FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Exact and Heuristic Approaches for Unrelated Parallel Machine Scheduling

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Maximilian Moser, BSc
Matrikelnummer 01326252

an der Fakultät für Informatik

der Technischen Universität Wien

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

Wien, 8. Oktober 2019

Maximilian Moser　　Nysret Musliu

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Exact and Heuristic Approaches for Unrelated Parallel Machine Scheduling

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Maximilian Moser, BSc
Registration Number 01326252

to the Faculty of Informatics

at the TU Wien

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

Vienna, 8th October, 2019

Maximilian Moser   Nysret Musliu

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

Erklärung zur Verfassung der Arbeit

Maximilian Moser, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 8. Oktober 2019

Maximilian Moser

Acknowledgements

I want to express my sincere gratitude to my advisor Priv.-Doz. Dr. Nysret Musliu along with DI Felix Winter and Dr. Andrea Schaerf for their valuable feedback and patience. This thesis would not have been possible without their everlasting support.

Special thanks go to Georg Faustmann for motivating me to finish the work, and for keeping the mood well above sea level in the office. Additional special thanks go to Ines Beneder and Anaïs Golder for painstakingly proof-reading the entire document and helping me with the formulations.

I would also like to express my thanks to my family for enabling me to pursue my studies in the first place and supporting me under all circumstances.

Further, I would like to thank Mathias Cammerlander, Denise Berthold and Katrin Deisl because I promised to mention them in the Acknowledgements.

The work in this thesis was performed in the Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling. The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

Kurzfassung

Parallel-Machine-Scheduling-Probleme wurden vielfach in der wissenschaftlichen Literatur untersucht und sind häufig in der Industrie anwendbar. Diese Diplomarbeit beschäftigt sich mit einem praktisch auftretendem Problem, welches als Unrelated Parallel-Machine-Scheduling-Problem mit sequenzabhängigen Umrüstzeiten, Fälligkeitsterminen und Einschränkungen für die Verwendung der Maschinen beschreiben lässt. Das Ziel ist die Minimierung der gesamten Verspätung und Produktionsdauer.

Da bereits existierende mathematische Formulierungen nicht direkt auf unser Problem anwendbar sind, erweitern wir verschiedene bestehende Formulierungen für verwandte Probleme und passen sie auf unser Problem an. Des weiteren stellen wir mehrere Varianten von Simulated Annealing vor, welche wir zum Lösen sehr großer Problem-Instanzen verwenden. Als Teil dieser Algorithmen verwenden wir verschiedene Suchnachbarschaften und untersuchen die Verwendung zusätzlicher innovativer Heuristiken zur Auswahl benachbarter Lösungen.

Wir verwenden die neu generierten Probleminstanzen zusammen mit bestehenden Instanzen aus der Literatur, um die vorgestellten mathematischen Modelle und metaheuristischen Algorithmen auszuwerten. Die praktischen Resultate zeigen, dass unsere gewählten Methoden in der Lage sind, die Resultate der bisher besten Methoden zu verbessern.

Abstract

Parallel Machine Scheduling problems have been subject of intensive research and have many applications in the manufacturing industry. In this thesis, we study a real-life scheduling problem that can be formulated as an Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times, Due Dates, and Machine Eligibility Constraints. The aim is to minimise total tardiness and makespan.

As existing formulations from the literature cannot be directly applied to our problem, we extend and adapt different mathematical models for related problems to approach the problem. Furthermore, we propose several variants of Simulated Annealing to solve very large-scale problem instances as they appear in practice. In these algorithms, we utilise several different search neighbourhoods and additionally investigate the use of innovative heuristic neighbourhood move selection strategies. Further, we provide a set of real-life problem instances as well as a random instance generator that we use to generate a large number of instances.

Using the novel datasets together with existing instances from the literature, we perform a thorough evaluation of the mathematical models and meta-heuristic algorithms that are studied in this thesis. Experimental results show that our methods are able to improve the results produced with state-of-the-art approaches for a large number of instances.

Contents

CHAPTER 1

Introduction

Finding optimised machine schedules in manufacturing is an important and challenging task, as a large number of jobs need to be processed every day. While a manual approach performed by human experts can be used to deal with a small number of jobs, the large-scale requirements of modern factories introduce the need for efficient automated scheduling techniques. In the literature, such scheduling problems that deal with the assignment of jobs to multiple machines which operate in parallel have previously been described as Parallel Machine Scheduling Problems (PMSPs, e.g. Allahverdi et al. [2008], Allahverdi [2015]).

As usually several types of machines are used in the industry that can perform different sets of operations, each job can only be assigned to a specified set of eligible machines. Furthermore, varying machine efficiency has to be considered. PMSPs that consider *eligible machines* are well-known in the literature and have been studied in several publications (e.g. Afzalirad and Rezaeian [2016], Perez-Gonzalez et al. [2019]). Similarly, problems with varying machine efficiency have been previously described as *Unrelated PMSP* (UPMSP, e.g. Vallada and Ruiz [2011], Avalos-Rosales et al. [2015], Allahverdi [2015]). After a job has reached its completion on a machine, in practice it is often required to perform a change of the tooling or machine maintenance before the next job can be processed. The corresponding changeover times between jobs have been referred to as *sequence-dependent setup times* in previous publications (e.g. Vallada and Ruiz [2011], Perez-Gonzalez et al. [2019]).

In the problem we investigate in this thesis, which emerges from a company in the packaging industry, each job corresponds to a customer order with an associated due date. Therefore, the problem's objective function aims to minimise the total *tardiness* of all jobs in addition to the total *makespan*.

In summary, the real-life problem we investigate in this thesis can be characterised as UPMSP with sequence-dependent setup times and eligible machines that aims to minimise both tardiness and makespan. Although a large number of different variants of UPMSP that feature all of the mentioned attributes to some extent have been described in the literature, our problem includes a novel combination of the different constraints and objectives.

1.1 Aims of This Thesis

The main objectives of this thesis are:

- Development of a random instance generator for the problem under study and provision of a large set of instances.
- Adaptation and evaluation of Mixed-Integer Programming formulations for related problems from the literature.
- Development and implementation of meta-heuristic approaches using several neighbourhood operators and guidance strategies in the move generation procedure.
- Computational evaluation of the described approaches on generated instances and instances from the literature.
- Comparison with state-of-the-art approaches for a related problem.

1.2 Contributions of This Thesis

We adapt existing mathematical models for related problems and compare several different formulations for our problem. However, the evaluated exact approaches can only solve small instances in reasonable time. To solve large practical instances, we deeply investigate several variants of Simulated Annealing (SA, Kirkpatrick et al. [1983]). The variants we investigate include different cooling schemes such as a dynamic cooling rate and a reheating mechanism. We investigate different search neighbourhoods based on shift and swap moves, which are commonly used in the context of PMSP. Additionally, we propose a novel neighbourhood operator for UPMSP that operates on blocks of consecutively scheduled jobs. To increase the effectiveness of the move generation procedure, we guide the search towards more promising regions of the search space by incorporating domain knowledge into the neighbourhood move selection strategy.

We provide realistic problem instances which are based on real-life scheduling scenarios that have been provided to us by an industrial partner. Furthermore, we propose a random instance generator to generate diverse datasets that together with the realistic instances form a large pool of instances which we use in our experimental evaluation. Experimental results show that the Simulated Annealing approach is able to generate high-quality solutions for both randomly generated instances as well as real-life instances.

To show the robustness of our method we compare to the state-of-the-art approach (Perez-Gonzalez et al. [2019]) that was proposed recently for a similar problem. The problem specification provided by Perez-Gonzalez et al. [2019] uses the same set of constraints as the problem that we investigate in this thesis and also aims to minimise total tardiness. However, the authors do not consider the minimisation of the makespan. As the minimisation of total tardiness is incomparably more important than the makespan in our problem specification, we can use our solution methods to approach the problem instances provided by Perez-Gonzalez et al. [2019] without problems. Our comparison on a huge set of instances that have been provided by Perez-Gonzalez et al. [2019], shows that our approach produces improved results for a large number of the instances.

1.3 Organisation

The structure of the thesis is as follows: In Chapter 2, we describe our problem and give an overview of the existing related literature. We provide six different Mixed Integer Programming models for the problem in Chapter 3. Our meta-heuristic algorithms are presented in Chapter 4. Chapter 5 contains the description of our instance generation procedure and the generated instances, and we give a detailed overview of our computational experiments and evaluations. Finally, we present our conclusions and remarks on possible future research in Chapter 6.

CHAPTER 2

Parallel Machine Scheduling

The problem we investigate in this thesis can be characterised as an *Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times and Machine Eligibility Constraints* with the objective of minimising both *total tardiness* as well as *makespan*. A Parallel Machine Scheduling Problem (PMSP) aims to assign jobs to a given number of machines to create an optimised production schedule, where every machine is continuously available and can process one job at a time. The execution of jobs cannot be interrupted and resumed later, and there are no precedence constraints between jobs. Furthermore, every machine is assumed to start immediately the execution of its assigned schedule without any break or interruption.

Instances of the PMSP are stated using a set of machines M and a set of jobs J. For every job $j \in J$, we have a due date d_j denoting its latest acceptable completion time and a set of eligible machines $M_j \subseteq M$ on which the job can be processed. The processing time p_{jk} of each job $j \in J$ depends on the machine $k \in M$ on which it is executed. Further, setup times s_{ijk} are defined for any pair of jobs i and j that are consecutively scheduled on a machine k. Additionally, an initial setup time s_{0jk} is required before the execution of job j can begin when it is scheduled as the first job on machine k. Analogously, a clearing time s_{j0k} is required *after* the execution of job j, if it is the last scheduled job on machine k.

A solution to a PMSP assigns a schedule for every machine k, which is represented by a permutation of a subset of all jobs J. If a job i occurs directly before another job j in the schedule for machine k, then i is called the predecessor of j (and j is the successor of i). Since the solution representation of the schedule is sequence-based, the *completion times* of the jobs can be computed as follows: The *completion time* C_j of a job j is the sum of the predecessor's completion time C_i, the appropriate setup time s_{ijk}, and the job's processing time p_{jk}. If a job is the first in its schedule, its completion time is defined by the initial setup time s_{0jk} plus its own processing time p_{jk}. Furthermore, the *tardiness* of a job is defined to be the difference between its completion time and its due date

($T_j := \max(0, C_j - d_j)$). The *machine span* O_k for a machine k is set to the completion time of the job l which is scheduled last plus the final clearing time s_{l0k}. Note that the final clearing times only affect the machine spans, but not the completion times of jobs. Finally, the *makespan* C_{max} is defined to be equal to the maximum of all machine spans.

Graham et al. [1979] propose a three-field $\alpha|\beta|\gamma$ notation to categorise variants of the PMSP, where α, β and γ describe the machine environment, additional constraints and the solution objectives, respectively. The machine environment for PMSPs typically consists of identical machines (P_m) or unrelated machines (R_m). Because we are dealing with an environment consisting of unrelated machines, the problem we investigate can, therefore, be characterised as $R_m|s_{ijk}, M_j|Lex(\Sigma_j(T_j), C_{max})$. The objective function $Lex(\Sigma_j(T_j), C_{max})$ in this case means that both tardiness and makespan should be minimised, with the tardiness being incomparably more important than the makespan (i.e. the objectives are lexicographically ordered). This particular problem variant can be seen as a generalisation of the basic PMSP with identical machines ($P_m||C_{max}$), which has been shown to be NP-hard even with only two machines (see Garey and Johnson [1979]).

2.1 Related Work

PMSPs have been the subject of thorough research in the past, and two surveys by Allahverdi et al. [2008] and Allahverdi [2015] give an overview of the related literature. Sequence-dependent setup times on unrelated machines have been described for many problems that have been studied in the literature. A well-known problem in this area is, for example, the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times, aiming to minimise the makespan ($R_m|s_{ijk}|C_{max}$).

Among the first to investigate this problem variant is Al-Salem [2004], who proposes a Partitioning Heuristic as solution method and Rabadi et al. [2006] who tackle the problem using a Meta-Heuristic for Randomised Priority Search. Arnaout et al. [2010] propose an Ant Colony Optimisation algorithm, which they further improved later (Arnaout et al. [2014]). Vallada and Ruiz [2011] propose Genetic Algorithms for the problem and create a set of benchmark instances for their experiments. Avalos-Rosales et al. [2015] propose two new mathematical formulations and a Variable Neighbourhood Descent meta-heuristic. They show that their proposed MIP models outperform existing models on the set of benchmark instances provided by Vallada and Ruiz [2011]. More recent contributions include Santos et al. [2019] who use Stochastic Local Searches on Vallada's instances. They find that Simulated Annealing offers good performance over all evaluated instance sizes. Tran et al. [2016] apply both Logic-based Benders Decomposition and Branch and Check as exact methods for solving the problem. Gedik et al. [2018] propose a Constraint Programming formulation of the problem, leveraging the benefits of interval variables. Fanjul-Peyro et al. [2019] propose a new MIP model for the problem and an algorithm based on mathematical programming. They replace the sub-tour elimination constraints in the MIP model from Avalos-Rosales et al. [2015] by constraints adapted from previous

TSP formulations. This results in a more efficient mathematical formulation for the problem which does not compute all of the jobs' completion times. Other contributions for this problem variant include a Tabu Search approach (Helal et al. [2006]) and Simulated Annealing (Ying et al. [2012]).

PMSPs that include machine eligibility constraints have been considered several times in the literature. Rambod and Rezaeian [2014] consider a PMSP with sequence-dependent setup times and machine eligibility constraints that focuses on minimising the makespan ($R_m|s_{ijk}, M_j|C_{max}$). Additionally, they include the likeliness of manufacturing defects in their objective function. Afzalirad and Rezaeian [2017] minimise a bi-criterion objective consisting of mean weighted tardiness (MWT) and mean weighted flow time ($MWFT$) in a PMSP with sequence-dependent setup times and machine eligibility. They assume different release times of jobs (r_j) and precedence constraints ($prec$) among jobs ($R_m|s_{ijk}, M_j, r_j, prec|MWT, MWFT$). Afzalirad and Rezaeian [2016] try to minimise the makespan for a similar problem where the execution of jobs requires additional resources (res) with limited availability ($R_m|s_{ijk}, M_j, r_j, prec, res|C_{max}$). Bektur and Saraç [2019] consider a problem variant similar to the one we investigate in this thesis, where they minimise the total weighted tardiness. Additionally, they require the availability of a single server (S_1) to perform the setups between jobs ($R_m|s_{ijk}, M_j, S_1|\Sigma_j(w_j \cdot T_j)$). Chen [2006] considers a problem with machine eligibility, where fixed setup times are only required if two consecutive jobs produce different product families. Their objective is to minimise the maximum tardiness of all jobs. Afzalirad and Shafipour [2018] try to minimise the makespan in a PMSP with machine eligibility and resource restrictions and assume that setup times are included in the processing times.

The problem statements most closely resembling the problem considered in this thesis are studied by Caniyilmaz et al. [2015], Adan et al. [2018] and Perez-Gonzalez et al. [2019]. All three papers use Sequence-dependent setup times, due dates and machine eligibility constraints. However, each of these papers focuses on the minimisation of a slightly different objective function.
Caniyilmaz et al. [2015] try to minimise the sum of makespan and cumulative tardiness ($C_{max} + \Sigma_j T_j$). They implement an Artificial Bee Colony algorithm and compare its performance against a Genetic Algorithm on a real-life instance originating from a quilting work centre. This most closely resembles our objective of minimising tardiness as the primary target and makespan as the secondary target. Adan et al. [2018] try to minimise a three-part objective function, consisting of a weighted sum of total tardiness, setup times and processing times ($\alpha \cdot \Sigma_j T_j + \beta \cdot \Sigma_{jk} p_{jk} + \gamma \cdot \Sigma_{ijk} s_{ijk}$), where α, β and γ are weights. This objective function coincides with our objective function when there is only a single machine available and the weights are chosen appropriately. They implement a Genetic Algorithm very similar to the one described by Vallada and Ruiz [2011] and apply it to three real-life datasets. Perez-Gonzalez et al. [2019] also consider a problem that is similar to the problem investigated in this thesis, but they only take the tardiness of jobs into consideration and disregard the makespan. Furthermore, they propose a MIP model for their problem which is based on the mathematical formulation from Vallada

and Ruiz [2011], along with five different constructive heuristics and an immune-based meta-heuristic. They are the first to create a dataset that is available for other researchers.

