-)	1774	(-)	**1636**	(-)	1596	(-)
s-1-3-100-1	63208	(0)	104804	(0)	**0** *	(0)	**0** *	(0)
	8568	(-)	9444	(-)	**10746** *	(10746)	**10746** *	(10746)
s-10-120-180-1	21195	(0)	17664	(0)	8372	(0)	**316**	(0)
	1761	(-)	1811	(-)	1853	(-)	**1787**	(-)
s-15-80-80-2	**0** *	(0)	0 *	(0)	0 *	(0)	0 *	(0)
	346	(217)	364	(219)	353	(149)	367	(61)
s-15-80-80-3	0 *	(0)	**0** *	(0)	0 *	(0)	0 *	(0)
	307	(209)	**300**	(208)	389	(80)	401	(99)
s-22-149-160-1	3278	(0)	**1589**	(0)	64026	(0)	101822	(0)
	780	(-)	**743**	(-)	3932	(-)	4818	(-)
s-4-16-20-1	**0** *	(0)	**0** *	(0)	**0** *	(0)	**0** *	(0)
	409 *	(409)	**409** *	(409)	**409** *	(409)	**409** *	(409)
t-10-24-40-1	**0** *	(0)	**0** *	(0)	**0** *	(0)	**0** *	(0)
	373 *	(373)	**373** *	(373)	**373** *	(373)	**373** *	(373)
t-15-77-80-1	119	(0)	36	(0)	0 *	(0)	**0** *	(0)
	737	(-)	502	(-)	378	(105)	**358**	(45)
t-18-56-100-1	1400	(0)	504	(0)	56	(0)	**43**	(0)
	521	(-)	563	(-)	435	(-)	**890**	(-)
t-20-76-100-1	446	(0)	222	(0)	267	(0)	**0** *	(0)
	458	(-)	528	(-)	348	(-)	**397**	(31)
t-28-34-100-1	85	(0)	**0** *	(0)	84	(0)	52	(0)
	514	(-)	**231**	(199)	365	(-)	407	(-)
t-3-12-200-1	204137	(0)	175792	(0)	**25675**	(0)	80054	(0)
	6433	(-)	7035	(-)	**4790**	(-)	6872	(-)

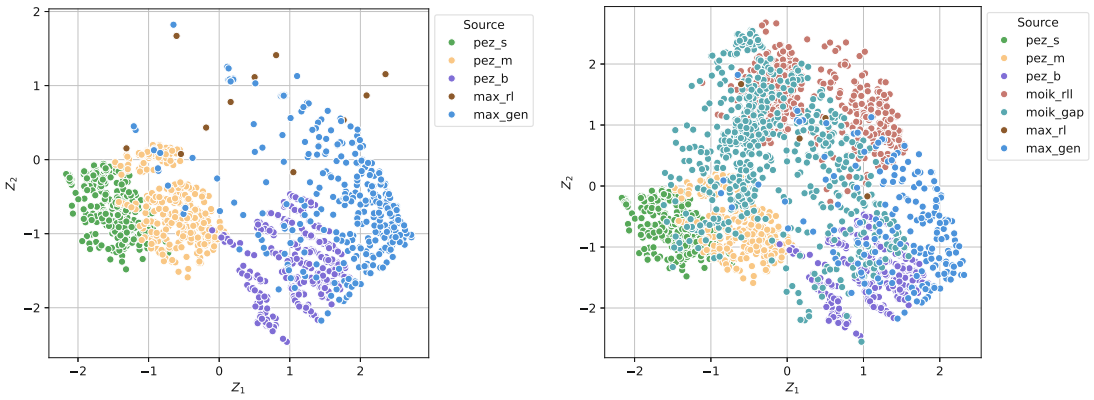
Table 5.1: Performance results of the reimplemented MIP models and novel CP formulations

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

Instance	SAC	SAC (Moser)	SAI	SAI (Moser)	SAR	SAR (Moser)
t-18-820-820-2	28 2645	**20** **2094**	**20** **2547**	45 2557	**25** **1960**	25 2071
s-18-763-800-1	0 2543	**0** **2485**	0 3203	**0** **2799**	**0** **1976**	0 2094
s-22-519-740-1	**0** **1549**	0 1801	0 2268	**0** **1916**	**0** **1451**	0 1503
t-6-540-540-1	22 5532	**6** **4055**	81 5723	**19** **4641**	**36** **4439**	54 4243
s-7-166-600-1	0 4882	**0** **4100**	80 5984	**0** **4304**	67 4072	**0** **4072**
s-12-720-720-1	**0** **2788**	0 3084	0 4339	**0** **3794**	**0** **2662**	0 2846
p-30-332-340-1	**419** **589**	434 563	461 640	**396** **489**	510 566	**431** **552**
t-10-230-700-1	**3** **3774**	8 3144	6 4049	**1** **2970**	14 3222	**2** **3137**
s-14-940-940-1	**0** **3479**	0 3539	0 4842	**0** **4716**	**0** **2997**	0 3141
t-2-490-800-1	8 20990	**0** **21485**	**0** **30450**	0 30975	**0** **22044**	0 31120
t-2-380-380-1	0 11130	**0** **9347**	4 13600	**0** **12374**	**0** **9542**	0 9835
t-20-76-100-1	0 336	**0** **298**	0 303	**0** **282**	0 275	**0** **270**
p-10-376-680-1	**451** **4299**	518 3235	508 4250	**418** **4140**	**410** **3553**	438 3590
t-15-77-80-1	0 414	**0** **330**	0 341	**0** **317**	0 321	**0** **313**
t-28-620-620-1	29 1368	**4** **1037**	11 1235	**2** **1011**	68 979	**3** **934**
s-20-794-800-1	**0** **1862**	0 2224	0 2836	**0** **2616**	**0** **1756**	0 1873
t-26-334-480-1	**58** **1065**	93 823	**5** **991**	49 720	**5** **752**	52 1045
t-24-400-400-1	38 981	**26** **733**	34 903	**34** **663**	90 693	**34** **678**
s-27-249-660-1	**0** **1119**	0 1317	0 1613	**0** **1089**	**0** **1022**	0 1086
s-24-900-900-1	**0** **1729**	0 2016	0 2721	**0** **2701**	**0** **1592**	0 1731
t-28-371-520-1	9 935	**5** **800**	0 943	**0** **714**	0 743	**0** **741**
p-21-62-740-1	572 1517	**392** **1601**	555 2045	**444** **1929**	**494** **1673**	562 2151
t-10-389-940-1	75 4321	**63** **4424**	60 5635	**45** **4894**	**56** **4388**	156 4706
t-27-360-360-1	2 792	**0** **623**	**4** **685**	22 544	4 579	**2** **550**
s-4-16-20-1	0 422	**0** **409**	**0** **409**	**0** **409**	**0** **409**	**0** **409**

Table 5.2: Results of the reimplemented SA approaches compared to the reported results in Moser et al. [2022], with the same number of iterations

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



(a) Existing instances

(b) All utilized instances

Figure 5.1: Instances separated by their source in the instance space for exact solvers

5.2 Instance Space Analysis for Exact Methods

In the following, we present and interpret the obtained results by analyzing the instance space in the context of exact methods. As already mentioned in Section 4.6, we exclude instances where none of the solvers were able to find a feasible solution. In total, the dataset utilized to create the instance space contains 2254 instances. This means that in total, 296 instances were removed from the whole set. The vast majority of these removed instances come from the source `max_gen`, which contains the generated instances proposed by Moser et al. [2022]. The reason no solution was found by any solver is that the instances simply were too large. The order of magnitude where this occurs is a combination of around 500 or more jobs with dozens of machines.

Instance Source Visualization

Figure 5.1 visualizes the projected instances in the instance space, using colors for different instance sources. On the left side, only the instances known from the literature are visualized, while the right figure also contains our generated instances.

One can see that the real-life instances from `max_rl` are placed separately from most of the known randomly generated instances. Also, there is a substantial gap between them.

The additional instances we propose lead to a more complete coverage of the instance space. `moik_rll` overlaps quite well with the position of the instances from `max_rl`. This indicates that we are able to generate instances with similar properties to the real-life ones, which is crucial to evaluating algorithms in this region of the space. Also `moik_gap` is able to fill the space between the real-life and generated instances, with only small gaps left.

However, it overlaps quite heavily with `moik_rll` around $Z_1 = 0.5$ and $Z_2 = 1,5$ as it can be seen in Figure 5.1b. This might indicate that more than six features might be

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.

Feature	Z_1	Z_2
mat_ct	0.162	−0.3945
job_mach_rat	0.6399	0.6867
min_proc_j_agg_median	−0.1323	0.2524
elig_m_ct_agg_mean	0.5782	−0.1024
jj_deg_agg_q3	0.0685	0.0174
j_bal_mksp	0.3173	−0.0912

Table 5.3: Features with their corresponding projection coefficients obtained by MATILDA, for the instance space for exact solvers

needed to separate these instances, or additional features have to be proposed. Also, there is still a sparse area at the center around $Z_1 = 0$ and $Z_2 = 0$, with very few instances, indicating that more instances should have been generated.

5.2.1 Selected Features

The selected features by MATILDA are listed with their corresponding projections in Table 5.3. With Figure 5.2, visualizing the feature values in the instance space, and Figure 5.3 focusing on the feature distribution per source, we can interpret the selection. Note that the feature values in all graphics are preprocessed using a Box-Cox transformation and normalization, as it is utilized by MATILDA itself (Smith-Miles and Muñoz [2023]). The correlation matrix in Figure 5.4 also adds to the analysis of the selected features.

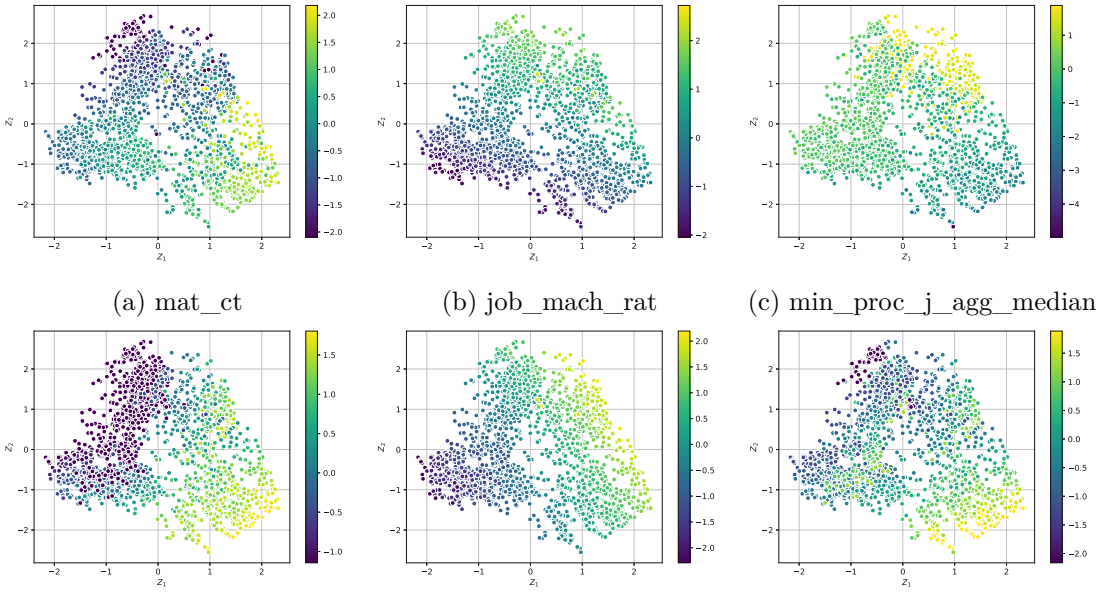
- `mat_ct` The material count of an instance seems to be highest for instances from `pez_b` and `max_gen`, as it can be seen in Figure 5.3a. A low number of materials means that there are fewer unique setup times for the jobs, which might make scheduling easier for some approaches. Of course, a high number of materials is only possible in combination with many jobs, so there exists a correlation between these basic features.

- `job_mach_rat` The job-machine-ratio, visualized by source in Figure 5.3b, is lower for instances from `pez_s`, `pez_m`, and `pez_b` than for the other instance sets, allowing a separation between instances from `pez` and the other sources.

- `min_proc_j_agg_median` The median of the minimal processing times per job is able to distinguish between generated and real-life, or real-life-like instances quite well, as it can be seen in Figure 5.3c.

