Scheduling Flexible Job Shops under Workforce Constraints

Dissertation

zur Erlangung des akademischen Grades

Doctor rerum politicarum (Dr. rer. pol.)

der Fakultät III – Wirtschaftswissenschaften, Wirtschaftsinformatik und Wirtschaftsrecht der Universität Siegen

vorgelegt von

Dipl.-Wirt.Inform. David Michael Müller

Erstgutachter:	Prof. Dr. Dominik Kreß
Zweitgutachter:	Prof. Dr. Erwin Pesch
Dekan der Fakultät III:	Prof. Dr. Marc Hassenzahl
Datum der Disputation:	16.07.2021

Acknowledgements

I am deeply grateful for the opportunity to do my PhD at the Chair of Management Information Science at the University of Siegen. This PhD study was a long and challenging process, but I was fortunate enough to be accompanied by many people. I would like to thank all of the people who have accompanied and supported me during this process.

First, I would like to express my profound gratitude to my supervisor, Prof. Dr. Dominik Kreß. He was very open-minded in his supervision and continuously supported me with his great expertise in the field of operations research. His extensive feedback and invaluable advice enabled me to substantially improve the scientific quality of this thesis and to learn tremendously for my professional development. I would also like to express my sincere gratitude to Prof. Dr. Erwin Pesch for giving me the chance to realize my research interests and to conduct my PhD study. I would like to thank him for his valuable feedback and the possibility to participate in different conferences as well as workshops.

I would like to express my appreciation to my co-author, Dr. Jenny Nossack, for the considerable discussions and constructive suggestions. In addition, I would like to thank my (former) colleagues PD Dr. Sergei Chubanov, Roswitha Eifler, Alexander Herbst, Jan-Erik Justkowiak, Christoph Kotthaus, Xiyu Li, Mikhail Lukashevich, Dr. Sebastian Meiswinkel, Prof. Dr. Alena Otto, and Jonas Weber for their support and their contributions.

Last but not least, I would like to express my deepest gratitude to my family and friends. In particular, I want to thank my parents, who continuously supported and encouraged me and always believed in me. A special thanks goes to my sister and my brother for their unconditional support and proofreading my thesis. I would also like to thank Jackie Law for proofreading my thesis. I am also thankful to Julian Dax for motivating me to pursue this PhD opportunity. Finally, I especially would like to thank Lydia, my girlfriend, who supported and encouraged me during this journey.

David Michael Müller

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Preliminaries

Scheduling problems are of high importance in both theory and practice since they occur in numerous application areas, especially in production and logistic environments. As stated by Blazewicz et al. (2019), scheduling problems can be understood as the problem of assigning resources over time to tasks that need to be completed. There exists a variety of different classes of scheduling problems in various domains, which have been widely addressed by researches over the last decades (see, e.g., Blazewicz et al., 2019, for an overview).

A well known class of scheduling problems that has attracted a lot of attention in the literature is the *job shop scheduling problem* (JSP) (see, e.g., Blazewicz et al., 2019). It occurs in traditional manufacturing systems and the classical JSP can be stated as follows. A set of machines and a set of jobs are given. Each job is composed of a set of operations that have to be processed in a predefined sequence in order to complete the job. It is assumed that there are no precedence constraints among the operations of different jobs. Each operation is assigned to a specific machine that must be used for its processing as well as associated with a corresponding processing time needed on that machine. Moreover, each machine can process only one operation at a time and preemption of operations is permitted. Given these restrictions, the problem is to find a schedule, in which the operations are sequenced on the corresponding assigned machines such that some performance measure is optimized. A typical performance measure in scheduling problems is the minimization of the so-called makespan, i.e., the maximum completion time among all jobs. With respect to the complexity, it is well known, that the JSP minimizing the makespan is strongly NP-hard (Lenstra and Rinnooy Kan, 1979).

Nowadays, in the face of global and dynamic markets, manufacturing companies are confronted with continuously growing challenges, e.g., short product life cycles, huge product varieties up to tailor-made products, and demand fluctuations. In order to react rapidly to market changes, manufacturing companies oftentimes improve the level of flexibility of their manufacturing processes by introducing multi-purpose machines that are able to process different types of manufacturing operations (see, e.g., Beach et al., 2000; Jain et al., 2013). These kinds of manufacturing systems are often referred to as flexible manufacturing systems (FMS) (see Browne et al., 1984).

This is taken account of in the *flexible job shop scheduling problem* (FJSP), which was originally stated by Brucker and Schlie (1990). It generalizes the JSP by assuming that each operation is associated with a set of *eligible machines*, which are capable of processing the respective manufacturing operation type. That is, in comparison to the JSP, a feasible schedule must additionally take account of the allocation of operations to machines. The FJSP occurs in many real-world manufacturing systems and has been addressed by many researches in the literature (see, e.g., the survey by Chaudhry and Khan, 2016).

Due to the manufacturing flexibility induced by the use of multi-purpose machines, which are capable of processing different types of operations, the machines usually need to be prepared before they are able to process a specific operation. The time needed to prepare a machine is referred to as a *setup time*. This typically includes, for example, cleanup tasks, the change of tools or technical machine configurations. The integration of setup times in scheduling problems is of great practical relevance to leverage the productivity of the manufacturing processes (see, e.g., Allahverdi, 2015; Allahverdi et al., 1999, 2008; Allahverdi and Soroush, 2008). In general, setup times can be classified into two types (see, e.g., Allahverdi, 2015). Sequence-independency indicates that the setup time for an operation only depends on the operation itself. In contrast, sequence-dependent refers to when the setup time depends on both the operation itself and on the operation, which has been immediately prior processed by the machine.

Moreover, the incorporation of workforce constraints in real-world manufacturing systems oftentimes plays an important role, especially in the face of a set of employees with heterogenous skills. On a shop floor level, a machine usually needs to be operated by a *machine operator*. In the case of considering setup times, the needed setup is executed by a *setup operator*. In many cases, setup operators are also involved in resolving machine defects. However, machine operators or setup operators are often not capable for executing all machines, or they posses differing qualifications in executing a machine, which results in varying processing times as well as setup times. Therefore, optimization techniques in workforce scheduling can lead to productivity enhancements (see, e.g., De Bruecker et al., 2015; Ernst et al., 2004; Van den Bergh et al., 2013).

In light of the increasing automatization and digitalization of manufacturing processes, the need to provide applicable solution approaches and methods that promise a good trade-off between computational time and solution quality is becoming increasingly important.

Against this background, the overall objective of this thesis is to investigate important FJSP settings that incorporate workforce constraints and are motivated by real-world manufacturing systems. On the one hand, new problem variants that are of practical relevance are analyzed, and on the other hand, novel solution approaches are proposed for solving these corresponding problem variants.

1.1 Research Questions

In recent years, FJSP settings incorporating workforce constraints have been attracting more attention from researches. A brief overview of the related literature is provided in Table 1.1 (FJSP settings incorporating setup operators) and Table 1.2 (FJSP settings considering machine operator restrictions). Note, that these settings are sometimes also referred to as dual-resource constrained (DRC) systems (see, e.g., Treleven, 1989; Xu et al., 2011).

Table 1.1: Overview of literature concerned with FJSP settings incorporating setup operators

Publication	Objective	Solution approach
Chen et al. (2003)[a,b]	On-time delivery of products, reduction of inventory and the number of setups	Decomposition based heuristic
Morinaga et al. (2014)	Total weighted tardiness and workload of setup operators	Genetic algorithm
Morinaga et al. (2016)	Total weighted tardiness, workload of setup operators, and work-in-process inventory costs	Hybrid genetic algorithm
Li et al. (2020)[c]	Makespan	Hybrid genetic algorithm

[a] : Incorporation of group-dependent setup times
[b] : Incorporation of machine operators
[c] : Consideration of sequence-dependent setup times

The majority of the literature is concerned with workforce constraints focusing on machine operators. Despite the practical importance of the integration of setup times as well as setup operators, only very few researches consider FJSPs with setup operators. With regard to optimality criteria, most of the papers consider either makespan minimization or multiple objectives. Objectives considering the on-time delivery of customer orders, e.g., the minimization of the total tardiness of jobs, is only addressed by a few researchers. Therefore, there still exists a wide range of different problem variants in this research stream that should be further studied in order to close the gap between research methodologies and practice. Hence, this thesis is motivated by the following research question:

Which FJSP settings incorporating workforce constraints are relevant from a practical point of view and have attracted less attention in the literature?

Based on the above literature overview, mainly metaheuristic solution approaches, in particular, population-based methods, have been considered for solving FJSP settings incorporating workforce constraints. With respect to exact solution approaches, only very few researches introduced mixed-integer programming (MIP) models that are actually evaluated in computational tests. However, the implementation of exact approaches would be fruitful for computing optimal solutions and measuring the solution quality of heuristic approaches. Also important is the development of promising heuristic approaches that are applicable in practice. Therefore, the second research question is, as follows:

What are novel exact as well as heuristic solution approaches for scheduling flexible job shops with workforce restrictions?

Table 1.2: Overview of literature concerned with FJSP settings incorporating machine operators

Publication	Objective	Solution approach
Lang and Li (2011)	Delivery satisfaction, process cost, energy consumption, and noise pollution	Genetic algorithm
Liu et al. (2011)	Makespan and production cost	Hybrid genetic algorithm
Xianzhou and Zhenhe (2011)	Makespan	Genetic algorithm
Zhang et al. (2013)	Makespan and production cost	Hybrid discrete particle swarm optimisation
Lei and Guo (2014)	Makespan	Variable neighbourhood search
Yazdani et al. (2015)	Makespan	Simulated annealing and vibration damping optimisation, MIP
Zhang et al. (2015)	Makespan	Particle swarm optimisation
Lei and Tan (2016)	Makespan and total tardiness	Local search
Paksi and Ma'ruf (2016)	Total tardiness	Genetic algorithm
Zheng and Wang (2016)	Makespan	Knowledge-guided fruit fly optimisation
Gong et al. (2018a)	Makespan, total worker cost and green-production factors	Hybrid genetic algorithm
Gong et al. (2018b)	Makespan, maximum workload of machines, and total workload of all machines	Memetic algorithm
Peng et al. (2018)	Makespan	Genetic algorithm
Vallikavungal Devassia et al. (2018)[a]	Makespan	Variable neighbourhood search, MIP
Wu et al. (2018)[b]	Makespan	Hybrid genetic algorithm
Cunha et al. (2019)[c]	Makespan	MIP
Meng et al. (2019)	Energy consumption	Variable neighbourhood search, MIPs
Yang et al. (2019)[d]	Lateness, makespan and deviation of the workload among the machines	Local search
Yazdani et al. (2019)	Makespan, critical machine workload, and total workload of machines	Genetic algorithms
Andrade-Pineda et al. (2020)	Makespan and mean tardiness	Iterated greedy algorithm, MIP
Wu et al. (2020)[e]	Makespan and total setup time	Genetic algorithm
Zhu et al. (2020)[b]	Makespan, total carbon emission, and total cost of workers	Memetic algorithm

[a] : Consideration of resource recovery constraints
[b] : Consideration of learning effects of workers
[c] : Incorporation of additional time constraints
[d] : Consideration of multilevel product structures
[e] : Consideration of loading and unloading time constraints of fixture resources

It is often observed in computational studies comparing different solution approaches on some optimization problem that there exists no single best solution approach that outperforms all others on all problem instances. Therefore, it would be useful to know in advance, which of a given set of solution approaches performs best on a given instance. This problem is also referred as the *algorithm selection problem*, which was originally introduced by Rice (1976). In recent years, algorithm selection approaches have been successfully applied on different combinatorial problems, for example, the propositional satisfiability problem, the traveling salesman problem, and the multi-mode resource-constrained project scheduling problem, by making use of machine learning techniques (see, e.g., Kerschke et al., 2018; Messelis and De Causmaecker, 2014; Xu et al., 2008).

Motivated by the success of algorithm selection approaches in different problem domains, the following additional question is raised:

> *How can an algorithm selection approach be designed that leverage the performance complementary of a given set of algorithms for solving the FJSP?*

1.2 Basics on the Flexible Job Shop Scheduling Problem

This section provides a detailed problem definition concerning the classical FJSP, which serves as a basis for the different problem variants investigated in this thesis.

The FJSP is defined as follows. A set of I of *jobs* $|I| = n$, and set of M of *machines* are given. Each job $i \in I$ consists of a set of q_i operations $O_i = \{i_1, \ldots, i_{q_i}\}$. The sets O_i are assumed to be ordered for all $i \in I$, which relates to the fact that for any pair of operations $i_j, i_k \in O_i$ with $j < k$, i_k can only start to be processed after the processing of i_j is completed. Each operation $i_j \in O_i$, $i \in I$, must be processed on exactly one machine out of a set of *eligible machines* $M_{i_j} \subseteq M$ and each machine can only process one operation at a time. The processing time of an operation $i_j \in O_i$ of a job $i \in I$ sequenced on an eligible machine $m \in M_{i_j}$ is denoted by $p_{i_j}^m \in \mathbb{N}^+$. The completion time of an operation $i_j \in O_i$ of job $i \in I$ is denoted by C_{i_j} which describes a point in time in which the processing of an operation i_j is completed. Analogous, the completion time of job $i \in I$ is denoted by C_i. Moreover, a job is completed if all of its operations are completed, i.e. $C_i = C_{i_{q_i}}$ for all $i \in I$. It is assumed, that all jobs and machines are available at time zero. Furthermore, the preemption of some operation processed on some eligible machine is not permitted.

The problem is to find a schedule, i.e. an assignment of operations to the eligible machines, and to sequence each operation on its corresponding machine, that is feasible to the restrictions stated above, such that some performance measure is optimized.

Due to the large variety of scheduling problems, Graham et al. (1979) introduce a widely used *classification scheme* that enables a great notation of scheduling problems in the literature. The notation consists of three fields $\alpha|\beta|\gamma$, where α states the machine environment, β describes job characteristics, and γ relates to a performance measure (optimality criterion). Over the years, the proposed three-field notation has been adapted, for example, by Allahverdi (2015) and Błażewicz et al. (1983) to include setup information and additional resource constraints.

1.3 Contribution and Outline of the Thesis

This cumulative thesis consist of four papers that were published or submitted to different journals and one conference publication (see Table 1.3). Papers P1 to P4 investigate FJSPs incorporating workforce restrictions arising in manufacturing systems and propose different exact and heuristic solution approaches for scheduling the respective problems. Additionally, Paper P5 presents algorithm selection approaches for the FJSP. Each paper is associated with a respective

chapter along with this thesis and has only been reformatted for the purpose of consistency. An overview of the different chapters presenting the research contributions of the according papers is provided as follows:

Table 1.3: Overview of papers

	Chapter	Paper
P1	2	Kress, D., Müller, D., and Nossack, J. (2019). A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, 41(1): 179–217.
P2	3	Kress, D. and Müller, D. (2019). Mathematical models for a flexible job shop scheduling problem with machine operator constraints. *IFAC-PapersOnLine*, 52(13): 94—99.
P3	4	Müller, D. and Kress, D. (2019). Filter-and-fan approaches for scheduling flexible job shops under workforce constraints. Working Paper. University of Siegen (Submitted).
P4	5	Kress, D. and Müller, D. (2020). Semiconductor final-test scheduling under setup operator constraints. Working Paper. University of Siegen (Submitted).
P5	6	Müller, D., Müller, M. G., Kress, D., and Pesch, E. (2021). An algorithm selection approach for the flexible job shop scheduling problem: choosing constraint programming solvers through machine learning. Working Paper. University of Siegen (Submitted).

In Chapter 2, we study a FJSP with sequence-dependent setup-times that takes account of heterogenous machine operator qualifications. The problem is motivated by a real-world scheduling problem as part of a research project conducted with a manufacturing company. In this study, we analyze two objective functions, minimizing the makespan and minimizing the total tardiness. We propose exact and heuristic solution approaches that are based on a decomposition of the problem into a vehicle routing problem with precedence constraints and a machine operator assignment problem. These problems are connected via logic inequalities. In computational tests, we evaluate our solution approaches on randomly generated test instances as well as real-world test instances that are based on data that has been provided by the manufacturing company (see Kress et al., 2019b).

In Chapter 3 and 4, we solely consider a FJSP that incorporates machine operator constraints and aims to minimize the makespan. In the former chapter, we provide a comprehensive literature review on FJSPs in the presence of machine operators and conclude that mainly metaheuristic solution approaches have been considered for solving these problem settings. Against this background, we propose two mathematical models, a MIP model and a constraint programming (CP) model, that are evaluated by using the standard solvers provided by IBM ILOG CPLEX. In a computational study, we show that the CP solver clearly outperforms the MIP solver given the developed modelling approaches. Moreover, it also tends to outperform a state-of-the-art metaheuristic solution approach (see Kress and Müller, 2019). Motivated by these research results, Chapter 4 introduces filter-and-fan based heuristic solution approaches that prove to be competitive when compared with a standard CP solver. These methods incorporate a local search procedure with a tree search procedure that are applied alternately. In general, the local search procedure is used to obtain local optima, while a tree search procedure aims to improve these locally optimal solutions by generating compound transitions in order explore

larger neighborhoods. Our solution approaches make use of the main ideas of the decomposition based approach proposed by Kress et al. (2019b) that makes use of neighborhood structures that have proven to perform well for the FJSP (see Mastrolilli and Gambardella, 2000). In an extensive computational study, we show that our filter-and-fan based heuristic solution approaches are competitive when compared with the use of the standard CP solver provided by IBM ILOG CPLEX. Moreover, they tend to outperform a tabu search benchmark heuristic as well as existing metaheuristic approaches from the literature (see Müller and Kress, 2019).