In summary, we can see that a large variety of PMSPs has been studied in the past. However, the particular problem variant that considers a lexicographically ordered minimisation of total tardiness and makespan under machine eligibility constraints and sequence-dependent setup times has, to the best of our knowledge, not been investigated yet.

CHAPTER 3

Mixed-Integer Programming Approaches

In this Chapter, we review Mixed-Integer Programming models for related Parallel Machine Scheduling Problems from the literature and adapt them to be applicable to our problem formulation. We adapt the model proposed by Perez-Gonzalez et al. [2019] for the $R_m|s_{ijk}, M_j|\Sigma_j T_j$ problem by constraints regarding the makespan as well as explicitly modelled constraints for machine eligibility. Additionally, we adapt one of the models proposed by Avalos-Rosales et al. [2015] for the $R_m|s_{ijk}|C_{max}$ problem by machine eligibility constraints and the calculation of tardiness. Furthermore, we replace one of the constraint sets in the latter model with another formulation found in the related literature to derive additional models.

3.1 Extended MIP Models from the Related Literature

In their work, Perez-Gonzalez et al. [2019] propose a mathematical formulation for a similar Parallel Machine Scheduling Problem that is based on the model described by Vallada and Ruiz [2011]. As their objective function does not consider the makespan, their model does not include corresponding constraints. Therefore, we extend their proposed model by constraint sets (3.5) and (3.8), to calculate the makespan (which includes the clearing times). The variables used in the formulation are described in Table 3.1. The set J_0 includes in addition to all jobs a dummy job (0), that represents the start and end points of each machine schedule. The predecessor of the first job assigned to each machine is set to be the dummy job. Similarly, the successor of the last job assigned to each machine is set to be the dummy job. $X_{i,j,m}$ are binary decision variables which are set to 1 if and only if job j is scheduled directly after job i on machine m (and 0 otherwise). $C_{j,m}$ denotes the completion time of job j on machine m and variables T_j

represent the tardiness of job j. C_{max} is set to the total makespan which includes the clearing times.

Parameter	Additional Information	Description
J	–	Set of Jobs
J_0	–	Set of Jobs, including Dummy Job 0
M	–	Set of Machines
E_j	$j \in J, E_j \subseteq M$	Eligible Machines of job j
d_j	$j \in J$	Due Date of job j
$p_{j,m}$	$j \in J$, $m \in M$	Processing Time of job j on machine m
$s_{i,j,m}$	$i, j \in J_0$, $m \in M$	Setup Time between jobs i and j on machine m
Variable	Additional Information	Description
$X_{i,j,m}$	$i, j \in J_0$, $m \in M$	Job i is the predecessor of job j on machine m
$C_{j,m}$	$j \in J_0, m \in M$	Completion time of job j on machine m
T_j	$j \in J$	Tardiness of job j
C_{max}	–	Makespan
V	–	Large Constant, e.g. Upper Bound for Makespan

Table 3.1: Variables Used in MIP Models **M1** and **M2**

The resulting model **M1** can be stated as follows:

$$\text{minimise } Lex(\Sigma_{j \in J}(T_j), C_{max}), \text{ subject to}$$

$$T_j \geq C_{j,m} - d_j, \forall m \in M, j \in J \tag{3.1}$$

$$\Sigma_{m \in M} \Sigma_{i \in J_0, i \neq j} X_{i,j,m} = 1, \forall j \in J \tag{3.2}$$

$$\Sigma_{m \in M} \Sigma_{j \in J_0, i \neq j} X_{i,j,m} \leq 1, \forall i \in J \tag{3.3}$$

$$\Sigma_{j \in J_0} X_{0,j,m} \leq 1, \forall m \in M \tag{3.4}$$

$$\Sigma_{m \in M} \Sigma_{i \in J_0, i \neq j} X_{i,j,m} = \Sigma_{m \in M} \Sigma_{i \in J_0, i \neq j} X_{j,i,m}, \forall j \in J \tag{3.5}$$

$$\Sigma_{k \in J_0, k \neq i}(X_{k,i,m}) \geq X_{i,j,m}, \forall i, j \in J, m \in M, i \neq j \tag{3.6}$$

$$C_{j,m} + V \cdot (1 - X_{i,j,m}) \geq C_{i,m} + s_{i,j,m} + p_{j,m}, \forall i \in J_0, j \in J, m \in M, i \neq j \tag{3.7}$$

$$\Sigma_{i \in J_0} \Sigma_{j \in J_0} (s_{i,j,m} + p_{j,m}) \leq C_{max}, \forall m \in M \quad (3.8)$$

$$C_{0,m} = 0, \forall m \in M \quad (3.9)$$

$$X_{i,j,m} \in \{0, 1\}, \forall i, j \in J, m \in M \quad (3.10)$$

$$T_j \geq 0, \forall j \in J \quad (3.11)$$

$$C_{j,m} \geq 0, \forall m \in M \quad (3.12)$$

Constraint set (3.1) binds the tardiness of each job. Constraint set (3.2) ensures that every job has exactly one predecessor and is scheduled on one machine, while constraint set (3.3) restricts every job to have only one successor. They are not instantiated for the dummy job, because it is shared over all machines and thus can have multiple predecessors (successors) in the solution. Constraint set (3.4) restricts every machine to schedule at most one job at position one. Constraint set (3.5) forces each job (except for the dummy job) to have a single predecessor and successor on the machine where it is scheduled. Constraint set (3.6) checks that if a job j is scheduled after another job i on the same machine, then i has at least one predecessor as well. Constraint set (3.7) calculates the completion time $C_{j,m}$ for every job j on machine m. Note that $V \cdot (1 - X_{i,j,m})$ evaluates to 0, if job i is the predecessor of job j on some machine. Otherwise, it evaluates to V and thereby fulfils the inequality. Given that every job has a processing time greater than zero on every machine, constraint set (3.7) also enforces *sub-tour elimination* regarding the predecessor relations. Constraint set (3.8) calculates the makespan which includes the clearing time after the final job. Constraint set (3.9) forces the completion time of the dummy job to be 0 on every machine. Constraint sets (3.10) to (3.12) restrict the domains of the decision variables.

Note that constraint set (3.2) does not ensure that jobs are scheduled on one of their eligible machines. Instead, the original model from Perez-Gonzalez et al. [2019] handles machine eligibility by assigning very high processing times to the jobs on their ineligible machines. Machine eligibility can be included directly in the model by replacing constraint set (3.2) by constraint sets (3.13) and (3.14). We refer to the resulting model as **M2**.

$$\Sigma_{m \in E_j} \Sigma_{i \in J_0, i \neq j} X_{i,j,m} = 1, \forall j \in J \quad (3.13)$$

$$\Sigma_{m \in M \setminus E_j} \Sigma_{i \in J_0, i \neq j} X_{i,j,m} = 0, \forall j \in J \quad (3.14)$$

Constraint set (3.13) ensures that every job is scheduled on exactly one of its eligible machines, while constraint set (3.14) prohibits the jobs from being scheduled on any other machine.

Avalos-Rosales et al. [2015] propose new mathematical formulations for the problem described by Vallada and Ruiz [2011]. Most notably, they include a new set of binary decision variables $Y_{j,m}$ to describe whether or not job j is scheduled on machine m and they additionally introduce the notion of *machine spans*. Further, they replace the decision variables for the completion time $C_{j,m}$ of job j on machine m by the variables C_j.

In addition to the above-mentioned models **M1** and **M2**, we extend the model proposed by Avalos-Rosales et al. [2015] to our problem statement to derive alternative mathematical formulations. These extensions include machine eligibility, due dates and the incorporation of final clearing times.

The resulting model **M3** can be stated as:

$$\text{minimise } Lex(\Sigma_{j \in J}(T_j), C_{max}), \text{ subject to}$$

$$\Sigma_{m \in M}(Y_{j,m}) = 1, \forall j \in J \tag{3.15}$$

$$\Sigma_{i \in J_0, i \neq j}(X_{i,j,m}) = Y_{j,m}, \forall j \in J, m \in M \tag{3.16}$$

$$\Sigma_{j \in J_0, i \neq j}(X_{i,j,m}) = Y_{i,m}, \forall i \in J, m \in M \tag{3.17}$$

$$C_j \geq C_i + s_{i,j,m} + p_{j,m} + V \cdot (X_{i,j,m} - 1) \forall i \in J_0, j \in J, m \in M \tag{3.18}$$

$$\Sigma_{j \in J}(X_{0,j,m}) \leq 1, \forall m \in M \tag{3.19}$$

$$\Sigma_{i \in J_0, j \in J, i \neq j}(s_{i,j,m} \cdot X_{i,j,m}) + \Sigma_{i \in J}(p_{i,m} \cdot Y_{i,m} + s_{i,0,m} \cdot X_{i,0,m}) \leq C_{max}, \forall m \in M \tag{3.20}$$

$$T_j \geq C_j - d_j, \forall j \in J \tag{3.21}$$

$$T_j \geq 0, \forall j \in J \tag{3.22}$$

$$C_0 = 0 \tag{3.23}$$

$$X_{i,j,m} \in \{0,1\}, \forall i,j \in J, m \in M \tag{3.24}$$

$$Y_{i,m} \in \{0,1\}, \forall j \in J, m \in M \tag{3.25}$$

Constraint set (3.15) ensures that every job is scheduled on exactly one machine. Constraint sets (3.16) and (3.17) ensure that every job has predecessors and successors on a machine if and only if the job is scheduled on this machine. Constraint set (3.18) connects the completion time for each job to its predecessors. Constraint set (3.19) ensures that there is at most one first job on each machine. Constraint set (3.20) binds the machine span for each machine by summing up the setup times between its scheduled jobs and their processing times. Constraint sets (3.21) and (3.22) involve the tardiness of each job and force it to be non-negative. Constraint (3.23) sets the dummy job's completion time to 0. Constraint sets (3.24) and (3.25) enforce that variables $X_{i,j,m}$ and $Y_{i,m}$ have a binary domain.

This model once again incorporates machine eligibility via penalisation of the corresponding processing times. To explicitly model machine eligibility in the formulation, constraint set (3.15) can be replaced by constraint sets (3.26) and (3.27). The resulting model is labelled **M4**.

$$\Sigma_{m \in E_j}(Y_{j,m}) = 1, \forall j \in J \tag{3.26}$$

$$\Sigma_{m \in M \setminus E_j}(Y_{j,m}) = 0, \forall j \in J \tag{3.27}$$

Constraint set (3.26) ensures that every job is scheduled on exactly one of its eligible machines, while constraint set (3.27) prohibits jobs from being scheduled on any non-eligible machines.

Helal et al. [2006] use a different formulation for constraint set (3.18) which aggregates the machines via sums instead of instantiating the constraints for every machine. Replacing constraint set (3.18) in models **M3** and **M4** by constraint set (3.28) result in models **M5** and **M6**.

$$C_j \geq C_i + \Sigma_{m \in M}(X_{i,j,m} \cdot (s_{i,j,m} + p_{j,m})) + V \cdot (\Sigma_{m \in M}(X_{i,j,m}) - 1), \forall i \in J_0, j \in J \tag{3.28}$$

The implementation of models **M1** - **M6** and their effectiveness will be discussed in Section 5.2.1.

CHAPTER 4

Meta-Heuristic Approaches

In addition to the Mixed-Integer Programming formulations, we propose several Simulated Annealing variants to quickly find high-quality solutions for large instances as they appear in the real world. We first describe how initial solutions for local search can be generated in Section 4.1. Afterwards, we propose neighbourhood operators for the PMSP in Section 4.2, and finally, we describe three Simulated Annealing variants in Section 4.3.

4.1 Constructing Initial Solutions

One way to create an initial solution for local search is to randomly assign jobs to machines. We can do this by selecting one of the eligible machines for each job randomly and then scheduling all jobs in random order on the selected machines.

An alternative to using a random construction of initial solutions is to greedily build an initial schedule. In our case, we propose a constructive greedy heuristic which aims to minimise both tardiness and makespan as follows: First, we order the set of jobs in ascending order by the due dates. Afterwards, we process the ordered jobs and schedule one job after the other on one of its eligible machines. To decide which machine should be selected for a job, the greedy heuristic compares the total machine spans that would be caused by each of the feasible machine assignment and finally selects the assignment that leads to the lowest machine span (ties are broken randomly). If multiple jobs exist that have exactly the same due date, we will compare possible machine assignments for all of these jobs in a single step instead of processing them in random order. In such a case we then also select the job to machine assignment that leads to the lowest increase in machine span. The detailed pseudo-code for this heuristic can be seen in Algorithm 1.

Algorithm 1 Constructive Heuristic (CH)

function ConstructSolution(Jobs, Machines)
  for all $m \in Machines$ **do**
    $Schedule_m \leftarrow$ empty schedule ▷ initialise empty machine schedules
    $t_m \leftarrow 0$ ▷ set machine span to 0
    $l_m \leftarrow 0$ ▷ set last scheduled job ID to 0 (no job) at first
  end for

  $G \leftarrow$ sort and group Jobs by due dates
  for all $g \in G$ **do**
    while $|g| > 0$ **do**
      $j, m \leftarrow \text{argmin}_{i \in g, n \in M}(t_n + s_{l_n i m} + p_{in})$ ▷ find job/machine resulting in lowest machine span
      $t_m \leftarrow t_m + s_{l_m j m} + p_{jm}$
      $l_m \leftarrow j$
      $Schedule_m$.Append(j) ▷ schedule job j on machine m
      $g \leftarrow g \setminus \{j\}$
    end while
  end for
end function

4.2 Search Neighbourhoods

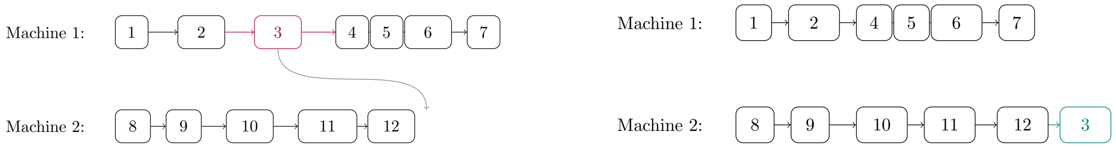
In this section, we introduce the neighbourhood relations that we use in our search method. We begin with the atomic neighbourhoods *Shift* and *Swap*, and then we describe the more complex *block moves*, called *BlockShift* and *BlockSwap*. Finally, we discuss the general notion of *guidance* used to bias the random selection toward promising moves.

4.2.1 Shift Neighbourhood

A *Shift* move is configured to shift a given job j onto machine m at position p. In other words, the job j is first removed from its original location in the current solution. Any successor on the associated machine is then shifted by one position towards the front of the schedule. Finally, job j is re-inserted into the solution at its target position p in the schedule of machine m. Any job that is present on the target schedule at a later or equal position is shifted towards the end of the schedule.

We call a shift move an *intra-machine* shift move if job j is already scheduled on machine m in the current solution. Otherwise, if j is currently assigned to a machine different to m, it is called an *inter-machine* shift move. An example of an *inter-machine* shift move is visualised in Figure 4.1.

Shift moves are validity-preserving as long as the target machine m is eligible for the selected job j.