- `elig_m_ct_agg_mean` The average number of eligible machines shows a gradual increase from the top left to the bottom right in Figure 5.2d. Since there are instances with only a single eligible machine per job, this feature might be used to separate them from the others. In Figure 5.3d, we can see this, since `max_rl`, and

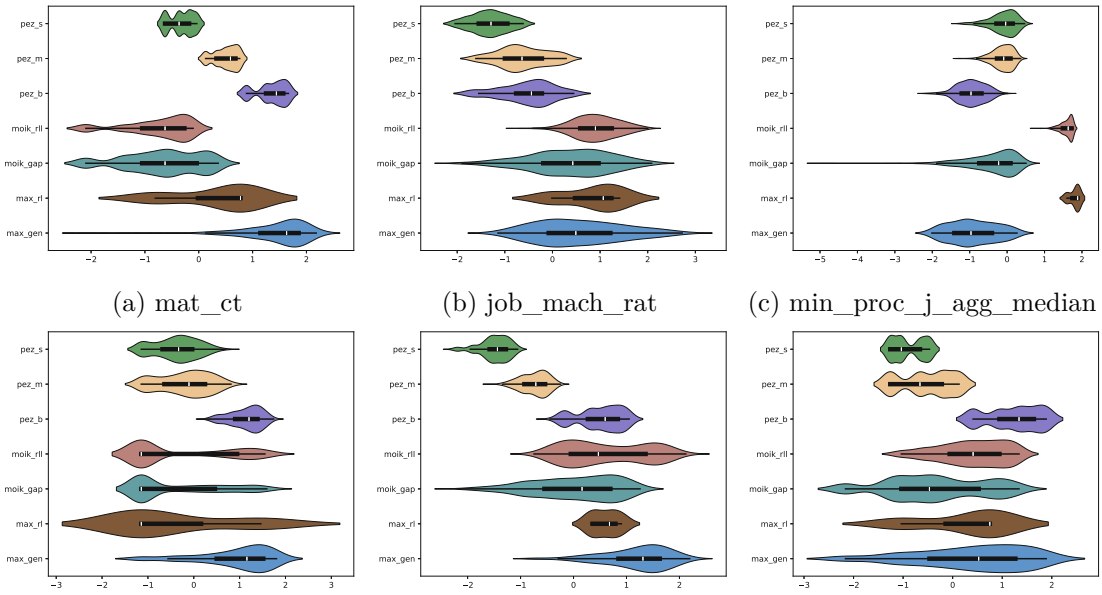
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) mat_ct (b) job_mach_rat (c) min_proc_j_agg_median

(d) elig_m_ct_agg_mean (e) jj_deg_agg_q3 (f) j_bal_mksp

Figure 5.2: Processed feature values visualized in the instance space for exact solvers



(a) mat_ct (b) job_mach_rat (c) min_proc_j_agg_median

(d) elig_m_ct_agg_mean (e) jj_deg_agg_q3 (f) j_bal_mksp

Figure 5.3: Distribution of features split by source

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.

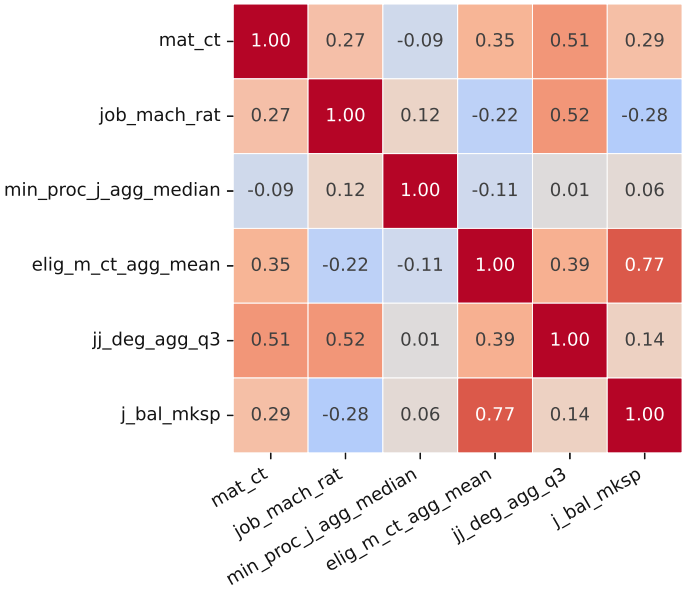


Figure 5.4: Correlation between the selected features for the instance space construction for exact solvers

our generated datasets show an accumulation of instances at the lower end, and also slightly at the upper end, with fewer instances in between.

- `jj_deg_agg_q3` The third quartile of the node degrees of the `jj_graph` correlates with `mat_ct` and `job_mach_rat` when inspecting the correlation matrix in Figure 5.4. When looking at the feature values in the instance space in Figure 5.2e, this feature increases from the bottom left to the top right. The node degree of this graph depends heavily on the number of jobs, but also indirectly on the eligible machines per job, since edges connect jobs running on the same machines. It might be a good representation of these two fundamental properties of an instance.
- `j_bal_mksp` The balancing of the machine makespans correlates a lot with `elig_m_ct_agg_mean` as Figure 5.4 shows. However, it is still interesting to see a possible connection between the number of eligible machines and the ability to distribute jobs equally on the machines by the construction heuristic.

5.2.2 Algorithm Performance

We shift the focus onto the performance of the individual algorithms in the instance space.

Figure 5.5 visualizes where an algorithm reaches the best solution among all other algorithms. It is further distinguished between a uniquely best solution and a tied best solution. Tied means that there is another algorithm that was able to find a solution 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.

the exact same objective value, whereas unique means that this algorithm was the only one that found the best solution among all algorithms. Grayed out instances indicate that the algorithm was not able to find the best known solution, and another algorithm outperformed it.

First, for the data sources `pez_s`, `pez_m` as well as parts of `moik_gen`, all algorithms perform equally well for most instances. This is a strong indication that optimal solutions were found for these instances. But one can see that there are some regions where the algorithms perform differently.

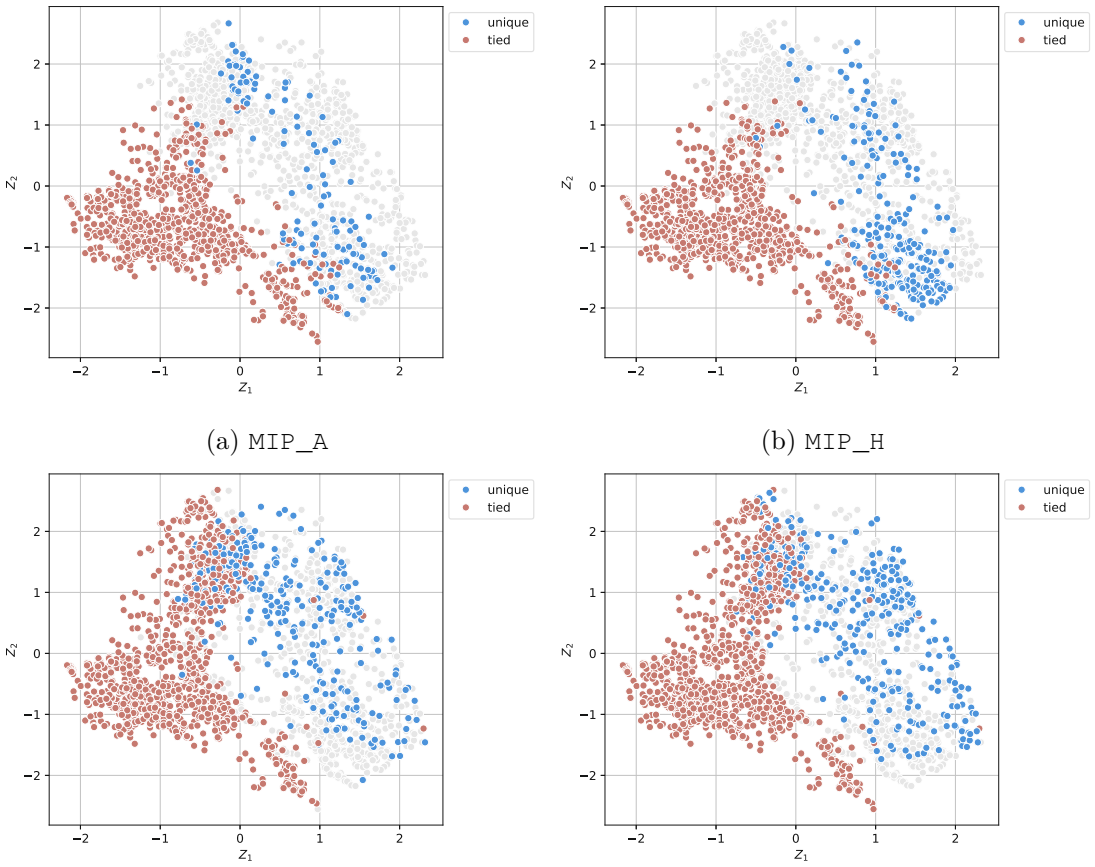
At the top of the instance space in Figure 5.5, around $Z_1 = 0.5$ and $Z_2 = 1.5$, both MIP variations perform worse than the CP approaches. However, `CP_C` and `CP_S` tie most of the time, and are able to produce a few uniquely best solutions in this area. With the feature distribution from the last section in mind, this region is characterized by its low values in `elig_m_ct_agg_mean`, in combination with higher values of `jj_deg_agg_q3`. So the CP models seem to have an advantage over the MIP approaches for instances with many jobs, but a small number of eligible machines per job. The fact that they are able to tie on so many instances in this area also suggests that they are able to find optimal solutions for many of these instances.

On the other hand, instances located around $Z_1 = 1.5$ and $Z_2 = -1.5$ in 5.5 are dominated by the MIP approaches. Especially, `MIP_H` is able to find many unique best solutions. This area mainly includes instances from `pez_b`, but also from `max_gen` and `moik_gap`. The instances in this area can also be described by using the same features again. A high value in `elig_m_ct_agg_mean`, in combination with a medium value of `jj_deg_agg_q3`, places an instance in this part of instance space. But with even higher values for the latter, the CP methods start to outperform the MIP solvers again. However, there is no clear distinction between `CP_S` and `CP_C`, meaning that either the features do not allow a clear separation, or they perform equally well in general.

So in total, `jj_deg_agg_q3` seems to have the biggest influence on the algorithm performance overall. For small values, all solvers are able to find an equally good solution, while for large values, CP yields better results than both MIP formulations. There is a region in between, where MIP and CP solvers outperform each other, depending on `elig_m_ct_agg_mean`.

Figure 5.6 shows the information from the plots above condensed by counting the number of algorithms that were able to find the best known solution for each instance. The hardness seems to increase from left to right in the instance space, looking very similar to the distribution of `jj_deg_agg_q3` in Figure 5.2e. Figure 5.1b shows that `max_gen`, as well as parts of `pez_b`, `moik_gen`, and `moik_rll` are projected to this area, where only one solver finds a good solution. Note that a good solution does not automatically mean that the solution is optimal, or close to the optimum.

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



(a) MIP_A (b) MIP_H

(c) CP_C (d) CP_S

Figure 5.5: Best performances of each exact approach

5.2.3 Algorithm Selection

Finally, we come to the training and evaluation of the algorithm selection models. Table 5.4 lists the resulting tuned parameters from the grid search cross-validation of the models described in Section 4.7.

Table 5.5 shows the F_1-scores achieved by the trained models on the test set, which contains 575 instances. By looking at these scores, one can see that all three classifiers are able to outperform MF. However, the differences in F_1-scores are not that high. This is simply due to the large number of instances, where multiple approaches are able to the same best solution, and the choice does not matter. This favors the most frequent classifier MF.

Figure 5.7 compares the number of instances where the models were able to choose one of the best-performing algorithms on the test set. Note that we shifted the y-axis to exclude instances where all algorithms were able to find the same best solution, and

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

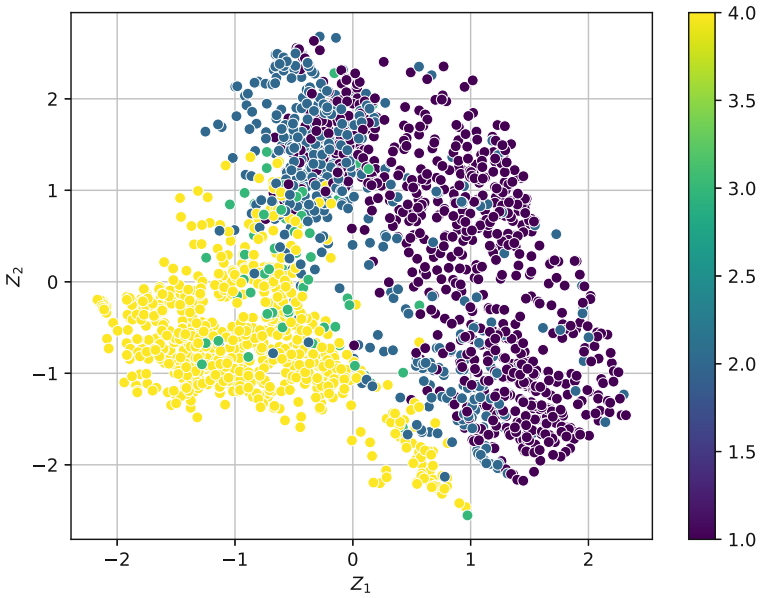


Figure 5.6: Number of good exact solvers per instance

Model	Parameters	Tuning result
KNN	n_neighbors	7
	weights	distance
	p	1
RF	n_estimators	200
	criterion	entropy
	min_samples_leaf	2
SVM	C	5
	kernel	rbf
	degree	-
	gamma	scale