Given the fact that the explicitly incorporation of setup operators in flexible job shop settings is rarely addressed in the literature, we consider a FJSP with sequence-dependent setup times under setup operator constraints in Chapter 5. Specifically, we study a semiconductor final-test scheduling problem that has been brought to our attention during a project with a developer and manufacturer of semiconductor-based system solutions. Given the main goal of the considered manufacturing company, which is the on-time delivery of customer orders, we therefore analyze the objective of minimizing the total weighted tardiness. We present a MIP model and a tabu search heuristic framework which applies the main ideas of the decomposition based approaches introduced by Kress et al. (2019b) and Müller and Kress (2019). Our computational tests show that our proposed tabu search heuristic approaches clearly outperform a MIP model using a standard solver and tend to outperform the current scheduling practice on real-world problem instances that mimic settings at the considered manufacturing company. Moreover, we present managerial insights on the application of our heuristic framework in dynamic environments concerning frequently changing customer requests and common test machine failures (see Kress and Müller, 2020).

An overview of the above mentioned studies contributing to different FJSP settings incorporating workforce constraints is presented in Table 1.4.

Table 1.4: Overview of studies contributing to FJSP settings incorporating workforce constraints

	Incorporation of:				
Paper	Machine operators	Setup operators	Setup times	Objective	Solution approach
P1	✓	-	✓	Makespan, Total tardiness	Decomposition-based exact and heuristic solution approaches
P2	✓	-	-	Makespan	MIP and CP model
P3	✓	-	-	Makespan	Filter-and-fan based heuristic solution approaches
P4	-	✓	✓	Total weighted tardiness	MIP model and tabu search heuristic solution approaches

As observed in our conducted studies, oftentimes, there exists no single best solution approach which outperforms all other approaches on all problem instances. Therefore, Chapter 6 deals with algorithm selection approaches that leverage the complementary performance of CP

solvers for solving the classical FJSP minimizing the makespan. Therefore, we introduce a set of algorithm selection models by making use of different machine learning techniques. In a computational study, we show that our proposed approaches tend to provide a better performance than using a single solver (see Müller et al., 2021).

Finally, the thesis closes with a summary and an outlook of future research in Chapter 7.

Chapter 2

A Worker Constrained Flexible Job Shop Scheduling Problem with Sequence-Dependent Setup Times

Abstract

We consider a flexible job shop scheduling problem with sequence-dependent setup times that incorporates heterogeneous machine operator qualifications by taking account of machine- and operator-dependent processing times. We analyze two objective functions, minimizing the makespan and minimizing the total tardiness, and present exact and heuristic decomposition based solution approaches. These approaches divide the scheduling problem into a vehicle routing problem with precedence constraints and an operator assignment problem, and connect these problems via logic inequalities. We assess the quality of our solution methods in an extensive computational study that is based on randomly generated as well as real-world problem instances.

Authors	Dominik Kress[a,b] dominik.kress@hsu-hh.de David Müller[a] david.mueller@uni-siegen.de Jenny Nossack[c] jenny.nossack@googlemail.com [a] *University of Siegen, Management Information Science, Kohlbettstraße 15, 57068 Siegen, Germany* [b] *Helmut Schmidt University - University of the Federal Armed Forces Hamburg, Business Administration, especially Procurement and Production, Friedrich-Ebert-Damm 245, 22159 Hamburg, Germany* [c] *HHL Leipzig, Center for Advanced Studies in Management, Jahnallee 59, 04109 Leipzig, Germany*

Publication details	*OR Spectrum*, Volume 41(1), 2019, Pages 179–217
Full citation	Kress, D., Müller, D., and Nossack, J. (2019). A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, 41(1): 179–217.

This is an Accepted Manuscript of an article published by Springer in OR Spectrum on 15 November 2018, available online: `http://dx.doi.org/10.1007/s00291-018-0537-z`.

2.1 Introduction

The well known *job shop scheduling problem* (JSP) is composed of a set of jobs and a set of machines (see, e.g., Błażewicz et al., 2007). Each job consists of a set of operations that must be processed in a given sequence in order to complete the job. That is, the processing of an operation of a job cannot be started before the preceding operation of that job is completed. There are no precedence relations among the operations of different jobs. Each operation must be processed on a specific machine and is associated with a corresponding processing time. Operations may not be preempted and each machine can process only one operation at a time. Given these restrictions, the problem is to determine sequences of the operations that are associated with the machines with the objective of optimizing some performance measure. It is well known that the JSP is strongly NP-hard for minimizing the makespan or the total tardiness (Graham et al., 1979; Lenstra and Rinnooy Kan, 1979).

Real-world manufacturing systems are usually more complex than the systems that can be represented by the classical JSP (see, e.g., Günther and Lee, 2007). Factory work floors, for example, oftentimes feature multiple machines of the same type as well as multi-purpose machines that allow for processing different types of operations. This is taken account of in the *flexible job shop scheduling problem* (FJSP), which generalizes the JSP by assuming that each operation must be processed by exactly one machine out of a given set of *eligible machines* (Brucker and Schlie, 1990; Hurink et al., 1994). Additionally, machines must oftentimes be prepared in order to be able to process a specific operation. The time needed for this preparation is referred to as a *setup time*. The importance of explicitly incorporating setup times into real-world scheduling problems has been discussed in the literature since the mid-1960s (see Allahverdi, 2015; Allahverdi et al., 1999, 2008; Allahverdi and Soroush, 2008). One distinguishes two classes of setup times (see, e.g., Allahverdi et al., 1999). If the setup time needed for some operation solely depends on the operation itself, it is referred to as sequence-independent. If it additionally depends on the immediately preceding operation that has been processed by the machine, it is referred to as sequence-dependent. Furthermore, real-world manufacturing systems oftentimes feature a heterogenous workforce which, for example, induces the need to take account of differently skilled *machine operators or workers* (De Bruecker et al., 2015).

2.1.1 Problem Setting and Motivation

In this paper, we address a FJSP with sequence-dependent setup times that takes account of differing worker skills. We will refer to this problem as the *worker constrained FJSP with sequence-dependent setup times*, and denote it by WSFJSP.

Our research is motivated by a real-world scheduling problem that has been brought to our attention during a project with a manufacturing company located in North Rhine-Westphalia, Germany. The company is specialized in the production of construction components – primarily cardan shaft mounts – made of aluminium, stainless steel, and steel. Its customers are mainly automotive suppliers. The products are fabricated in predefined lots that we will refer to as

jobs. That is, each job is composed of multiple items of a specific product. The raw materials (usually aluminum profiles) are delivered by suppliers and subsequently run through various manufacturing operations, e.g. sawing, lathing, milling and punching, in predefined sequences in order to complete the final products. The items of each lot are jointly stored and moved in steel box pallets. Therefore, a specific operation of a job must be completed for the entire lot before the processing of the next operation of the job can be started. The production process of an exemplary job is illustrated in Figure 2.1.

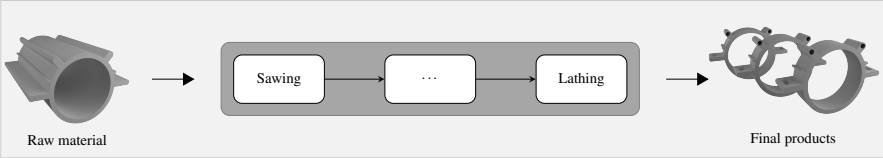


Figure 2.1: Exemplary production process

There are several kinds of multi-purpose machines that are capable of performing different manufacturing operations, so that each operation of a job is associated with a set of eligible machines as described above. Cleaning operations, the change of tools, and process configurations result in sequence-dependent setup times between two consecutive operations processed on the same machine. Setup operators are not considered to be scarce. Machine operators, on the other hand, are considered to be a scarce resource and possess differing skills that our industry partner implements by making use of worker-dependent processing times. The processing time of a specific operation therefore depends on both, the machine chosen from the set of eligible machines and the worker assigned to the machine. A worker is assigned to an operation for its entire processing time and can only process one operation at a time.

Currently, the scheduling of the production processes at our industry partner is a daily manual task with a planning horizon of about one week. We were asked to implement scheduling algorithms that are capable of taking account of larger planning horizons. Additionally, we were asked to analyze two different objectives. First, based on the assumption that each job is associated with a due date at which the production of the job is intended to be completed, the aim is to minimize the total tardiness. Second, we were asked to provide results for minimizing the makespan. Clearly, due to the computational complexity of the JSP for both of these objectives, both variants of WSFJSP are strongly NP-hard.

According to the classical three-field notation by Graham et al. (1979) that was adapted by Allahverdi (2015) and Błażewicz et al. (1983) to include setup information and additional resource constraints, the two variants of WSFJSP considered in this article fall into the categories $FJ|res1 \cdot 1, ST_{sd}|C_{max}$ and $FJ|res1 \cdot 1, ST_{sd}|\sum T_i$, respectively.

2.1.2 Related Literature

The WSFJSP combines two variants of the FJSP that have been addressed in the literature, i.e. the FJSP with sequence-dependent setup times and the FJSP with explicit incorporation of machine operators. We will denote these variants by SFJSP and WFJSP, respectively, and

summarize the relevant literature regarding these settings in this section. Note that machine scheduling problems with two types of resources, e.g. machines and machine operators, are sometimes also referred to as dual-resource constrained (DRC) systems (see, e.g. Treleven, 1989; Xu et al., 2011).

Surveys on scheduling problems with setup considerations for various machine environments are provided by Allahverdi et al. (1999, 2008) and Allahverdi (2015). With respect to flexible job shops, Shen et al. (2017) consider the SFJSP with the objective of minimizing the makespan. They present a mixed-integer programming (MIP) formulation and a tabu search algorithm. Corresponding ant colony optimization approaches are presented by Zhang and Liu (2012) and Rossi (2014). Alternative MIP models are given by Nourali et al. (2012) and Saidi-Mehrabad and Fattahi (2007). The latter authors furthermore suggest another tabu search algorithm. Defersha and Chen (2010) additionally consider time lag requirements and machine release dates. They present a MIP model and a genetic algorithm. Some articles deal with objective functions that differ from minimizing the makespan. Mousakhani (2013), for example, aims at minimizing the total tardiness. The author presents a MIP model and proposes a metaheuristic algorithm based on iterated local search. Özgüven et al. (2012) present MIP formulations for the SFJSP with the objective of minimizing a weighted sum of the makespan and a specific measure for the degree of unbalancedness of the machine workloads. Similarly, Bagheri and Zandieh (2011) consider minimizing a weighted sum of the makespan and the mean tardiness. They present a variable neighborhood search algorithm.

The existing articles on the WFJSP with the objective of minimizing the makespan focus on the development of metaheuristic approaches. Examples include Lei and Guo (2014) (variable neighbourhood search), Yazdani et al. (2015) (simulated annealing, vibration damping optimization), Zhang et al. (2015) (particle swarm optimization), and Zheng and Wang (2016) (fruit fly optimization). Paksi and Ma'ruf (2016) analyze the objective of minimizing the total tardiness and propose a genetic algorithm. Multiple objectives are considered by Lang and Li (2011) and Lei and Tan (2016), the former of which also take account of uncertain processing times.

There exist only a few papers in the scheduling literature that explicitly take account of setup considerations as well as machine operator related constraints. Venditti et al. (2010) and Behnamian (2014) consider open shop and flow shop settings, respectively. Chen et al. (2003) address a FJSP which is very closely related to our setting. The authors consider machine operator related constraints and group-dependent setup times, where setup operations are necessary whenever switching between predefined groups of operations. Furthermore, they allow the generation of so called transfer lots. That is, each lot (as defined above for the WSFJSP) may be divided into multiple transfer lots, each of which can move to the next operation as soon as all parts within that transfer lot are completed. The objective function relates to maximizing the on-time delivery of products and the reduction of inventory and the number of setups. The problem is solved via a heuristic framework that makes use of a decomposition of the overall problem into smaller subproblems.

2.1.3 Contribution and Overview

Based on the above literature review, it can be concluded that FJSPs with sequence-dependent setup times and explicit incorporation of machine operators have received little attention. We will therefore contribute to the literature by analyzing a corresponding setting, namely the WSFJSP, which – as outlined above – is based on a real-world scheduling problem. We will propose an exact solution approach that decomposes the WSFJSP into a *vehicle routing problem (VRP) with precedence constraints* and a *worker assignment problem*. These models are combined by using *logic inequalities*. This exact approach will turn out to outperform an integrated MIP model and is able to solve small instances to optimality. Moreover, in order to be able to determine feasible solutions for medium and large instances within reasonable time, we will present heuristic algorithms based on the above decomposition. Our approaches are evaluated based on randomly generated test instances as well as real-world test instances that are based on data that has been provided by our industry partner.

The remainder of this article is structured as follows. In Section 2.2, we provide a formal definition of the WSFJSP and introduce the notation. The exact decomposition approach is presented in Section 2.3, while the heuristics are subject of Section 2.4. The computational tests are presented in Section 2.5. Finally, Section 2.6 concludes the paper.

2.2 Notation and Detailed Problem Description

We are given a set $I = \{1, \dots, n\}$ of jobs. Each job $i \in I$ is associated with a set of q_i operations $O_i = (i_1, \dots, i_{q_i})$ that have to be sequenced on a set M of machines and that may not be preempted. The sets O_i are assumed to be ordered for all jobs $i \in I$, which relates to the fact that, for any pair of operations $i_j, i_k \in O_i$ with $j < k$, i_j must be completed before the processing of i_k may start. Each operation $i_j \in O_i$, $i \in I$, must be processed by exactly one machine out of a set of eligible machines $M_{i_j} \subseteq M$ in order to be completed. To simplify the notation, we define $M_{i_j,k_l} := M_{i_j} \cap M_{k_l}$ for all $i, k \in I$, $i_j \in O_i$, $k_l \in O_k$. An operation can only be processed by a machine if exactly one worker out of a given set W of workers is assigned to the machine during the entire processing time of the operation. Each machine and each worker can process at most one operation at a time. A job is completed if all of its operations are completed. The completion time of an operation $i_j \in O_i$ of job $i \in I$ is denoted by C_{i_j}. The completion time of job $i \in I$ is denoted by C_i. Obviously, $C_i = C_{i_{q_i}}$ for all $i \in I$. The processing time of an operation $i_j \in O_i$ of a job $i \in I$, which we denote by $p_{i_j}^{m,w} \in \mathbb{Q}_0^+ \cup \{\infty\}$, is assumed to vary over different machines $m \in M_{i_j}$ and workers $w \in W$, which enables us to take account of differently skilled workers. We assume that all workers are available during the entire planning horizon. In case of a shift-based system and a planning horizon larger than one shift, we must therefore assume that all shifts are staffed with equally skilled workers, so that they can replace each other at shift changeovers. $p_{i_j}^{m,w} = \infty$ represents the case that worker $w \in W$ is not allowed to process an operation $i_j \in O_i$ of a job $i \in I$ on machine $m \in M_{i_j}$. We assume that for each operation $i_j \in O_i$ of a job $i \in I$ and each corresponding machine $m \in M_{i_j}$, there exists at least one worker $w \in W$

that can process the operation within finite time. Moreover, we assume that sequence-dependent setup times $s^m_{i_j,k_l} \in \mathbb{Q}^+_0$ occur when an operation $k_l \in O_k$, $k \in I$, is processed immediately after an operation $i_j \in O_i$, $i \in I$, on machine $m \in M_{i_j,k_l}$. Note that these setup times may differ among the machines $m \in M_{i_j,k_l}$. Setup operations do not require the assignment of a worker. Note that we consider an offline setting where all jobs, machines, and workers are available at the beginning of the planning horizon. Furthermore, there are no precedence relations between jobs.

The WSFJSP is to find a schedule, i.e. an allocation of operations to machines, a sequence of the operations that have been allocated to each machine, and a corresponding assignment of workers to operations, that is feasible with respect to the restrictions stated above, such that an objective function is addressed. We will consider two objectives. First, we will consider the minimization of the makespan $C_{max} := \max_{i \in I} C_i$. Second, given a due date $d_i \in \mathbb{Q}^+_0$ for each job $i \in I$, we will consider the minimization of the total tardiness $\sum_{i \in I} T_i$, where the tardiness T_i of job $i \in I$ is defined as $T_i := \max\{C_i - d_i, 0\}$.

Our notation is summarized in Table 2.1.

Table 2.1: Notation used throughout the paper

I	set of jobs	$I = \{1, \ldots, n\}$, $\lvert I\rvert = n$
M	set of machines	
W	set of workers	
O_i	set of operations of job $i \in I$	$O_i = (i_1, \ldots, i_{q_i})$, $\lvert O_i\rvert = q_i$
M_{i_j}	set of eligible machines for operation $i_j \in O_i$ of job $i \in I$	$M_{i_j} \subseteq M$
M_{i_j,k_l}	intersection of M_{i_j} and M_{k_l}, where $i, k \in I$, $i_j \in O_i$, and $k_l \in O_k$	$M_{i_j,k_l} := M_{i_j} \cap M_{k_l}$
d_i	due date of job $i \in I$	$d_i \in \mathbb{Q}^+_0$
$p^{m,w}_{i_j}$	processing time of operation $i_j \in O_i$ of job $i \in I$ when processed by worker $w \in W$ on machine $m \in M_{i_j}$	$p^{m,w}_{i_j} \in \mathbb{Q}^+_0 \cup \{\infty\}$
$s^m_{i_j,k_l}$	setup time when processing operation $k_l \in O_k$ of job $k \in I$ immediately after operation $i_j \in O_i$ of job $i \in I$ on machine $m \in M_{i_j,k_l}$	$s^m_{i_j,k_l} \in \mathbb{Q}^+_0$
C_{i_j}	completion time of operation $i_j \in O_i$ of job $i \in I$	
C_i	completion time of job $i \in I$	$C_i = C_{i_{q_i}}$
C_{max}	makespan of the schedule	$C_{max} := \max_{i \in I} C_i$
T_i	tardiness of job $i \in I$	$T_i := \max\{C_i - d_i, 0\}$

2.3 Decomposition Approach

There exist similarities between VRPs and machine scheduling problems, which has resulted in several articles that have addressed machine scheduling settings from a vehicle routing perspective. Bigras et al. (2008), for example, analyze relationships between single machine scheduling problems with sequence-dependent setup times and the time-dependent traveling salesman problem (TSP). Similarly, Balas et al. (2008) model single machine problems that arise in their adapted shifting bottleneck procedure for a JSP with sequence-dependent setup times as TSPs with time windows. A similar method is applied by Tran and Beck (2012), who propose a logic-based Benders decomposition approach for scheduling unrelated parallel machines with

sequence-dependent setup times.