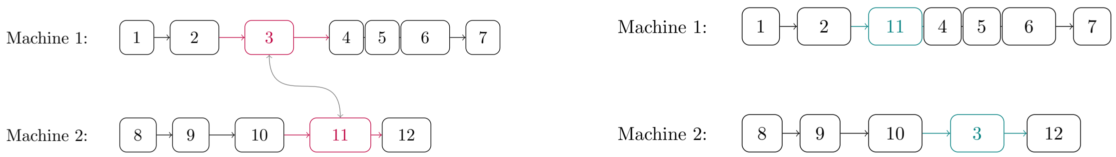


(a) Before Move Application (b) After Move Application

Figure 4.1: An Example Inter-Machine Shift Move

4.2.2 Swap Neighbourhood

A *Swap* move swaps the position of two distinct jobs j_1 and j_2 when applied to a solution. If both jobs are scheduled on the same machine, we refer to such a move as an *intra-machine* swap. Otherwise, if jobs are scheduled on different machines, we call it an *inter-machine* swap. An example of an inter-machine swap move is visualised in Figure 4.2.



(a) Before Move Application (b) After Move Application

Figure 4.2: An Example Inter-Machine Swap Move

Note that in the case of *inter-machine* swaps it is likely that the processing times and associated setup times from both jobs will change. Furthermore, completion times of all jobs that are scheduled on the affected machines after the swapped jobs need to be updated. When performing *intra-machine* swaps, the processing times of the swapped jobs do not change. However, the completion times of other jobs on the same schedule still need to be updated.

To preserve solution validity for swap moves, it has to be ensured that the first job's machine has to be eligible for the second job and vice versa.

4.2.3 Block Moves

We introduce the notion of *block moves*, as a variant of the basic shift and swap neighbourhoods: A *block* is defined as a set of jobs that are scheduled consecutively on a single machine. Therefore, a block move operates on a set of jobs instead of a single job. This concept can be applied to both *Swap* and *Shift* moves, leading to two new neighbourhoods that we call *BlockSwap* and *BlockShift* respectively. *BlockShift* moves are similar to regular shift moves but include an additional parameter l determining the length of the block. In turn, every *BlockSwap* move uses two additional attributes l_1 and l_2 representing the length of the blocks.

Block moves are motivated by our real-life application, where usually several jobs process the same material type and thus are in the best case scheduled consecutively to avoid any unnecessary setup times. Therefore, moving blocks of jobs at once can be beneficial to preserve low setup times when searching for neighbourhood solutions. To the best of our knowledge, using such block moves to approach parallel machine scheduling problems has not been considered in the literature before.

To be validity-preserving, the target machine of a *BlockShift* move has to be included in the intersection of all eligible machine sets of the affected blocks. For intra-machine *BlockSwap* moves, the blocks may not overlap and the second block's machine has to be contained in the intersection of the eligible machine sets of all jobs in the first block and vice versa.

4.2.4 Guidance in Random Move Generation

As customary for many practical scheduling problems, the size of the search neighbourhood becomes tremendously large for real-world instances. Therefore, it might be infeasible to explore all possible neighbourhood moves in reasonable time and the probability of randomly guessing an improving move usually is very low. Thus, it can be beneficial to introduce problem-specific strategies that guide neighbourhood exploration towards promising areas in the search neighbourhood. The general idea is to preferably select moves affecting jobs and machines that are currently involved in constraint or soft constraint violations.

Santos et al. [2019] propose a guidance strategy for their Simulated Annealing approach that focuses on reducing the makespan. They point out that a move can only improve the makespan if it involves the machine with the longest machine span. For this reason, they generate moves where at least one of the involved machines is fixed to be this machine, in addition to randomly generated moves.

Similarly, in our problem, the tardiness of a solution can only be improved if a tardy job is either rescheduled at an earlier time, or its predecessors are shifted to other machines or later positions. Therefore, we propose a move selection strategy that is biased towards moves that shift tardy jobs to earlier positions in the schedule as follows: Whenever a tardy job exists in the schedule, either the job itself or one of its predecessors is selected randomly. Otherwise, if no tardy job exists, any random job is chosen. Additionally, in case of an intra-machine (block) shift move we restrict the target position to be earlier in the schedule than the source position. The detailed procedure to select a job that if moved can improve tardiness is described in Algorithm 2.

To generate the moves we include, in addition to a completely random move generation, both the makespan guidance and the tardiness guidance strategies. One of the mentioned generation strategies is chosen during a search iteration based on random probabilities that are configured by parameters.

Algorithm 2 Job Selection Procedure Aiming to Minimise Tardiness

function PickJobToImproveTardiness(M)
  for $i \leftarrow |M|$ until 1 **do** ▷ iterate backwards over all job positions scheduled on machine M
    $j \leftarrow M[i]$
    if $T_j > 0$ **then**
      $k \leftarrow$ Random$(1, i)$ ▷ select the tardy job or one of its predecessors
      return $M[k]$
    end if
  end for
  $i \leftarrow$ Random$(1, |M|)$ ▷ no tardy job could be found: pick one at random
  return $M[i]$
end function

4.3 Simulated Annealing

Simulated Annealing is a meta-heuristic procedure which is inspired by the cooling processes appearing in metallurgy and has first been proposed by Kirkpatrick et al. [1983]. The main idea is to generate random neighbourhood moves and determine the probability of move acceptance based on the change in solution quality caused by the move. Moves that lead to an improved objective function or no change in solution cost are accepted in any case. To determine whether or not a move that weakens the solution quality should be accepted, the notion of temperature is used. Simply put, the higher the temperature, the higher is the probability to accept also worsening moves. As the search goes on, the temperature lowers its value according to some cooling scheme and eventually reaches values close to zero. Towards the end of search Simulated Annealing therefore evolves into a Hill-Climber as only improving moves are accepted.

In the remainder of this section, we propose three different variants of Simulated Annealing with different cooling schemes.

4.3.1 Reheating Simulated Annealing (SA-R)

This variant uses a geometric cooling scheme, i.e. $t_{i+1} := t_i \cdot \alpha$, where the cooling rate α is a predefined constant. At each temperature, a number of moves N_s is generated before the next cooling step is applied. To determine the neighbourhood from which to sample the next move, we use a set of hierarchical probabilities (p_I, p_S, and p_B). When the next move is determined to be a *block move*, we determine its size by uniformly sampling from the set $\{2, ..., B_{max}\}$. Further options for move generation are whether or not to enable guidance towards minimising tardiness or makespan. This is again handled by corresponding probabilities, p_T and p_M.

If the generated move improves the solution, it is accepted immediately. The probability of accepting a non-improving move is calculated via eq. (4.1) where δ denotes the cumulative weighted delta cost introduced by the move and t_c is the current temperature. Since improving moves are accepted unconditionally, only positive values for δ can occur in this formula. It should be noted that higher values for δ lead to lower values in the exponent and thus lower acceptance probabilities.

$$p := e^{-\frac{\delta}{t_c}} \tag{4.1}$$

When the algorithm reaches its minimum temperature T_{min}, a reheat occurs and its temperature is set to the initial temperature T_{max}. Execution stops when a time limit is reached. The pseudo-code for Reheating Simulated Annealing can be seen in Algorithm 3.

Algorithm 3 Simulated Annealing with Reheating

```
function SA-R(Solution)
    c ← Solution
    b ← c
    t_c ← T_0
    while ¬timeout do
        for i ← 0 until N_s do
            x ← GenerateNeighbour(c)
            if Accept(x, t_c) then
                c ← x
                if Cost(x) < Cost(b) then
                    b ← x
                end if
            end if
        end for
        t_c ← t_c · α
        if t_c < T_min then
            t_c ← T_0                    ▷ reheat if the minimum temperature is reached
        end if
    end while
    return b
end function
```

The parameters for this variant of Simulated Annealing are the following:

- T_{max}: Initial Temperature
- T_{min}: Minimum Temperature
- N_s: Samples per Temperature
- α: Cooling Rate
- p_I: Probability of generating inter-machine moves (as opposed to intra-machine moves)
- p_S: Probability of generating shift moves (as opposed to swap moves)
- p_B: Probability of generating block moves (as opposed to single-job moves)

- p_T: Probability of applying tardiness guidance in the move generation
- p_M: Probability of applying makespan guidance in the move generation
- B_{max}: Maximum size of blocks to generate

4.3.2 Simulated Annealing with Dynamic Cooling (SA-C)

This variant tries to continually adapt its cooling rate in such a way that the minimum temperature is reached at the end of the algorithm's time limit. Therefore, an estimate of how many iterations can still be done within the time limit is calculated after each iteration. Before every cooling step, the current cooling rate α_i is computed according to eq. (4.2), where x is the number of cooling steps left, T_{min} and t_c are the minimum and current temperature respectively. Pseudo-code for this dynamic cooling procedure can be seen in Algorithm 4.

$$\alpha_i := \sqrt[x]{\frac{T_{min}}{t_c}} \tag{4.2}$$

Algorithm 4 Dynamic Cooling Procedure

function COOLOFF(t_c)
  $i \leftarrow$ number of iterations done
  $e \leftarrow$ elapsed time
  $r \leftarrow$ remaining time
  $l \leftarrow \frac{i}{e} \cdot l$ ▷ estimate number of remaining iterations
  $q \leftarrow \frac{T_{min}}{t_c}$
  $\alpha \leftarrow \sqrt[l]{q}$
  return $t_c \cdot \alpha$
end function

In order to minimise the time spent in high temperatures, we further apply a cut-off mechanic. This counts the number of *accepted* moves at every temperature and forces an immediate cooling step if a specified threshold N_a is exceeded. This threshold is often specified as ratio $\rho = \frac{N_a}{N_s}$ and a typical value is 0.05. Overall, *SA-C* requires the same set of parameters as *SA-R*, except for α, which is replaced by ρ. The pseudo-code for SA-C can be seen in Algorithm 5.

4.3.3 Simulated Annealing with Iteration Budget (SA-I)

This variant uses a constant cooling rate to determine the temperature for each iteration. In addition to a time limit, it uses a fixed iteration budget $\mathcal{I}$ to limit its run time. The idea is to choose the value for $\mathcal{I}$ is to estimate the possible number of iterations within the given time limit, based on the execution speed of sample iterations on the benchmark machine (in our experiments, we set $\mathcal{I}$ to $185 \cdot L$, where L is the time limit in milliseconds). The provided iteration budget is split evenly over the temperatures,

Algorithm 5 Simulated Annealing with Dynamic Cooling

```
function SA-C(Solution)
    c ← Solution
    b ← c
    t_c ← T_0
    while ¬timeout do
        n_s ← 0
        n_a ← 0
        while n_s < N_s ∧ n_a < N_a do                ▷ apply cut-off if N_a is exceeded
            x ← GenerateNeighbour(c)
            if Accept(x, t_c) then
                c ← x
                if Cost(x) < Cost(b) then
                    b ← x
                end if
                n_a ← n_a + 1                          ▷ increase the counter of accepted moves
            end if
            n_s ← n_s + 1                              ▷ increase the counter of generated moves
        end while
        t_c ← CoolOff(t_c)
    end while
    return b
end function
```

such that the minimum temperature is reached when its iteration budget is exhausted. As such, the number of samples per temperature is defined as the value of a function depending on the maximum and minimum temperature, which can be seen in eq. (4.3).

$$N_s := \frac{\mathcal{I}}{\log_{\alpha}(\frac{T_{min}}{T_{max}})} \tag{4.3}$$

Once again, we apply the same cut-off mechanism as in **SA-C**. Since the cooling rate is adjusted differently than for *SA-C*, this may cause the temperature to fall below the actual minimum temperature. Due to N_s being calculated directly by *SA-I*, it does not have to be provided as a parameter. However, the cooling rate α has to be stated as a parameter for SA-I. Pseudo-code for SA-I can be seen in Algorithm 6.

Algorithm 6 Simulated Annealing with Iteration Budget

```
function SA-I(Solution)
    c ← Solution
    b ← c
    t_c ← T_0
    N_s ← I / log_α(T_min / T_max)
    c ← 0
    while ¬timeout ∧ c < I do        ▷ terminate when the iteration budget is exhausted
        c ← c + 1
        n_s ← 0
        n_a ← 0
        while n_s < N_s ∧ n_a < N_a do        ▷ apply cut-off if N_a is exceeded
            x ← GenerateNeighbour(c)
            if Accept(x, t_c) then
                c ← x
                if Cost(x) < Cost(b) then
                    b ← x
                end if
                n_a ← n_a + 1        ▷ increase the counter of accepted moves
            end if
            n_s ← n_s + 1        ▷ increase the counter of generated moves
        end while
        t_c ← t_c · α
    end while
    return b
end function
```

CHAPTER 5

Experimental Evaluation

In this Chapter, we propose a random instance generator for the problem under investigation and use it to generate a large set of instances. Automated parameter tuning is performed for the meta-heuristics described in Chapter 4 using a subset of the generated instances. We use another subset of the generated instances for conducting computational experiments with the meta-heuristics and the Mixed-Integer Programming formulations described in Chapter 3 and give a detailed evaluation of the results. Additionally, we use instances from the related literature and show that our approaches improve the state-of-the-art approaches.

5.1 Instances

To evaluate the proposed approaches, we perform a large number of experiments with a set of randomly generated instances, a set of real-life instances, and instances on a related PMSP from the literature. In Section 5.1.1, we provide information on our instance generator before we describe the set of randomly generated instances in Section 5.1.2. Later in Section 5.1.3, we present the real-life instances that have been provided to us by our industrial partner. Additional information on the problem instances from the literature are given in Section 5.1.4.

5.1.1 Instance Generator

To describe the generation of instances we use the notion of *materials*, as our real-life problem data determines the setup times using materials instead of jobs. As every job processes a single material, converting a given material-based setup time matrix into a job-based setup time matrix is a simple preprocessing step. The main reason for this kind of specification of the setup times is to reduce space requirements of the instance files. A similar instance format has also been used previously in the literature (e.g. Caniyilmaz et al. [2015]).

To create a large pool of instances we implement a random instance generator which is configured by parameters that specify the desired instance size (i.e. the number of machines, materials and jobs) as well as a random seed. This instance generator is based on a similar one proposed by Vallada and Ruiz [2011], but we extend it to include also the generation of machine eligibility constraints and due dates. The detailed pseudo-code is provided in Algorithm 7.

The processing time of each job on each machine and the setup time between every pair of materials are drawn from uniform distributions $[1, 100)$ and $[1, 125)$ respectively. Setup times between two jobs sharing the same material is set to zero.

Exactly one material is assigned to each job as follows: At first, one job after the other gets matched to an unused material until every material has been assigned exactly once. Afterwards, a randomly selected material is assigned to any job that has not been matched to a material yet. Thus, no two jobs will process the same material if the number of jobs is lower or equal to the number of materials.

For every job, the corresponding set of eligible machines is determined in the following way: First, the eligible machine count m is sampled from a uniform distribution $[1, M]$, where M is the total number of machines. Then, the set of available machines is sampled m times without replacement to determine the eligible machines for the job.

We further use three different procedures to assign randomly due dates to create different sets of instances.

In the first two procedures, the due dates are determined by constructing a reference solution for the problem and afterwards setting the scheduled completion time of each job as the corresponding due date. Thus, it is ensured by construction that a feasible schedule exists for every generated problem instance even though the generated reference solution may not be optimal with respect to makespan. The construction of such a reference solution consists of the following steps (see Algorithm 7, lines 34 - 44): First, we determine a random order of jobs in which we will schedule them one after the other. Then, for every job we randomly select one of its eligible machines according to an independently specified selection strategy. The job is then appended to the selected machine's schedule and the job's due date is set to its completion time based on the current schedule.

Our first due date generation procedure (*S-style*) constructs a solution as described above with the use of a random machine selection strategy.

The second due date generation procedure (*T-style*) also constructs a reference solution but aims to obtain tighter due dates through the use of an alternative greedy machine selection heuristic (see Algorithm 8). This alternative strategy greedily selects the machine that causes the lowest setup time when the corresponding job is scheduled.