Table 5.4: Tuned Parameters for the algorithm selection models for exact solvers

	MF	KNN	RF	SVM
MIP_A	0	0.90	0.91	0.90
MIP_H	0.71	0.86	0.86	0.86
CP_C	0.76	0.85	0.86	0.84
CP_S	0.78	0.87	0.87	0.87

Table 5.5: F_1-scores of the algorithm selection models for the exact solvers

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

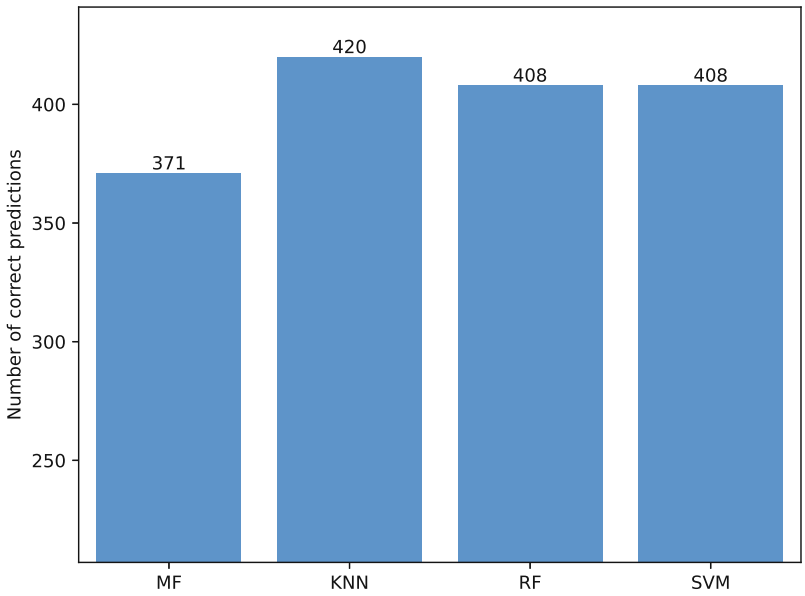


Figure 5.7: Comparison of the number of correct predictions of the algorithm selection models on the test set

therefore, algorithm selection does not have any impact. The plot shows that the trained models are able to outperform the baseline. Again, since there are a lot of instances where two or more algorithms perform equally well and the choice does not matter, the differences are only minor. The best model, the KNN classifier, selects one of the best exact approaches for 420 out of the 575 instances, while the baseline finds 371. This is an increase of around 8%.

The confusion matrices for the predictions of the KNN model are visualized in Figure 5.8. One can see that the model is able to make correct predictions most of the time for all algorithms. Both CP versions outperform the MIP models. And there is also a difference between the models of the same type. MIP_H performs better than MIP_A, as it was already suggested in the reported results by Moser et al. [2022]. Our approach, CP_C, which utilizes the separate objectives, outperforms the version with a combined objective function. This is an indicator that optimizing these two-valued, lexicographically ordered objectives separately can lead to improved results compared to combining the objective values into a single measurement.

The algorithm selection using KNN on the train and test sets is visualized in Figure 5.9. All models are used by this model, with CP_C dominating. However, this is only the case because we decided to select the model that was best on the training set to break ties. Around $Z_1 = 1.5$ and $Z_2 = -1.5$ is the only area where MIP_H dominates a whole cluster, which was already expected by analyzing the performance of the train set in Figure 5.5. However, there exists a large area with no clear pattern that the 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.

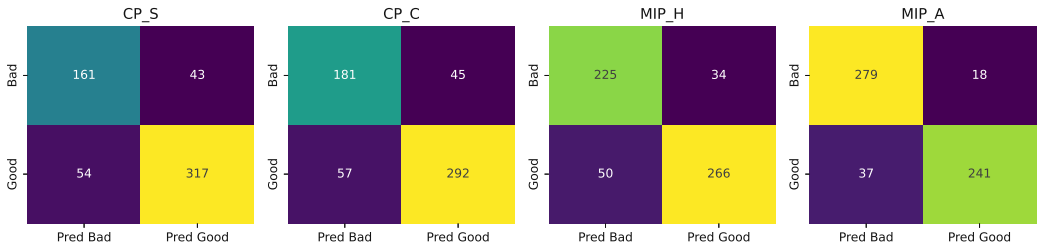
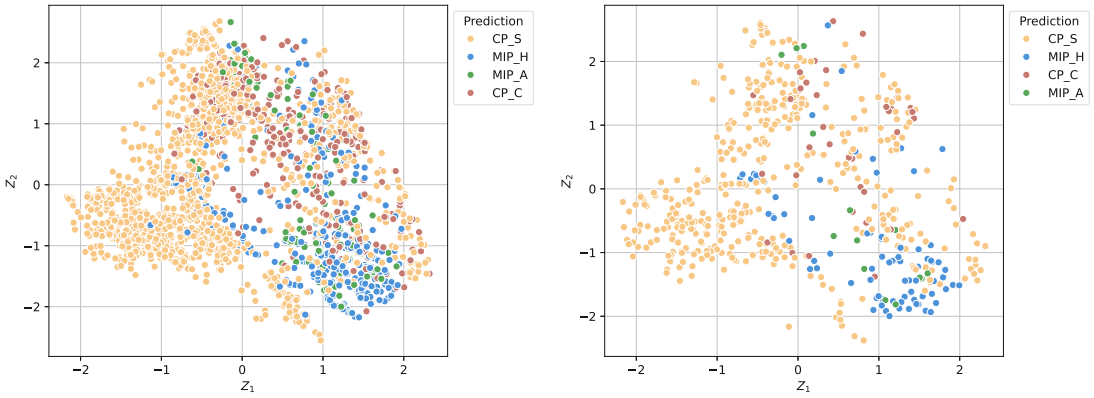


Figure 5.8: Confusion matrices of KNN for exact optimizers



(a) Train set

(b) Test set

Figure 5.9: Selected exact methods by KNN for train and test instances

selection follows.

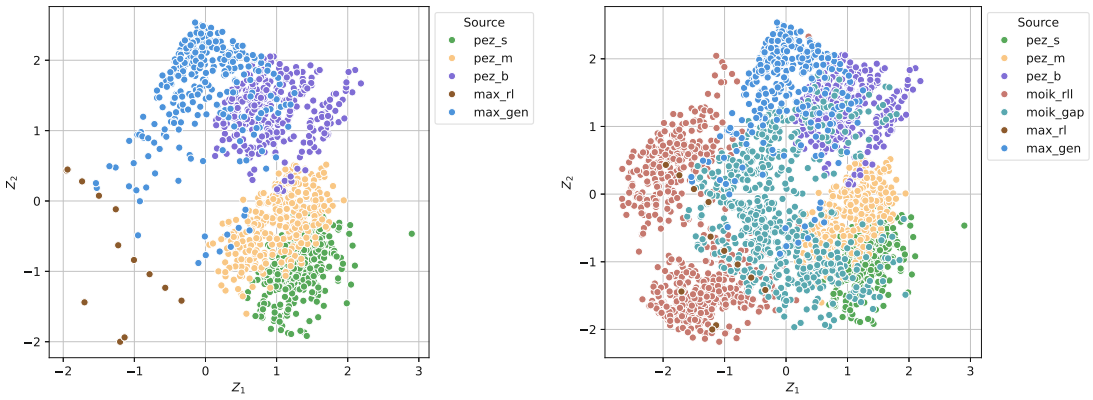
5.3 Instance Space Analysis for Heuristics

We repeat the same analysis for the heuristic methods discussed in Section 3.2, which includes Greedy, SAC, SAR, SAI, and LNS. All 2552 instances from the train set, as well as the whole test set containing 638 instances, will be utilized.

5.3.1 Instance Source Visualization

Again, before going into details about feature selection and algorithm performance, we visualize the distribution of instances from different data sources in the instance space. To show the significance and novelty of our own generated instance sets, Figure 5.10 visualizes the existing instances in the instance space on the left, while the right plot includes all utilized instances. The existing instances are differentiated quite well by their source. However, in the same way as for the selected features for the exact solvers in Section 5.2, the few real-life instances from max_rl are located apart from the other instances. With the instances generated by us, we solve this issue. moik_gap is located

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



(a) Existing instances

(b) All utilized instances

Figure 5.10: Instance separated by their source in the instance space for heuristics

Feature	Z_1	Z_2
max_proc_m_agg_std	−0.0986	−0.2878
max_setup_pre_agg_range	−0.3029	−0.1479
min_spare_j_agg_std	−0.5977	−0.1136
elig_m_ct_agg_range	0.0096	0.4919
jj_deg_agg_mean	−0.4799	0.4640
rat_lb	0.2598	0.2561

Table 5.6: Features with corresponding projection coefficients obtained by MATILDA, for the instance space for heuristic approaches

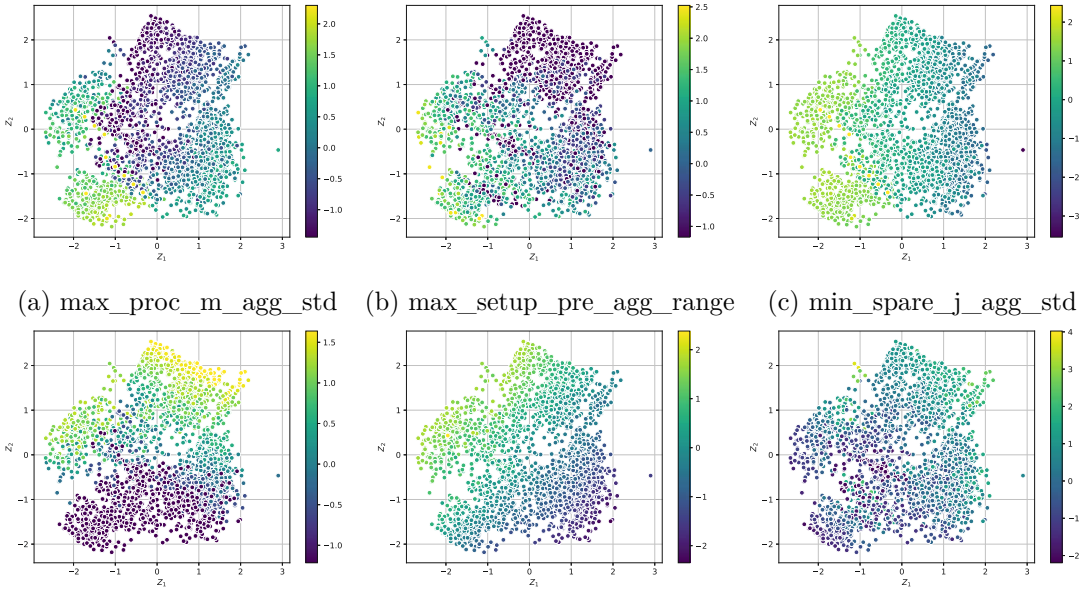
quite central to fill the gap, and moik_rll is located in the same area as max_rl as before. Similarly, with the utilization of other instance features for the construction of the instance space, our instance sets are able to complement the existing ones.

5.3.2 Selected Features

The features listed in Table 5.6 were selected by MATILDA to project the instances into a two-dimensional space. We visualize the distribution of these features split by source in Figure 5.12, as well as in the instance space in Figure 5.11. The correlation between the selected features can be seen in Figure 5.13.

- max_proc_m_agg_std The standard deviation of maximal processing time per machine is highest for instances from max_rl and moik_rll, as one can see in Figure 5.12a. This feature is able to separate the real-life and real-life-like instances from the rest quite well. In the instance space in Figure 5.11a, the highest feature values occur on the bottom left at around $Z_1 = -1$ and $Z_2 = -2$. The Top and the center show the lowest values.

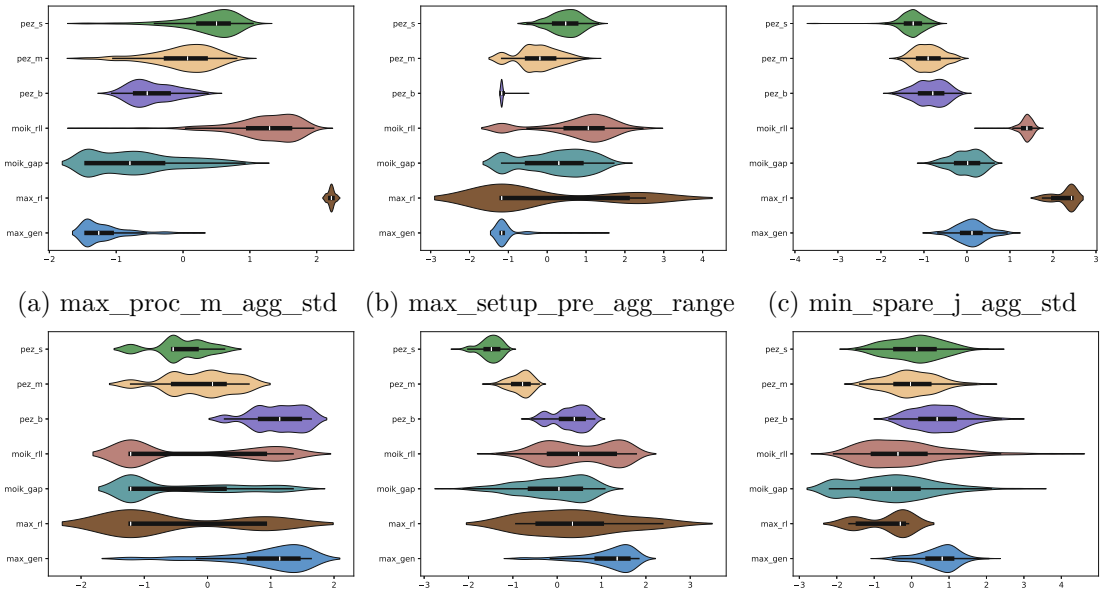
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) max_proc_m_agg_std (b) max_setup_pre_agg_range (c) min_spare_j_agg_std

(d) elig_m_ct_agg_mean (e) jj_deg_agg_mean (f) rat_lb

Figure 5.11: Feature values visualized in the instance space for heuristics



(a) max_proc_m_agg_std (b) max_setup_pre_agg_range (c) min_spare_j_agg_std

(d) elig_m_ct_agg_mean (e) jj_deg_agg_mean (f) rat_lb

Figure 5.12: Distribution of selected features in the instance space for heuristics per data source

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.