In line with the aforementioned research, we propose to solve the WSFJSP in a branch-and-cut framework by decomposing it into a VRP with precedence constraints (master problem, MP) and a worker assignment problem (subproblem). The MP explicitly addresses the allocation of operations to machines and the sequencing of operations on each machine. The effect of the assignment of workers to operations on the objective function value is embedded into the MP by making use of logic inequalities, a class of constraints that is inspired by the logic-based Benders decomposition approach by Hooker and Ottosson (2003). Within our branch-and-cut framework, these inequalities are consecutively obtained by the subproblem, which explicitly determines an assignment of workers to operations such that the makespan or the total tardiness is minimized based on a given allocation and sequencing decision of (a relaxed version of) the MP.

To ease the notation in the remainder of this paper, we define a dummy job 0 with due date $d_0 = \infty$ and exactly one operation 0_1 with $M_{0_1} = M$. We set $M_{0_1,i_j} = M_{i_j,0_1} = M_{i_j}$ for all jobs $i \in I$ and operations $i_j \in O_i$. Furthermore, we set $p_{0_1}^{m,w} = 0$ for all machines $m \in M$ and workers $w \in W$. The setup times s_{0_1,i_j}^m can take arbitrary nonnegative rational values for all machines $m \in M$, jobs $i \in I$, and operations $i_j \in O_i$, which allows for modelling the first setup operation on each machine. Furthermore, $s_{i_j,0_1}^m = 0$ for all $m \in M$, $i \in I$, and $i_j \in O_i$.

2.3.1 Master Problem

As mentioned above, the master problem does not explicitly take account of the assignment of workers to operations. Hence, we make use of lower bounds for the processing times, and define $p_{i_j}^{m,min} := \min_{w \in W} p_{i_j}^{m,w}$ for all jobs $i \in I \cup \{0\}$, operations $i_j \in O_i$, and machines $m \in M_{i_j}$. Furthermore, we set $p_{i_j}^{min} := \min_{m \in M_{i_j}} p_{i_j}^{m,min}$ for all jobs $i \in I \cup \{0\}$ and operations $i_j \in O_i$.

When considering each machine as a vehicle and each operation as a distinct customer that must be visited (processed) exactly once, the MP corresponds to a variant of the VRP, where precedence relations capture the fact that the operations of each job have to follow a predefined order. The dummy operation 0_1 represents the depot of the VRP and allows modelling the start and end configurations of the machines. Following well established VRP formulations, we define the MP on a weighted, directed (multi-) graph $G = (V, E)$ with vertex set $V := \bigcup_{i \in I} O_i \cup \{0_1\}$ and the edge set being composed of $2 \cdot |M_{i_j,k_l}|$ edges between any pair of vertices $i_j, k_l \in V$, $i \neq k$, and $|M_{i_j,i_v}|$ edges between any pair of vertices $i_j, i_v \in V$, $v > j$, as illustrated in Figure 2.2. Here,

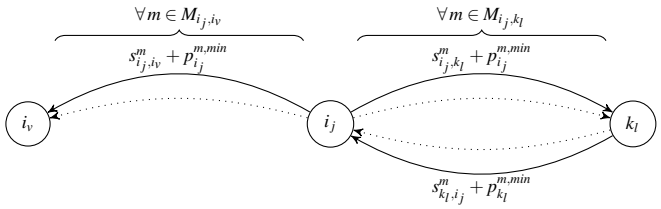


Figure 2.2: Illustration of the graph representation, $i_v, i_j, k_l \in V$, $i \neq k$, $v > j$

solid lines represent exemplary edges of the multigraph for some specific eligible machine, while dotted lines indicate that there potentially exist a number of additional edges that correspond

to the remaining eligible machines.

Define $\bar{V} := V \setminus \{0_1\}$ and let B be a large positive number. Furthermore, for the sake of notational convenience, define $V_{i_j} := V \setminus \{i_k | k \leq j\}$ (set of potential direct successors of i_j), $\tilde{V}_{i_j} := V \setminus \{i_k | k \geq j\}$ (set of potential direct predecessors of i_j), and $\bar{V}_{i_j} := V_{i_j} \setminus \{0_1\}$ for all $i_j \in V$.

For the sake of clarity, the additional notation for the MP is summarized in Table 2.2.

Table 2.2: Additional notation for the master problem, defined on $G = (V, E)$

$p_{i_j}^{m,min}$	minimum processing time of operation $i_j \in O_i$ of job $i \in I$ processed on machine $m \in M_{i_j}$	$p_{i_j}^{m,min} := \min_{w \in W} p_{i_j}^{m,w}$
$p_{i_j}^{min}$	minimum processing time of operation $i_j \in O_i$ of job $i \in I$	$p_{i_j}^{min} := \min_{m \in M_{i_j}} p_{i_j}^{m,min}$
V	set of vertices	$V := \bigcup_{i \in I} O_i \cup \{0_1\}$
$\bar{V}$	set of vertices without depot vertex	$\bar{V} := V \setminus \{0_1\}$
V_{i_j}	set of potential direct successors of $i_j \in V$	$V_{i_j} := V \setminus \{i_k \mid k \leq j\}$
$\tilde{V}_{i_j}$	set of potential direct predecessors of $i_j \in V$	$\tilde{V}_{i_j} := V \setminus \{i_k \mid k \geq j\}$
$\bar{V}_{i_j}$	set of potential direct successors of $i_j \in V$ without depot vertex	$\bar{V}_{i_j} := V_{i_j} \setminus \{0_1\}$
B	large positive number	

Now, define a continuous variable $t_{i_j} \in \mathbb{R}_0^+$ for all operations $i_j \in V$. It represents the time when i_j is started to be processed on one of the machines. Moreover, define a continuous variable $C_{max} \in \mathbb{R}_0^+$ that represents the makespan, and a binary variable

$$y_{i_j,k_l}^m := \begin{cases} 1, & \text{if operation } k_l \text{ is processed} \\ & \text{directly after operation } i_j \\ & \text{on machine } m \\ 0, & \text{else} \end{cases} \quad \forall\, i_j \in V, k_l \in V_{i_j}, m \in M_{i_j,k_l}. \tag{2.1}$$

Then, when considering makespan minimization, a MIP formulation for the MP is as follows:

$$\min C_{max} \tag{2.2}$$

s.t.

$$t_{i_{q_i}} + \sum_{k_l \in V_{i_{q_i}}} \sum_{m \in M_{i_{q_i},k_l}} y_{i_{q_i},k_l}^m \cdot p_{i_{q_i}}^{m,min} \leq C_{max} \qquad \forall\, i \in I, \tag{2.3}$$

$$\sum_{i_j \in \tilde{V}_{k_l}} \sum_{m \in M_{i_j,k_l}} y_{i_j,k_l}^m = 1 \qquad \forall\, k_l \in \bar{V}, \tag{2.4}$$

$$\sum_{i_j \in \bar{V}} y_{0_1,i_j}^m \leq 1 \qquad \forall\, m \in M, \tag{2.5}$$

$$\sum_{k_l \in \{a_b \in \tilde{V}_{i_j} | m \in M_{a_b}\}} y_{k_l,i_j}^m - \sum_{k_l \in \{a_b \in V_{i_j} | m \in M_{a_b}\}} y_{i_j,k_l}^m = 0 \quad \forall\, i_j \in V, m \in M_{i_j}, \tag{2.6}$$

$$t_{i_j} + s_{i_j,k_l}^m + p_{i_j}^{m,min} - t_{k_l} \leq \left(1 - y_{i_j,k_l}^m\right) B \qquad \forall\, i_j \in V, k_l \in \bar{V}_{i_j}, m \in M_{i_j,k_l}, \tag{2.7}$$

$$t_{i_j} + \sum_{k_l \in V_{i_j}} \sum_{m \in M_{i_j,k_l}} y_{i_j,k_l}^m \cdot p_{i_j}^{m,min} \leq t_{i_{j+1}} \qquad \forall\, i_j \in \bar{V} \text{with } j \leq q_i - 1, \tag{2.8}$$

$$t_{i_j} + p_{i_j}^{min} \leq t_{i_{j+1}} \qquad \forall\, i_j \in \bar{V} \text{with } j \leq q_i - 1, \tag{2.9}$$

$$\tilde{C}_{max}^{h} \cdot \left(1 - \sum_{(i_j,k_l,m)\in P^h} (1 - y_{i_j,k_l}^{m})\right) \leq C_{max} \qquad \forall \text{ logic inequalities } h, \tag{2.10}$$

$$y_{i_j,k_l}^{m} \in \{0,1\} \qquad \forall\, i_j \in V, k_l \in V_{i_j}, m \in M_{i_j,k_l}, \tag{2.11}$$

$$t_{i_j} \in \mathbb{R}_0^+ \qquad \forall\, i_j \in V, \tag{2.12}$$

$$C_{max} \in \mathbb{R}_0^+. \tag{2.13}$$

The objective function (2.2) minimizes the makespan of the schedule. Constraints (2.3) set a lower bound on the makespan. Constraints (2.4) guarantee that each operation is scheduled exactly once. Inequalities (2.5) ensure that there is at most one operation that is processed first on each machine. Flow conservation is enforced by constraints (2.6). Constraints (2.7) enforce a time increase of at least $s_{i_j,k_l}^{m} + p_{i_j}^{m,min}$ (setup and processing) compared to t_{i_j}, if $y_{i_j,k_l}^{m} = 1$. Constraints (2.8) guarantee that an operation $i_{j+1} \in O_i$ of job $i \in I$ cannot start before its preceding operation $i_j \in O_i$ has been processed completely. Constraints (2.9) are redundant to constraints (2.8), but have shown to improve the computational performance in our tests. The logic inequalities (2.10) correct a potential underestimation of the objective function value that is caused by minimizing over all workers when determining the processing times that (partly) define the edge weights of the underlying graph representation. They are explained in more detail in Section 2.3.4, where we also introduce the related notation. Finally, constraints (2.11)–(2.13) define the domains of the variables.

Model (2.2)–(2.13) can easily be adapted for minimizing the total tardiness instead of the makespan. To do so, define a continuous variable $T_i \in \mathbb{R}_0^+$, representing the tardiness of job i, for all $i \in I$. We get:

$$\min T = \sum_{i\in I} T_i \tag{2.14}$$

$$\text{s.t. } (2.4) - (2.9),\ (2.11),\ (2.12),$$

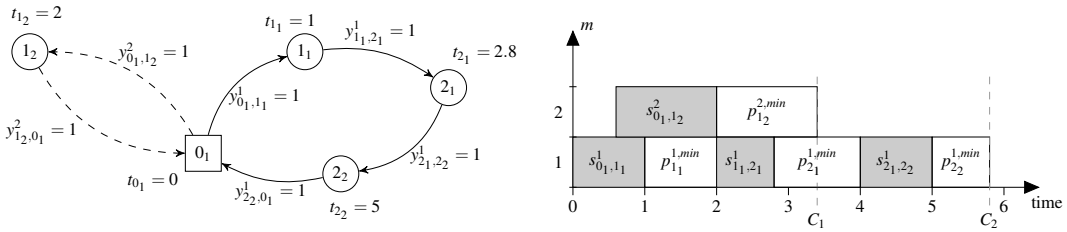
$$t_{i_{q_i}} + \sum_{k_l \in V_{i_{q_i}}} \sum_{m \in M_{i_{q_i},k_l}} y_{i_{q_i},k_l}^{m} \cdot p_{i_{q_i}}^{m,min} - d_i \leq T_i \quad \forall\, i \in I, \tag{2.15}$$

$$\tilde{T}^{h} \cdot \left(1 - \sum_{(i_j,k_l,m)\in P^h} (1 - y_{i_j,k_l}^{m})\right) \leq \sum_{i\in I} T_i \qquad \forall \text{ logic inequalities } h, \tag{2.16}$$

$$T_i \in \mathbb{R}_0^+ \qquad \forall\, i \in I. \tag{2.17}$$

The objective function (2.14) minimizes the total tardiness, which we will hereafter denote by T, of the jobs. Constraints (2.15) set a lower bound on the tardiness of each job. Constraints (2.16) are logic inequalities that are defined in analogy to constraints (2.10). Constraints (2.17) define the domains of the newly introduced variables.

We denote a relaxed version of the MP, which results from considering only a (potentially empty) subset of constraints (2.10) or (2.16), by RMP. Based on a given solution of the MP or a RMP, we can construct a directed graph with vertex set V that includes only those edges of G that are associated to a positive variable (2.1). We will refer to this graph as the *supporting graph* of the solution. The supporting graph of any feasible solution of the MP or a RMP is composed of at most $|M|$ edge-disjoint cycles, each of which includes the depot vertex 0_1, such that every vertex of the set $\bar{V}$ is included in exactly one of the cycles. Figure 2.3 illustrates the supporting graph of a feasible solution of the MP (Figure 2.3a) and the corresponding Gantt chart that represents the processing of the operations on the machines (Figure 2.3b) for an example instance with two jobs and two machines. The solid and dashed lines of Figure 2.3a



(a) Supporting graph (b) Gantt chart

Figure 2.3: Representation of a feasible solution of the MP

represent positive variables (2.1) that are associated to the first or second machine, respectively. The values of the variables (2.12) are given next to the corresponding vertices of the set V. Hence, in the depicted solution, machine 1 processes operations 1_1, 2_1, and 2_2, while machine 2 solely processes operation 1_2. As illustrated in the Gantt chart in Figure 2.3b, the precedence constraints among the operations of job 1 result in idle time that precedes the setup operation which is necessary to process 1_2 on machine 2.

2.3.2 Subproblem

Based on a feasible allocation and sequencing decision of the MP or a RMP and the true processing times, the subproblem determines an assignment of workers to operations such that the makespan or the total tardiness is minimized. It is important to note that there always exists such a feasible assignment, because the assignment of workers to operations can only cause temporal shifts of the operations on the machines.

Let $\bar{\mathbf{y}} := (\bar{y}^m_{i_j,k_l} | i_j \in V, k_l \in V_{i_j}, m \in M_{i_j,k_l})$ denote the vector of variables (2.1) of a solution of the MP or a RMP and refer to the corresponding value of the objective function (2.2) or (2.14) by $\bar{C}_{max}$ and $\bar{T}$, respectively. Based on $\bar{\mathbf{y}}$, we can construct parameters that specify the machines that process the operations:

$$\bar{z}^m_{i_j} := \begin{cases} 1, & \text{if } \sum_{k_l \in \{a_b \in \tilde{V}_{i_j} | m \in M_{a_b}\}} \bar{y}^m_{k_l,i_j} = 1 \\ 0, & \text{else} \end{cases} \qquad \forall\, i_j \in \bar{V}, m \in M_{i_j}. \tag{2.18}$$

In analogy to the variables of the MP, we additionally define continuous variables $\tilde{t}_{i_j} \in \mathbb{R}_0^+$ for all operations $i_j \in V$, as well as continuous variables $\tilde{C}_{max} \in \mathbb{R}_0^+$ and $\tilde{T}_i \in \mathbb{R}_0^+$ for all jobs $i \in I$. Moreover, we define the following binary variables:

$$x_{i_j}^w := \begin{cases} 1, & \text{if operation } i_j \text{ is processed by worker } w \\ 0, & \text{else} \end{cases} \qquad \forall\, i_j \in \bar{V}, w \in W, \tag{2.19}$$

$$x_{i_j,k_l}^w := \begin{cases} 1, & \text{if operations } i_j \text{ and } k_l \text{ are processed by worker } w \\ 0, & \text{else} \end{cases} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, w \in W, \tag{2.20}$$

$$v_{i_j,k_l} := \begin{cases} 1, & \text{if the processing of operation } k_l \text{ starts before the processing of operation } i_j \text{ finishes} \\ 0, & \text{else} \end{cases} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l. \tag{2.21}$$

We furthermore define $x_{0_1}^w := 1$ for all workers $w \in W$.

The subproblem for makespan minimization is then defined as follows:

$$\min \tilde{C}_{max} \tag{2.22}$$

s.t.

$$\tilde{t}_{i_{q_i}} + \sum_{m \in M_{i_{q_i}}} \sum_{w \in W} \bar{z}_{i_{q_i}}^m \cdot x_{i_{q_i}}^w \cdot p_{i_{q_i}}^{m,w} \leq \tilde{C}_{max} \qquad \forall\, i \in I, \tag{2.23}$$

$$\sum_{w \in W} x_{i_j}^w = 1 \qquad \forall\, i_j \in \bar{V}, \tag{2.24}$$

$$\tilde{t}_{i_j} + s_{i_j,k_l}^m + p_{i_j}^{m,w} - \tilde{t}_{k_l} \leq \left(1 - \bar{y}_{i_j,k_l}^m \cdot x_{i_j}^w\right) B \qquad \forall\, i_j \in V, k_l \in \bar{V}_{i_j}, m \in M_{i_j,k_l}, w \in W, \tag{2.25}$$

$$\tilde{t}_{i_j} + \sum_{m \in M_{i_j}} \sum_{w \in W} \bar{z}_{i_j}^m \cdot x_{i_j}^w \cdot p_{i_j}^{m,w} \leq \tilde{t}_{i_{j+1}} \qquad \forall\, i_j \in \bar{V} \text{with } j \leq q_i - 1, \tag{2.26}$$

$$x_{i_j,k_l}^w \geq x_{i_j}^w + x_{k_l}^w - 1 \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, w \in W, \tag{2.27}$$

$$\tilde{t}_{i_j} + \sum_{m \in M_{i_j}} \sum_{w \in W} \bar{z}_{i_j}^m \cdot x_{i_j}^w \cdot p_{i_j}^{m,w} - \tilde{t}_{k_l} \leq B \cdot v_{i_j,k_l} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, \tag{2.28}$$

$$x_{i_j,k_l}^w \leq 2 - v_{i_j,k_l} - v_{k_l,i_j} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, w \in W, \tag{2.29}$$

$$x_{i_j}^w \in \{0,1\} \qquad \forall\, i_j \in V, w \in W, \tag{2.30}$$

$$x_{0_1}^w = 1 \qquad \forall\, w \in W, \tag{2.31}$$

$$x_{i_j,k_l}^w \in \{0,1\} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, w \in W, \tag{2.32}$$

$$v_{i_j,k_l} \in \{0,1\} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, \tag{2.33}$$

$$\tilde{t}_{i_j} \in \mathbb{R}_0^+ \qquad \forall\, i_j \in V, \tag{2.34}$$

$$\tilde{C}_{max} \in \mathbb{R}_0^+. \tag{2.35}$$

Given the vector $\bar{\mathbf{y}}$ of a solution of the MP or a RMP, the objective function (2.22) minimizes the makespan of the schedule. Constraints (2.23) bound the makespan from below. Restrictions (2.24) ensure that each operation is assigned to exactly one worker. Constraints (2.25) and (2.26) are defined in analogy to restrictions (2.7) and (2.8) of the MP. Inequalities (2.27) and (2.28) ensure that the variables (2.20) and (2.21) are set to one when needed. Based on these variables, constraints (2.29) guarantee that each worker is assigned to at most one operation at a time. Finally, constraints (2.30)–(2.35) define the domains of the variables.