The third due date generation procedure (*P-style*) does not rely on constructing a reference solution but assigns random due dates by sampling the values from a uniform distribution $[\widetilde{C}_{max} \cdot (1 - \tau - R/2), \widetilde{C}_{max} \cdot (1 - \tau + R/2)]$. The variables τ and R in this case are parameters determining the tightness and variance of the generated due dates. For the calculation of the approximate makespan $\widetilde{C}_{max}$, we use the formula suggested by Perez-Gonzalez et al. [2019] (see eq. (5.1)). Similar approaches to randomly sample due dates have been used by Potts and Wassenhove [1982], Chen [2006] and Lee et al. [2013].

$$\widetilde{C}_{max} := \max_{j \in J} \frac{\Sigma_{m \in E_j}(p_{j,m}) + \frac{\Sigma_{i \in J}(s_{i,j,m})}{n}}{|E_j|} \cdot \frac{n}{m} \tag{5.1}$$

5.1.2 Generated Instances

Using the previously described instance generator, we generated 560 instances that can be separated into six different categories. The instances of each category have been generated with a differently configured random instance generator (see Table 5.1).

	S-Style Due Dates	T-Style Due Dates	P-Style Due Dates
Unique Materials	110	110	60
Shared Materials	110	110	60

Table 5.1: Number of Generated Instances per Category

When every job in an instance is assigned to a different material (i.e. no pair of jobs use the same material), then the instance is classified as using *Unique Materials*. In this case, between any pair of jobs, the setup time is greater than zero. This is the case commonly found in the literature.

Instances with less materials than jobs are classified as using *Shared Materials* because at least one material is shared between multiple jobs. As mentioned earlier, in such a case the setup time between pairs of jobs is zero, because no change in tooling is required. This reflects the structure observed in real-life instances.

Two of the three due date generation variants use heuristics to arrive at reference solutions from which we can derive the due dates (a random selection procedure and Algorithm 8). For the third due date generation procedure, we set the parameters $\tau = 0.25$ and $R = 0.5$, to generate instances that are unlikely to have zero-cost solutions with respect to tardiness.

We sample the input parameters for our instance generator (i.e. the number of jobs, machines and materials) from the uniform distributions described in Table 5.2. For instances with shared materials, both the number of jobs and number of materials are sampled separately from a uniform distribution. For instances with unique materials, the number of materials is set to be equal to the number of jobs. The bounds for the value ranges have been chosen to generate instances with realistic size.

As our meta-heuristic approaches rely on a number of parameters, we use the automated parameter configuration tool SMAC to find efficient parameter configurations. To avoid over-fitting of the tuned parameters on the instances we use in our experiments, we

Algorithm 7 Instance Generator

```
function GenerateInstance(NumMachines, NumMaterials, NumJobs)
    Machines ← list from 1 to NumMachines
    Materials ← list from 1 to NumMaterials
    Jobs ← list of jobs from 1 to NumJobs

    for i ← 1 until NumJobs do
        n ← UniformRandom(1, NumMachines + 1)
        E ← select n random elements from Machines
        p ← empty dictionary
        for all m ∈ E do
            processingTime ← UniformRandom(1, 100)
            p.Add(m, processingTime)
        end for
        t ← select element from Materials
        Jobs[i].eligibleMachines ← E
        Jobs[i].processingTimes ← p
        Jobs[i].material ← t
    end for

    SetupTimes ← array with dimensions (NumMaterials + 1) × (NumMaterials + 1) × (NumMachines)
    for all pred ∈ Materials ∪ {0} do
        for all succ ∈ Materials ∪ {0} do
            for all m ∈ Machines do
                if pred = succ then
                    st ← 0
                else
                    st ← UniformRandom(1, 125)
                end if
                SetupTimes[pred, succ, m] = st
            end for
        end for
    end for

    S ← empty dictionary
    for all m ∈ Machines do
        S.Add(m, empty schedule)
    end for

    for all j ∈ Shuffle(Jobs) do                         ▷ Determine the Due Dates
        m ← SelectMachine(j)
        S[m].Append(j)
        c ← completion time of j on S[m]
        j.dueDate = c
    end for

    return NewInstance(Machines, Materials, Jobs, SetupTimes)
end function
```

Algorithm 8 Greedy Machine Selection

```
function SelectGreedyMachine(j)
    m ← null
    s ← 0
    for all e ∈ j.eligibleMachines do
        i ← ID of last scheduled job on e or 0 if the schedule is empty
        t ← SetupTimes[i, j, m]                ▷ SetupTimes as defined in Algorithm 7
        if (m = null) ∨ (t < s) then
            s ← t
            m ← e
        end if
    end for
    return m
end function
```

split the 560 randomly generated instances into a training set for parameter tuning and a validation set that is used in our final experimental evaluation. To create our training set we uniformly sampled 90 *S-style* instances with shared materials and 90 *S-style* instances with unique materials from the randomly generated instances. Similarly, we further sampled two sets of 90 instances from the *T-style* instances. Finally, we also sampled 40 *P-style* instances with shared materials and another 40 *P-style* instances with unique materials. Our training set therefore consists of 440 instances in total, while the validation set consists of the remaining 120 instances. Further details on the parameter tuning are given in Section 5.2.2.

The instance generator and the generated instances are available at `http://cdlab-artis.dbai.tuwien.ac.at/instances/parallel-machine-scheduling`.

Variable	Distribution Range	Remarks
Number of Machines	$[1, 30]$	Max. $\frac{1}{3} \cdot \lvert Jobs \rvert$
Number of Jobs	$[20, 1000]$	Step Size: 20
Number of Materials	$[1, \lvert Jobs \rvert)$	Only with Shared Materials

Table 5.2: Ranges for Specifics of Generated Instances

5.1.3 Real-Life Instances

For testing purposes, our industrial partners provided us with three real-life instances (A, B and C) that represent planning scenarios from industrial production sites. The characteristics of these instances can be seen in Table 5.3.

Based on the real-life instances, we further create additional instances as follows. Instances A and B originally contained due dates that lie in the past and thus included negative values. Therefore, we created two additional instances (A-fixed, B-fixed) by changing such due dates to non-negative values. Based on instance C, we further created four additional instances by scaling up instance C (up to 16 times the original size). Additionally, we include instance C-assigned into our instance set. It uses the same set of jobs and machines as instance C, but predetermines a machine assignment for each job. Similar to instance C, we scaled up instance C-assigned to create another four instances. The set of all real-life based instances used in our benchmark experiments can be seen in the first Column of Table 5.10.

Instance	Machines	Jobs	Materials
A	3	29	29
B	3	187	4
C	13	172	40

Table 5.3: Real-life Instance Specifics

5.1.4 Instances from the Literature

We evaluate the performance of our meta-heuristics on instances for another PMSP that has been described by Perez-Gonzalez et al. [2019][1]. The only difference to the problem we investigate is that their objective function only considers the minimisation of total tardiness and does not include the makespan. Since the makespan is the secondary objective in our problem and thus incomparably less important than total tardiness, we can directly use our meta-heuristics to approach the problem from Perez-Gonzalez et al. [2019] by simply ignoring any output on the makespan.

5.2 Computational Results

To evaluate our approaches we performed a large number of experiments based on two sets of instances. The first set of instances contains 120 randomly generated instances as well as 14 real-life instances that have been provided to us by an industrial partner. The second set of instances was previously proposed by Perez-Gonzalez et al. [2019].

5.2.1 Comparison of Mixed-Integer Programming Formulations

We implemented the MIP models described in Chapter 3 with Gurobi 8.1.1. Experiments with the MIP models were performed on a computer with an AMD Ryzen 2700X Eight-Core CPU and 16GB RAM.

All six models were evaluated on the 25 smallest generated instances from the validation set under a time limit of 1800 seconds. Table 5.4 summarises the results for each instance. The upper values per entry denote the tardiness and the lower values denote the makespan of the found solution. The best found solutions per instance are highlighted in bold and asterisks mark proven optimal solutions.

[1]The instances we use in our experiments have been kindly provided to us by the authors

It can be seen that the models with explicitly modelled machine eligibility constraints (**M2**, **M4**, and **M6**) are able to provide solutions for more instances than their counterparts that rely on pre-processed data. In fact, models **M2** and **M6** are able to provide solutions for all 25 instances, while **M4** can provide solutions for all instances except one. Interestingly, the solutions found by **M1** exhibit often higher quality than the corresponding solutions found by **M2**. This is not the case when comparing models **M3** and **M5** with their counterparts **M4** and **M6**. The novel model adaptation **M6** that we described in Chapter 3 is able to find the best solutions for 14 instances. Nine of these instances are *P-style* instances, four are *S-style* and one is a *T-style* instance. **M4** provides the best solution two times for *P-style* instances, three times for *S-style* instances and four times for *T-style* instances. Further, **M4** is able to prove optimality for two solutions, while **M6** is able to find one optimal solution. Overall, **M4** and **M6** provide the best solutions for 22 instances. This indicates that both models outperform the other in different areas of the instance space.

To calculate the relative performance, we use the Relative Percentage Deviation (RPD) for every instance I and solution S as defined in eq. (5.2). RPD values have also been used as a performance measure in publications on related problems (e.g. Vallada and Ruiz [2011]).

$$RPD_{I,S} := \frac{\text{cost}_{I,S} - \text{best}_I}{\text{best}_I} \tag{5.2}$$

Figure 5.1 visualises the RPD values of each model in the form of box plots. In Figure 5.1a, results for all models except for **M3** are presented. Figure 5.1b shows RPD values of models with explicit machine eligibility constraints and Figure 5.1c shows RPD values for models **M4**, **M5** and **M6**. Model **M3** is excluded from the comparisons, as it was unable to solve most instances. Only those instances that could be solved by all compared models within the time limit are included in the plots to avoid missing values.

Overall, we conclude that models **M4** and **M6** provide the best results for the majority of instances in our experiments. The best model depends on the instance characteristics: For *T-style* instances, **M4** shows the best performance, while **M6** produces the best results for *P-style* instances.

5.2.2 Comparison of Meta-Heuristic Approaches

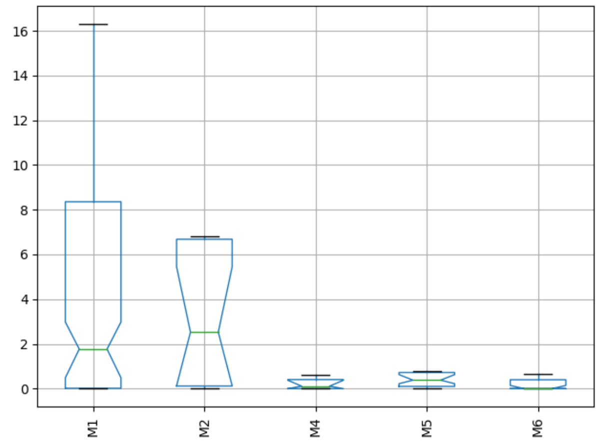
Both parameter tuning and benchmark experiments for the meta-heuristics have been executed on a computer with an Intel Xeon E5-2650 v4 12-Core processor that has 24 logical cores and 250 gigabytes of RAM.

Parameter Tuning

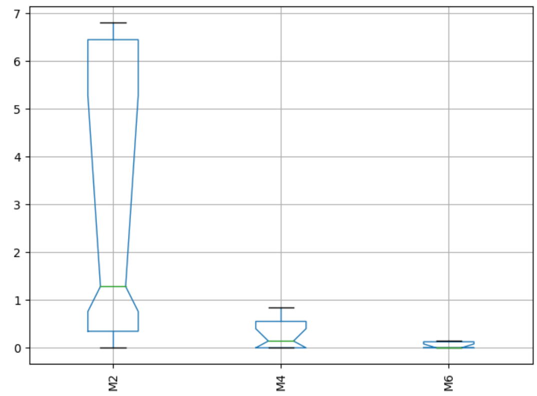
As our methods include various parameters which have an impact on the final results, we performed automated parameter tuning for each proposed meta-heuristic algorithm

Instance	M1	M2	M3	M4	M5	M6
P_13-80-80_1	5280 601	10015 879	–	2125 520	4234 673	**1327** **471**
P_15-60-60_1	1096 379	1349 341	1480 406	964 427	1323 430	**958** **371**
P_15-63-80_1	4156 568	8901 918	–	1900 496	**1256** **496**	2960 495
P_16-100-100_1	12208 838	12412 997	–	3736 545	4505 597	**2917** **523**
P_16-180-180_1	–	60749 1939	–	78940 2555	–	**47234** **1776**
P_17-100-100_1	10103 723	11783 726	–	5122 475	6536 588	**3649** **532**
P_18-80-80_2	4393 457	5815 669	1787 372	2322 427	2323 368	**1649** **401**
P_20-180-180_1	–	49713 2046	–	64061 2159	–	**34619** **1457**
P_22-140-140_1	–	24091 1226	–	**10542** **683**	–	11655 677
P_29-140-140_1	–	14375 657	–	9947 664	–	**7130** **464**
P_3-17-20_1	**927** **520**	1029 576	957 611	**927** **520**	**927** **520**	1029 576
P_7-19-40_1	**1955** **514**	2204 497	1976 453	2173 472	2419 529	2237 503
P_9-180-180_1	–	138861 3339	–	80059 2784	–	**78859** **2645**
S_10-120-180_1	–	50869 2467	–	57902 2998	–	**33756** **2427**
S_1-3-100_1	35660 7664	35660 7664	**34442** **7589**	**34442** **7589**	56919 8097	56919 8097
S_15-80-80_2	857 868	6108 999	–	94 859	71 721	**59** **734**
S_15-80-80_3	1084 944	1689 871	110 649	0 368	2 961	**0** **322**
S_22-149-160_1	–	47583 2483	–	**6467** **1072**	–	14175 1383
S_4-16-20_1	**0** **409** *	**0** **409** *	**0** **409** *	**0** **409** *	**0** **409** *	**0** **409** *
T_10-24-40_1	**0** **373** *	**0** **373** *	32 512	**0** **373** *	**0** **373** *	47 456
T_15-77-80_1	613 700	5970 813	–	**14** **644**	617 751	38 614
T_18-56-100_1	25167 1515	11365 1123	–	**1455** **524**	1870 662	1542 735
T_20-76-100_1	–	9036 647	–	**877** **580**	1283 573	1106 526
T_28-34-100_1	–	2475 468	–	154 382	–	**23** **478**
T_3-12-200_1	**192462** **7190**	297785 8044	–	–	–	192624 6408

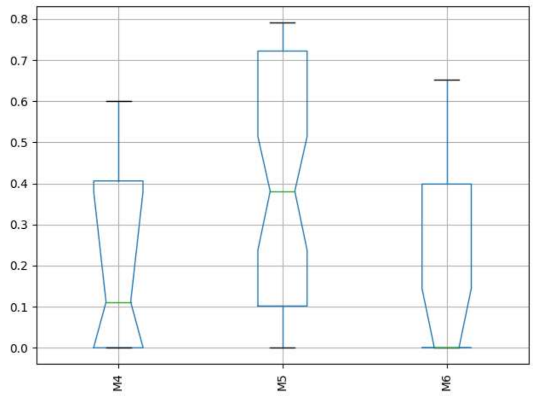
Table 5.4: Comparison of Solutions Found by MIP Formulations. Tardiness and makespan are denoted by the top and bottom values respectively. The best results per instance are highlighted in boldface and asterisks mark solutions that have been proven optimal by MIP. Dashes mean that no valid solution was found within the time limit.



(a) All Models (Except **M3**)



(b) **M2**, **M4**, and **M6**



(c) **M4**, **M5**, and **M6**

Figure 5.1: RPD Values of MIP Models

using Sequential Model-based Algorithm Configuration (SMAC) as proposed by Hutter et al. [2011].

	T_{max}	T_{min}	N_s	p_I	p_S	p_B	p_T	p_M	B_{max}	α	ρ
Minimum Value	500	0.0001	1000	0.0	0.0	0.0	0.0	0.0	2	0.75	0.01
Maximum Value	10000	10.0	250000	1.0	1.0	1.0	1.0	1.0	200	0.999	1.0
Default Value	1000	0.01	10000	0.9	0.5	0.1	0.1	0.1	100	0.9	0.05

Table 5.5: Parameter Value Ranges for SMAC

For every Simulated Annealing variant, we started 24 parallel SMAC runs with a wallclock time limit of 18 hours per run. The initial configurations used for the starting point of SMAC are based on intuition about plausible parameter values. Table 5.5 shows the value ranges for all parameters as well as their initial values. The best tuned parameters from all parallel SMAC runs are listed in Table 5.6.