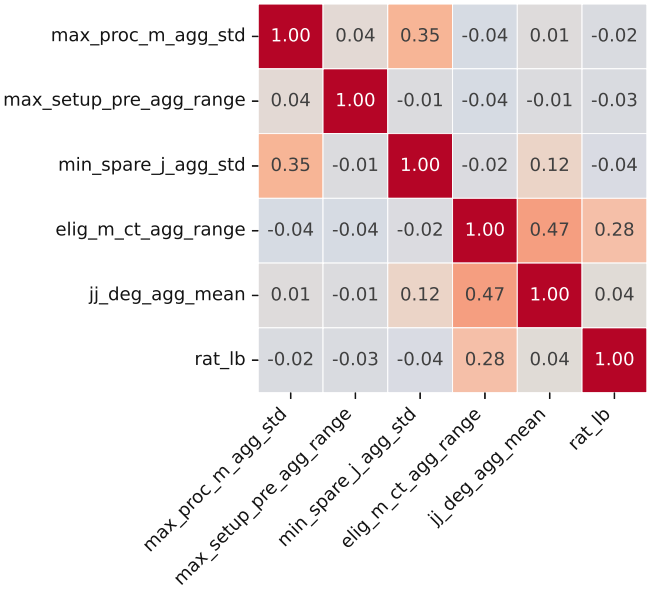


Figure 5.13: Correlation between the features selected for heuristics

- `max_setup_pre_agg_range` The second selected feature, the range of the maximal setup times preceding a job, shows a gradual increase from the top right to the bottom left in the instance space, with a few exceptions, as it can be seen in Figure 5.11b. In general, this feature seems to group the instances at the top together with their low values for this feature. Figure 5.12b suggests that these are mainly from `pez_b` and `max_gen`, since these are concentrated at a low value.

- `min_spare_j_agg_std` The standard deviation of the minimal spare time per job shows a clear increase from right to left in the instance space in Figure 5.11c. This again means that both, `max_rl` and `moik_rll`, reach the highest values like it is visible in Figure 5.12c. In comparison to `max_proc_m_agg_std`, which also reached its highest values in this area, this feature is lower on the right side of the instance space, while the first one shows an increase on both sides.

- `elig_m_ct_agg_range` The range of the eligible machine count for an instance increases gradually from bottom to the top in Figure 5.11d. This feature is suitable for separating instances with one or multiple eligible machines. This can be seen in Figure 5.12d, where most sources show two accumulation regions. The selection of this feature indicates an impact of the number of eligible machines on the hardness of an instance.

- `jj_deg_agg_mean` The mean node degree of the `jj_graph` shows a trend from bottom right to top left in Figure 5.11e. This feature, in combination with `elig_m_ct_agg_range`, also shows the highest correlation between any of the

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.

selected features in the correlation matrix in Figure 5.13. In general, the features are only weakly correlated, since the highest correlation coefficient is 0.47.

- `rat_lb` The ratio of the greedy solution to a naive lower bound is the only probing feature included. A higher ratio indicates a larger optimization potential for the heuristics, starting from the solution provided by `Greedy`. However, it might also be an indicator of the hardness of an instance. Figure 5.12f shows that the lowest values for this ratio are achieved by the real-life instances and our own generated instance set. So this feature could be coupled to the sparse setup times utilized in these sets.

5.3.3 Algorithm Performance

Figure 5.14 visualizes the best performances of each algorithm. Again, tied means that there exists another algorithm that found the same best solution, while unique indicates that this particular algorithm was the only one that found the best solution. Grayed out instances indicate that another algorithm was able to find a better solution.

In the bottom part of the instance cloud, all algorithms manage to find the same best solution, suggesting that these instances are quite easy to solve. This is also backed by the fact that `pez_s` and `pez_m`, the two instance sets containing smaller instances, are located in this area, as it can be seen in Figure 5.10. Instances from `moik_gap` and `moik_rll` are also part of these easy instances. They seem to be instances where each job has very few eligible machines, as suggested by the feature distribution in Figure 5.11d. For the other areas in the instance space, `SAC` and `SAR` dominate, while `SAI` and `LNS` only achieve uniquely best solutions for a few instances, but not for a whole cluster.

`SAR`, which is the SA variant that reheats after the minimum temperature is reached, achieves many uniquely best solutions in the top right around $Z_1 = 1$ and $Z_2 = 1.5$. This area contains mostly instances from `pez_b` when looking at Figure 5.10. `SAC`, which adapts the cooling speed according to the time left, outperforms all other algorithms on the rest of the instances most of the time.

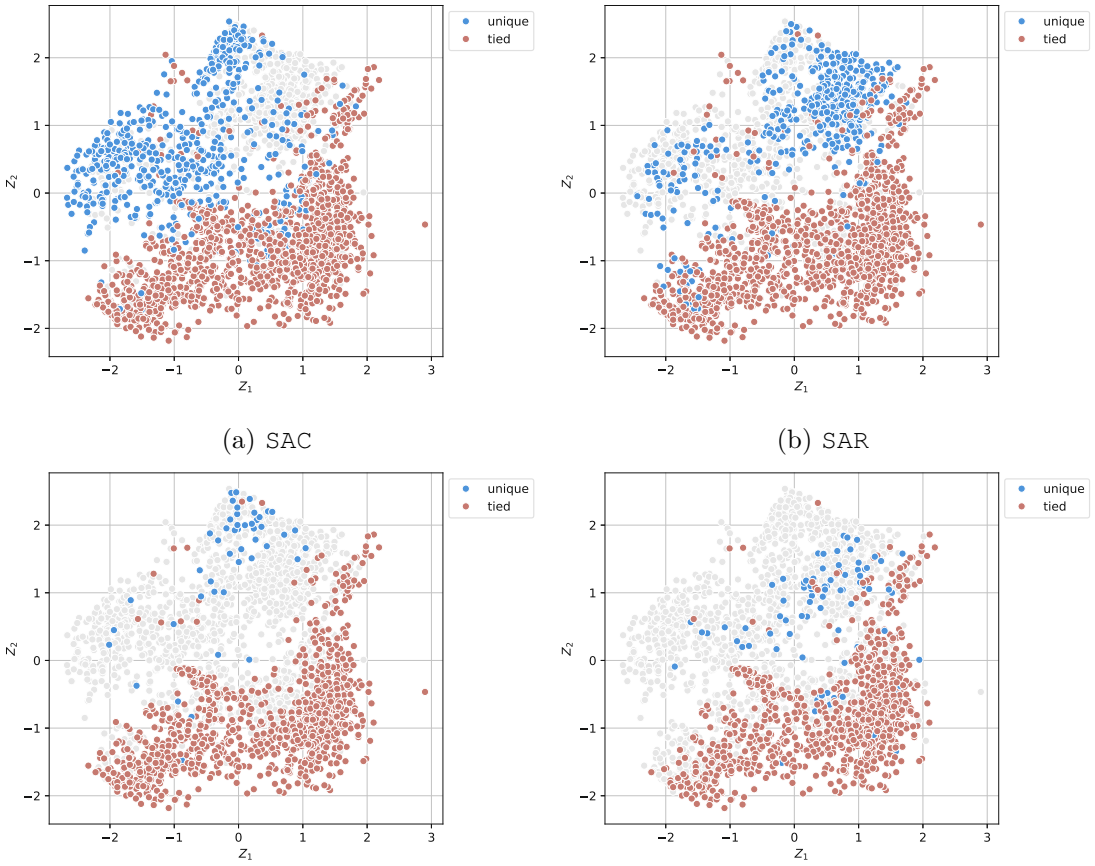
In Figure 5.15 we visualize the number of good algorithms per instance. The trend from easy instances at the bottom to harder ones on the top can be seen clearly here. Occasionally, even `Greedy` produces the same results as the more sophisticated heuristics.

5.3.4 Algorithm Selection

Finally, we train algorithm selection models for the heuristic optimizers. The tuning of the models according to the setup described in Section 4.7 yielded the parameters listed in Table 5.7.

All models outperform the baseline classifier when comparing the F_1-scores in Table 5.8. The macro-averaged F_1-score of `MF` reaches 0.34., while the best solver, which is `RF`, scores 0.87. Again, the scores of the `MF` classifier do not differ much for `SAC` and `SAR`,

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



(a) SAC (b) SAR

(c) SAI (d) LNS

Figure 5.14: Best performances of each heuristic approach, excluding greedy

Model	Parameters	Tuning result
KNN	n_neighbors	7
	weights	distance
	p	1
RF	n_estimators	200
	criterion	log_loss
	min_samples_leaf	5
SVM	C	5
	kernel	rbf
	degree	-
	gamma	scale

Table 5.7: Tuned Parameters for the algorithm selection models for heuristic approaches

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

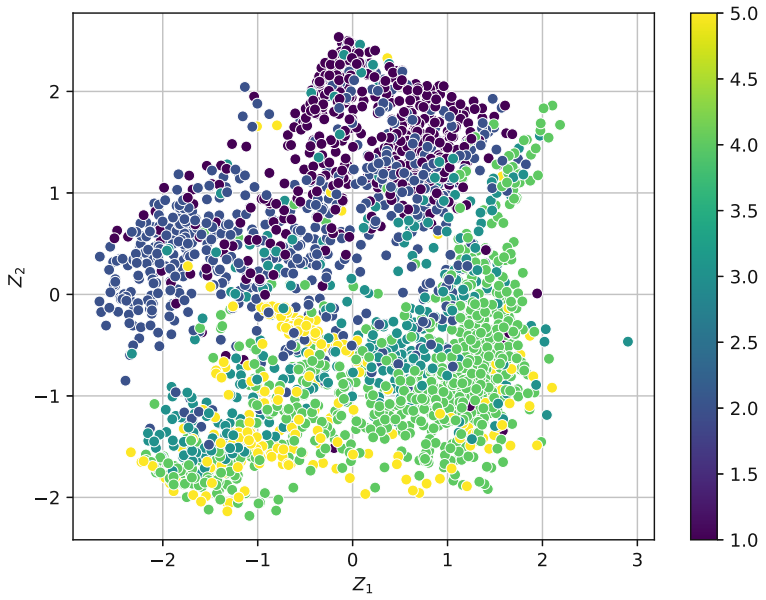


Figure 5.15: Number of good heuristic algorithms per instance

	MF	KNN	RF	SVM
SAC	0.85	0.92	0.92	0.91
SAR	0.83	0.91	0.92	0.88
SAI	0	0.88	0.91	0.84
LNS	0	0.83	0.87	0.80
Greedy	0	0.68	0.73	0.12

Table 5.8: F_1-scores for the algorithm selection models for the heuristic approaches

because these algorithms are able to perform well on the majority of instances. Still, small improvements can be seen in the other models.

In Figure 5.16, a closer inspection is possible by looking at the confusion matrices for each algorithm for the predictions from RF.

In Figure 5.17, we visualize the number of instances each model was able to identify as one of the best algorithms. In this visualization, we accounted for the instances where all algorithms (except Greedy) are able to find the same best solution, as these instances are not interesting for algorithm selection. For this, we moved the y-axis so that the irrelevant instances are excluded. The number of correct predictions by RF is 548, which is an improvement of around 11% over the baseline.

The visualization of the predictions of RF on train and test instances is given in Figure 5.18. As the analysis of the algorithm performances in Subsection 5.3.3 already suggested, SAR is considered the best algorithm for a small cluster around $Z_1 = 0.5$ and $Z_2 = 1.5$.