For the case of minimizing the total tardiness, the subproblem is similarly defined as follows:

$$\min \tilde{T} = \sum_{i \in I} \tilde{T}_i \tag{2.36}$$

$$\text{s.t. } (2.24) - (2.34),$$

$$\tilde{t}_{i_{q_i}} + \sum_{m \in M_{i_{q_i}}} \sum_{w \in W} \bar{z}^m_{i_{q_i}} \cdot x^w_{i_{q_i}} \cdot p^{m,w}_{i_{q_i}} - d_i \leq \tilde{T}_i \quad \forall\, i \in I, \tag{2.37}$$

$$\tilde{T}_i \in \mathbb{R}^+_0 \qquad \forall\, i \in I. \tag{2.38}$$

Here, the objective function (2.36) minimizes the total tardiness of the jobs based on the vector $\bar{\mathbf{y}}$ of a solution of the MP or a RMP. Constraints (2.37) bound the tardiness of each job from below, and constraints (2.38) define the domains of the tardiness variables.

2.3.3 Subtour Elimination Cuts

As described in Section 2.3.1, the supporting graph of any feasible solution of the MP or a RMP is composed of a set of at most $|M|$ edge-disjoint cycles, each of which includes the depot vertex 0_1, such that every vertex of the set $\bar{V}$ is included in exactly one of the cycles. For fractional solutions, that may arise during the solution process of a standard MIP solver that relaxes the integrality constraints of the MP or a RMP, we define the supporting graph to include all vertices of the set V as well as all edges of G that are associated with positive relaxed variables (2.1). Here, the above property needs not be true, which is illustrated in Figure 2.4 based on a fractional solution of a RMP of an example instance with two machines and two jobs, both of which consist of two operations (refer to the previous example in Figure 2.3a for an explanation of the illustration). Each machine of the example instance is eligible for processing all operations. As can be seen, the supporting graph of the depicted fractional solution is composed of four edge-disjoint cycles, two of which do not contain the depot vertex 0_1.

We make use of the above fact to separate subtour elimination constraints that we add to our branch-and-cut framework in addition to the logic inequalities. More specifically, when a fractional solution arises at a node of the branch-and-bound tree, we construct the corresponding supporting graph as described above and determine the set A of connected components of the undirected pendent of this graph. To do so, we make use of a depth-first search approach provided by the Boost Graph Library (Boost, 2018). Each element of the set A is a set of vertices, each

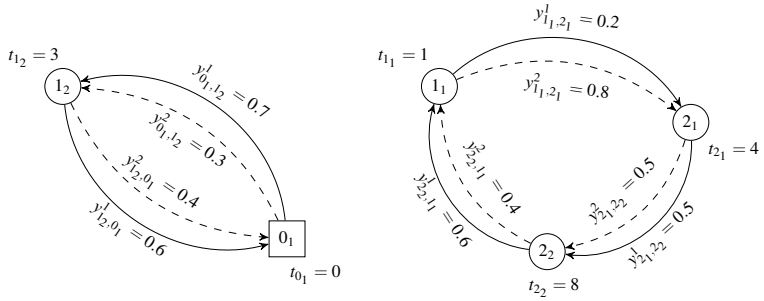


Figure 2.4: Supporting graph of a fractional solution of a RMP of an example instance

of which is included in the respective connected component. If $|A| > 1$, the subtour elimination constraints

$$\sum_{i_j \in S} \sum_{k_l \in V \setminus S} \sum_{m \in M_{i_j,k_l}} y^m_{i_j,k_l} \geq 1 \quad \forall S \in A \tag{2.39}$$

are generated and included in the branch-and-cut procedure.

2.3.4 Branch-and-Cut Framework

We propose to solve the WSFJSP in a branch-and-cut framework offered by a standard MIP solver. Modern solvers allow their users to guide a branch-and-cut solution process via so called callbacks. We make use of these callbacks to consecutively generate and add logic inequalities (2.10) or (2.16) and subtour elimination constraints (2.39) after having started the solver on the RMP with an empty set of logic inequalities. This is illustrated in Algorithm 2.1.

The logic inequalities are indexed by h and are derived by solving subproblems in step 3 of Algorithm 2.1. The basic idea is as follows. The RMP makes use of lower bounds of the processing times, so that the objective function value of a potential incumbent solution may be incorrect with respect to the real processing times. The corresponding logic inequality assures that, for this specific solution, the objective function value of the RMP implicitly takes account of the worker assignment restrictions and the real processing times which are explicitly modelled in the subproblem.

The overall process of handling a potential incumbent solution with objective value $\bar{C}_{max}$ ($\bar{T}$) in step 3 of Algorithm 2.1 is as follows. First, the corresponding subproblem is generated and solved by a standard solver. The resulting objective function value is referred to as $\tilde{C}_{max}$ ($\tilde{T}$). If the subproblem is feasible and $\bar{C}_{max} = \tilde{C}_{max}$ ($\bar{T} = \tilde{T}$), no logic inequality is violated and the branch-and-cut process continues without any modification. If, however, the subproblem is feasible and $\bar{C}_{max}$ ($\bar{T}$) is smaller than $\tilde{C}_{max}$ ($\tilde{T}$), we generate a logic inequality, add it to the RMP, and proceed with the branch-and-cut solution process.

The process of separating subtour elimination cuts in step 4 of Algorithm 2.1 proceeds as outlined in Section 2.3.3. Note, however, that we do not construct these cuts for all potential fractional solutions in order to balance the computational effort needed for their separation and their use within the branch-and-cut procedure. Instead, we generate subtour elimination cuts

Input: Instance $Inst$ of WSFJSP, parameter ρ_s
Output: Optimal solution of $Inst$

1. Initialize $h := 0$ (counter for logic inequalities).
2. Call standard branch-and-cut solver for the RMP of $Inst$ with an empty set of logic inequalities. Whenever a potential incumbent solution with vector $\bar{\mathbf{y}}$ and objective function value $\bar{C}_{max}$ ($\bar{T}$) is found, go to 3 (solve subproblem and potentially generate and add a logic inequality). Whenever a fractional solution arises at a node of the branch-and-bound tree and the node cannot be pruned, go to 4 (potentially generate and add subtour elimination constraints) with a given probability ρ_s.
3. Determine parameters (2.18) and solve the subproblem based on $\bar{\mathbf{y}}$ by making use of a standard solver. The resulting objective function value is referred to as $\tilde{C}_{max}$ ($\tilde{T}$).
   (a) If the subproblem is feasible and $\bar{C}_{max} < \tilde{C}_{max}$ ($\bar{T} < \tilde{T}$), set $h := h + 1$, $\tilde{C}^h_{max} := \tilde{C}_{max}$ ($\tilde{T}^h := \tilde{T}$), and construct the logic inequality

$$\tilde{C}^h_{max} \cdot (1 - \sum_{(i_j,k_l,m)\in P^h} (1 - y^m_{i_j,k_l})) \leq C_{max}$$

   (in case of makespan minimization), or

$$\tilde{T}^h \cdot (1 - \sum_{(i_j,k_l,m)\in P^h} (1 - y^m_{i_j,k_l})) \leq \sum_{i\in I} T_i$$

   (in case of minimization of the total tardiness). Here

$$P^h := \left\{(i_j,k_l,m) | \bar{y}^m_{i_j,k_l} = 1, i_j \in V, k_l \in V_{i_j}, m \in M_{i_j,k_l}\right\}.$$

   Add this constraint to the branch-and-cut process and continue in 2.
   (b) If the subproblem is feasible and $\bar{C}_{max} = \tilde{C}_{max}$ ($\bar{T} = \tilde{T}$), the potential incumbent solution becomes the new incumbent solution and the branch-and-cut process in 2 continues.
4. Construct the supporting graph of the fractional solution and determine the connected components A of this graph. If $|A| > 1$, generate subtour elimination constraints (2.39). Add these constraints to the branch-and-cut process and continue in 2.

Algorithm 2.1 Branch-and-cut framework for WSFJSP

with a given probability ρ_s for each fractional solution.

2.4 Decomposition Based Heuristics

Due to the computational complexity of WSFJSP, it cannot be expected that our branch-and-cut framework is able to solve medium- to large-sized instances to optimality within reasonable time. In this section, we will therefore introduce two heuristic approaches that are based on the above decomposition. First, in Section 2.4.1, we will present a simple approach for generating a feasible initial solution of WSFJSP. Then, in Section 2.4.2, we will develop a more sophisticated improvement procedure which is inspired by an idea presented by Della Croce et al. (2014) for a flow shop scheduling problem with two machines.

2.4.1 Generating an Initial Solution

In line with the decomposition introduced in Section 2.3, we generate an initial solution of a given instance of the WSFJSP by a hierarchical approach composed of two steps. First, based on the lower bounds of the processing times introduced in Section 2.3.1, our approach allocates operations to machines and decides on the sequences of the operations on the machines. This step corresponds to finding a solution to the RMP with an empty set of logic inequalities. Second, given this solution, the approach assigns workers to operations based on the correct processing times, which corresponds to determining a solution to the subproblem and, thus, to the given instance of the WSFJSP.

2.4.1.1 Allocation and Sequencing

The allocation and sequencing decisions are made by a priority-rule based heuristic that follows an algorithmic idea of Giffler and Thompson (1960) for the classical JSP. Our approach is outlined in detail in Algorithm 2.2.

Basically, the algorithm iteratively (loop 7–21) allocates operations that can start being processed (when applying lower bounds of the processing times as introduced in Section 2.3.1) at the respective point of time (represented by "release times", see lines 3–4 and 12–19), when taking account of the corresponding precedence constraints. Among all operations that compete for the same machine (chosen in lines 8–9) in some iteration, exactly one operation is chosen based on a priority rule (line 10). As there exists a vast amount of potential priority rules (see, for example, Haupt, 1989), we rely on a comparative study by Sels et al. (2012), who analyze the performance of multiple priority rules for JSPs and FJSPs under different objective functions, including settings with setup times. Based on the results of this analysis, we decided to make use of two priority rules. In case of the objective of minimizing the makespan, we apply a combination of the *flow due date* (FDD), the *most work remaining* (MWKR), and the *shortest setup time* (SS) priority rules. Formally, after having decided on a machine $m^* \in M$, we determine the last operation $a_b(m^*) \in V$ that is processed by m^* in its current processing

Input: RMP of an instance *Inst* of WSFJSP with an empty set of logic inequalities
Output: Feasible solution of RMP

Initialize $U := \bar{V}$; ▷ `Set of operations that have not been sequenced`
Initialize $L_m := 0 \ \forall m \in M$; ▷ `Load of machine` m
Initialize $r^m_{i_1} := s^m_{0_1,i_1} \ \forall i \in I, m \in M_{i_1}$; ▷ `Earliest starting time of operation` i_1 `of job` i `on machine` m
Initialize $r^m_{i_j} := \infty \ \forall i_j \in \bar{V} \setminus \{i_1 | i \in I\}, m \in M_{i_j}$; ▷ `Earliest starting time of operation` i_j `of job` i `on machine` m
Initialize $a_b(m) := 0_1 \ \forall m \in M$; ▷ `Last operation sequenced on machine` m
Initialize $c_i := 0 \ \forall i \in I$; ▷ `Completion time of the operation of job` i `that has been completed last`
repeat
  Determine $C^* := \min_{i_j \in U, m \in M_{i_j}} r^m_{i_j} + p^{m,min}_{i_j}$; ▷ `Smallest possible completion time of operations that have not been sequenced on their eligible machines`
  Let m^* denote a machine on which C^* is a possible completion time;
  Among all operations that have not been sequenced, $i_j \in U$, with $r^{m^*}_{i_j} < C^*$, choose an operation i^*_j based on a priority rule and sequence it on machine m^*, starting its processing at time $r^{m^*}_{i^*_j}$;
  Update $L_{m^*} := r^{m^*}_{i^*_j} + p^{m^*,min}_{i^*_j}$, $a_b(m^*) := i^*_j$, and $c_{i^*} := L_{m^*}$;
  ▷ `Update earliest starting times of operations that have not been sequenced`
  forall $i \in I$ **do**
    if $i = i^*$ *and* i^*_j *has a succeeding operation* **then**
      update $r^m_{i^*_{j+1}} := \max\{L_{m^*}, L_m + s^m_{a_b(m),i^*_{j+1}}\} \ \forall m \in M_{i^*_{j+1}}$;
    end
    else if $i \neq i^*$ **then**
      $r^{m^*}_{i_k} := \max\{c_i, L_{m^*} + s^{m^*}_{i^*_j,i_k}\} \ \forall i_k \in U$ with $r^{m^*}_{i_k} < \infty$;
    end
  end
  Update $U := U \setminus \{i^*_j\}$;
until $U = \emptyset$;

Algorithm 2.2 Obtaining an initial allocation and sequencing decision

sequence, where $a_b(m^*) = 0_1$ in case of the empty sequence. Among all relevant candidate operations, our priority rule then selects an operation i_j with smallest value

$$\frac{\sum\limits_{1 \leq k \leq j} p_{i_k}^{min}}{\sum\limits_{j \leq k \leq q_i} p_{i_k}^{min}} + s_{a_b(m^*),i_j}^{m^*}.$$

Similarly, in case of the objective of minimizing the total tardiness, we consider a combination of the SS and the *earliest due date* (EDD) priority rules, where an operation i_j with smallest value

$$d_i + s_{a_b(m^*),i_j}^{m^*}$$

among the relevant candidate operations is selected.

2.4.1.2 Worker Assignment

After having determined an initial allocation and sequencing decision, i.e. a feasible solution of a RMP as, for instance, constructed by Algorithm 2.2, we proceed by computing a corresponding feasible worker assignment and the resulting solution of the given instance of the WSFJSP with a *beam search* approach.

In general, beam search uses a graph representation of a solution process and applies breadth-first search with a filtering process to only expand the β (*beam width*) most promising nodes of the graph in each iteration. It was first used by Lowerre (1976). Our implementation of a beam search approach for the subproblem first sorts the operations of all jobs in non-decreasing order of the points in time when their processing is started in a feasible solution of a RMP. It then proceeds by assigning workers to the operations (and potentially shifting the corresponding starting times of the operations based on the correct values of the processing times) in this order by constructing a search tree in a breadth-first search manner as illustrated in Figure 2.5. That

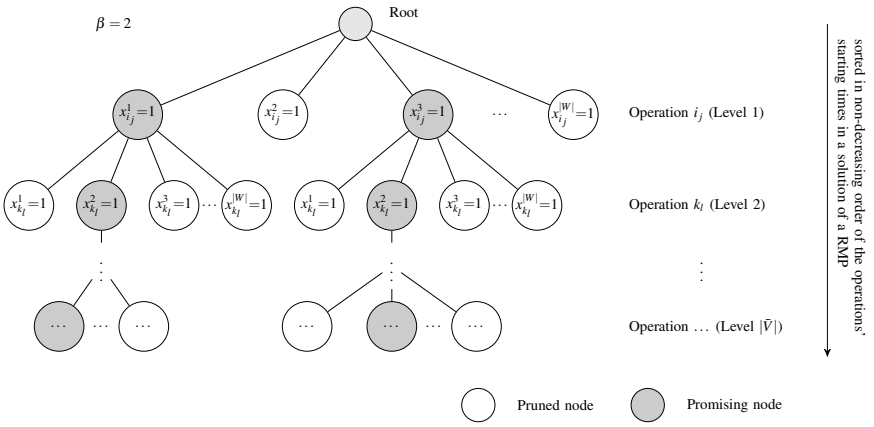


Figure 2.5: Illustration of the beam search algorithm for the worker assignment problem

is, the algorithm decides on the worker assignment for a single operation on each level of the tree and hereafter selects the β most promising nodes for further consideration, while all other

nodes are pruned. The root node represents a situation where no worker has been assigned. A node is evaluated based on the earliest possible completion time of the corresponding operation that results from the worker assignment decisions given in the node. On the last level of the search tree, the algorithm determines the objective function value, i.e. the makespan or the total tardiness, of the β most promising nodes and selects a solution with minimum value.

2.4.2 Decomposition Based Improvement Procedure

Our improvement procedure is based on dividing the relevant time horizon into time windows of equal length (see Section 2.4.2.1). Its main idea is to iterate over pairs of time windows and – for each pair – make use of a given reference solution of a RMP (which is altered in the course of the procedure, starting with the solution determined by Algorithm 2.2) to decide on fixing a subset of variables (2.1), i.e. fixing subsequences of operations on machines. The resulting problems are referred to as *fixed relaxed master problems* (FRMPs, see Section 2.4.2.2). They have a reduced number of free variables, which allows to quickly determine (potentially improved) feasible solutions with a standard MIP solver by solely reoptimizing the subsequences of operations that are (started to be) processed within the time windows in the reference solution. The worker assignment decisions are handled by solving the resulting subproblems either heuristically by the beam search approach introduced in Section 2.4.1.2 or as in our exact approach.