Configuration	T_{max}	T_{min}	N_s	p_I	p_S	p_B	p_T	p_M	B_{max}	α	ρ
SA-I	5485.42	9.17	–	0.49	0.78	0.01	0.31	0.81	100	0.95	0.78
SA-C	6188.72	5.18	191058	0.53	0.81	0.02	0.47	0.27	32	–	0.42
SA-R	2764.93	6.73	20339	0.66	0.84	0.04	0.85	0.71	26	0.93	–

Table 5.6: Configurations Obtained from Automated Parameter Tuning

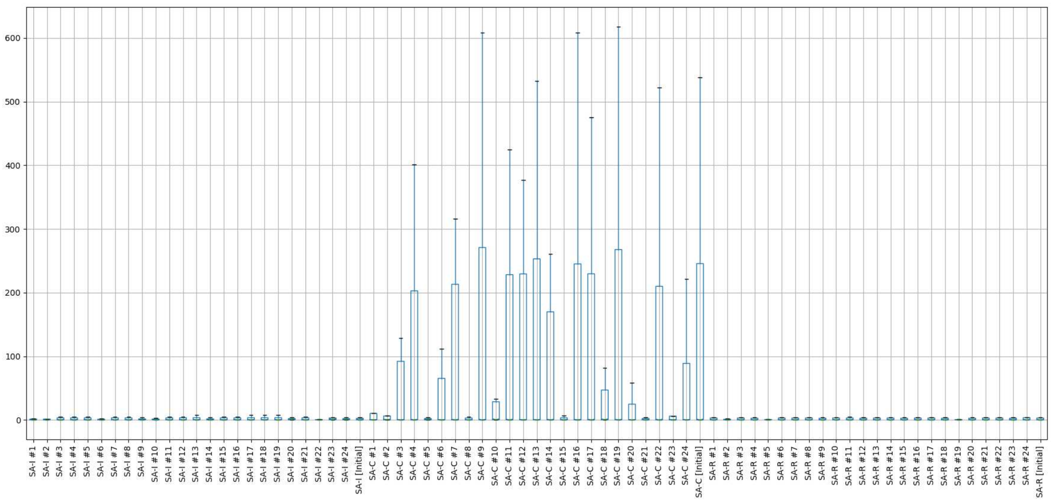
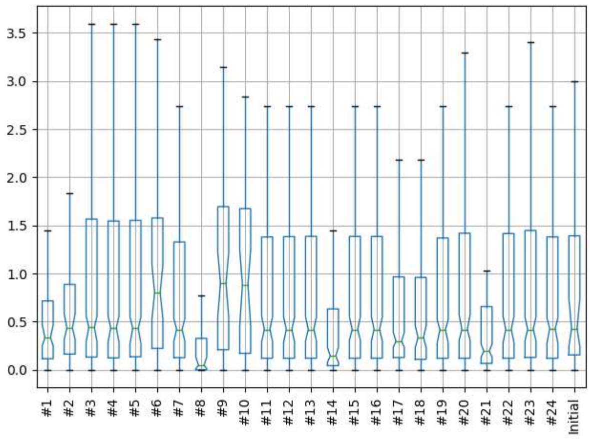


Figure 5.2: RPD Values for All Incumbent Configurations

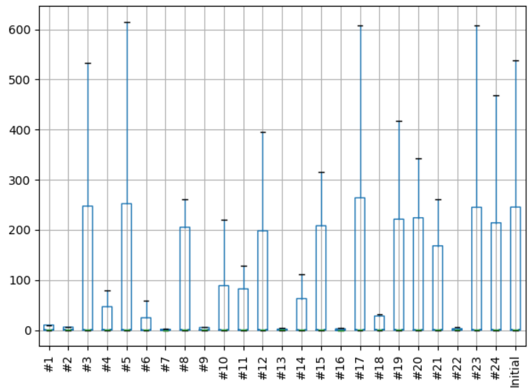
To investigate the robustness of the SA variants against different configurations, we conducted additional experiments for the so-called *incumbent configurations* (i.e. the best found configurations) of each parallel SMAC run. We let every SA variant run 20 times with every incumbent configuration on the validation set. Box plots for the resulting RPD values can be seen in Figure 5.2. It can be seen that SA-I and SA-R (on the left and right thirds of the plot respectively) are much more robust against changes in the configuration than SA-C. Figure 5.3 shows the RPD values for all incumbents, grouped per SA variant for a clearer visual representation. Inspection of the plot scales shows that SA-C and SA-R are the least and most robust of the compared SA variants, respectively. Figures 5.3a and 5.3c show that most configurations result in a very similar performance for both SA-I and SA-R. However, some configurations lead to significantly better results than the others. This suggests that both SA-I and SA-R are largely robust against changes in the configuration, but still have the potential for some optimisation.

Results on Randomly Generated Instances

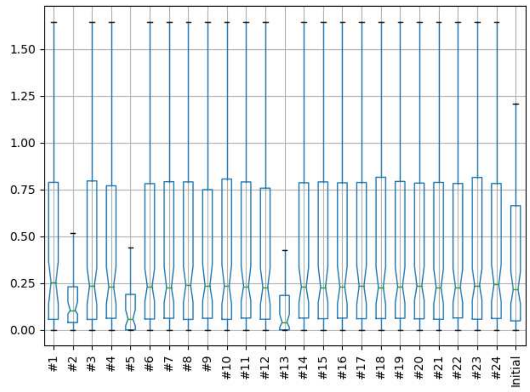
In this section, we present the results produced by the proposed Constructive Heuristic (Algorithm 1) and Simulated Annealing variants. For each of the investigated Simulated Annealing variants, we performed experiments with a randomly constructed initial solution (R) and initial solutions produced by our Constructive Heuristic (CH). All



(a) Incumbents for SA-I



(b) Incumbents for SA-C



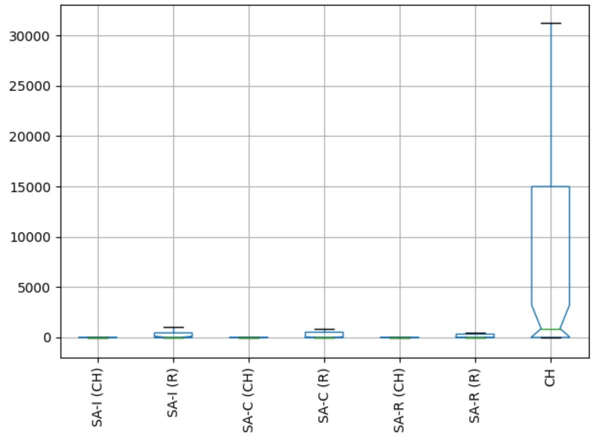
(c) Incumbents for SA-R

Figure 5.3: RPD Values for All Incumbent Configurations per SA Variant

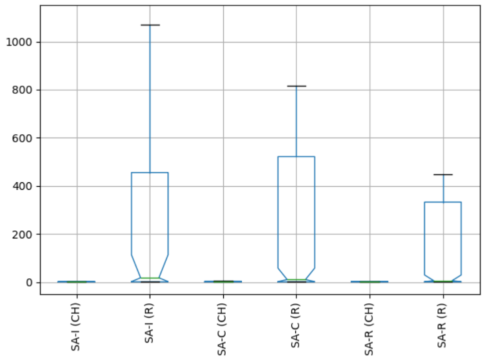
experiments were performed within a time limit of 60 seconds per run regardless of the instance size. For SA-I, we set an iteration budget of 11 100 000 iterations, as this corresponds to the average number of iterations that the other Simulated Annealing variants performed within the 60 seconds time limit.

Tables 5.7 to 5.9 show the detailed experimental results for all SA variants as *tardiness / makespan* pairs. The entries represent the low median solution cost found by the compared algorithms and the best results per instance are highlighted in boldface. Figure 5.4 further visualises an overview of the relative algorithm performances on the entire validation set as box plots. As expected, all Simulated Annealing variants are able to significantly improve the quality of their initial solution (Algorithm 1). Further, we can see in Figure 5.4b that SA produces significantly better results when starting from a good initial solution as opposed to a randomly constructed initial solution. Figure 5.4c shows box plots over the median RPD values per instance for each algorithm. The fact that the median RPD values are not notably different from the mean RPD values for SA (CH) indicates its robustness.

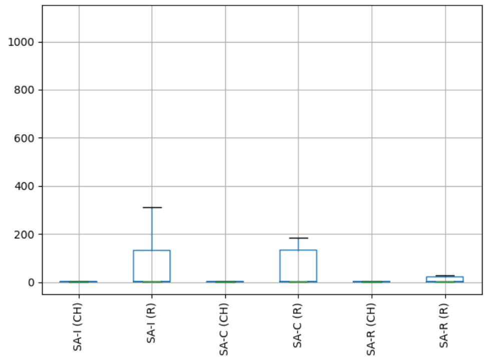
Comparing the SA variants with random initial solution, we conclude that SA-R shows a slightly better performance compared to the other two SA variants. We further observe that all SA variants produce similar results when starting from a greedily constructed initial solution.



(a) Means, Including Solutions from CH



(b) Means



(c) Medians

Figure 5.4: RPD Values for SA Variants, with Constructive Heuristic (CH) and Random Initial Solution (R)

For a more detailed evaluation, we divide the instances in the validation set into categories, according to their due date (*S-style*, *T-style* and *P-style*) and material usage (*shared* and *unique*) characteristics. The mean RPD values for *S-style*, *T-style* and *P-style* instances can be seen in Figures 5.5 to 5.7 respectively.

Table 5.7 shows that the majority of algorithm runs find solutions for *S-style* instances with zero tardiness, which means that the relevant objective in these instances is the minimisation of the makespan. As can be seen in Figure 5.5a, all SA variants find solutions that are significantly better than those provided by the Constructive Heuristic. However, the solutions created by CH provide a better starting point for SA than randomly generated solutions, which can be seen in Figure 5.5b. The relative performances of the SA variants with CH on *S-style* instances with unique and shared materials can be seen in Figure 5.5c respectively. In both cases, SA-R outperforms the other SA variants significantly.

Table 5.8 shows that it is harder to find zero-tardiness solutions for *T-style* than it is for *S-style* instances, as many results contain tardiness. The fact that the meta-heuristics struggle with finding zero-tardiness solutions in combination with the high penalty for tardiness leads to high RPD values (as can be seen in Figure 5.6). The results regarding the Constructive Heuristic and random initial solutions are similar to those for *S-style* instances. Simulated Annealing is again able to significantly improve solutions provided by the Constructive Heuristic (see Figure 5.6a) and CH again provides a better starting point for the search than random solutions (see Figure 5.6b). Interestingly, the compared meta-heuristics exhibit a much more consistent behaviour on *T-style* instances with shared materials than on instances with unique materials, resulting in significantly lower RPDs (see Figures 5.6c and 5.6d).

In contrast to the other instances, there are no guarantees for the existence of solutions without tardiness in **P-style** instances. This fact is underlined in Table 5.9, which shows that the compared meta-heuristics could find zero-tardiness solutions for only one instance. As almost all found solutions contain tardy jobs, the RPD values are dominated by the differences in total tardiness. From Table 5.9 we observe that the difference between solutions in terms of tardiness is usually only a fraction of the best solution's tardiness, which leads to the small RPD values that can be seen in Figures 5.7b to 5.7d. Even though the differences are much less pronounced than for *S-style* and *T-style* instances, it can still be seen that SA benefits from greedily constructed initial solutions.

Overall, it can be seen that Simulated Annealing is able to produce high-quality solutions and that it benefits significantly from good initial solutions. Further, we observe that the performance of the compared meta-heuristics is more consistent on instances with shared materials, as opposed to instances with unique materials.

Comparison with Generated Reference Solutions

For both *S-style* and *T-style* instances, our instance generator constructs reference solutions which have zero total tardiness per construction. Figures 5.8a and 5.8b compare the mean RPD of the SA variants on *S-style* and *T-style* instances with the RPD values of the reference solutions.

It can be seen that the SA variants are able to generate solutions that are considerably better than the sample solutions for *S-style* instances. For the majority of *T-style* instances, the SA meta-heuristics produce solutions with low total tardiness. The solutions produced

Instance	SA-I (CH)	SA-I (R)	SA-C (CH)	SA-C (R)	SA-R (CH)	SA-R (R)
1-3-100_1	**0 / 10746**	48 / 6362	**0 / 10746**	48 / 6328	**0 / 10746**	48 / 6362
10-120-180_1	0 / 771	0 / 761	0 / 838	0 / 806	0 / 771	**0 / 732**
10-468-800_1	0 / 4539	23 / 5294	0 / 3860	15 / 3798	**0 / 3640**	23 / 3879
11-680-680_1	0 / 3538	0 / 3823	0 / 3110	0 / 2944	**0 / 2833**	0 / 2906
12-380-380_2	**0 / 1407**	65 / 1442	5 / 1535	0 / 1576	5 / 1471	0 / 1465
12-700-700_1	0 / 3351	0 / 3727	0 / 2935	0 / 2804	**0 / 2682**	47 / 2788
12-720-720_1	0 / 3730	0 / 3948	0 / 3044	0 / 2945	**0 / 2846**	0 / 2851
13-476-540_1	0 / 1875	**0 / 1814**	0 / 2044	0 / 2026	0 / 1832	0 / 1851
14-940-940_1	0 / 4716	43 / 5009	0 / 3628	26 / 3311	**0 / 3141**	0 / 3164
15-500-500_1	0 / 1512	0 / 1486	0 / 1622	0 / 1631	0 / 1495	**0 / 1477**
15-80-80_2	0 / 336	0 / 333	0 / 337	0 / 340	**0 / 314**	0 / 320
15-80-80_3	0 / 272	0 / 275	0 / 288	0 / 282	0 / 274	**0 / 271**
16-646-660_1	0 / 2437	0 / 2632	0 / 2101	0 / 2015	0 / 1875	**0 / 1837**
18-460-660_1	0 / 1679	0 / 2271	0 / 1883	0 / 1823	0 / 1666	**0 / 1617**
18-763-800_1	0 / 2830	0 / 3194	0 / 2430	0 / 2157	0 / 2094	**0 / 2031**
19-800-800_1	41 / 2624	0 / 3055	33 / 2193	0 / 2193	**0 / 1881**	0 / 1977
20-794-800_1	0 / 2710	42 / 2888	0 / 2186	2 / 1998	**0 / 1873**	0 / 1883
21-530-560_1	0 / 1261	0 / 1504	0 / 1309	0 / 1319	**0 / 1186**	0 / 1195
22-149-160_1	0 / 363	0 / 369	0 / 396	0 / 402	**0 / 350**	0 / 352
22-519-740_1	0 / 1815	0 / 2334	0 / 1772	40 / 1756	**0 / 1503**	0 / 1522
22-600-600_1	0 / 1408	0 / 1774	0 / 1360	0 / 1392	0 / 1220	**0 / 1208**
23-612-640_1	0 / 1718	0 / 1790	0 / 1429	0 / 1392	0 / 1272	**0 / 1231**
24-900-900_1	0 / 2701	0 / 2807	0 / 2025	0 / 1856	0 / 1731	**0 / 1695**
25-300-300_1	0 / 575	0 / 576	0 / 655	0 / 631	0 / 558	**0 / 535**
25-660-660_1	0 / 1476	0 / 1819	0 / 1410	0 / 1345	**0 / 1133**	0 / 1190
26-181-500_1	**0 / 810**	0 / 904	0 / 992	0 / 977	0 / 836	0 / 833
26-414-460_1	3 / 768	**0 / 777**	0 / 885	0 / 898	0 / 796	0 / 785
27-249-660_1	**0 / 1089**	0 / 1238	0 / 1268	0 / 1253	0 / 1105	0 / 1105
28-105-420_1	0 / 667	**0 / 635**	0 / 771	0 / 787	0 / 668	0 / 652
29-580-580_1	0 / 1284	0 / 1266	0 / 1093	0 / 1032	**0 / 876**	0 / 892
3-500-500_1	5 / 11156	23 / 11373	2 / 8205	40 / 8167	**0 / 8485**	2 / 9237
30-720-720_2	0 / 1611	43 / 1746	0 / 1278	7 / 1200	**0 / 1081**	7 / 1105
30-760-760_1	0 / 1615	0 / 1912	0 / 1433	0 / 1291	0 / 1160	**0 / 1157**
4-16-20_1	**0 / 409**	**0 / 409**	**0 / 409**	**0 / 409**	**0 / 409**	**0 / 409**
4-460-460_1	0 / 6886	5 / 7018	**0 / 5741**	0 / 5779	0 / 5962	0 / 6046
5-14-620_1	0 / 7499	0 / 7737	0 / 5583	**0 / 5462**	0 / 5672	0 / 5570
5-500-500_1	0 / 5669	0 / 5834	**0 / 4907**	0 / 4909	0 / 5048	0 / 4978
7-166-600_1	0 / 4304	0 / 4829	0 / 4118	0 / 4108	**0 / 4072**	67 / 4282
7-50-360_1	**0 / 2221**	0 / 2274	0 / 2332	0 / 2344	0 / 2267	0 / 2245
8-780-780_1	0 / 6717	0 / 7034	0 / 4892	0 / 4785	**0 / 4769**	0 / 4847

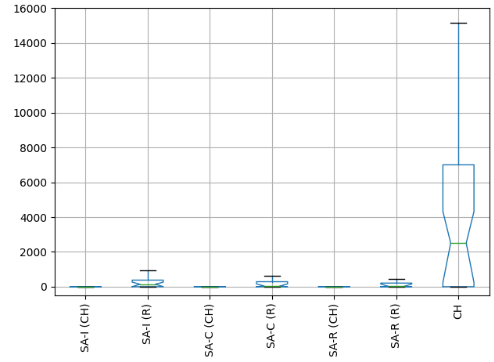
Table 5.7: Median Results as *Tardiness / Makespan* Pairs for *S-style* Instances in the Validation Set. The best results per instance are highlighted in boldface.