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 Bibliothek Your knowledge hub WIEN

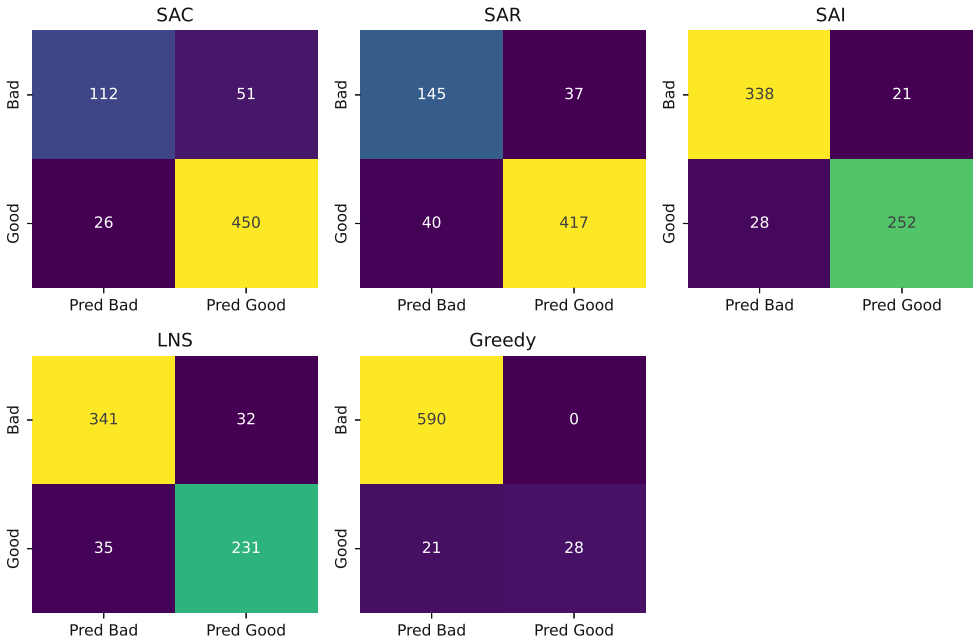


Figure 5.16: Confusion matrices of RF for heuristic optimizers

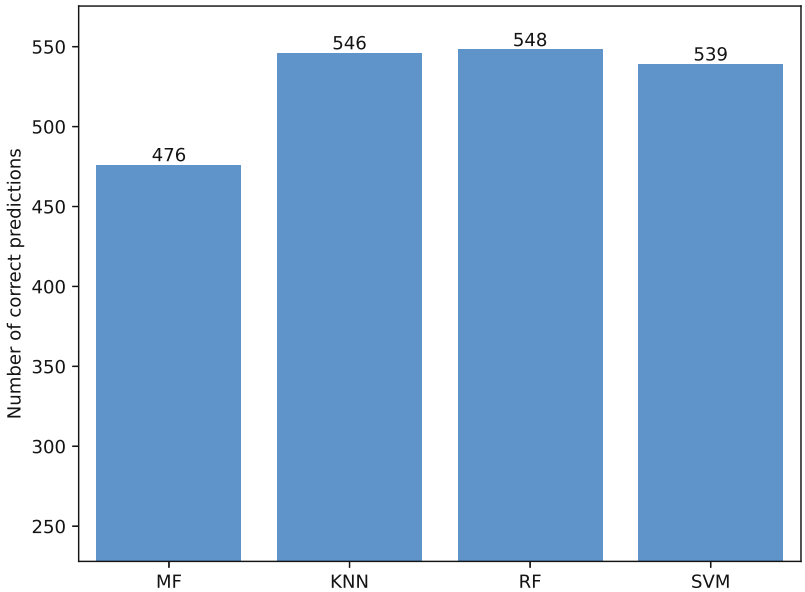
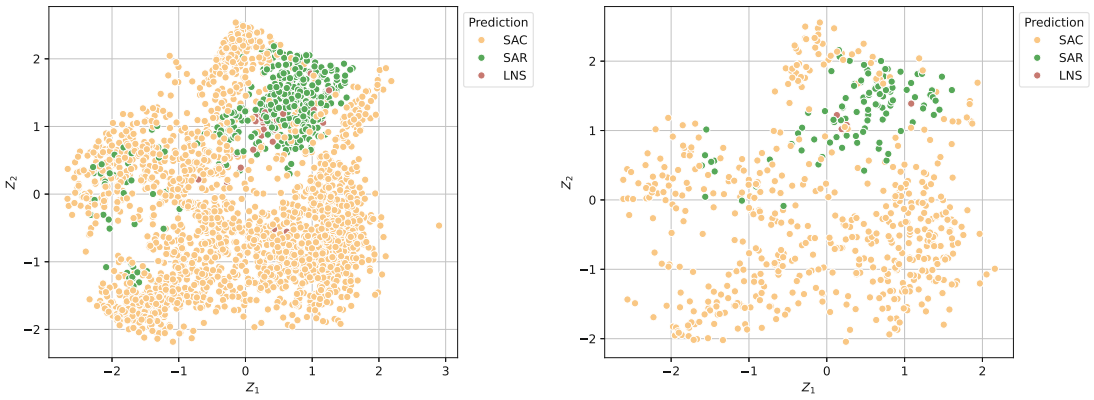


Figure 5.17: Comparison of the number of correct predictions of the algorithm selection models on the test set

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.



(a) Train set (b) Test set

Figure 5.18: Selected heuristics by RF for train and test instances

Otherwise, SAC is the algorithm of choice with few exceptions. With the feature analysis from Subsection 5.3.2, we can conclude that the average number of eligible machines per job has an impact on the algorithm selection between SAR and SAC.

5.4 Comparison Between Exact and Heuristic Instance Space

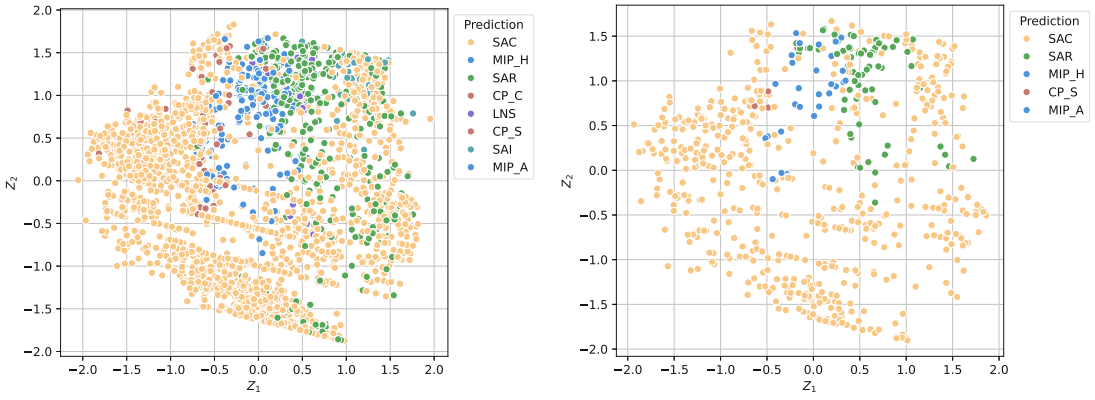
To finalize this chapter, we will briefly discuss the insights from both instance spaces to point out similarities and differences between them.

In comparison of the two analyses, the heuristics showed better separation of the instances regarding algorithmic performance. For the exact solvers, MATILDA could not select features that allow the same quality of separation. This might be due to missing features or simply because the models do not show enough differences in performance.

The most important features for distinguishing the performance of the various heuristics and exact methods are closely related. They focus on the number of eligible machines and the node degree of the jj_graph. It is also worth mentioning that the clusters where SAR is selected for the heuristics, and the MIP models perform best for the exact solvers, contain similar instances as well.

While experiments with exact methods and heuristics combined were conducted, there are only very few instances where exact solvers are chosen over heuristic approaches, as Figure 5.19 shows. The combined analysis also would not have allowed for the discovery of the performance difference between CP models and MIP models on instances with very few eligible machines per job, since the heuristics cover that area.

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.



(a) Train set (b) Test set

Figure 5.19: Selected approaches by RF for train and test instances

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

CHAPTER 6

CP Formulation for an Extension of the UPMSP

Very recently, the UPMSP we analyzed in the earlier chapters was adapted by introducing new constraints and objectives. In this chapter, we will propose a novel CP formulation, which will be implemented for the CP-SAT solver. We compare our new approach to the other known solvers for this problem from Horn et al. [2025]. The focus lies on the general approach for handling resource calendars and machine downtimes using interval variables.

6.1 Problem Description

This variant of the UPMSP has, in addition to the machine eligibility and setup time constraints that we already introduced in Section 2.1, job precedences and calendar-based cumulative resource constraints. Therefore, the problem is called the *Parallel Machine Scheduling Problem with Precedence Constraints and Resource Calendar Constraints* (short PMSP-PCRCC). These constraints arise in real-life applications of our industry partner and are therefore of practical importance.

The following problem description is directly taken from Horn et al. [2025], with a slightly adapted example problem, which makes the following explanation for resource handling easier to understand. We include the problem description in our work to make the constraint formulations in the next section easier to understand, with the necessary conditions and an example problem in mind.

Figure 6.1 illustrates a simple example schedule using two machines (M_1, M_2) and six jobs ($J_1 \dots J_6$). The jobs are drawn as red bars, where the horizontal length indicates the processing time. The thinner blue bars denote the predecessor-dependent setup time of the jobs. Note that J_1, J_5, and J_6 have a setup time of zero.

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

The diagonally striped bars within the schedule indicate a machine downtime or the unavailability of a resource. During a downtime, no work can be processed. However, jobs running before a downtime period can simply be paused during the downtime and continue right afterwards. Pausing a job randomly or due to a lack of resources is not allowed. This interruption may occur during the setup or processing time of the job. In the example, this is illustrated as J_5 is interrupted during the processing time but continues immediately after the end of the downtime. The downtimes at the end of the schedule reach to infinity, indicating the end of the input or machine and resource availabilities.

In any feasible schedule, job precedence constraints have to be respected. The job precedences in the example are illustrated by arrows between pairs of jobs. Thus, J_3 and J_5 must be scheduled after the completion of J_1. Precedences can optionally specify time lags between jobs. In the example, we see that J_5 must respect a short time lag after the completion of J_1 (otherwise, it could start directly after J_2).

In addition, two resources R_1 and R_2 are visualized in Figure 6.1. Cumulative resource constraints impose restrictions on feasible schedules as follows: For each time slot in the scheduling horizon, each resource provides a certain amount of capacity. Thus, resource calendars can be modeled by supplying different capacities in different time periods of the scheduling horizon. Each job and machine also specifies resource demands that must be fulfilled. Thus, for each time slot, all resources need to provide sufficient capacity to supply the running machines and all jobs that are processed at the given time. Note that a machine with a running job does not consume any resource capacities during machine downtimes, but jobs may not be paused outside of machine downtimes to purposely interrupt resource consumption.

In the example, R_1 provides a capacity of 4 in the time slots 0-4 and a capacity of 2 in the time slots 6-9. During the remaining times, there is no capacity supply, indicated again by a diagonal line pattern. R_2 provides a capacity of 2 during the time slots 2-9; in the remaining time, the resource is not available.

In the following, we provide the full formal specification of the problem.

6.1.1 Model Variables

Table 6.1 summarizes all instance parameters. We further define the following variables for the PMSP-PCRCC:

- Job start times: $start_j \in \mathbb{N} \quad \forall j \in J_0$
- Job completion times: $end_j \in \mathbb{N} \quad \forall j \in J_0$
- Indicator variables determining if a job is spanning across a machine downtime: $across_{ju} \in \{0, 1\} \, \forall j \in J, u \in U$

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

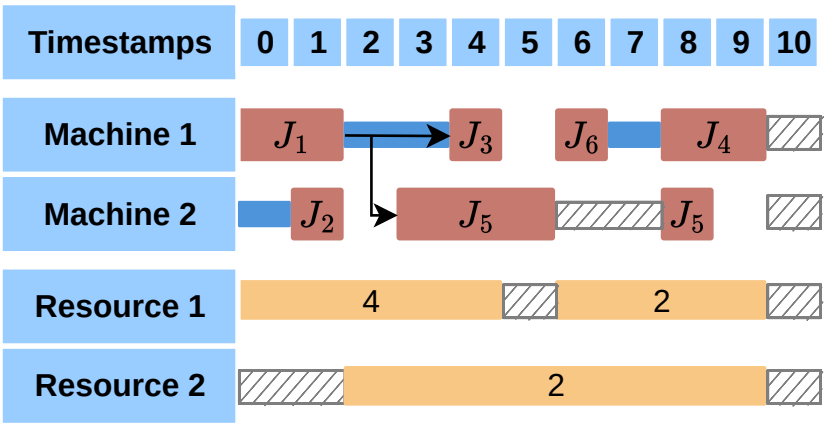


Figure 6.1: An example schedule with two parallel machines and six jobs, adapted from Horn et al. [2025]

- Previous job assignments (capture the last job previously scheduled before another job): $prev_j \in J \cup J_b \setminus \{j\} \, \forall j \in J$
- Machine assignments: $a_j \in E_j \quad \forall j \in J_0$
- Total tardiness: $T \in \mathbb{N}$
- Makespan: $C \in \mathbb{N}$
- Total setup time: $S \in \mathbb{N}$

6.1.2 Constraints

Several constraints impose restrictions on feasible schedules:

- All previous job assignments must be different:

$$prev_{(j_1)} \neq prev_{(j_2)} \quad \forall j_1, j_2 \in J, \, j_1 \neq j_2$$

- The starting dummy jobs, which represent the machine starts, are fixed to the corresponding machines:

$$a_{(b_m)} = m \quad \forall m \in M$$

- Set start- and end-times of starting dummy jobs to 0:

$$start_j = 0 \wedge end_j = 0 \quad \forall j \in J_b$$

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.