2.4.2.1 Time Windows

The length $twLen$ of the time windows is a crucial parameter of our improvement procedure. On one hand, large time windows result in a large amount of free variables and, consequently, induce a relatively large computational effort needed to solve the resulting FRMPs with a MIP solver. On the other hand, the solution space is less restricted when considering large time windows, which allows for potentially computing high quality solutions. In order to balance this trade-off, it is reasonable to take account of the average processing time of all operations when computing $twLen$, so that we define a multiset $P := \{p_{i_j}^{m,w} | i_j \in \bar{V}, m \in M_{i_j}, w \in W, 0 < p_{i_j}^{m,w} < \infty\}$, and set

$$twLen = \left\lceil \frac{\sum_{p \in P} p}{|P|} \right\rceil .$$

We then define time windows $tw_i := [(i-1) \cdot twLen, i \cdot twLen]$, $i = 1, 2, \ldots$. The number of relevant time windows is not fixed, but is implicitly specified by the current reference solution within our improvement procedure (see Section 2.4.2.3).

2.4.2.2 Fixed Relaxed Master Problem

Algorithm 2.3 presents our method of determining the variables (2.1) that are considered as fixed in the FRMP that is based on a reference solution Sol_{ref}^{RMP} of a RMP for a given pair of time windows tw_a and tw_b, $a \neq b$. Here, $\mathbf{t^{ref}} := (t_{i_j}^{ref} | i_j \in V)$ refers to the time variables

of Sol_{ref}^{RMP}, while $\mathbf{y^{ref}} := (y_{i_j,k_l}^{ref,m} | i_j \in V, k_l \in V_{i_j}, m \in M_{i_j,k_l})$ denotes the vector of variables (2.1) of Sol_{ref}^{RMP}. The algorithm iterates over all machines and all pairs of operations that are

Input: Solution Sol_{ref}^{RMP} ($\mathbf{y^{ref}}$, $\mathbf{t^{ref}}$) of a RMP of an instance $Inst$ of WSFJSP, time windows tw_a and tw_b, $a \neq b$
Output: Sets F_0 and F_1

1 Initialize $F_1 := \emptyset$; ▷ `Set of indices of variables that will be fixed to one`
2 Initialize $F_0 := \emptyset$; ▷ `Set of indices of variables that will be fixed to zero`
3 Initialize $\hat{t}_{i_j} := t_{i_j}^{ref}\ \forall i_j \in V$; ▷ `Auxiliary variables`
4 **forall** $i_j \in V$, $k_l \in V_{i_j}$, $m \in M_{i_j,k_l}$ *with* $y_{i_j,k_l}^{ref,m} = 1$ **do**
  ▷ `Update auxiliary variables if operation` 0_1 `is involved`
5 **if** $k_l = 0_1$ **then** $\hat{t}_{k_l} := \hat{t}_{i_j} + p_{i_j}^{m,min}$;
6 **else if** $i_j = 0_1$ **then** $\hat{t}_{i_j} := 0$;
  ▷ `Check time windows` tw_a `and` tw_b `and update` F_0 `and` F_1
7 **if** $\hat{t}_{i_j} \notin tw_a$ **and** $\hat{t}_{i_j} \notin tw_b$ **and** $\hat{t}_{k_l} \notin tw_a$ **and** $\hat{t}_{k_l} \notin tw_b$ **then**
8 $F_1 := F_1 \cup \{(i_j, k_l, m)\}$;
9 **forall** $q_v \in V_{i_j} \setminus \{k_l\}$ **do**
10 **if** $i_j \neq 0_1$ **then** $F_0 := F_0 \cup \{(i_j, q_v, m') | m' \in M_{i_j,q_v}\}$;
11 **else if** $m \in M_{q_v}$ **then** $F_0 := F_0 \cup \{(0_1, q_v, m)\}$;
12 **end**
13 **forall** $q_v \in \tilde{V}_{k_l} \setminus \{i_j\}$ **do**
14 **if** $k_l \neq 0_1$ **then** $F_0 := F_0 \cup \{(q_v, k_l, m') | m' \in M_{q_v,k_l}\}$;
15 **else if** $m \in M_{q_v}$ **then** $F_0 := F_0 \cup \{(q_v, 0_1, m)\}$;
16 **end**
17 **end**
18 **end**

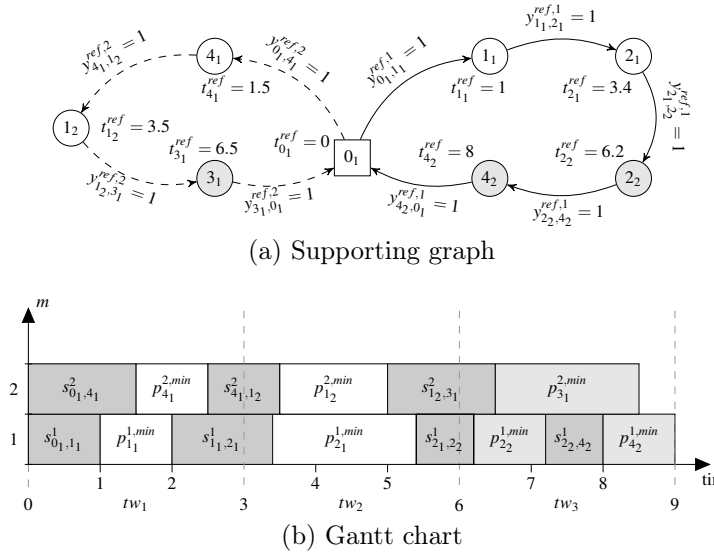
Algorithm 2.3 Determining the set of variables to be fixed

consecutively sequenced on these machines in the given solution (loop 4–18). If, for a given pair of operations, none of the starting times of these operations lie within one of the time windows (line 7), the corresponding variable (2.1) will later be fixed to one (line 8). Based on this decision, a number of additional, directly related variables (2.1) will necessarily later be fixed to zero (lines 9–16). Within the algorithm, the variables that will later be fixed are stored in the sets F_0 (fix to zero) and F_1 (fix to one) via their indices.

Consider an exemplary reference solution of a RMP as given in Figure 2.6 (refer to Figure 2.3 for or an explanation of the illustrations) and assume that the time windows $tw_1 = [0, 3]$ and $tw_2 = [3, 6]$ are considered. When, for example, analyzing $y_{2_2,4_2}^{ref,1}$, the algorithm inserts $(2_2, 4_2, 1)$ into the set F_1, i.e. the decision to process operation 4_2 directly after operation 2_2 on machine 1 will later be fixed, because both starting times $t_{2_2}^{ref}$ and $t_{4_2}^{ref}$ lie outside of the considered time windows in the reference solution. The set F_0 is updated accordingly. When, on the other hand, analyzing $y_{2_1,2_2}^{ref,1}$, no subsequence is fixed, because $t_{2_1}^{ref}$ lies inside of time window tw_2.

In order to obtain the mathematical model of the FRMP based on a given reference solution Sol_{ref}^{RMP} and the result of Algorithm 2.3, the following constraints are added to the RMP:

$$y_{i_j,k_l}^{m} = 1 \quad \forall\, (i_j, k_l, m) \in F_1,$$
$$y_{i_j,k_l}^{m} = 0 \quad \forall\, (i_j, k_l, m) \in F_0.$$



(a) Supporting graph

(b) Gantt chart

Figure 2.6: Exemplary reference solution of a RMP

A MIP solver will therefore reoptimize the subsequences of operations that are started to be processed within the given time windows in Sol^{RMP}_{ref} when being called on this FRMP.

2.4.2.3 Heuristic Framework

We are now ready to present the details of our decomposition based improvement procedure in Algorithm 2.4. The algorithm stores solutions of RMPs, FRMPs, and the WSFJSP instance in the data structures defined in Table 2.3. To ease the notation, we refer to the value of

Table 2.3: Solutions stored by the heuristic framework

Sol^{RMP}_{best}	Best solution found for RMP
Sol^{RMP}_{ref}	Current reference solution of RMP
Sol^{FRMP}	Current solution of FRMP
Sol	Current solution of WSFJSP
Sol_{best}	Best solution found for WSFJSP

the objective function of a solution Sol by using an additional label, i.e. $\ddot{Sol}$, throughout the algorithm.

In the *initialization phase* (step 1) of the algorithm, the length of the time windows is calculated as described in Section 2.4.2.1, the RMP is initialized with an empty set of logic inequalities, and an initial solution of the WSFJSP instance is determined as described in Section 2.4.1. During the process of generating this solution, the reference solution Sol^{RMP}_{ref} is set to the solution determined by Algorithm 2.2. It is potentially altered multiple times in the course of the algorithm (steps 2e i and 2f). In the *iteration phase* (step 2), the algorithm first computes the current pair of time windows (step 2a) based on the reference solution. If all pairs have been traversed, the algorithm exits (step 2a iii). Otherwise, the current pair of time windows is discarded with probability $1 - \rho_{tw}$. Then, in steps 2b and 2c, the FRMP is constructed based on the current

Input: Instance $Inst$ of WSFJSP, parameters ρ_{tw} and λ
Output: Solution Sol_{best} of $Inst$

1. Initialization:
   (a) Compute and initialize $twLen$.
   (b) Initialize the RMP of $Inst$ with an empty set of logic inequalities.
   (c) Determine Sol_{ref}^{RMP} by calling Algorithm 2.2 on the RMP. Set $Sol_{best}^{RMP} := Sol_{ref}^{RMP}$.
   (d) Compute a feasible solution for the subproblem based on Sol_{ref}^{RMP} with the beam search approach of Section 2.4.1.2. Retrieve the corresponding feasible solution Sol of the WSFJSP instance. Set $Sol_{best} := Sol$.
   (e) Initialize $counter := 1$, $a := 1$, and $b := 1$.
2. Iteration:
   (a) Define current pair of time windows as follows:
      i. Set $b := b + 1$.
      ii. If $(b-1) \cdot twLen$ is not smaller than the completion time of the last operation which is completed in Sol_{ref}^{RMP}, set $a := a + 1$ and $b := a + 1$.
      iii. If $(b-1) \cdot twLen$ is not smaller than the completion time of the last operation which is completed in Sol_{ref}^{RMP}, *exit algorithm.*
      iv. Define $tw_a := [(a-1) \cdot twLen, a \cdot twLen]$ and $tw_b := [(b-1) \cdot twLen, b \cdot twLen]$.
      v. Go to step 2a i with probability $1 - \rho_{tw}$.
   (b) Determine F_0 and F_1 by calling Algorithm 2.3 on Sol_{ref}^{RMP}.
   (c) Construct FRMP based on RMP (including the current set of logic inequalities), F_0, and F_1. Compute a feasible solution Sol^{FRMP}.
   (d) If $\ddot{Sol}^{FRMP} \leq \ddot{Sol}_{best}^{RMP}$, update $Sol_{best}^{RMP} := Sol^{FRMP}$.
   (e) If $\ddot{Sol}^{FRMP} < \ddot{Sol}_{best}$, do the following:
      i. Update $Sol_{ref}^{RMP} := Sol^{FRMP}$.
      ii. Compute a feasible solution for the subproblem based on Sol_{ref}^{RMP} and retrieve the corresponding WSFJSP solution Sol.
      iii. If $\ddot{Sol} < \ddot{Sol}_{best}$, update $Sol_{best} := Sol$. Else, set $counter := counter + 1$.
      iv. If $\ddot{Sol}_{ref}^{RMP} < \ddot{Sol}$, generate the corresponding logic inequality (2.10) or (2.16) and add it to RMP.
      v. If $counter \leq \lambda$, go to step 2b.
   (f) Update $Sol_{ref}^{RMP} := Sol_{best}^{RMP}$ and set $counter := 1$.
   (g) Go to step 2a.

Algorithm 2.4 Heuristic framework for WSFJSP

set of logic inequalities (see Section 2.4.2.2) and a solution Sol^{FRMP} of the resulting FRMP is computed by some algorithm. If promising, i.e. if $\ddot{Sol}^{FRMP} < \ddot{Sol}_{best}$, a feasible solution of the subproblem is computed by some algorithm (step 2e ii) and the best known solution of the WSFJSP instance is potentially updated (step 2e iii). A logic inequality is generated and added to the RMP (step 2e iv) if this is necessary. This process of generating the FRMP and solving the corresponding subproblem is repeated at most λ times for a given pair of time windows (steps 2e iii and 2e v).

2.5 Computational Study

In order to evaluate the performance of our solution approaches, we conducted extensive computational tests. The tests were performed on a PC with an Intel® Core^{TM} i7-4770 CPU, running at 3.4 GHz, with 16 GB of RAM under a 64-bit version of Windows 8. All algorithms were implemented in C^{++} (Microsoft Visual Studio 2015). We used IBM ILOG CPLEX in version 12.7 as a MIP solver.

We implemented six approaches:

1. E-DM refers to the exact branch-and-cut approach (Algorithm 2.1), using CPLEX as a standard solver.

2. E-IM refers to calling CPLEX in its standard settings on an integrated MIP model for WSFJSP. This approach is intended to be a benchmark for evaluating the performance of E-DM. The integrated model is presented in Section 2.5.1.

3. H-DM refers to the heuristic framework (Algorithm 2.4), using CPLEX in step 2c (determining feasible solutions of FRMPs, 10 seconds time limit) and in step 2e ii (determining feasible solutions of subproblems, the time limit is set to the remaining time with respect to the limit set for the overall heuristic framework).

4. H-DMB refers to the heuristic framework (Algorithm 2.4), using CPLEX in step 2c (10 seconds time limit) and applying the beam search approach of Section 2.4.1.2 in step 2e ii.

5. H-HIER refers to an adapted version of the heuristic framework (Algorithm 2.4, CPLEX time limit of 10 seconds in step 2c), where the subproblem is solved only once at the very end of the procedure. That is, steps 1d and 2e are replaced by a single call of the beam search approach on the best reference solution after having iterated over all relevant pairs of time windows.

6. LS refers to a local search procedure inspired by Mastrolilli and Gambardella (2000), which we use as a benchmark heuristic when evaluating the heuristic framework. It is presented in detail in Section 2.5.3.3.

Whenever calling CPLEX on a FRMP within H-DM, H-DMB, or H-HIER, we provide the vector $\mathbf{y^{ref}}$ of the corresponding solution Sol^{RMP}_{ref} as a warm start. Similarly, in H-DM, when

solving a subproblem with CPLEX, we provide a lower bound on the optimal objective function value, i.e. the objective function value of the corresponding FRMP. Furthermore, we provide CPLEX with the vector $\mathbf{t^{ref}}$ of the corresponding solution Sol_{ref}^{RMP}, which allows to quickly check if this partial solution remains feasible for the subproblem.

We set the large integer B to $\sum_{i_j \in \bar{V}}(p_{i_j}^{max} + s_{i_j}^{max})$ in constraint (2.7) of the MP and to $\sum_{i_j \in \bar{V}}(\tilde{p}_{i_j}^{max} + s_{i_j}^{max})$ in constraints (2.25) and (2.28) of the subproblem as well as in the relevant constraints of the integrated MIP model. Here, $s_{i_j}^{max} := \{s_{k_l,i_j}^{m} | k_l \in \tilde{V}_{i_j}, m \in M_{i_j,k_l}\}$, $p_{i_j}^{max} := \max\{p_{i_j}^{m,min} | m \in M_{i_j}\}$, and $\tilde{p}_{i_j}^{max} := \max\{p_{i_j}^{m,w} | p_{i_j}^{m,w} \neq \infty, m \in M_{i_j}, w \in W\}$, for all $i_j \in \bar{V}$.

2.5.1 Integrated MIP model

E-IM is based on an integrated MIP model for WSFJSP. This model is in line with the formulations of the MP and the subproblem in Section 2.3, but replaces the variables (2.19) with variables

$$x_{i_j}^{m,w} := \begin{cases} 1, & \text{if operation } i_j \text{ is processed} \\ & \text{by worker } w \text{ on machine } m \\ 0, & \text{else} \end{cases} \qquad \forall\, i_j \in \bar{V}, m \in M_{i_j}, w \in W.$$

The model for minimizing the makespan is as follows.

$$\min C_{max} \tag{2.40}$$

s.t. (2.4)–(2.6), (2.9), (2.11)–(2.13), (2.29), (2.32)–(2.33),

$$t_{i_{q_i}} + \sum_{m \in M_{i_{q_i}}} \sum_{w \in W} x_{i_{q_i}}^{m,w} \cdot p_{i_{q_i}}^{m,w} \leq C_{max} \qquad \forall\, i \in I, \tag{2.41}$$

$$\sum_{i_j \in \{a_b \in \tilde{V}_{k_l} | m \in M_{a_b}\}} y_{i_j,k_l}^{m} = \sum_{w \in W} x_{k_l}^{m,w} \qquad \forall\, k_l \in \bar{V}, m \in M_{k_l}, \tag{2.42}$$

$$t_{i_j} + s_{i_j,k_l}^{m} + \sum_{w \in W} x_{i_j}^{m,w} \cdot p_{i_j}^{m,w} - t_{k_l} \leq \left(1 - y_{i_j,k_l}^{m}\right) B \quad \forall\, i_j \in V, k_l \in \bar{V}_{i_j}, m \in M_{i_j,k_l}, \tag{2.43}$$

$$t_{i_j} + \sum_{m \in M_{i_j}} \sum_{w \in W} x_{i_j}^{m,w} \cdot p_{i_j}^{m,w} \leq t_{i_{j+1}} \qquad \forall\, i_j \in \bar{V} \text{with } j \leq q_i - 1, \tag{2.44}$$

$$x_{i_j,k_l}^{w} \geq \sum_{m \in M_{i_j}} x_{i_j}^{m,w} + \sum_{m \in M_{k_l}} x_{k_l}^{m,w} - 1 \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, w \in W, \tag{2.45}$$

$$t_{i_j} + \sum_{m \in M_{i_j}} \sum_{w \in W} x_{i_j}^{m,w} \cdot p_{i_j}^{m,w} - t_{k_l} \leq B \cdot v_{i_j,k_l} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, \tag{2.46}$$

$$x_{0_1}^{m,w} = 0 \qquad \forall\, m \in M_{i_j}, w \in W, \tag{2.47}$$

$$x_{i_j}^{m,w} \in \{0,1\} \qquad \forall\, i_j \in \bar{V}, m \in M_{i_j}, w \in W. \tag{2.48}$$

The objective function and most constraints are either identical to the ones in the MP and the subproblem or have been adapted in a straightforward manner. Constraint (2.42) connects the

sequencing variables with the worker assignment variables. The MIP model for minimizing the total tardiness modifies the objective function and constraint (2.41) in analogy to Section 2.3. We do not present it in detail for the sake of brevity.