Instance	SA-I (CH)	SA-I (R)	SA-C (CH)	SA-C (R)	SA-R (CH)	SA-R (R)
1-73-620_1	0 / 51539	0 / 51486	0 / 36170	**0 / 35794**	0 / 37602	0 / 37505
10-230-700_1	**1 / 3246**	36 / 3813	55 / 3126	167 / 3100	2 / 3137	68 / 3444
10-24-40_1	**18 / 319**	33 / 319	**18 / 319**	63 / 341	68 / 319	**18 / 319**
10-389-940_1	**45 / 4996**	101 / 5641	53 / 4274	222 / 4239	156 / 4706	234 / 4929
11-400-400_1	**46 / 1795**	345 / 1565	214 / 1682	76 / 1654	95 / 1771	365 / 1610
12-351-1000_1	74 / 4648	449 / 4876	**42 / 3726**	79 / 3696	75 / 4086	278 / 5058
13-940-940_1	**26 / 4029**	598 / 4262	39 / 3290	222 / 3120	66 / 3272	141 / 4416
14-520-520_1	3 / 1470	8 / 1451	18 / 1634	13 / 1582	**0 / 1502**	0 / 1537
14-680-680_1	65 / 2625	230 / 2555	60 / 2273	50 / 2149	**31 / 2206**	190 / 2989
14-680-680_2	0 / 2342	364 / 2755	0 / 2209	146 / 2160	**0 / 2108**	115 / 2409
15-400-400_1	**0 / 1081**	5 / 1128	29 / 1233	3 / 1189	29 / 1162	29 / 1190
15-740-740_2	0 / 2242	388 / 2763	1 / 2192	50 / 2093	**0 / 2231**	93 / 2345
15-77-80_1	0 / 317	0 / 337	0 / 335	0 / 334	**0 / 313**	0 / 317
17-520-520_1	138 / 1250	325 / 1376	**124 / 1332**	464 / 1331	147 / 1331	403 / 1706
18-56-100_1	64 / 318	51 / 316	45 / 318	110 / 340	10 / 316	**5 / 313**
18-820-820_2	45 / 2557	271 / 2570	34 / 2110	253 / 1937	**25 / 2071**	247 / 2053
18-90-880_1	14 / 2415	123 / 2560	**14 / 2099**	205 / 2096	23 / 2157	122 / 2276
2-380-380_1	0 / 12602	98 / 12371	0 / 9408	**0 / 9325**	0 / 9835	51 / 10619
2-490-800_1	0 / 30975	7 / 30893	0 / 21557	**0 / 21226**	0 / 31120	0 / 22518
20-76-100_1	0 / 282	0 / 291	0 / 294	35 / 300	0 / 270	**0 / 261**
22-471-480_1	130 / 864	80 / 892	92 / 996	126 / 945	**33 / 967**	292 / 974
22-760-760_1	22 / 1720	200 / 1941	40 / 1561	205 / 1516	**8 / 1489**	31 / 1597
24-280-280_1	4 / 508	8 / 512	11 / 558	92 / 571	**0 / 512**	43 / 700
24-400-400_1	**34 / 663**	119 / 817	35 / 742	301 / 734	34 / 678	173 / 747
25-352-680_1	5 / 1061	189 / 1463	0 / 1206	174 / 1178	**0 / 1122**	122 / 1182
26-220-240_1	**0 / 388**	67 / 421	0 / 452	62 / 461	0 / 400	62 / 591
26-334-480_1	49 / 720	89 / 902	**5 / 828**	45 / 809	52 / 1030	9 / 977
26-540-540_1	10 / 787	134 / 1003	10 / 875	26 / 864	10 / 837	**9 / 833**
27-360-360_1	22 / 544	65 / 537	4 / 597	69 / 607	**2 / 550**	51 / 595
27-540-540_1	35 / 784	94 / 997	**0 / 863**	257 / 873	29 / 835	93 / 907
28-189-240_1	0 / 384	54 / 371	0 / 419	0 / 418	**0 / 379**	65 / 383
28-34-100_1	**0 / 231**	0 / 241	**0 / 231**	**0 / 231**	**0 / 231**	39 / 237
28-371-520_1	**0 / 714**	110 / 858	6 / 808	16 / 781	0 / 741	135 / 788
28-620-620_1	**2 / 980**	72 / 1266	85 / 990	73 / 1004	3 / 934	154 / 1193
29-137-340_1	21 / 454	52 / 469	5 / 521	79 / 524	**2 / 492**	6 / 479
3-12-200_1	2 / 3133	107 / 3138	**2 / 3006**	109 / 3067	107 / 3210	86 / 3241
30-580-580_1	16 / 929	104 / 1040	**11 / 872**	175 / 838	16 / 812	141 / 1017
6-540-540_1	19 / 4516	146 / 4944	**0 / 4131**	54 / 4105	54 / 4243	54 / 4267
6-860-860_1	92 / 9345	87 / 9744	**45 / 6971**	111 / 6861	45 / 7966	78 / 7911
8-3-260_1	178 / 1229	299 / 1250	**162 / 1266**	221 / 1182	189 / 1259	220 / 1231

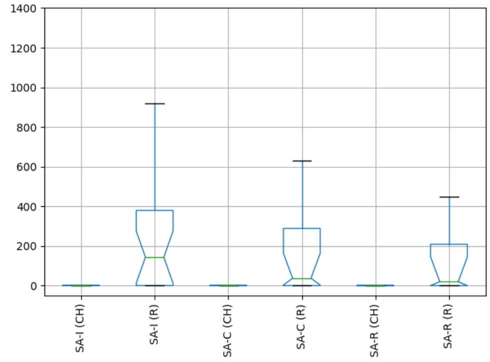
Table 5.8: Median Results as *Tardiness / Makespan* Pairs for *T-style* Instances in the Validation Set. The best results per instance are highlighted in boldface.

Instance	SA-I (CH)	SA-I (R)	SA-C (CH)	SA-C (R)	SA-R (CH)	SA-R (R)
1-364-580_1	0 / 41388	0 / 41487	0 / 32825	**0 / 32504**	0 / 33940	0 / 34177
1-829-860_1	**3314 / 62953**	15353 / 62253	5432 / 51146	8441 / 50725	4194 / 60661	7716 / 57869
10-376-680_1	**418 / 4076**	473 / 3755	527 / 3270	582 / 3154	438 / 3590	667 / 3550
13-760-760_1	710 / 3583	934 / 3467	749 / 2806	701 / 2730	**698 / 3157**	776 / 2950
13-80-80_1	665 / 359	618 / 377	436 / 359	**414 / 334**	514 / 361	423 / 370
15-60-60_1	656 / 313	778 / 392	656 / 313	**626 / 326**	658 / 398	718 / 307
15-63-80_1	548 / 369	536 / 386	600 / 369	515 / 424	**488 / 339**	540 / 391
15-640-640_1	340 / 2607	**281 / 2536**	298 / 2034	350 / 1960	406 / 2105	324 / 2598
16-100-100_1	350 / 335	492 / 359	**314 / 334**	599 / 326	321 / 336	354 / 327
16-180-180_1	597 / 577	566 / 572	564 / 571	**552 / 577**	623 / 540	605 / 540
16-233-260_1	**430 / 932**	487 / 781	485 / 776	436 / 768	503 / 759	543 / 785
17-100-100_1	1346 / 353	1213 / 350	1342 / 355	1110 / 330	1287 / 342	**901 / 322**
18-80-80_2	851 / 306	931 / 327	803 / 307	756 / 307	805 / 307	**752 / 312**
19-540-540_1	565 / 1656	551 / 1670	552 / 1337	546 / 1307	659 / 1783	**544 / 1411**
2-760-760_2	1874 / 26221	2215 / 26225	2190 / 20551	1208 / 20646	**1177 / 23483**	1267 / 22758
20-180-180_1	487 / 416	552 / 426	**419 / 483**	523 / 460	497 / 462	547 / 465
20-340-340_1	658 / 685	830 / 686	**621 / 794**	679 / 791	709 / 799	664 / 796
21-62-740_1	**444 / 1929**	641 / 1977	525 / 1591	458 / 1627	562 / 2190	602 / 2118
22-140-140_1	456 / 317	681 / 324	474 / 333	431 / 318	457 / 315	**375 / 341**
23-394-420_1	294 / 792	285 / 899	**205 / 817**	246 / 908	335 / 868	349 / 863
23-840-840_1	**282 / 2214**	705 / 2311	290 / 1743	910 / 1714	347 / 2056	553 / 2181
26-223-260_1	673 / 411	599 / 428	593 / 482	803 / 486	**591 / 491**	755 / 469
27-268-860_1	113 / 1786	146 / 1925	**104 / 1498**	148 / 1462	134 / 1862	224 / 1434
28-340-340_1	808 / 576	662 / 615	823 / 598	749 / 605	**563 / 719**	884 / 709
28-594-760_1	512 / 1590	616 / 1621	443 / 1277	690 / 1258	**430 / 1682**	507 / 1363
29-140-140_1	665 / 307	**546 / 305**	630 / 307	742 / 310	814 / 307	634 / 287
29-170-760_1	885 / 1432	1406 / 1540	751 / 1187	840 / 1210	**714 / 1303**	1110 / 1549
3-17-20_1	**937 / 525**	1212 / 546	945 / 568	1073 / 527	954 / 512	957 / 611
3-580-580_1	**293 / 12336**	611 / 12517	626 / 9913	424 / 9683	602 / 10740	367 / 10852
30-13-560_1	658 / 954	674 / 957	656 / 800	678 / 795	643 / 819	**640 / 946**
30-148-220_1	**490 / 312**	622 / 316	500 / 359	591 / 337	500 / 348	555 / 338
30-332-340_1	**396 / 489**	483 / 518	399 / 565	472 / 521	431 / 552	564 / 660
30-52-420_1	447 / 666	327 / 606	**327 / 595**	349 / 628	357 / 644	465 / 637
4-620-620_1	8435 / 9794	8864 / 9925	7571 / 7772	7259 / 7829	**7011 / 9728**	7931 / 9793
5-83-360_1	218 / 3271	458 / 4520	**208 / 3398**	290 / 3341	261 / 3736	233 / 3688
6-680-680_2	**30 / 6559**	212 / 6874	185 / 5179	96 / 5125	142 / 5577	140 / 6027
7-19-40_1	2002 / 520	**1937 / 507**	1999 / 511	2188 / 483	2035 / 477	2134 / 533
7-249-980_1	**898 / 8503**	1487 / 8748	1092 / 6388	1146 / 6262	1035 / 7311	1026 / 8730
7-431-480_1	985 / 4200	886 / 4159	977 / 3440	897 / 3316	**832 / 3625**	876 / 4332
9-180-180_1	769 / 898	797 / 889	835 / 952	929 / 961	**711 / 994**	1160 / 1237

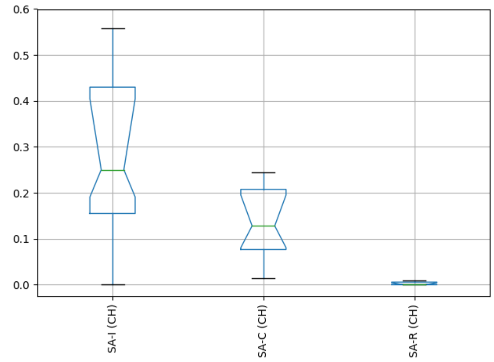
Table 5.9: Median Results as *Tardiness / Makespan* Pairs for *P-style* Instances in the Validation Set. The best results per instance are highlighted in boldface.



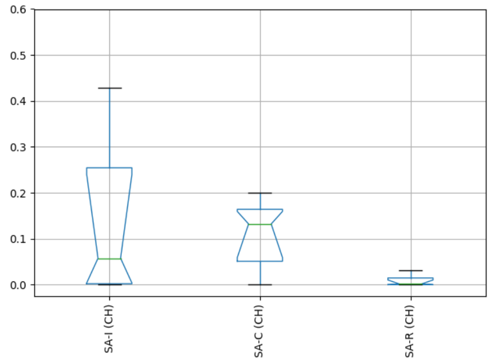
(a) Means Including Constructive Heuristic



(b) Means without Constructive Heuristic



(c) Means over Instances with Unique Materials



(d) Means over Instances with Shared Materials

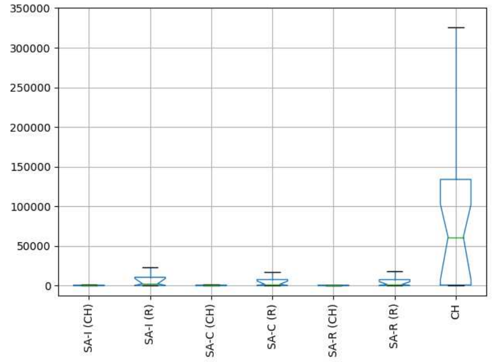
Figure 5.5: RPD Values for S-style Instances

by SA with greedily constructed initial solutions have 33 total tardiness on average (and a median of 13). On average, these SA variants produce zero-tardiness solutions for 13 instances. However, the makespans in the reference solutions are significantly higher than in the solutions produced by SA, as can be seen in Figure 5.8c.

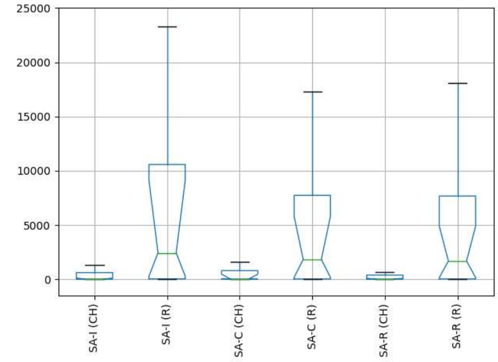
Overall, it can be seen that it is indeed more difficult to find zero-tardiness solutions for the *T-style* instances than for *S-style* instances. Further, the fact that the solutions produced by the meta-heuristics have significantly lower makespan than the reference solutions while having relatively low tardiness values suggests that there exist solutions with a lower cost than the reference solutions.