Parameter	Description
M, $\|M\| = k$	Set of machines
J, $\|J\| = n$	Set of jobs
$b_m \, \forall m \in M, J_b = \{b_m \mid m \in M\}$	Starting dummy jobs
$J_0 = J \cup J_b$	Jobs with dummies
$E_j \subseteq M \quad \forall j \in J_0$	Eligible machines
$d_j \in \mathbb{N} \quad \forall j \in J$	Due date of each job
$rd_j \in \mathbb{N} \quad \forall j \in J$	Release date of each job
$jp_{jm} \in \mathbb{N} \quad \forall j \in J_0, m \in M$	Job processing times
$s_{ijm} \in \mathbb{N} \, \forall m \in M, i, j \in J_0$	Job setup times
R, $\|R\| = s$	Set of resources
$rc_{rt} \in \mathbb{N} \quad \forall r \in R, t \in H$	Resource capacity
$z_{jr} \in \mathbb{N} \quad \forall j \in J, r \in R$	Job demands
$v_{mr} \in \mathbb{N} \quad \forall m \in M, r \in R$	Machine demands
$U = \bigcup_{m \in M} U_m$	Machine downtimes
$us_u \in H \quad \forall u \in U$	Downtime start
$ue_u \in H \quad \forall u \in U$	Downtime end
$P_j \quad \forall j \in J$	Set of preceding jobs
$l_{jp} \in \mathbb{N} \quad \forall j \in J, p \in P_j$	Precedence time lag
$prec_p \in J \quad \forall j \in J, p \in P_j$	Predecessor job
$V \in \mathbb{N}$	Horizon upper bound
$H = [0, V]$	Scheduling horizon

Table 6.1: Input parameters of the PMSP-PCRCC

- Jobs must be assigned to the same machine as their previous job:

$$a_{(prev_j)} = a_j \quad \forall j \in J$$

- Determine if a job is spanning across a machine downtime:

$$(start_j \leq us_u \wedge end_j \geq us_u) \Leftrightarrow across_{ju} = 1 \quad \forall u \in U_{(a_j)}, j \in J$$

- Calculate the job end times, including pauses (note that the job's start time is set to the beginning of the setup time):

$$end_j = start_j + s_{(a_j)(prev_j),j} + jp_{j(a_j)} + \sum_{u \in U_{(a_j)}} across_{ju} \cdot (ue_u - ue_s) \quad \forall j \in J$$

- Job start times must be greater than or equal to the release date:

$$start_j \geq rd_j \quad \forall j \in J$$

- Job start times must be greater than the previous job's end time:

$$start_j \geq end_{(prev_j)} \quad \forall j \in J$$

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 Bibliothek Your knowledge hub WIEN

- Job starts and ends cannot lie within machine downtimes:

$$(start_j < us_u \vee start_j \geq ue_u) \quad \forall j \in J, u \in U_{(a_j)}$$
$$(end_j < us_u \vee end_j \geq ue_u) \quad \forall j \in J, u \in U_{(a_j)}$$

- Sufficient resource capacities must be supplied for all running jobs at any point (except for paused jobs during machine downtimes):

$$J_t = \{j \in J | t \in [start_j, end_j] \wedge \forall u \in U_{(a_j)} (t \notin [us_u, ue_u])\}$$
$$\sum_{j \in J_t} z_{jr} + v_{(a_j)r} \leq rc_{rt} \quad \forall t \in H, r \in R$$

- Job precedence constraints must be fulfilled with respect to the precedence time lag:

$$start_j \geq end_{(prec_p)} + l_{jp} \quad \forall j \in J, p \in P_j$$

- Calculate total tardiness:

$$T = \sum_{j \in J} \max\{0, end_j - d_j\}$$

- Calculate makespan:

$$C = \max\{end_j \mid \forall j \in J\}$$

- Calculate total setup time:

$$S = \sum_{j \in J} s_{(a_j)(prev_j)j}$$

6.1.3 Objective Function

The objective function minimizes a weighted sum of the total tardiness, the makespan, and the total setup time (with weights $w_1, w_2, w_3 \in \mathbb{R}$):

$$\texttt{minimize}(w_1 \cdot T + w_2 \cdot C + w_3 \cdot S)$$

6.2 CP Formulation

In the following, the model constraints will be given and explained. We again make use of the `circuit` and `no_overlap` global constraints, which were introduced in Subsection 3.1.2. Additionally, we use the `cumulative` global constraint to handle the resource calendar and machine downtimes. The concept underlying this global constraint was first proposed by Aggoun and Beldiceanu [1993]. A modern implementation of the global constraint for lazy clause generation based CP solvers was proposed by Schutt et al. [2011]. This constraint takes a value as the maximal capacity, as well as a set of

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.

interval variables with corresponding demands as input. Intervals use up the quantity of demanded resources from their start to end time. If two intervals overlap, their demanded resources add up. The cumulative constraint enforces that at no time, the cumulative demanded resources of all intervals exceed the given capacity.

Before we define all variables used in this constraint formulation, we introduce another notation. The set $Seg_r,\ r \in R$ contains all segments, where the capacity of resource r stays constant. Working with such segments instead of individual timesteps makes it easier to define interval variables in the following. Also cap_{rs}, the capacity of resource $r \in R$ in segment $s \in Seg_r$, as well as ss_{rs} and se_{rs}, which are the start and end time of segment s of resource r, are used in the constrain formulation. Table 6.2 lists all model variables necessary for the following constraint formulation.

We start with the definition of the interval variables modeling the jobs. In this modeling approach, we extend the job interval duration if machine downtimes are crossed and provide additional resources during these downtimes instead of splitting the job into multiple intervals to avoid such crossings. With the same notation as it was introduced in Subsection 3.1.2, we define a global interval variable in Constraint (6.1), as well as machine-specific ones in Constraint (6.2) for each job. Note that the machine-specific interval variable is not of fixed size because it is dependent on whether or not machine downtimes are crossed.

$$I_j = \texttt{interval}(start_j, dur_j, end_j) \quad \forall j \in J \tag{6.1}$$

$$I_{jm} = \texttt{opt\_interval}(start_{jm}, dur_{jm}, end_{jm}, active_{jm}) \quad \forall j \in J, m \in E_j \tag{6.2}$$

With this and Constraints (6.3) - (6.5), it is ensured that no two job intervals overlap on the same machine. Implications are enforced using `only_enforce_if` available for CP-SAT. Note that, in contrast to the separation of setup- and processing time in the formulation in Subsection 3.1.2, the start of an interval is the start of the preceding setup time of the corresponding job in this case.

$$\texttt{exactly\_one}(\{active_{jm} | m \in E_j\}) \quad \forall j \in J \tag{6.3}$$

$$active_{jm} \Rightarrow start_j = start_{jm} \quad \forall j \in J, m \in E_j \tag{6.4}$$

$$active_{jm} \Rightarrow dur_j = dur_{jm} \quad \forall j \in J, m \in E_j \tag{6.5}$$

The tardiness, as well as the precedence constraints for each job, are enforced by Constraints (6.6) - (6.7). Note that because of the non-negative variable domain of T_j and the minimization of the objective function, it will automatically be zero if the job finishes before its due date.

$$T_j \geq end_j - d_j \quad \forall j \in J \tag{6.6}$$

$$start_j \geq end_p + l_{pj} \quad \forall j \in J, p \in P_j \tag{6.7}$$

To get an ordering of the jobs on each machine, we use the `circuit` global constraints given in Constraint (6.8). To exclude non-present jobs from the cycle on a given machine,

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.

Variable	Description
$start_j$	Start of setup of job j
end_j	End of job j
dur_j	Duration of job j
T_j	Tardiness of job j
S_j	Duration of setup time of job j
$active_{jm}$	Boolean indicating if job j is scheduled on machine m
$start_{jm}$	Start time of job j on machine m
dur_{jm}	Duration of job j on machine m
end_{jm}	End time of job j on machine m
C_m	Makespan of machine m
$delay_j$	Additional time that job j spends during machine downtimes
I_j	Interval variable corresponding to job j
I_{jm}	Optional interval variable for job j on machine m
$Ical_{rs}$	Interval variable for resource r during segment s
$Icrosses_{rju}$	Optional fixed size interval variable to complement usage of resource r of job j during downtime u
$Idown_{ru}$	Optional fixed size interval variable for usage of resource r if no job crosses downtime u
Iup_{ru}	Fixed-size interval variable for usage of resource r during machine uptime immediately after downtime u
arc_{ijm}	Boolean indicating if the arc from job i to job j on machine m in present
$SBDTS_{ju}$	Boolean indicating if job j starts before downtime u starts
$SBDTE_{ju}$	Boolean indicating if job j starts before downtime u ends
$EADTS_{ju}$	Boolean indicating if job j ends after downtime u starts
$crosses_{ju}$	Boolean indicating if job j crosses the downtime u
$crosses_u$	Boolean indicating, if any job crosses downtime u
C	Maximal machine makespan
T	Cumulative tardiness of all jobs
S	Cumulative setup times of all jobs

Table 6.2: Utilized variables in the CP model for the PMSP-PCRCC with their description

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.

Constraint (6.9) is added. The dummy job 0 in each circle marks the start and end point of the schedule and also allows for modeling the initial setup time in the next steps.

$$\texttt{circuit}(\{arc_{ijm} | i,j \in J_0, m \in E_i \cap E_j\}) \quad \forall m \in M \tag{6.8}$$

$$arc_{jjm} = \neg active_{jm} \quad \forall j \in J, m \in E_j \tag{6.9}$$

Starting out with helper variables in Constraints (6.10) -(6.12), Constraints (6.13) - (6.14) can then enforce $crosses_{ju}$ and $crosses_u$ to take on their correct values.

$$SBDTS_{ju} = start_j < us_u \quad \forall j \in J, u \in U \tag{6.10}$$

$$SBDTE_{ju} = start_j < ue_u \quad \forall j \in J, u \in U \tag{6.11}$$

$$EADTS_{ju} = end_j > us_u \quad \forall j \in J, u \in U \tag{6.12}$$

$$crosses_{ju} = SBDTS_{ju} \wedge EADTS_{ju} \wedge active_{jm} \quad \forall j \in J, m \in M, u \in U_m \tag{6.13}$$

$$crosses_u = \bigvee_{j \in J} (crosses_{ju}) \quad \forall u \in U \tag{6.14}$$

Using the same variables, we enforce that no job starts during a machine downtime in Constraint (6.15). With the other constraints, it is ensured that no job ends during a downtime, so this constraint does not have to be included explicitly. Also note that with this implementation, there is no differentiation if a job is interrupted by a downtime during setup or processing because it is not relevant to the final schedule or the objective function.

$$SBDTS_{ju} \vee \neg SBDTE_{ju} \vee \neg active_{jm} = True \quad \forall j \in J, m \in M, u \in U_m \tag{6.15}$$

For later usage, the additional time that a job spends in machine downtimes is given with Constraint (6.16).

$$delay_j = \sum_{u \in U_m} crosses_{ju} \cdot (ue_u - us_u) \quad \forall j \in J, m \in M \tag{6.16}$$

To enforce the correct start, setup, and processing times, the ordering from the circuit constraint (represented by the presence of arcs in the cycle) is used in Constraints (6.17) - (6.20). Also, the machine makespans are enforced with Constraint (6.21).

$$start_j \geq rd_j \quad \forall j \in J \tag{6.17}$$

$$arc_{ijm} \Rightarrow S_j = s_{ijm} \quad \forall i \in J_0, j \in J, m \in M, i \neq j \tag{6.18}$$

$$arc_{ijm} \Rightarrow start_j \geq end_i \quad \forall i,j \in J, m \in M, i \neq j \tag{6.19}$$

$$arc_{ijm} \Rightarrow end_j = start_j + s_{ijm} + p_{jm} + delay_j \quad \forall i \in J_0, j \in J, m \in M, i \neq j \tag{6.20}$$

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.

	J_1	J_2	J_3	J_4	J_5	J_6	M_1	M_2
R_1	1	1	2	1	0	1	1	0
R_2	0	0	1	1	1	1	0	0

Table 6.3: Example resource demands in addition to schedule from Figure 6.1

$$arc_{j0m} \Rightarrow C_m \geq end_j \quad \forall j \in J, m \in M \tag{6.21}$$

Finally, we come to the resources. The resource calendar and machine availability constraints are enforced using a single global `cumulative` constraint per resource. We propose a general formulation in this thesis, which can also be used for other variants, where any kind of resource calendar, machine downtimes, or similar are present.