2.5.2 Instance Generation and Parameter Settings

Our computational tests were performed on randomly generated test instances as well as real-world test instances based on data of our industry partner.

2.5.2.1 Random Testbed

Our random testbed is composed of three classes of instance sets. These classes differ in the size of the included instances (small, medium, and large), which is expressed by an identifier $size \in \{s, m, l\}$. Each class is composed of eight sets of test instances with differing numbers of jobs $|I|$, machines $|M|$, and workers $|W|$. Each set is composed of ten instances and is denoted by $size_{|I|,|M|,|W|}$, where $size \in \{s, m, l\}$. For each instance, the number of operations q_i of jobs $i \in I$, the number of eligible machines $|M_{i_j}|$ for operations $i_j \in O_i$ of jobs $i \in I$, and the integer setup times $s^m_{i_j,k_l}$ (including $s^m_{0_1,k_l}$) when processing operation $k_l \in O_k$ of job $k \in I$ immediately after operation $i_j \in O_i$ of job $i \in I$ on machine $m \in M_{i_j,k_l}$ are drawn from uniform distributions over the intervals given in Table 2.4. Our process of generating the processing times of the operations is

Table 2.4: Random testbed

small instances					medium instances					large instances				
set	q_i	$\lvert M_{i_j}\rvert$	p_{i_j}	$s^m_{i_j,k_l}$	set	q_i	$\lvert M_{i_j}\rvert$	p_{i_j}	$s^m_{i_j,k_l}$	set	q_i	$\lvert M_{i_j}\rvert$	p_{i_j}	$s^m_{i_j,k_l}$
$s_{3,2,2}$	$[2,3]$	$[2,2]$	$[3,6]$	$[1,4]$	$m_{10,5,4}$	$[2,3]$	$[2,3]$	$[4,8]$	$[1,5]$	$l_{20,15,12}$	$[5,10]$	$[4,6]$	$[5,15]$	$[1,10]$
$s_{4,2,2}$	$[2,3]$	$[2,2]$	$[3,6]$	$[1,4]$	$m_{10,5,5}$	$[2,3]$	$[2,3]$	$[4,8]$	$[1,5]$	$l_{20,15,15}$	$[5,10]$	$[4,6]$	$[5,15]$	$[1,10]$
$s_{5,3,3}$	$[2,3]$	$[2,2]$	$[3,6]$	$[1,4]$	$m_{10,8,6}$	$[4,6]$	$[3,4]$	$[4,8]$	$[1,5]$	$l_{25,20,16}$	$[5,10]$	$[5,7]$	$[5,15]$	$[1,10]$
$s_{6,3,3}$	$[2,3]$	$[2,2]$	$[3,6]$	$[1,4]$	$m_{10,8,8}$	$[4,6]$	$[3,4]$	$[4,8]$	$[1,5]$	$l_{25,20,20}$	$[5,10]$	$[5,7]$	$[5,15]$	$[1,10]$
$s_{7,4,3}$	$[2,3]$	$[2,2]$	$[3,6]$	$[1,4]$	$m_{15,10,8}$	$[4,6]$	$[3,5]$	$[5,10]$	$[1,7]$	$l_{30,20,16}$	$[5,7]$	$[5,7]$	$[5,15]$	$[1,10]$
$s_{7,4,4}$	$[2,3]$	$[2,2]$	$[3,6]$	$[1,4]$	$m_{15,10,10}$	$[4,6]$	$[3,5]$	$[5,10]$	$[1,7]$	$l_{30,20,20}$	$[5,7]$	$[5,7]$	$[5,15]$	$[1,10]$
$s_{8,4,3}$	$[2,3]$	$[2,2]$	$[3,6]$	$[1,4]$	$m_{20,10,8}$	$[4,6]$	$[3,5]$	$[5,10]$	$[1,7]$	$l_{40,20,16}$	$[4,6]$	$[5,7]$	$[5,15]$	$[1,10]$
$s_{8,4,4}$	$[2,3]$	$[2,2]$	$[3,6]$	$[1,4]$	$m_{20,10,10}$	$[4,6]$	$[3,5]$	$[5,10]$	$[1,7]$	$l_{40,20,20}$	$[4,6]$	$[5,7]$	$[5,15]$	$[1,10]$

as follows. We first draw auxiliary integer parameters p_{i_j} for all jobs $i \in I$ and operations $i_j \in O_i$ from uniform distributions over the intervals given in Table 2.4. Based on these parameters, we construct varying processing times over the corresponding eligible machines $m \in M_{i_j}$ by drawing integer values $p^m_{i_j}$ from uniform distributions over $[\lfloor 0.9 \cdot p_{i_j} + 0.5 \rfloor, \lceil 1.1 \cdot p_{i_j} - 0.5 \rceil]$ and, in the last step, we incorporate dependencies on the workers $w \in W$ by drawing integer values $p^{m,w}_{i_j}$ from uniform distributions over $[\lfloor 0.9 \cdot p^m_{i_j} + 0.5 \rfloor, \lceil 1.1 \cdot p^m_{i_j} - 0.5 \rceil]$. Finally, the integer due dates d_i are drawn from uniform distributions over the interval $[\lfloor \mu_i \cdot \left(1 - \frac{\Phi}{2}\right) + 0.5 \rfloor, \lceil \mu_i \cdot \left(1 + \frac{\Phi}{2}\right) - 0.5 \rceil]$ for all jobs $i \in I$ (cf. Vilcot and Billaut, 2008). Here,

$$\mu_i := \left(1 + \frac{\Omega \cdot |I|}{|M|}\right) \cdot \frac{\sum\limits_{p \in P_i} p}{|P_i|} \quad \forall i \in I,$$

where P_i is a multiset $\{p_{i_j}^{m,w} | i_j \in O_i, m \in M_{i_j}, w \in W, 0 < p_{i_j}^{m,w} < \infty\}$ for all jobs $i \in I$. We set $\Phi = 0.5$ and $\Omega = 0.3$ for all small instances, $\Phi = 0.5$ and $\Omega = 0.4$ for all medium instances, and $\Phi = 0.3$ and $\Omega = 0.5$ for all large instances.

2.5.2.2 Real-World Instances

As mentioned in Section 2.1.1, our research is motivated by a real-world problem setting. Our industry partner provided us with data on its production processes, including the processing times of the manufacturing operations on their eligible machines and the sequence-dependent setup times, as well as the customer demands (including due dates) over a time period of several months. Additionally, we received the relevant information on worker qualifications needed to derive all relevant processing times. Based on this data, we constructed ten realistic scheduling scenarios for testing our algorithms. Each scenario consists of a set of jobs relating to the company's product portfolio, including the respective lot sizes and their due dates. Currently, the product portfolio consists of more than one hundred products and the company uses 16 different multi-purpose machines for processing the manufacturing operations. The number of eligible machines for the operations varies between one and two. The machine operators work on a shift-based system. Each shift is staffed with nine workers of similar qualifications. We will therefore assume that nine representative machine workers are available at all time in all of our scenarios.

In line with the random testbed, we denote each scheduling scenario – or problem instance – by $r_{|I|,|M|,|W|}$. Table 2.5 lists all instances, including their total number of operations. Note

Table 2.5: Real-world instances

instance	$r_{14,13,9}$	$r_{15,11,9}$	$r_{16,14,9}$	$r_{25,16,9}$	$r_{35,16,9}$	$r_{45,16,9}$	$r_{55,16,9}$	$r_{60,16,9}$	$r_{70,16,9}$	$r_{80,16,9}$
$\sum_{i \in I} \lvert O_i \rvert$	34	34	38	63	84	106	136	146	162	188

that the number of machines is smaller than 16 in the three smallest instances. This is because not all machines are needed for producing the respective products of the scenario. These three scenarios are representative instances that feature planning horizons of at most one week, which is what our industry partner is currently capable of scheduling manually. All remaining instances have been constructed to analyze the capability and performance of our algorithms for larger planning horizons in a real-world context. As mentioned in Section 2.1.1, our industry partner wishes to compute schedules on a daily basis. Hence, when wanting to take account of these planning horizons, our approaches will have to be embedded into a rolling horizon planning approach. While we will not explicitly analyze this type of approach in our computational study, we note at this point that all of our algorithms are flexible enough to support a rolling horizon procedure that allows rescheduling decisions if the WSFJSP parameters are set to appropriate values. Most important, by appropriately setting the setup times associated to the dummy operation, one can take account of machines that are currently processing operations at the beginning of the planning horizon. Our model also allows for interrupting the processing of these operations (as sometimes decided by experts of the manufacturing companies) by appropriately

(re-)defining the set of operations and the associated setup times.

The detailed data of all instances is available in supplementary files that accompany this paper. The processing times of the operations for the representative workers on the machines vary between 15 and 5280 minutes (1090 minutes on average). The setup times range from zero to 300 minutes. Finally, the due dates of the jobs vary between one and six weeks.

2.5.2.3 Parameter Settings

With respect to the parameters of Algorithms 2.1 and 2.4, we set $\rho_s = 0.005$ and $\lambda = 3$. The remaining parameters were set based on the size of the instances (see Table 2.6). Finally, we set

Table 2.6: Setup of the algorithms

	small instances	medium and real-world instances	large instances
ρ_{tw}	1	0.4	0.2
β	25	16	8

a time limit of 3,600 seconds for each call of an algorithm. Note, however, that we check the current runtime of the algorithmic framework solely at the beginning of step 2b of Algorithm 2.4, so that the overall runtime of the framework upon termination may slightly exceed the time limit.

2.5.3 Results and Analysis

This section presents and analyzes the results of our computational study. Section 2.5.3.1 analyzes the performance of the exact approaches on the random testbed. Hereafter, Section 2.5.3.2 presents the corresponding results of the heuristic approaches H-DM, H-DMB, and H-HIER. In Section 2.5.3.3, we describe LS and evaluate the aforementioned heuristics against this procedure on the random testbed. Section 2.5.3.4 focuses on the results for the real-world instances.

2.5.3.1 Exact Approaches

Table 2.7 presents the computational results for the exact approaches E-IM and E-DM for the small test instances of our random testbed. It includes information about the percentage of instances within each set that were solved to optimality within the given time limit (columns "opt.") as well as the average computational time needed to compute the optimal solutions (columns "t_{avg}") for both objectives, minimizing the makespan C_{max} or the total tardiness T. The decomposition based approach E-DM clearly outperforms the integrated approach E-IM both with respect to the number of instances solved to optimality and the runtimes. While this is true for both objectives, the benefit of using the decomposition based approach is more pronounced in case of minimizing the makespan. The maximum number of logic inequalities generated over all calls of E-DM for the small test instances was 1,503 (235) for the objective of minimizing the makespan (total tardiness). On average, a logic inequality was generated and added to the branch-and-cut process in 29% (38%) of the calls of step 3 of Algorithm 2.1.

Table 2.7: Performance of exact approaches for small instances

	C_{max}				T			
	E-IM		E-DM		E-IM		E-DM	
set	opt. [%]	t_{avg} [s]	opt. [%]	t_{avg} [s]	opt. [%]	t_{avg} [s]	opt. [%]	t_{avg} [s]
$s_{3,2,2}$	100	0.9	100	0.3	100	1	100	0.5
$s_{4,2,2}$	100	47.5	100	13.3	100	58.2	100	19.7
$s_{5,3,3}$	100	36.8	100	5	100	51.1	100	12.4
$s_{6,3,3}$	60	756.7	100	425.7	70	882.8	80	482.6
$s_{7,4,3}$	10	2142.5	70	913.1	20	1952.9	40	944.3
$s_{7,4,4}$	80	1161.6	100	112.2	50	1074.2	90	839.3
$s_{8,4,3}$	-	-	20	2773.1	-	-	-	-
$s_{8,4,4}$	-	-	40	622.7	-	-	10	3132.4

Unfortunately, neither of the exact approaches was able to solve instances of medium or large size to optimality within the time limit. Table 2.8 therefore focuses on illustrating the percentage of instances for which E-DM found a feasible solution. It can be seen that E-DM returns feasible

Table 2.8: Capability of finding feasible solutions with E-DM within the time limit of 3,600 seconds

small instances			medium instances		
set	C_{max} [%]	T [%]	set	C_{max} [%]	T [%]
$s_{3,2,2}$	100	100	$m_{10,5,4}$	100	100
$s_{4,2,2}$	100	100	$m_{10,5,5}$	100	100
$s_{5,3,3}$	100	100	$m_{10,8,6}$	90	100
$s_{6,3,3}$	100	100	$m_{10,8,8}$	90	100
$s_{7,4,3}$	100	100	$m_{15,10,8}$	-	-
$s_{7,4,4}$	100	100	$m_{15,10,10}$	-	10
$s_{8,4,3}$	100	100	$m_{20,10,8}$	-	-
$s_{8,4,4}$	100	100	$m_{20,10,10}$	-	-

solutions for instances up to the size of the ones in the set $m_{10,8,8}$ and for one instance in the set $m_{15,10,10}$.

We conclude that, while E-DM is certainly not a reasonable choice when facing large instances of WSFJSP in practice, it clearly outperforms E-IM and is able to compute benchmark solutions that allow to assess the performance of heuristic approaches for instances of medium size.

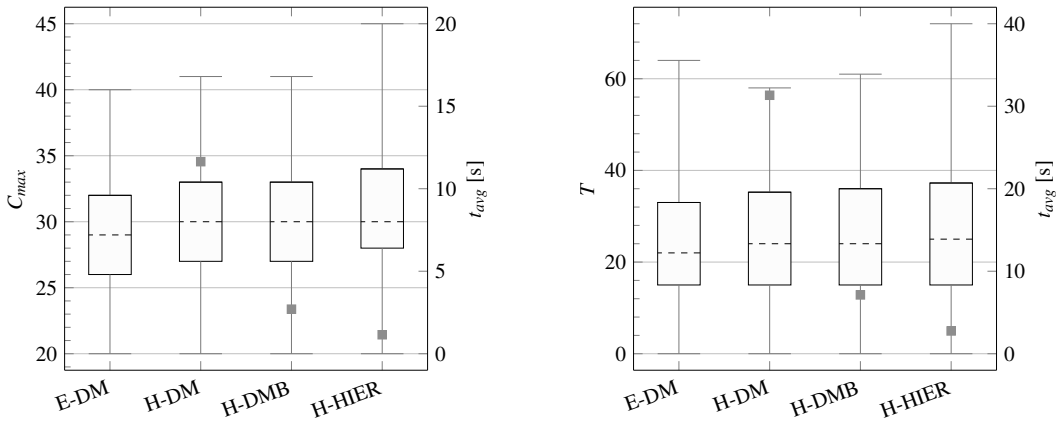
2.5.3.2 Heuristic Approaches

The performance of the heuristic approaches H-DM, H-DMB, and H-HIER in comparison to E-DM is illustrated in Table 2.9 for the small test instances. For each objective function, each instance set, and each solution approach, the table presents information on the average objective function values of the best solutions returned by the respective algorithms (columns "C_{max}^{avg}" and "T^{avg}") as well as the average runtimes for computing these solutions (columns "t_{avg}"). Entries "tl" denote average computational times that correspond to or exceed the time limit of 3,600 seconds. Bold entries highlight the best heuristic approaches with respect to the average solution quality.

Table 2.9: Performance of heuristic approaches for small instances

	C_{max}								T							
	E-DM		H-DM		H-DMB		H-HIER		E-DM		H-DM		H-DMB		H-HIER	
set	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]
$s_{3,2,2}$	23.9	0.3	**24.2**	0.2	24.3	0.2	24.5	0.1	8.6	0.5	**8.6**	0.2	**8.6**	0.2	8.7	0.2
$s_{4,2,2}$	30	13.3	**30.4**	0.4	30.5	0.4	30.7	0.3	16.2	19.7	**17.3**	0.5	17.4	0.4	17.4	0.3
$s_{5,3,3}$	27.1	5	**27.7**	0.5	**27.7**	0.4	28	0.3	14.3	12.4	**15.4**	1	15.5	0.9	16	0.4
$s_{6,3,3}$	29.9	425.7	**31**	1.5	31.1	1.2	31.8	0.9	23.8	1106.1	26.3	2.3	**25.8**	1.8	26.6	1.3
$s_{7,4,3}$	31	1719.2	**32.1**	22	32.9	3.8	33.5	1.1	29.3	2537.7	**30.9**	42.5	31.4	9.7	35.3	3.2
$s_{7,4,4}$	27.5	112.2	**28.2**	1.4	**28.2**	1.3	28.4	0.8	24.4	1115.4	**26.2**	6.4	26.5	6.8	26.9	2.7
$s_{8,4,3}$	34.5	3436.6	**35.4**	63.2	36	11.0	37.6	3.4	44.7	tl	**45.7**	187.6	46.6	27.1	53.7	6.5
$s_{8,4,4}$	29.7	2409.1	31.1	4	**31**	3.4	31.1	2.3	34.7	3553.3	**36.6**	10.2	37	10.1	36.8	7.5

Figure 2.7 complements Table 2.9 by presenting boxplots of the objective function values returned by the algorithms (left ordinate) as well as data points (gray squares) on the overall average of the corresponding runtimes (right ordinate) for the heuristic approaches. Each boxplot



(a) Minimizing the makespan (b) Minimizing the total tardiness

Figure 2.7: Computational results for small instances

depicts the first quartile (bottom of the box), the third quartile (top of the box), the median (dotted line within the box), the minimum (bottom whisker), and the maximum (top whisker) of the objective function values.