Results on Real-Life Instances

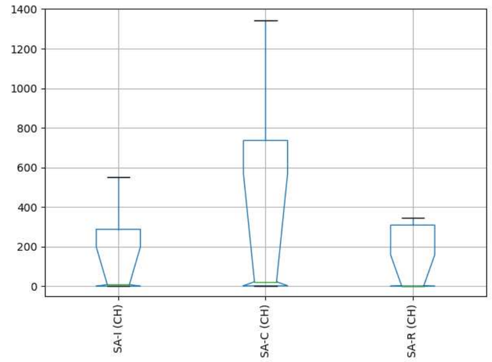
To evaluate our approaches on the real-life instances, we use the same computational environment and time limits as in Sections 5.2.1 and 5.2.2. Table 5.10 shows the best solution costs obtained by MIP and the median solution costs provided by each SA variant. All methods obtained the same solution costs for instances A-fixed and C-assigned, which



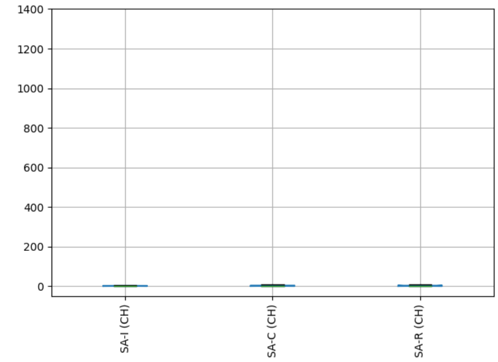
(a) Means Including Constructive Heuristic



(b) Means without Constructive Heuristic



(c) Means over Instances with Unique Materials



(d) Means over Instances with Shared Materials

Figure 5.6: RPD Values for T-style Instances

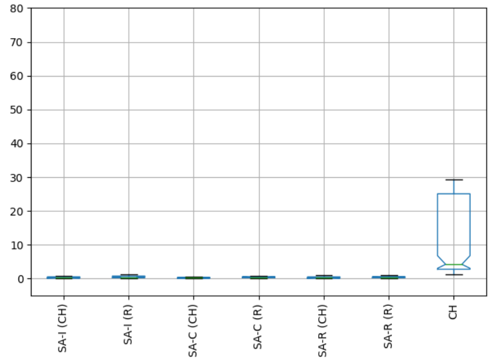
also could be solved to optimality by MIP. For the rest of the instances, SA produced better results than MIP in our experiments. All SA variants achieved the same result in all 20 runs for six of the instances (A, A-fixed, B, B-fixed, C-restricted and C-restricted x2). Only for the larger instances, we observe varying solution qualities between the different solution approaches. SA-R exhibits a slightly better performance on the larger instances (C-x8, C-x16 and C-assigned-x8), while SA-I performs best on smaller instances (C-x2 and C-x4). The box plots in Figure 5.9 show that all SA variants perform very similarly, with RPDs close to zero.

Comparison of Configurations

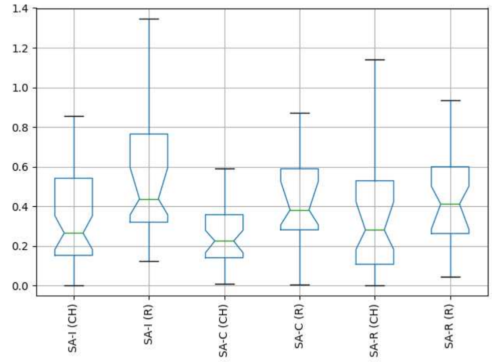
As can be seen in Section 5.2.2, the chosen configuration for an algorithm can have a large impact on its performance. Thus, the possibility exists that the configuration proposed by SMAC for one of the SA variants is significantly better or worse than the configurations for the other variants. To exclude this case, we conducted experiments for all combinations of SA variants and configurations. We label each of the algorithm/configuration pairs

Instance	MIP	SA-I	SA-C	SA-R
A	–	**1476042115** **1091223**	**1476042115** **1091223**	**1476042115** **1091223**
B	338062834 3128548	**334900552** **3128548**	**334900552** **3128548**	**334900552** **3128548**
C	12884400 2671200	**12884400** **2394000**	12884400 2404800	**12884400** **2394000**
A-fixed	**76593** **1091223** *	**76593** **1091223**	**76593** **1091223**	**76593** **1091223**
B-fixed	15328158 3128548	**11634212** **3128548**	**11634212** **3128548**	**11634212** **3128548**
C-x2	–	**25768800** **4748400**	25768800 4766400	25768800 4777200
C-x4	–	**51537600** **9511200**	51537600 9565200	51537600 9608400
C-x8	–	344577600 19152000	346960800 19425600	**343814400** **19407600**
C-x16	–	5000468400 39020400	4951357200 39369600	**4867243200** **39106800**
C-assigned	**12884400** **5094000** *	**12884400** **5094000**	**12884400** **5094000**	**12884400** **5094000**
C-assigned-x2	40485600 10512000	**25952400** **9907200**	**25952400** **9907200**	**25952400** **9907200**
C-assigned-x4	–	**96285600** **19598400**	**96285600** **19598400**	**96285600** **19598400**
C-assigned-x8	–	1601834400 38851200	1601834400 38851200	**1600657200** **38829600**
C-assigned-x16	–	11827620000 77421600	**11811006000** **77464800**	11812179600 77486400

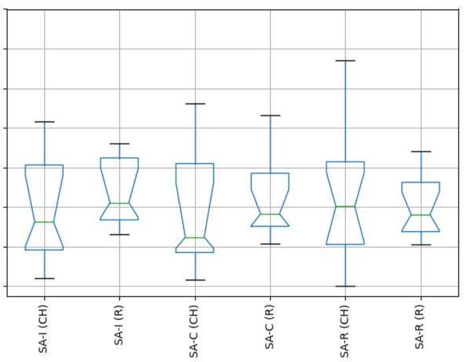
Table 5.10: Median Solution Cost of MIP and SA (with CH) on Real-Life Instances. Tardiness and makespan are denoted by the top and bottom values respectively. The best results per instance are highlighted in boldface and asterisks mark solutions that have been proven optimal by MIP. Dashes mean that no valid solution was found within the time limit.



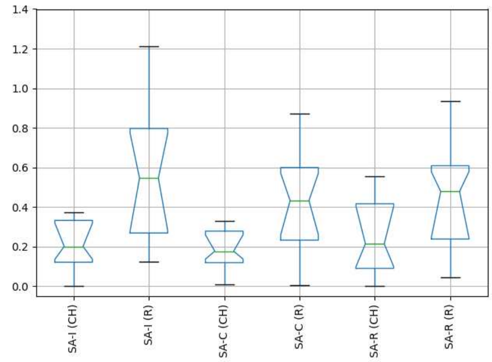
(a) Means Including Constructive Heuristic



(b) Means without Constructive Heuristic



(c) Means over Instances with Unique Materials



(d) Means over Instances with Shared Materials

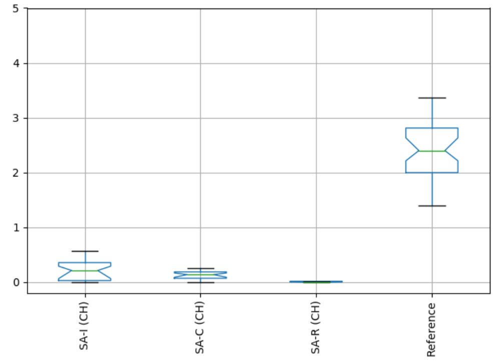
Figure 5.7: RPD Values for P-style Instances

with the algorithm name as prefix and the configuration name in square brackets as suffix (e.g. SA-I with the tuned configuration for SA-C is labelled *SA-I [C]*). The results in Figure 5.10 show that none of the configurations leads to better results for all algorithms. Further, comparing the mean and median RPD values in Figures 5.10a and 5.10b shows that the results for both SA-I and SA-R are relatively stable as opposed to SA-C.

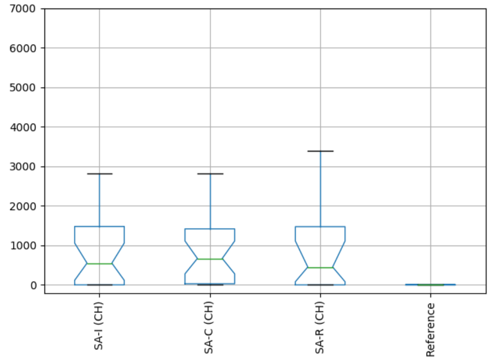
Influence of the Features

To investigate the influence of block moves and guidance strategies on the performance of the meta-heuristics, we performed additional experiments with different configurations on the validation set. We created further configurations for each SA variant by changing the value for a single feature in the configuration while leaving everything else fixed. The derived configurations are the following:

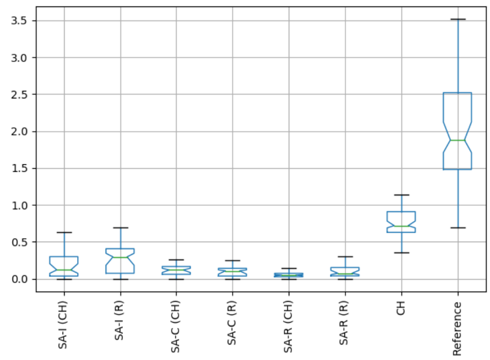
- **Standard**: The configuration proposed by SMAC



(a) S-style Instances



(b) T-style Instances



(c) RPD for Makespans on T-style Instances

Figure 5.8: Comparison of SA with Reference Solutions

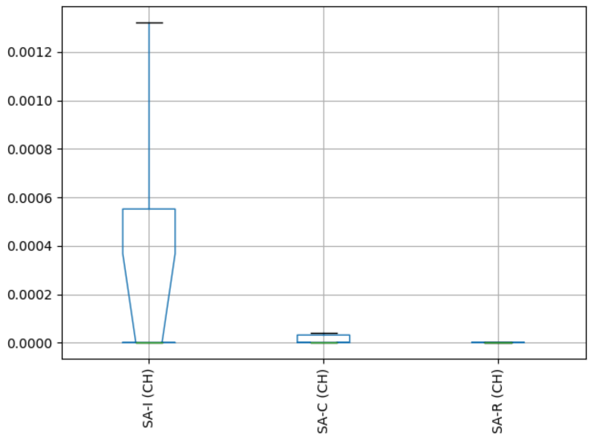
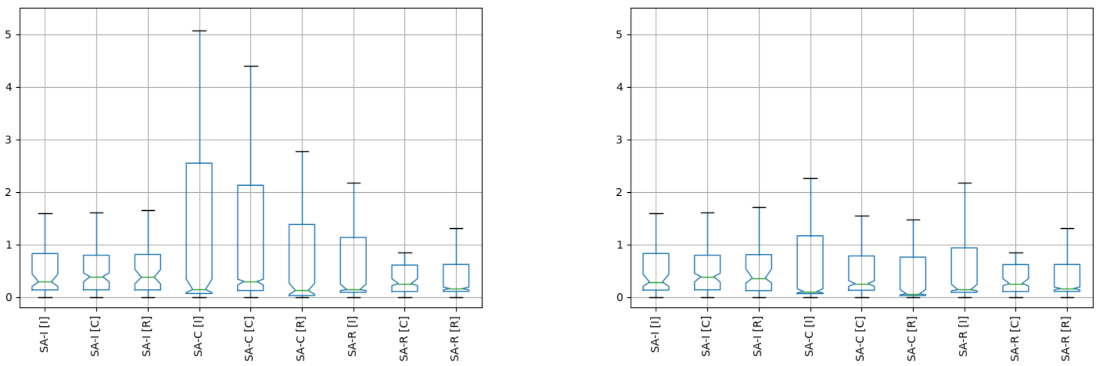


Figure 5.9: RPD Values for SA on Real-Life Instances



(a) Mean RPD Values

(b) Median RPD Values

Figure 5.10: RPD Values for the SA Variants, with All Configurations

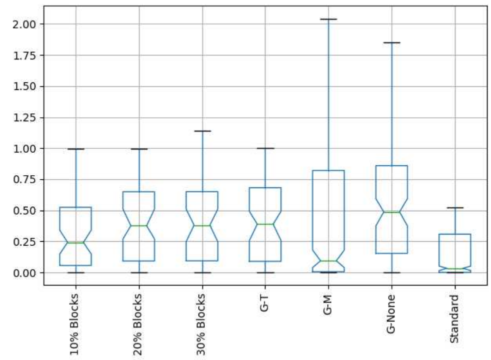
- **X% Blocks**: The probability for generating block moves p_B is set to 10%, 20% and 30%
- **G-T**: The probability for applying guidance towards minimising makespan p_M is set to zero
- **G-M**: The probability for applying guidance towards minimising tardiness p_T is set to zero
- **G-None**: Both guidance probabilities (p_M and p_T) are set to zero

The RPD values for all three SA variants over the validation set instances can be seen in Figure 5.11. For SA-I, any modification of the standard configuration leads to performance degradation as can be seen in Figure 5.11a. For SA-C and SA-R, the results are not as obvious (see Figures 5.11b and 5.11c). In the case of SA-C, block moves apparently have the most negative impact, as increases in block move probabilities lead to increases in RPD values. SA-R loses some performance when the guidance is reduced in any way but does not perform significantly different with higher probabilities for block moves.

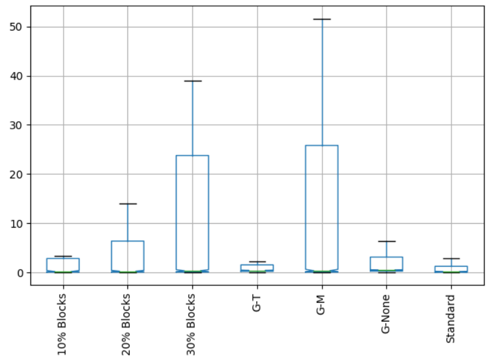
Overall, there is no configuration for any of the SA variants that outperforms the standard configuration. Thus, we conclude that a limited amount of guidance towards minimisation of both makespan as well as tardiness is favourable over a purely unguided search. The block moves in their current implementation, however, do not provide a benefit and are even detrimental to the performance of SA-I and SA-C.

Comparison to Results from the Literature

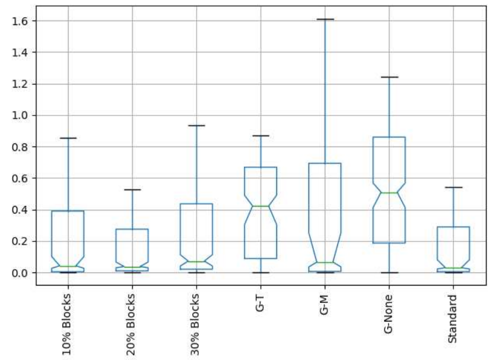
To show the robustness of our methods, we compare them to the state of the art approach that was proposed recently for a similar problem provided by Perez-Gonzalez et al. [2019]. The particular problem uses the same constraints and also aims to minimise total



(a) SA-I



(b) SA-C



(c) SA-R

Figure 5.11: RPD for the SA Variants, with Different Features

tardiness, but does not consider the minimisation of the makespan. However, we can still use our meta-heuristics to approach the problem, as the makespan objective can simply be ignored in this case. The instances used in our experiments were provided by Perez-Gonzalez et al. [2019].