Recall that jobs can be interrupted when they are crossing a machine downtime and do not use any resources during that time. Instead of splitting the job into individual intervals, we fix this with additional dummy jobs that are running during machine up- and downtimes.

To complement the constraint formulation, Figure 6.2 shows the scheduling and demands of all dummy and job intervals for the example schedule in Figure 6.1, which is included again below the interval schedules for easier reference. For this, we assume the resource usages of jobs and machines given in Table 6.3.

In Constraint (6.22) we set an upper bound of the maximal resource demand at any time. This acts as the capacity in the `cumulative` global constraints in Constraint 6.31. The first term of $maxCap_r$ is used to handle the changes in the resource calendars, while the second term is needed for resource demands of machines and jobs. We calculate an upper bound for the resource demand and then block as much as necessary to recreate the actual availabilities, depending on the schedule.

$$maxCap_r = \max_{s \in Seg_r}(cap_{rs}) + |M| \cdot \left(\max_{j \in J}(z_{jr}) + \max_{m \in M}(v_{mr})\right) \quad \forall r \in R \tag{6.22}$$

For our example with values from Table 6.3 this means $maxCap_1 = 4 + 2 \cdot (2 + 1) = 10$ and $maxCap_2 = 2 + 2 \cdot (1 + 0) = 4$.

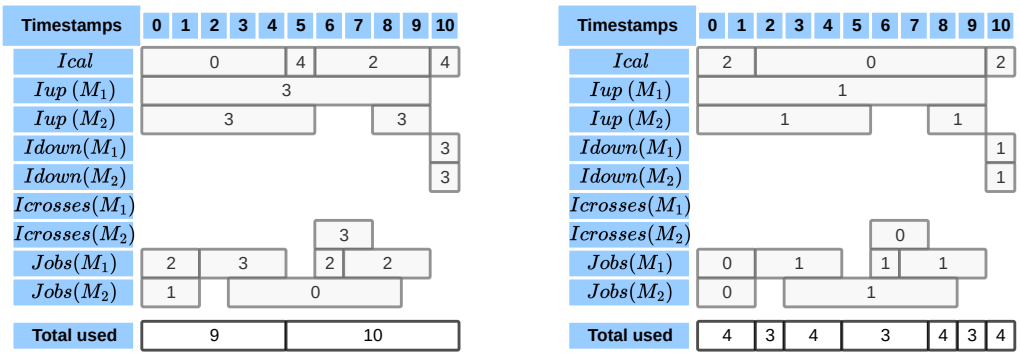
Next, we introduce the dummy jobs for the resource calendar and machine up- and downtimes. The interval definition follows the same notation as it was introduced in Subsection 3.1.2. In addition to each interval, we define a corresponding demand.

To enforce the resource calendar, Constraints (6.23) - (6.24) are used. While $Ical_{rs}$ is an interval variable, corresponding to a part of the resource calendar, $IcalD_{rs}$ is the demand of resource r for $Ical_{rs}$. This notation of using the letter D for the demand corresponding to an interval is also used in the following constraints.

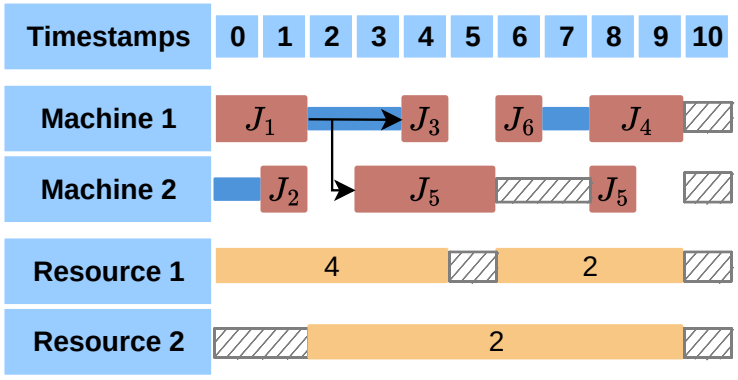
The intervals are of fixed size, not optional, and span across each period where the resource availability stays constant. This interval is defined by the start time and duration.

$$Ical_{rs} = \texttt{fixed\_interval}(ss_{rs}, se_{rs} - ss_{rs}) \quad \forall r \in R, s \in Seg_r \tag{6.23}$$

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



(a) Resource 1 (b) Resource 2



(c) Schedule from Figure 6.1 as reference

Figure 6.2: Visualization of dummy intervals scheduled to handle resource calendars and machine downtimes

$$IcalD_{rs} = \max_{s \in Seg_r}(cap_{rs}) - cap_{rs} \quad \forall r \in R, s \in Seg_r \tag{6.24}$$

In this approach, job intervals are still active during machine downtimes. Therefore, we start with a higher capacity, which is blocked by combining the job interval and a dummy interval during downtimes. The dummy intervals with their demands in Constraints (6.25) - (6.26) during downtimes act complementary to the job, so that at all times, both combined use $\max(z_{jr}) + \max(v_{mr})$ resources. If no job crosses the downtime, the dummy jobs use as many resources as are allocated for them, as it is given in Constraints (6.27) and (6.28).

$$Icrosses_{rju} = \texttt{opt\_fixed\_interval}(us_u, ue_u - us_u, crosses_{ju}) \quad \forall r \in R, j \in J, u \in U \tag{6.25}$$

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.

$$IcrossesD_{rju} = \max_{i \in J}(z_{ir}) + \max_{n \in M}(v_{nr}) - z_{jr} - v_{mr} \quad \forall r \in R, j \in J, m \in M, u \in U_m \tag{6.26}$$

$$Idown_{ru} = \texttt{opt\_fixed\_interval}(us_u, ue_u - us_u, \neg crosses_u) \quad \forall r \in R, u \in U \tag{6.27}$$

$$IdownD_{ru} = \max_{j \in J}(z_{jr}) + \max_{m \in M}(v_{mr}) \quad \forall r \in R, u \in U \tag{6.28}$$

Note that these intervals are not merely preprocessing steps, because the intervals and corresponding demands are dependent on whether a job interval crosses a machine downtime or not.

To make this clearer, we go through the calculation to reach the values for $Icrosses(M_2)$ shown in Figure 6.2. Since J_5 crosses a downtime, its resource usage has to be complemented by the dummy intervals $Icrosses_{rju}$ to use up the additionally allocated resources. Since J_5 does not use resource 1, calculation with the example values from Table 6.3 yields $IcrossesD_{rju} = \max_{i \in J}(z_{ir}) + \max_{n \in M}(v_{nr}) - z_{jr} - v_{mr} = 2 + 1 - 0 - 0 = 3$. But J_5 utilizes resource 2. The same calculation for this resource results in $IcrossesD_{rju} = \max_{i \in J}(z_{ir}) + \max_{n \in M}(v_{nr}) - z_{jr} - v_{mr} = 1 + 0 - 1 - 0 = 0$.

During machine uptimes, only resources are used where they are supposed to be, so we use up the additionally assigned resources with dummy jobs. In the preprocessing, we add dummy downtimes at timestamp 0 up until the first uptime and at the end after the last uptime to the horizon so that each machine's uptime is enclosed by two downtimes. Now let U^- be the set of all downtimes, excluding the final downtime that indicates the end of the scheduling horizon. Using downtime $u \in U^-$, we have its succeeding downtime $succ(u)$. With this notation, we define dummy jobs with corresponding demands with Constraints (6.29) - (6.30) to use up the extra resources given for the handling of machine downtimes that are not needed during machine uptimes.

$$Iup_{ru} = \texttt{fixed\_interval}(ue_u, us_{u_{succ}} - ue_u) \quad \forall r \in R, u \in U^- \tag{6.29}$$

$$IupD_{ru} = \max_{j \in J}(z_{jr}) + \max_{m \in M}(v_{mr}) \quad \forall r \in R, u \in U^- \tag{6.30}$$

The cumulative constraint gets passed the interval variables defined above with their corresponding capacity, as well as the interval variables I_{jm} representing jobs with their demand $z_{jr} + v_{mr}$ for resource r. In Constraint (6.31), the intervals and demands are condensed into the expressions $Intervals_r$ and $Demands_r$, respectively, for each resource $r \in R$.

$$\texttt{cumulative}(maxCap_r, Intervals_r, Demands_r) \quad \forall r \in R \tag{6.31}$$

Finally, we can define the separate objectives T, C, and S in Constraints (6.32) - (6.34), followed by the whole objective function as the weighted sum of them with weights given in the problem input in Constraint (6.35)

$$C = \max_{m \in M}(C_m) \tag{6.32}$$

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

Measurement	MiniZinc	CP_SAT	SA
Optimum found	118	120	108
Proven Infeasibility	40	40	-
Unknown if solution exists	1	0	47

Table 6.4: Results for CP_SAT , MiniZinc and SA on gen_s

$$S = \sum_{j \in J} S_j \tag{6.33}$$

$$T = \sum_{j \in J} T_j \tag{6.34}$$

$$minimize(w_1 \cdot T + w_2 \cdot C + w_3 \cdot S) \tag{6.35}$$

6.3 Experimental Evaluation

Finally, we performed tests to compare our CP model to the MiniZinc model on the CP-SAT solver and the SA approach in Horn et al. [2025]. We call our model CP_SAT, as it is directly implemented for this solver. While Minizinc also utilizes CP-SAT, it is translated automatically and does not use interval variables. We call this simply MiniZinc. The SA approach is referred to as SA. We performed the experiments on identical hardware to the original work so that the results are comparable. The experiments are conducted on a computing cluster with 13 nodes, each featuring two Intel Xeon E5-2650 v4 CPUs (12 cores @ 2.20GHz). CP_SAT is run with OR-Tools version v9.10 on a single CPU using the interleave mode, a parameter for CP-SAT, which allows the solver to combine multiple search strategies effectively instead of relying on a single one. We set a runtime limit of 1 hour and conducted a single run per instance.

We utilize the provided instances from Horn et al. [2025]. These consist of a total of 345 instances. They can be divided into three groups:

- gen_s: 140 randomly generated instances with 10 jobs, and 20 randomly generated instances with 20 jobs.
- gen_m: 160 randomly generated instances with 100 jobs.
- real-life: 25 instances that were extracted from real-life industry data.

Tables 6.4, 6.5 and 6.6 show the comparison between our CP model, with the results of Horn et al. [2025], which were kindly provided to us in form of the raw experiment output, on each group of instances.

One can see that both exact methods find almost all optimal solutions for the small instances, with only MiniZinc missing two of them. SA only reached the optimum for

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.

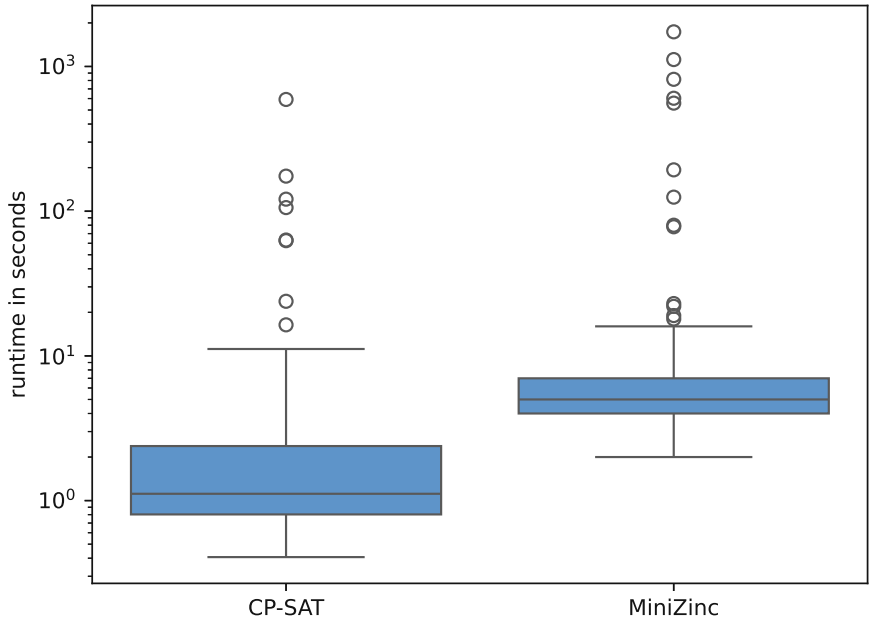


Figure 6.3: Comparison of time needed to find the optimal solution for solved instances from `gen_s`

108 instances, while for five, it found feasible but not optimal solutions. For the other 47 instances, no feasible solution is found in the given runtime. Because both `MiniZinc` and `CP_SAT` yield nearly perfect results on `gen_s`, we also compare the time needed for them to find the optima. The results can be seen in Figure 6.3, showing a log-scaled box plot of the runtimes. One can clearly see that `CP_SAT` finds the optimum much faster. By calculating the ratio of the `MiniZinc` runtime to the `CP_SAT` runtime and calculating the median of these ratios, we see that the latter is faster by a factor of four on average. Also, there is only one instance where `MiniZinc` is faster than `CP_SAT` among all 120 feasible instances.