Based on Table 2.9 and Figure 2.7, we observe that the solution quality of all heuristic approaches is competitive when compared with the (mostly optimal, see Section 2.5.3.1) solutions computed by E-DM for both objectives. That is, all heuristics provide high quality solutions. H-DM tends to provide the best solutions at the cost of substantially larger running times than the ones of the other heuristic approaches. This is a result of determining optimal solutions to the subproblems within the heuristic framework. Applying beam search (H-DMB) results in slightly worse solutions when compared with the solutions determined by H-DM but clearly pays off with respect to the running times. A similar but less pronounced effect can be observed when comparing H-DMB and H-HIER.

Table 2.10 presents the computational results for the instances of medium size. In contrast to

Table 2.10: Performance of heuristic approaches for medium instances

	C_{max}								T							
	E-DM		H-DM		H-DMB		H-HIER		E-DM		H-DM		H-DMB		H-HIER	
set	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]
$m_{10,5,4}$	41.9	tl	**42.2**	158.8	43.6	14.3	44.4	5.8	43	tl	**40.4**	543.5	42.4	31.8	45.5	7
$m_{10,5,5}$	38.2	tl	**39.3**	12.5	39.6	11.5	39.7	6.4	22.9	tl	**24.3**	32.9	24.7	26.2	27.3	7
$m_{10,8,6}$	165	tl	57.7	tl	**56.4**	49.6	57.8	13.5	746.3	tl	25.5	3157.4	**17.8**	75.1	19	17.2
$m_{10,8,8}$	198.1	tl	52.9	317.1	**52.6**	32.9	53.4	15.7	656.2	tl	5.8	901.4	**4.6**	53.8	9.3	14.4
$m_{15,10,8}$	-	-	84.2	tl	**78.9**	186.3	80	45.3	-	-	154.5	tl	**66.4**	417.2	84.2	106.5
$m_{15,10,10}$	-	-	80.8	3105.2	**75.7**	118.1	77.7	47.9	2367	tl	64.1	2361.1	**34.5**	255.2	42.8	85.4
$m_{20,10,8}$	-	-	107.3	tl	**102**	525.8	103.3	140.7	-	-	286.2	tl	**164.5**	895.3	178	216.5
$m_{20,10,10}$	-	-	98.6	tl	**93.8**	409.9	**93.8**	156.5	-	-	224	3287.5	**143**	618.7	156.5	233.6

the results for the small instances, H-DMB now clearly outperforms H-DM, both with respect to solution quality and computational time. Again, this mainly results from solving the subproblems to optimality within H-DM, which causes high computational effort for increasing instance sizes, so that less pairs of time windows are traversed within the given time limit. Moreover, note that all heuristics outperform E-DM for most instance sets. When comparing H-DMB and H-HIER, we observe larger differences in the average solution qualities as in case of the small test instances, particularly for the case of minimizing total tardiness. Nevertheless, the runtimes are in ranges that allow the use of both H-DMB and H-HIER in real-world scenarios. The same results hold for the large test instances, as can be seen in Table 2.11, where we restrict ourselves to comparing H-DMB and H-HIER due to the above results for the medium instances.

Table 2.11: Performance of H-DMB and H-HIER for large instances

	C_{max}				T			
	H-DMB		H-HIER		H-DMB		H-HIER	
set	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]
$l_{20,15,12}$	**146.6**	217.3	149.7	54.1	**37.7**	466.7	55.8	113.3
$l_{20,15,15}$	**150.2**	206.1	152	62.5	**32.7**	393.8	53.5	124.9
$l_{25,20,16}$	**141.3**	461.6	144.5	95.3	**17.4**	934.4	32.4	213.2
$l_{25,20,20}$	**137.5**	295.1	139.7	101.1	**29**	480.8	39.8	193.2
$l_{30,20,16}$	**125.2**	357.6	127.7	95.3	**146.8**	889.3	181	208.1
$l_{30,20,20}$	**119**	246	120.1	98.8	**90.4**	721.5	145.6	208.4
$l_{40,20,16}$	**131.8**	588.8	134.8	135.6	**247.6**	1320.4	285.4	306.7
$l_{40,20,20}$	**126.2**	359.8	127.9	117.2	**247.3**	998.1	302.1	287.8

Summing up the above results, the frequency of computing solutions of subproblems within the heuristic framework as well as the algorithms applied to determine these solutions are components of major importance when wanting to balance the trade-off between runtime and solution quality. When facing medium or large instances, it does not pay off to try to solve the subproblems to optimality, but one must make use of a heuristic approach. Calling this approach more often has a positive effect on solution quality at the cost of larger runtimes. Moreover, as to be expected, we observe that the staffing level, i.e. the ratio of the number of workers and the number of machines, influences the performance of the algorithms. Large staffing levels allow to

compute better solutions with less computational effort.

In order to analyze the sensitivity of our heuristic framework to a change of the initial reference solution of RMP (determined in step 1c of Algorithm 2.4), we have implemented an additional simple priority rule that can be used in Algorithm 2.2 (line 10). In contrast to the priority rules described in Section 2.4.1.1, that make use of most of the relevant problem parameters, this priority rule simply selects the operation with *smallest job index* (SI) among the operations that have not been sequenced and that can start to be processed at the respective point of time. Table 2.12 compares the performance of the heuristic framework for the case of applying SI and the standard setting introduced in Section 2.4.1.1. For the sake of brevity, we

Table 2.12: Sensitivity of H-DMB and H-HIER to a change of the initial reference solution

	C_{max}								T							
	standard setting				SI				standard setting				SI			
	H-DMB		H-HIER		H-DMB		H-HIER		H-DMB		H-HIER		H-DMB		H-HIER	
class	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]
small	30.2	2.7	30.7	1.2	30.7	2.9	31.5	1.3	26.1	7.1	27.7	2.8	29.8	8.1	31.7	3
medium	67.8	168.5	68.8	54	72.5	120.2	75.8	30.2	62.2	296.7	70.3	85.9	114.9	302.3	133.9	87.4
large	134.7	341.5	137.1	95	166.8	446.3	175.6	122.8	106.1	775.6	137	206.9	355.5	910.3	408.7	228.8

restrict ourselves to presenting the average objective function values and computational times over all instances of the three instance classes (small, medium, and large) for H-DMB and H-HIER. We observe large differences of the average objective function values, particulary for the case of minimizing total tardiness, so that we conclude that it pays off to make use of a well designed procedure to determine high quality initial allocation and sequencing decisions in our heuristic framework.

In light of the fact that H-DMB has shown to be a promising approach but uses a heuristic to determine solutions for the subproblems in step 2e ii of Algorithm 2.4, one may wonder if the use of logic inequalities within this setup of the heuristic framework actually pays off. Table 2.13 therefore illustrates the effect of adding logic inequalities by comparing the standard setting of H-DMB (see Section 2.4.2.3) with the setting where the generation of logic inequalities, i.e. step 2e iv of Algorithm 2.4, is disabled. Again, we restrict ourselves to presenting average values for

Table 2.13: Effect of using logic inequalities in H-DMB

	C_{max}					T				
	standard setting			no logic inequalities		standard setting			no logic inequalities	
class	C_{max}^{avg}	t_{avg} [s]	LI [%]	C_{max}^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	LI [%]	T^{avg}	t_{avg} [s]
small	30.2	2.7	39	30.4	3.5	26.1	7.1	47	26.8	8.8
medium	67.8	168.5	91	68.5	157.9	62.2	296.7	92	63.2	280
large	134.7	341.5	95	135	357	106.1	775.6	88	113.7	849.9

the three classes of instances. For the standard setting, the columns "LI" present the percentage of executions of step 2e of Algorithm 2.4 that result in the generation of a logic inequality. The results indicate that the use of logic inequalities within H-DMB has a positive effect on the

objective function values and tends to reduce (or only slightly increase) runtimes for the small and large (medium) instances.

2.5.3.3 Evaluation of the Heuristic Framework against a Local Search Heuristic

As indicated above, we implemented a local search approach (LS) inspired by Mastrolilli and Gambardella (2000) in order to gain additional insights into the performance of our heuristic framework. In line with Algorithm 2.4, LS is called on a feasible allocation and sequencing decision, referred to as Sol^{RMP}, initially determined by Algorithm 2.2. In the course of the algorithm, Sol^{RMP} is altered in a first-fit manner and initially evaluated by dropping the worker assignment restrictions and making use of the lower bounds of the processing times introduced in Section 2.3.1. Given some current solution Sol^{RMP}, LS restricts the search process by computing a *critical path* of the so called *solution graph* of this solution as described by Mastrolilli and Gambardella (2000) (see also Błażewicz et al., 2007), where sequence-dependent setup times are taken account of in a straightforward manner. Basically, a critical path identifies a sequence of operations, any of which fulfills the property that a delayed start of its processing would immediately cause an increase of the makespan (without considering worker assignment restrictions) under the given allocation and sequencing decision. Note that we make use of the makespan criterion independently of the considered objective function when computing critical paths. A neighbor of the current solution Sol^{RMP} is then determined by moving an operation of the critical path to another feasible position on the same machine or to a feasible position in the sequence of operations on any other eligible machine. As outlined above, the objective function value (makespan or total tardiness) of a neighbor (as well as the initial solution determined by Algorithm 2.2) is first evaluated based on the lower bounds of the processing times. If this objective function value is smaller than the one of the currently best known solution of the WSFJSP instance (including worker restrictions), a feasible worker assignment and the corresponding objective function value is determined with the beam search approach of Section 2.4.1.2. If the resulting objective function value remains smaller than the one of the currently best WSFJSP solution, Sol^{RMP} and the overall best WSFJSP solution are updated and LS proceeds by computing a critical path for Sol^{RMP}. LS terminates if none of the neighbors results in an improved objective function value.

Table 2.14 illustrates the average runtimes and objective function values determined by LS in comparison with H-DMB and H-HIER, i.e. the most promising approaches identified in Section 2.5.3.2, for the three instance classes. As can be seen, H-DMB and H-HIER outperform

Table 2.14: Performance of LS on random testbed

	C_{max}						T					
	H-DMB		H-HIER		LS		H-DMB		H-HIER		LS	
class	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	C_{max}^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]	T^{avg}	t_{avg} [s]
small	30.2	2.7	30.7	1.2	32	0.1	26.1	7.1	27.7	2.8	34.4	0.1
medium	67.8	168.5	68.8	54	71.4	10.7	62.2	296.7	70.3	85.9	119.8	7.4
large	134.7	341.5	137.1	95	138.7	66.2	106.1	775.6	137	206.9	234.4	87.6

LS with respect to the average objective function values. This effect is especially pronounced

for the objective of minimizing total tardiness, which is probably caused by restricting LS to compute neighbors based on critical paths as outlined above. The average computational times of LS are smaller than the ones of H-DMB and H-HIER. However, as our practical application allows runtimes in the ranges of the ones of the latter approaches, this effect is less of an issue. Hence, the remainder of this computational study focusses on the decomposition based solution approaches.

2.5.3.4 Results for Real-World Instances

Table 2.15 presents the results on the performance of E-DM as well as the heuristic approaches for the real-world problem instances. For each instance and objective, the table provides results

Table 2.15: Computational results for real-world instances

	C_{max}								T							
	E-DM		H-DM		H-DMB		H-HIER		E-DM		H-DM		H-DMB		H-HIER	
inst.	C_{max}	t [s]	C_{max}	t [s]	C_{max}	t [s]	C_{max}	t [s]	T	t [s]	T	t [s]	T	t [s]	T	t [s]
$r_{14,13,9}$	**6256**	tl	**6256**	14.3	**6274**	15.9	6676	10.2	**0***	18.5	**0***	3.6	170	4.2	1293	2.2
$r_{15,11,9}$	5375	tl	**5272**	13.5	5392	17.5	5475	3.3	**0***	6.8	**0***	0.4	**0***	0.4	**0***	0.2
$r_{16,14,9}$	**7584**	tl	**7584**	18.7	**7584**	3	**7584**	0.2	**864**	tl	**864**	25.2	**864**	5.2	**864**	0.2
$r_{25,16,9}$	8392	tl	**8180**	2190.4	8551	157.2	11596	66.2	2150	tl	**0***	33.3	227	36.8	1716	12.1
$r_{35,16,9}$	11993	tl	**9045**	tl	9333	358.7	9572	76.6	19337	tl	**0***	50.9	**0***	18.8	322	3
$r_{45,16,9}$	25019	tl	**12722**	tl	12820	378.6	12884	95.5	105019	tl	233	822.9	**0***	40.9	761	11.7
$r_{55,16,9}$	29126	tl	15288	tl	**14608**	916.1	16391	225.4	-	-	**0***	1906.4	**0***	310	289	20.5
$r_{60,16,9}$	33993	tl	18953	tl	**17180**	1444.4	18953	253.1	327669	tl	947	tl	**190**	1300.8	1900	14.9
$r_{70,16,9}$	34716	tl	21202	tl	**19590**	1691.2	21202	280.6	353140	tl	4675	tl	**669**	823.7	878	67.8
$r_{80,16,9}$	-	-	23492	tl	**21869**	2694.4	23998	490.7	-	-	10230	tl	**4670**	1152.5	5880	105.8

*: optimal objective function value

on the objective function values of the solutions determined by the algorithms. Bold elements highlight the best approaches in each row of the table. Note that, in contrast to the results for the random testbed, the objective function values represent minutes of real time (see Section 2.5.2.2).

Recall that the three smallest instances are representative instances that our industry partner is currently capable of scheduling manually. For all of these instances, H-DM or H-DMB are able to compute solutions in at most 26 seconds while performing similar to the exact approach E-DM and, in case of H-DM, even determining optimal solutions for two instances when minimizing total tardiness. Even in case of medium time horizons, H-DM is able to determine optimal solutions for minimizing total tardiness within reasonable time, i.e. instances $r_{25,16,9}$, $r_{35,16,9}$ and $r_{55,16,9}$. When wanting to make quick decisions for medium and large time horizons, however, it seems appropriate to make use of H-DMB or H-HIER, both of which are capable of computing high quality solutions. For these instances, H-DMB clearly outperforms the other approaches with respect to solution quality, which is in line with our results for the random testbed.

Summing up, we can conclude that our algorithms are well suited for daily use when schedules for time horizons of about one week must be computed at our industry partner. Furthermore, when using H-DMB and H-HIER, our industry partner will be able to make scheduling decisions for demand forecasts covering several weeks or months.

2.6 Summary

In this paper, we have introduced a worker constrained flexible job shop scheduling problem with sequence-dependent setup times that takes account of heterogeneous machine operator qualifications. We have analyzed two objective functions, minimizing the makespan and the total tardiness, and proposed to solve the problem in a branch-and-cut framework by decomposing it into a vehicle routing problem with precedence constraints (master problem) and a worker assignment problem (subproblem) that are connected via logic inequalities. In addition to this exact approach, we have presented decomposition based heuristic approaches. In an extensive computational study, we have shown that our exact decomposition approach outperforms an integrated approach and that it allows to compute benchmark solutions for assessing the performance of heuristic approaches for instances of medium size. Our heuristic approaches have shown to provide high-quality solutions within reasonable time and have proven well suited for daily use at our industry partner, who provided us with real-world data. When setting up the decomposition based heuristics, the decisions on the frequency of computing solutions of subproblems as well as on the algorithms applied to determine these solutions are of major importance for finding a suitable trade-off between runtime and solution quality.

Acknowledgements
This work has been supported by the European Union and the state North Rhine-Westphalia through the European Fund for Regional Development (EFRD). It has been conducted as part of the project "EKPLO: Echtzeitnahes kollaboratives Planen und Optimieren" (EFRE-0800463).

Bibliography

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

Allahverdi, A., Gupta, J. N. D., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239.

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

Allahverdi, A. and Soroush, H. M. (2008). The significance of reducing setup times/setup costs. *European Journal of Operational Research*, 187(3):978–984.

Bagheri, A. and Zandieh, M. (2011). Bi-criteria flexible job-shop scheduling with sequence-dependent setup times—variable neighborhood search approach. *Journal of Manufacturing Systems*, 30(1):8–15.

Balas, E., Simonetti, N., and Vazacopoulos, A. (2008). Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, 11(4):253–262.

Behnamian, J. (2014). Scheduling and worker assignment problems on hybrid flowshop with cost-related objective function. *The International Journal of Advanced Manufacturing Technology*, 74(1-4):267–283.

Bigras, L.-P., Gamache, M., and Savard, G. (2008). The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, 5(4):685–699.

Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., and Węglarz, J. (2007). *Handbook on Scheduling: From Theory to Applications*. Springer, Berlin.

Błażewicz, J., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24.

Boost (2018). The Boost Graph Library (BGL). `https://www.boost.org/doc/libs/1_67_0/libs/graph/doc/index.html`. Last accessed 2018-06-22.

Brucker, P. and Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375.

Chen, D., Luh, P. B., Thakur, L. S., and Moreno Jr, J. (2003). Optimization-based manufacturing scheduling with multiple resources, setup requirements, and transfer lots. *IIE Transactions*, 35(10):973–985.

De Bruecker, P., Van den Bergh, J., Beliën, J., and Demeulemeester, E. (2015). Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243(1):1–16.

Defersha, F. M. and Chen, M. (2010). A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. *The International Journal of Advanced Manufacturing Technology*, 49(1-4):263–279.

Della Croce, F., Grosso, A., and Salassa, F. (2014). A matheuristic approach for the two-machine total completion time flow shop problem. *Annals of Operations Research*, 213(1):67–78.

Giffler, B. and Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Günther, H.-O. and Lee, T. E. (2007). Scheduling and control of automated manufacturing systems. *OR Spectrum*, 29(3):373–374.

Haupt, R. (1989). A survey of priority rule-based scheduling. *OR Spektrum*, 11(1):3–16.

Hooker, J. N. and Ottosson, G. (2003). Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60.

Hurink, J., Jurisch, B., and Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, 15(4):205–215.

Lang, M. and Li, H. (2011). Research on dual-resource multi-objective flexible job shop scheduling under uncertainty. In *Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce*, pages 1375–1378. IEEE.

Lei, D. and Guo, X. (2014). Variable neighbourhood search for dual-resource constrained flexible job shop scheduling. *International Journal of Production Research*, 52(9):2519–2529.