Since the optimal solutions for the majority of these instances have an objective function value of zero, we use the Relative Deviation Index (RDI, eq. (5.3)) as performance measure instead of the RPD. Similar performance measures have been used in previous publications (e.g. Perez-Gonzalez et al. [2019]).

$$RDI_{I,S} := \frac{\text{cost}_{I,S} - \text{best}_I}{\text{worst}_I - \text{best}_I} \tag{5.3}$$

We ran each of our SA variants 20 times on every instance within a time limit that is based on the run time formula used by Perez-Gonzalez et al. [2019]. The parameters for each of our meta-heuristics are set to the ones determined by SMAC.

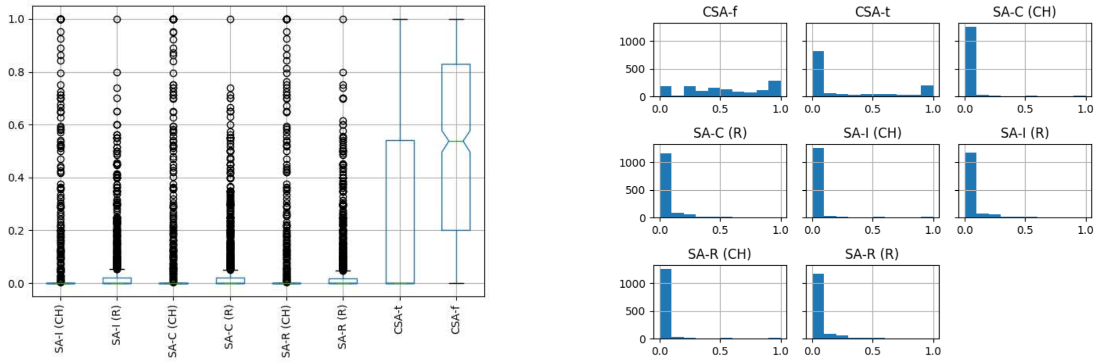
The results for the Clonal Selection Algorithms (CSA_t and CSA_f) described by Perez-Gonzalez et al. [2019] have been kindly provided to us by the authors. Publicly available CPU benchmarks[2] show that the CPU used in our experiments is roughly 1.5 times faster than the Intel i7 7700 processor that was used by Perez-Gonzalez et al. [2019] with respect to single-thread performance. Therefore, we used a time limit of $M \cdot N \cdot \frac{20}{2}$ milliseconds per instance, where M is the number of machines and N is the number of jobs (Perez-Gonzalez et al. [2019] used a time limit of $M \cdot N \cdot \frac{30}{2}$ in their experiments).

For the majority of instances (2469 small, 5058 medium and 5909 big), all runs of the Simulated Annealing variants and all runs of the state-of-the-art Clonal Selection Algorithms produced solutions with the exact same solution cost. Figures 5.12 to 5.14 show box plots and histograms for all evaluated algorithms on the remaining 1371 small, 702 medium and 91 big instances, respectively. In the aggregated results it can be seen that all SA variants outperform the state-of-the-art CSA algorithms regarding solution quality for most of the instances. This observation is further supported by Mann-Whitney-Wilcoxon tests using a confidence level of 0.95 which show that the proposed SA variants produce significantly improved results than both CSA_t and CSA_f in our experiments. Figures 5.12 to 5.14 show that the SA variants with greedily constructed initial solution significantly outperform those which use a random initial solution. Comparing the different SA variants with the same initial solution generation procedure shows no statistically significant differences.

When looking at the best results per instance for all SA and both CSA variants respectively, we observe that in 3291 small, 5548 medium and 5979 big instances, the best SA result has the same cost as the best CSA result. For 506 small, 208 medium and 21 big instances, the best SA result is better than the best CSA result. On the other hand, for 43 small and 4 medium instances, the best CSA result is better than the best SA result. Figure 5.15 shows the number of instances where CSA outperforms SA, grouped by the number of jobs per instance. We observe that the number of instances where CSA outperforms SA declines with instance size, which indicates that SA scales better with increasing instance size.

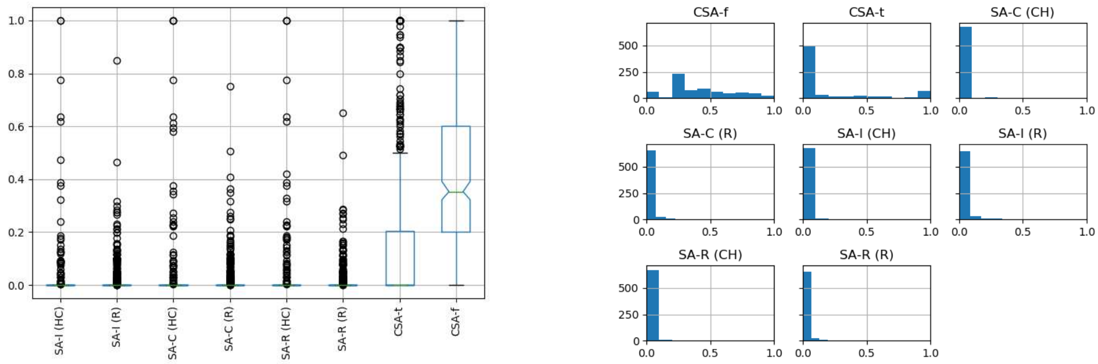
Detailed results of all our experiments are available for download at `http://cdlab-artis.dbai.tuwien.ac.at/instances/parallel-machine-scheduling`.

[2] `https://www.cpubenchmark.net/compare/Intel-i7-7700-vs-Intel-Xeon-E5-2650-v4/2905vs2797`



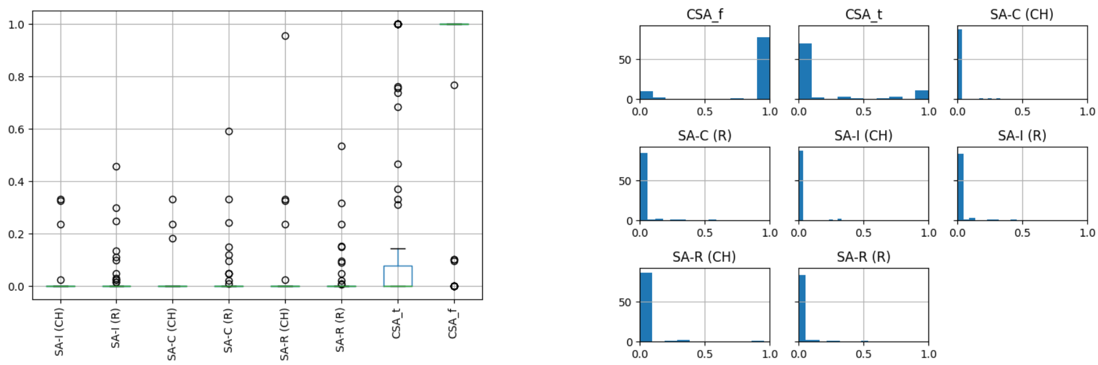
(a) Box Plots for All Meta-Heuristics (b) Histograms

Figure 5.12: RDI Values over 1371 Small Perez Instances



(a) Box Plots for All Meta-Heuristics (b) Histograms

Figure 5.13: RDI Values for 702 Medium Perez Instances



(a) Box Plots for All Meta-Heuristics (b) Histograms

Figure 5.14: RDI Values for 91 Big Perez Instances

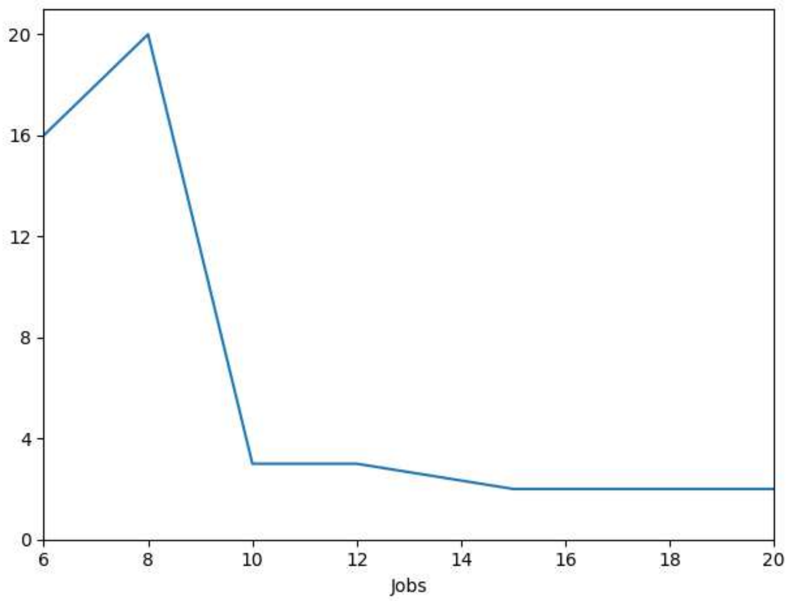


Figure 5.15: Number of Instances Where CSA Outperforms SA

CHAPTER 6

Conclusions

In this thesis, we have investigated several solution approaches for a novel variant of the Unrelated Parallel Machine Scheduling Problem. To approach the problem, we proposed several variants of Simulated Annealing that utilise different neighbourhood operators and cooling schemes, and we adapted several Mixed-Integer Programming formulations for related problems. The adapted models include the formulation proposed by Perez-Gonzalez et al. [2019] for the $R_m|s_{ijk}, M_j|\Sigma_j T_j$ problem and a formulation for the $R_m|s_{ijk}|C_{max}$ problem proposed by Avalos-Rosales et al. [2015]. Additionally, we derived further models by replacing one set of constraints by another formulation found in the related literature.

To evaluate the effectiveness of our approaches, we implemented an instance generator for the problem and created a large pool of instances. Based on the set of randomly generated and real-life instances, we performed a thorough evaluation of all investigated methods. Furthermore, we compared the performance of the proposed techniques to the state-of-the-art on a set of existing instances.

From the experiments conducted in Section 5.2.1, we can see that the models which we adapted from the work of Avalos-Rosales et al. [2015] outperform those that we adapted from Perez-Gonzalez et al. [2019]. Also, we see that the reformulation of constraints leads to performance improvements in some areas of the instance space.

Regarding the meta-heuristics, the experimental results show that the compared SA variants produce high-quality solutions within short run times and outperform the exact approaches for the large majority of instances. All Simulated Annealing variants produce significantly better and more consistent results when they are provided with greedily constructed instead of randomly generated initial solutions. Comparing the different Simulated Annealing variants shows only minor differences when they are paired with the same Constructive Heuristic. Furthermore, we observe that the behaviour of all compared Simulated Annealing variants is more consistent on instances with shared materials.

Further, we conclude that a limited amount of guidance towards the minimisation of both makespan and tardiness in the generation of neighbourhood moves is preferable to purely random move generation. In contrast to the guidance, *Block Moves* do not provide a significant benefit to Simulated Annealing in our implementation and are mostly ignored by SMAC. We think that the generation of Block Moves requires additional information about the current solution in order to be effective.

A closer inspection of the results shows that it is significantly more difficult for the meta-heuristics to find zero-tardiness solutions for the *T-style* instances than for the *S-style* instances. Furthermore, we observe that most solutions produced by SA have a significantly lower makespan than the reference solutions. This suggests that the generated reference solutions are generally not the global optimum. Overall, we conclude that our instance generator is well-suited for the generation of hard instances.

Finally, our experiments on the instances provided by Perez-Gonzalez et al. [2019] show that Simulated Annealing is able to improve many of the solutions produced by state-of-the-art approaches.

For future work, it may be interesting to use Block Moves in a more systematic context than Simulated Annealing. For instance, a more sophisticated heuristic for the selection of job blocks could be employed by defining a distance measure between jobs and building blocks based on this measure. Additionally, it would be interesting to investigate new neighbourhood operators and hybrid algorithms for this problem.

List of Figures

List of Tables

List of Algorithms

Bibliography

Ali Allahverdi, C. T. Ng, T. C. Edwin Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

Ali Allahverdi. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378, 2015.

Mojtaba Afzalirad and Javad Rezaeian. Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers & Industrial Engineering*, 98:40–52, 2016.

Paz Perez-Gonzalez, Victor Fernandez-Viagas, Miguel Zamora García, and Jose M. Framiñan. Constructive heuristics for the unrelated parallel machines scheduling problem with machine eligibility and setup times. *Computers & Industrial Engineering*, 131:131–145, 2019.

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

Oliver Avalos-Rosales, Francisco Angel-Bello, and Ada Alvarez. 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(9-12):1705–1718, 2015.

Scott Kirkpatrick, D. Gelatt Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

A. Al-Salem. Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, 17 (1):177–187, 2004.

Ghaith Rabadi, Reinaldo J. Moraga, and Ameer Al-Salem. Heuristics for the unrelated parallel machine scheduling problem with setup times. *J. Intelligent Manufacturing*, 17(1):85–97, 2006.

Jean-Paul Arnaout, Ghaith Rabadi, and Rami Musa. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *J. Intelligent Manufacturing*, 21(6):693–701, 2010.

Jean-Paul Arnaout, Rami Musa, and Ghaith Rabadi. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines - part II: enhancements and experimentations. *J. Intelligent Manufacturing*, 25(1):43–53, 2014.

Haroldo G. Santos, Túlio A. M. Toffolo, Cristiano L. T. F. Silva, and Greet Vanden Berghe. Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem. *ITOR*, 26(2):707–724, 2019.

Tony T. Tran, Arthur Araujo, and J. Christopher Beck. Decomposition methods for the parallel machine scheduling problem with setups. *INFORMS Journal on Computing*, 28(1):83–95, 2016.

Ridvan Gedik, Darshan Kalathia, Gokhan Egilmez, and Emre Kirac. A constraint programming approach for solving unrelated parallel machine scheduling problem. *Computers & Industrial Engineering*, 121:139–149, 2018.

Luis Fanjul-Peyro, Rubén Ruiz, and Federico Perea. Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers & OR*, 101:173–182, 2019.

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

Kuo-Ching Ying, Zne-Jung Lee, and Shih-Wei Lin. Makespan minimization for scheduling unrelated parallel machines with setup times. *J. Intelligent Manufacturing*, 23(5): 1795–1803, 2012.

Mahdi Rambod and Javad Rezaeian. Robust meta-heuristics implementation for unrelated parallel machines scheduling problem with rework processes and machine eligibility restrictions. *Computers & Industrial Engineering*, 77:15–28, 2014.

Mojtaba Afzalirad and Javad Rezaeian. A realistic variant of bi-objective unrelated parallel machine scheduling problem: NSGA-II and MOACO approaches. *Appl. Soft Comput.*, 50:109–123, 2017.

Gulcin Bektur and Tugba Saraç. A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server. *Computers & OR*, 103:46–63, 2019.

Jeng-Fung Chen. Minimization of maximum tardiness on unrelated parallel machines with process restrictions and setups. *The International Journal of Advanced Manufacturing Technology*, 29(5):557–563, 2006.

Mojtaba Afzalirad and Masoud Shafipour. Design of an efficient genetic algorithm for resource-constrained unrelated parallel machine scheduling problem with machine eligibility restrictions. *J. Intelligent Manufacturing*, 29(2):423–437, 2018.

Erdal Caniyilmaz, Betül Benli, and Mehmet S. Ilkay. An artificial bee colony algorithm approach for unrelated parallel machine scheduling with processing set restrictions, job sequence-dependent setup times, and due date. *The International Journal of Advanced Manufacturing Technology*, 77(9):2105–2115, 2015.

Jelle Adan, Ivo J. B. F. Adan, Alp Akcay, Rick Van den Dobbelsteen, and Joep Stokkermans. A hybrid genetic algorithm for parallel machine scheduling at semiconductor back-end production. In *ICAPS*, pages 298–302. AAAI Press, 2018.

Chris N. Potts and Luk N. Van Wassenhove. A decomposition algorithm for the single machine total tardiness problem. *Oper. Res. Lett.*, 1(5):177–181, 1982.

Jae-Ho Lee, Jae-Min Yu, and Dong-Ho Lee. A tabu search algorithm for unrelated parallel machine scheduling with sequence-and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, 69 (9-12):2081–2089, 2013.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2011.