We continue with the medium-sized instances. `MiniZinc` is not able to find a single solution, which is why we only compare `CP_SAT` and `SA`. Table 6.5 contains the gathered insights. Whether `SA` is able to find an optimal solution or not can only be determined if `CP_SAT` is able to prove the optimality first. In the tables showing the results, we denote this with a + behind the reported value for `SA` to emphasize that there might be more optimal solutions. However, we see that there are 19 instances where infeasibility is proven by `CP_SAT`. Because of the lack of optimal solutions, we report the number of instances in which one algorithm outperforms the other. As one can see, `SA` found a better solution on 55 instances, and `CP_SAT` on 51 instances. We can conclude that for

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.

Measurement	CP_SAT	SA
Optimum found	0	0+
Proven Infeasibility	19	-
Unknown if solution exists	89	93
Better than other	51	55

Table 6.5: Results for CP_SAT and SA on gen_m

Measurement	CP_SAT	SA
Optimum found	6	6+
Proven Infeasibility	0	-
Unknown if solution exists	19	0
Better than other	0	19

Table 6.6: Results for CP_SAT and SA on real-life

this instance size (around 100 jobs), the performance of both algorithms is comparable.

Finally, we come to the 25 real-life instances. These results are presented in Table 6.6. For the smallest six instances, our solver is able to prove optimality. SA is also able to find the same solution. For the other instances, no feasible solution or lower bound is found by CP_SAT. Here, SA finds at least one feasible solution for all instances. The smallest six instances of this set have a maximum of 31 jobs, with the next higher one having 107 jobs to schedule, with up to 685 jobs for the largest instance. So, one can see again that the dominance of SA starts at about 100 jobs.

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

CHAPTER 7

Conclusion

In this thesis, we analyze the instance space for a challenging variant of the Unrelated Parallel Machine Scheduling Problem to detect biases in existing instance sets and to deeply analyze the strengths and weaknesses of exact solvers and heuristics. We also introduce a novel Constraint Programming formulation for a second variant of the Unrelated Parallel Machine Scheduling Problem with an extended set of constraints. The Instance Space Analysis yields valuable insights regarding the performance of algorithms based on different instance types, while the Constraint Programming model for the extended problem shows the strength of interval variables in CP modeling for Parallel Machine Scheduling Problems.

First, we identify existing state-of-the-art approaches to the problem. These are various Mixed Integer Programming models and Simulated Annealing heuristics. We complement them with other approaches to the problem, to enable a broader analysis and comparison of algorithmic performance. We introduce a Constraint Programming formulation, which makes use of interval variables, as well as a Large Neighborhood Search with novel destroy and repair operators for the problem. We also gather existing instances used in past benchmarks for our analysis. These instances come from Moser et al. [2022] and Perez-Gonzalez et al. [2019].

To visualize the instances in the instance space, we propose various instance features. This includes graph-based features, which are derived from two graphs constructed from the problem input. Additionally, probing features, based on the construction heuristic, are introduced. This set, containing 150 features, is used to map instances into the instance space. The online tool MATILDA, proposed by Smith-Miles and Muñoz [2023], projects the instances into a two-dimensional space by selecting a subset of features that is still able to separate instances well, and also takes into account the performance of the evaluated algorithms. Visualizing this projection reveals gaps between existing instances, as well as a lack of instances resembling real-life instances. We address these issues by introducing a novel set of 1562 instances to complement the existing ones and allow 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

better coverage of the instance space. This is done by adapting an existing instance generator, which allows the generation of instances with unseen feature combinations. The adaptation is inspired by the structure of the setup times from real-life instances and, therefore, is also able to produce instances that resemble the sparse real-life instances that are available to us.

Evaluating all algorithms and solvers on a sampled subset containing 2552 instances allows for a detailed performance analysis, depending on instance features. In general, all included methods are able to solve the vast majority of small instances included in our experiments, with only a few exceptions. Our proposed Constraint Programming models are able to reach better results than the Mixed Integer Programming methods on most instances included in our experiments, which have a low average number of eligible machines per job. For instances with many eligible machines per job in combination with a higher number of jobs, the Mixed Integer Programming approaches show better results. However, the Constraint Programming models yield better results if the number of jobs is increased even further. The comparison of the Simulated Annealing variants and the Large Neighborhood Search leads to similar outcomes. For instances with a low number of eligible machines per job, all algorithms are able to find good solutions. With a higher number of jobs and multiple eligible machines, the Simulated Annealing variant SAR yields the best performance. This variant uses a cooling scheme using a constant cooling factor and resets the temperature again to the maximum temperature if the minimum temperature is reached. However, in the same way as before, for instances with an even higher number of jobs, the advantage goes to the Simulated Annealing variant SAC, which adapts the cooling factor based on the current runtime, to reach the minimum temperature by the timeout. Unfortunately, the Large Neighborhood Search is not able to reach uniquely best results on a cluster of instances to play a relevant role in the algorithm selection. We train machine learning models to identify the most promising exact method and heuristic for a given instance. While the best individual exact method is able to find the best solution for 64% of the instances from the test set, the algorithm selection model is able to obtain the best solution for 73%. For the heuristic approaches, the best individual algorithm finds 74% of the best solutions, and the algorithm selection model obtains the best solution for 86% of the test instances.

In addition to the Instance Space Analysis, we propose a novel Constraint Programming formulation for an extension of the Unrelated Parallel Machine Scheduling Problem. Since the interval variable models from the initial problem yield better results for most instances than the Mixed Integer Programming models, we utilize the same approach again. Interval variables are used to model resource calendars and machine availability constraints in a generally applicable way by introducing dummy jobs to block resources or machines a the right times. Our model formulation, implemented for the CP-SAT solver, is able to prove optimality of solutions for the evaluated small instances four times faster than the existing exact solvers on average. For medium-sized instances, our approach is also able to compete with Simulated Annealing and provides new best solutions to some instances.

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

The Instance Space Analysis conducted in this thesis opens up many possibilities for future work. Novel instance generators could be investigated to populate the instance space even more densely. Regarding the exact solvers, one could utilize a novel performance measurement that incorporates the best bounds found. This could make the differences between the models clearer and would allow for a better separation in the instance 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 Bibliothek Your knowledge hub WIEN

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

Overview of Generative AI Tools Used

All of the work presented in this thesis was done by me. For the implementation, ChatGPT only helped with the generation of regular expressions used to read input files. It was also used to translate single words or reformulate some sentences while writing this thesis. However, the generated output only acted as a basis for the final formulation and was never used directly. In addition, Grammarly was used as a spellchecker in the finalizing steps of the writing process.

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

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

List of Figures

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

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

List of Tables

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

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

List of Algorithms

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

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

Bibliography

Aggoun, A. and Beldiceanu, N. (1993). Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73.

Ahuja, R. K., Ergun, O., Orlin, J. B., and Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1):75–102.

Avalos-Rosales, O., Angel-Bello, F., and Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 76(9):1705–1718.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42–47.

Dang, Q.-V., Herps, K., Martagan, T., Adan, I., and Heinrich, J. (2023). Unsupervised parallel machines scheduling with tool switches. *Computers & Operations Research*, 160:106361.

Dang, Q.-V., van Diessen, T., Martagan, T., and Adan, I. (2021). A matheuristic for parallel machine scheduling with tool replacements. *European Journal of Operational Research*, 291(2):640–660.

De Coster, A., Musliu, N., Schaerf, A., Schoisswohl, J., and Smith-Miles, K. (2022). Algorithm selection and instance space analysis for curriculum-based course timetabling. *Journal of Scheduling*, 25(1):35–58.

Du, J. and Leung, J. Y.-T. (1990). Minimizing Total Tardiness on One Machine is NP-Hard. *Mathematics of Operations Research*, 15(3):483–495. Publisher: INFORMS.

Francis, K. G. and Stuckey, P. J. (2014). Explaining circuit propagation. *Constraints*, 19(1):1–29.

Gedik, R., Kalathia, D., Egilmez, G., and Kirac, E. (2018). A constraint programming approach for solving unrelated parallel machine scheduling problem. *Computers & Industrial Engineering*, 121:139–149.

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

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. H. G. R. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In Hammer, P. L., Johnson, E. L., and Korte, B. H., editors, *Annals of Discrete Mathematics*, volume 5 of *Discrete Optimization II*, pages 287–326. Elsevier.

Gurobi Optimization, LLC (2024). Gurobi Optimizer Reference Manual. Accessed: 2025-04-12.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

Helal, M., Rabadi, G., and Al-Salem, A. (2006). A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, 3:182–192.

Horn, M., Lackner, M.-L., Malik, P., Mrkvicka, C., Musliu, N., Preininger, J., and Winter, F. (2025). Solving parallel machine scheduling with precedences and cumulative resource constraints with calendars. *Under Submission*.

Katial, V., Smith-Miles, K., and Hill, C. (2024). On the Instance Dependence of Optimal Parameters for the Quantum Approximate Optimisation Algorithm: Insights via Instance Space Analysis. arXiv:2401.08142.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

Kletzander, L., Musliu, N., and Smith-Miles, K. (2021). Instance space analysis for a personnel scheduling problem. *Annals of Mathematics and Artificial Intelligence*, 89(7):617–637.

Kotthoff, L. (2016). Algorithm Selection for Combinatorial Search Problems: A Survey. In Bessiere, C., De Raedt, L., Kotthoff, L., Nijssen, S., O'Sullivan, B., and Pedreschi, D., editors, *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, pages 149–190. Springer International Publishing, Cham.

Lauriere, J.-L. (1978). A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1):29–127.

Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P. (1977). Complexity of Machine Scheduling Problems. In Hammer, P. L., Johnson, E. L., Korte, B. H., and Nemhauser, G. L., editors, *Annals of Discrete Mathematics*, volume 1 of *Studies in Integer Programming*, pages 343–362.

Marko, D. and Jakobovic, D. (2023). Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: a survey. *Artificial Intelligence Review*, 56(4):3181–3289.

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

Messelis, T. and De Causmaecker, P. (2014). An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 233(3):511–528.

Moser, M., Musliu, N., Schaerf, A., and Winter, F. (2022). Exact and metaheuristic approaches for unrelated parallel machine scheduling. *Journal of Scheduling*, 25(5):507–534.

Muñoz, M. A., Villanova, L., Baatar, D., and Smith-Miles, K. (2018). Instance spaces for machine learning classification. *Machine Learning*, 107(1):109–147.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). MiniZinc: Towards a Standard CP Modelling Language. In Bessière, C., editor, *Principles and Practice of Constraint Programming – CP 2007*, Lecture Notes in Computer Science, pages 529–543, Berlin, Heidelberg. Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Perez-Gonzalez, P., Fernandez-Viagas, V., Zamora García, M., and Framinan, J. M. (2019). Constructive heuristics for the unrelated parallel machines scheduling problem with machine eligibility and setup times. *Computers & Industrial Engineering*, 131:131–145.

Perron, L. and Didier, F. (2024). CP-SAT. Accessed: 2025-04-12.

Rice, J. R. (1976). The Algorithm Selection Problem*. In Rubinoff, M. and Yovits, M. C., editors, *Advances in Computers*, volume 15, pages 65–118.

Rolim, G. A., Nagano, M. S., and Prata, B. d. A. (2023). Formulations and an adaptive large neighborhood search for just-in-time scheduling of unrelated parallel machines with a common due window. *Computers & Operations Research*, 153:106159.

Santoro, M. C. and Junqueira, L. (2023). Unrelated parallel machine scheduling models with machine availability and eligibility constraints. *Computers & Industrial Engineering*, 179:109219.

Saraç, T., Ozcelik, F., and Ertem, M. (2023). Unrelated parallel machine scheduling problem with stochastic sequence dependent setup times. *Operational Research*, 23(3):46.

Schutt, A., Feydy, T., Stuckey, P. J., and Wallace, M. G. (2011). Explaining the cumulative propagator. *Constraints*, 16(3):250–282.

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

Smith-Miles, K., Baatar, D., Wreford, B., and Lewis, R. (2014). Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24.

Smith-Miles, K. and Bowly, S. (2015). Generating new test instances by evolving in instance space. *Computers & Operations Research*, 63:102–113.

Smith-Miles, K. and Muñoz, M. A. (2023). Instance Space Analysis for Algorithm Testing: Methodology and Software Tools. *ACM Comput. Surv.*, 55(12):255:1–255:31.

Smith-Miles, K., Wreford, B., Lopes, L., and Insani, N. (2013). Predicting Metaheuristic Performance on Graph Coloring Problems Using Data Mining. In Talbi, E.-G., editor, *Hybrid Metaheuristics*, pages 417–432. Springer Berlin Heidelberg, Berlin, Heidelberg.

Strassl, S. and Musliu, N. (2022). Instance space analysis and algorithm selection for the job shop scheduling problem. *Computers & Operations Research*, 141:105661.

Vallada, E. and Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622.

Wu, X., Zhong, Y., Wu, J., Jiang, B., and Tan, K. C. (2024). Large language model-enhanced algorithm selection: towards comprehensive algorithm representation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24.

Yunusoglu, P. and Topaloglu Yildiz, S. (2022). Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 60(7):2212–2229.

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.