Lei, D. and Tan, X. (2016). Local search with controlled deterioration for multi-objective scheduling in dual-resource constrained flexible job shop. In *Proceedings of the 28th Chinese Control and Decision Conference*, pages 4921–4926. IEEE.

Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140.

Lowerre, B. T. (1976). *The HARPY speech recognition system.* PhD thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Mastrolilli, M. and Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20.

Mousakhani, M. (2013). Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness. *International Journal of Production Research*, 51(12):3476–3487.

Nourali, S., Imanipour, N., and Shahriari, M. R. (2012). A mathematical model for integrated process planning and scheduling in flexible assembly job shop environment with sequence dependent setup times. *International Journal of Mathematical Analysis*, 6(41-44):2117–2132.

Özgüven, C., Yavuz, Y., and Özbakır, L. (2012). Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times. *Applied Mathematical Modelling*, 36(2):846–858.

Paksi, A. B. N. and Ma'ruf, A. (2016). Flexible job-shop scheduling with dual-resource constraints to minimize tardiness using genetic algorithm. In *Proceedings of the 2nd International Manufacturing Engineering Conference and 3rd Asia-Pacific Conference on Manufacturing Systems*, page 012060. IOP Publishing.

Rossi, A. (2014). Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. *International Journal of Production Economics*, 153:253–267.

Saidi-Mehrabad, M. and Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. *The International Journal of Advanced Manufacturing Technology*, 32(5-6):563–570.

Sels, V., Gheysen, N., and Vanhoucke, M. (2012). A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions. *International Journal of Production Research*, 50(15):4255–4270.

Shen, L., Dauzère-Pérès, S., and Neufeld, J. S. (2017). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265(2):503–516.

Tran, T. T. and Beck, J. C. (2012). Logic-based Benders decomposition for alternative resource scheduling with sequence dependent setups. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 774–779. ACM.

Treleven, M. (1989). A review of the dual resource constrained system research. *IIE Transactions*, 21(3):279–287.

Venditti, L., Pacciarelli, D., and Meloni, C. (2010). A tabu search algorithm for scheduling pharmaceutical packaging operations. *European Journal of Operational Research*, 202(2):538–546.

Vilcot, G. and Billaut, J.-C. (2008). A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem. *European Journal of Operational Research*, 190(2):398–411.

Xu, J., Xu, X., and Xie, S. Q. (2011). Recent developments in dual resource constrained (DRC) system research. *European Journal of Operational Research*, 215(2):309–318.

Yazdani, M., Zandieh, M., Tavakkoli-Moghaddam, R., and Jolai, F. (2015). Two meta-heuristic algorithms for the dual-resource constrained flexible job-shop scheduling problem. *Scientia Iranica - Transactions E*, 22(3):1242–1257.

Zhang, J. and Liu, G. B. (2012). Hybrid ant colony algorithm for job shop schedule with unrelated parallel machines. In *Proceedings of the Conference on Frontiers of Advanced Materials and Engineering Technology*, pages 905–908. Trans Tech Publications.

Zhang, J., Wang, W., and Xu, X. (2015). A hybrid discrete particle swarm optimization for dual-resource constrained job shop scheduling with resource flexibility. *Journal of Intelligent Manufacturing*, 28(8):1961–1972.

Zheng, X.-L. and Wang, L. (2016). A knowledge-guided fruit fly optimization algorithm for dual resource constrained flexible job-shop scheduling problem. *International Journal of Production Research*, 54(18):5554–5566.

Chapter 3

Mathematical Models for a Flexible Job Shop Scheduling Problem with Machine Operator Constraints

Abstract

We consider a flexible job shop scheduling problem that incorporates machine operators and aims at makespan minimization. In a detailed overview of the related literature, we reveal the fact that the research in this field is mainly concerned with (meta-)heuristic approaches. Only few papers consider exact approaches. In order to promote the use of exact approaches and in order to facilitate the evaluation of the performance of heuristic approaches, we present two mathematical models, a mixed-integer programming model and a constraint programming model, that are analyzed and compared with a state-of-the-art heuristic in computational tests with a standard solver.

Authors	Dominik Kress[a,b] dominik.kress@hsu-hh.de David Müller[b] david.mueller@uni-siegen.de [a] *University of Siegen, Management Information Science, Kohlbettstraße 15, 57068 Siegen, Germany* [b] *Helmut Schmidt University - University of the Federal Armed Forces Hamburg, Business Administration, especially Procurement and Production, Friedrich-Ebert-Damm 245, 22159 Hamburg, Germany*
Publication details	*IFAC-PapersOnLine*, Volume 52(13), 2019, Pages 94–99
Full citation	Kress, D. and Müller, D. (2019). Mathematical models for a flexible job shop scheduling problem with machine operator constraints. *IFAC-PapersOnLine*, 52(13): 94–99.

This is an Accepted Manuscript of an article published by Elsevier in IFAC-PapersOnLine on 25 December 2019, available online: `https://doi.org/10.1016/j.ifacol.2019.11.144`.

3.1 Introduction

The *job shop scheduling problem* (JSP) is a well-known scheduling setting that arises in many traditional manufacturing systems (see, e.g., Błażewicz et al., 2007). It assumes that each job consists of a set of operations, each of which must be processed on a specific machine. Modern flexible manufacturing systems, however, oftentimes feature multi-purpose machines that allow for processing different types of manufacturing operations, so that operations are typically associated to a set of *eligible machines*. This is taken account of in the *flexible job shop scheduling problem* (FJSP) (Brucker and Schlie, 1990). A recent survey of solution approaches for the FJSP is presented by Chaudhry and Khan (2016). Usually, a machine needs to be operated by some *machine operator*, hereafter referred to as a *worker*. We will denote FJSP settings that explicitly incorporate workers by WFJSPs. Note, however, that machine scheduling problems that are concerned with two types of resources, e.g., machines and workers, are sometimes also referred to as dual-resource constrained (DRC) systems (see, e.g., Treleven, 1989; Xu et al., 2011).

3.1.1 Literature Overview

Most of the research on WFJSPs is concerned with the objective of minimizing the makespan. In this stream of publications, Xianzhou and Zhenhe (2011) combine a genetic algorithm with an immune algorithm. Lei and Guo (2014) introduce a variable neighborhood search. A mixed-integer programming (MIP) model and two metaheuristic approaches (simulated annealing and vibration damping optimization) are proposed by Yazdani et al. (2015). Zhang et al. (2015) introduce a hybrid discrete particle swarm optimization algorithm by incorporating a simulated annealing approach with a variable neighborhood structure. Zheng and Wang (2016) develop a knowledge-guided fruit fly optimization algorithm. Peng et al. (2018) propose a genetic algorithm. Vallikavungal Devassia et al. (2018) consider a WFJSP that incorporates recovery times for the resources. For this problem, a general variable neighborhood search is introduced. A further special case of the WFJSP is addressed by Wu et al. (2018). The authors assume that workers have the ability to learn, and thus incorporate learning effects into the WFJSP. They propose a hybrid genetic algorithm which combines a genetic algorithm with a variable neighborhood search. Paksi and Ma'ruf (2016) analyze the objective of minimizing the total tardiness and introduce a genetic algorithm. Kress et al. (2019) address a WFJSP that takes account of sequence-dependent setup times. They analyze two objectives, minimizing the makespan and minimizing the total tardiness, present an integrated MIP model, and propose exact and heuristic decomposition based solution approaches. A few researchers address multiple objectives for WFJSPs. Lang and Li (2011) address a multi-objective WFJSP, where delivery satisfaction, process cost, energy consumption and noise pollution are to be minimized. The authors also consider uncertain processing times and introduce a heuristic which combines simulation and a genetic algorithm. Liu et al. (2011) and Zhang et al. (2013) study a bi-criteria WFJSP, where both the makespan and the production cost are to be minimized. The former authors propose a hybrid genetic algorithm based on a Pareto approach, while a hybrid discrete particle swarm op-

timization algorithm is introduced by the latter authors. Lei and Tan (2016) address a WFJSP to simultaneously minimize the makespan and total tardiness. For this problem, the authors propose a local search approach with controlled deterioration. Gong et al. (2018a) consider green production indicators in a WFJSP and propose a hybrid genetic algorithm to minimize the makespan, the total worker cost and green production factors. Gong et al. (2018b) address a multi-objective WFJSP, where the makespan, the maximum workload of machines and the total workload of all machines are to be minimized. The authors propose a memetic algorithm.

Table 3.1 summarizes the above literature overview.

3.1.2 Contribution and Overview

Based on the above literature overview, we conclude that mainly metaheuristic approaches have been considered for solving WFJSPs. With respect to mathematical models, some researches introduce MIP or NLP formulations. However, only very few (Kress et al., 2019; Vallikavungal Devassia et al., 2018; Yazdani et al., 2015) actually make use of their models to design exact approaches or to evaluate their heuristic approaches in computational tests.

Recently, commercial *constraint programming* (CP) optimizers have shown to perform remarkably well for solving scheduling problems. Puget (2013), for instance, presents results of the IBM ILOG CPLEX CP Optimizer for solving the FJSP for 7 well-known instance sets from the literature. Interestingly, to the best of the authors' knowledge, there has been no attempt to provide a CP model as a benchmark for a WFJSP. In this paper, we therefore introduce two models, a MIP model and a CP model, for a WFJSP that aims to minimize the makespan. We evaluate the performance of both models by using CPLEX. Furthermore, we compare the performance of CPLEX on the CP model with a recently proposed heuristic approach. For an introduction to CP, we refer to Rossi et al. (2006).

The remainder of this paper is organized as follows. In Section 3.2, we provide a formal definition of the WFJSP considered in this paper. The MIP model and the CP model are presented in Sections 3.3 and 3.4, respectively. The computational tests are subject of Section 3.5. A conclusion and future research directions are provided in Section 3.6.

3.2 Problem Description

The WFJSP under consideration (hereafter referred to as *the* WFJSP for the sake of simplicity) is composed of a set I of jobs, a set M of machines and a set W of workers. We assume that all jobs, machines and workers are available at time zero. For each job $i \in I$, we are given an ordered set of q_i operations $O_i = (i_1, \ldots, i_{q_i})$. The ordering is such that, for any pair of operations $i_j, i_k \in O_i$ with $j < k$, i_k can only start to be processed after the processing of i_j has finished. Each operation $i_j \in O_i$, $i \in I$, must be processed without preemption on exactly one machine out of a set of eligible machines $M_{i_j} \subseteq M$. We define $M_{i_j,k_l} := M_{i_j} \cap M_{k_l}$ for all $i, k \in I$, $i_j \in O_i$, $k_l \in O_k$. An operation $i_j \in O_i$ of a job $i \in I$ can only be processed on a machine, if exactly one worker of the set W is assigned to the operation for the entire processing

Table 3.1: Overview of literature concerned with WFJSPs

Publication	Objective	Math. model[a]	Approach[b]	Special characteristics
Gong et al. (2018a)	Makespan, total worker cost and green production factors	NLP	HGA	Multiple objectives
Gong et al. (2018b)	Makespan, maximum workload of machines and total workload of all machines	NLP	MEM	Multiple objectives
Kress et al. (2019)	Makespan; Total tardiness	MIP	MIPS, DEC	Incorporation of sequence-dependent setup times
Lang and Li (2011)	Delivery satisfaction, process cost, energy consumption and noise pollution	-	GA	Multiple objectives, Uncertain processing times
Lei and Guo (2014)	Makespan	-	VNS	
Lei and Tan (2016)	Makespan and total tardiness	-	LS	Multiple objectives
Liu et al. (2011)	Makespan and production cost	-	HGA	Multiple objectives
Paksi and Ma'ruf (2016)	Total tardiness	-	GA	
Peng et al. (2018)	Makespan	MIP	GA	
Vallikavungal Devassia et al. (2018)	Makespan	MIP	MIPS, VNS	Consideration of resource recovery constraints
Wu et al. (2018)	Makespan	NLP	HGA	Consideration of learning effects for the workers
Xianzhou and Zhenhe (2011)	Makespan	-	GA	
Yazdani et al. (2015)	Makespan	MIP	MIPS, SA, VDO	
Zhang et al. (2013)	Makespan and production cost	NLP	HPSO	Multiple objectives
Zhang et al. (2015)	Makespan	NLP	HPSO	
Zheng and Wang (2016)	Makespan	MIP	KF	

[a] The mathematical models are abbreviated as follows: MIP = Mixed-integer programming model; NLP = Non-linear programming model.

[b] The approaches are abbreviated as follows: DEC = Decomposition-based approach; KF = Knowledge-guided fruit fly optimization; (H)GA = (Hybrid) genetic algorithm; LS = Local search; MEM = Memetic algorithm; MIPS = Standard MIP solver; (H)PSO = (Hybrid) particle swarm optimization; SA = Simulated annealing; VNS = Variable neighborhood search; VDO = Vibration damping optimization

time. We assume a heterogenous shop floor as well as workforce, so that the processing times of an operation may vary for different machines and workers. Therefore, we denote the processing time of an operation $i_j \in O_i$ of a job $i \in I$ that is processed on an eligible machine $m \in M_{i_j}$ by a worker $w \in W$ by $p_{i_j}^{m,w} \in \mathbb{N}^+$. If some worker must not process an operation of a job on an eligible machine, the corresponding processing time is set to infinity. Each machine and each worker can process at most one operation at a time. We denote the completion time of an operation $i_j \in O_i$ of job $i \in I$ by C_{i_j} and the completion time of job $i \in I$ by C_i. A job $i \in I$ is completed if all of its operations are completed, i.e., $C_i = C_{i_{q_i}}$.

The problem is to allocate the operations to eligible machines and workers and to determine corresponding feasible sequences of the operations, such that the makespan $C_{max} := \max_{i \in I} C_i$ is minimized. Based on the results of Lenstra and Rinnooy Kan (1979), it can easily be seen that this problem is strongly NP-hard.

3.3 Mixed-Integer Programming Model

Kress et al. (2019) propose a MIP model for a WFJSP with sequence-dependent setup times based on modelling approaches for the vehicle routing problem. We adjust this model to the simplified setting considered in this paper. To do so, we define a dummy job 0 with exactly one operation 0_1 and $M_{0_1} = M$. We set $M_{0_1,i_j} = M_{i_j,0_1} = M_{i_j}$ for all $i \in I$ and $i_j \in O_i$. Moreover, as in Kress et al. (2019), we define the following sets:

- $V := \bigcup_{i \in I} O_i \cup \{0_1\}$
- $\bar{V} := V \setminus \{0_1\}$
- $V_{i_j} := V \setminus \{i_k | k \leq j\}$ for all $i_j \in V$
- $\tilde{V}_{i_j} := V \setminus \{i_k | k \geq j\}$ for all $i_j \in V$
- $\bar{V}_{i_j} := V_{i_j} \setminus \{0_1\}$ for all $i_j \in V$

We furthermore define $p_{i_j}^{m,min} := \min_{w \in W} p_{i_j}^{m,w}$ for all $i \in I$, $i_j \in O_i$ and $m \in M_{i_j}$, and set $p_{i_j}^{min} := \min_{m \in M_{i_j}} p_{i_j}^{m,min}$ for all $i \in I$, $i_j \in O_i$.

Now, for all operations $i_j \in \bar{V}$, we define a continuous variable $t_{i_j} \in \mathbb{N}_0^+$ that represents the point in time at which operation i_j is started to be processed, a continuous variable $C_{max} \in \mathbb{N}_0^+$ that represents the makespan, as well as the following binary variables:

$$y_{i_j,k_l}^m := \begin{cases} 1, & \text{if } k_l \text{ is processed on } m \text{ directly after } i_j \\ 0, & \text{else} \end{cases} \qquad \forall\, i_j \in V, k_l \in V_{i_j}, m \in M_{i_j,k_l}, \quad (3.1)$$

$$x_{i_j}^{m,w} := \begin{cases} 1, & \text{if } i_j \text{ is processed by } w \text{ on } m \\ 0, & \text{else} \end{cases} \qquad \forall\, i_j \in \bar{V}, m \in M_{i_j}, w \in W, \quad (3.2)$$

$$x^w_{i_j,k_l} := \begin{cases} 1, & \text{if } i_j \text{ and } k_l \text{ are processed by } w \\ 0, & \text{else} \end{cases} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, w \in W, \tag{3.3}$$

$$v_{i_j,k_l} := \begin{cases} 1, & \text{if the processing of } k_l \text{ starts before the} \\ & \text{processing of } i_j \text{ finishes} \\ 0, & \text{else} \end{cases} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l. \tag{3.4}$$

Let B be a large positive number. Then, a MIP model for WFJSP is as follows.

$$\min C_{max} \tag{3.5}$$

s.t.

$$t_{i_{q_i}} + \sum_{m \in M_{i_{q_i}}} \sum_{w \in W} x^{m,w}_{i_{q_i}} \cdot p^{m,w}_{i_{q_i}} \leq C_{max} \qquad \forall\, i \in I, \tag{3.6}$$

$$\sum_{i_j \in \tilde{V}_{k_l}} \sum_{m \in M_{i_j,k_l}} y^m_{i_j,k_l} = 1 \qquad \forall\, k_l \in \bar{V}, \tag{3.7}$$

$$\sum_{i_j \in \bar{V}} y^m_{0_1,i_j} \leq 1 \qquad \forall\, m \in M, \tag{3.8}$$

$$\sum_{k_l \in \tilde{V}_{i_j} \text{ with } m \in M_{k_l}} y^m_{k_l,i_j} - \sum_{k_l \in V_{i_j} \text{ with } m \in M_{k_l}} y^m_{i_j,k_l} = 0 \qquad \forall\, i_j \in V, m \in M_{i_j}, \tag{3.9}$$

$$t_{i_j} + \sum_{w \in W} x^{m,w}_{i_j} \cdot p^{m,w}_{i_j} - t_{k_l} \leq \left(1 - y^m_{i_j,k_l}\right) B \qquad \forall\, i_j \in \bar{V}, k_l \in \bar{V}_{i_j}, m \in M_{i_j,k_l}, \tag{3.10}$$

$$t_{i_j} + \sum_{m \in M_{i_j}} \sum_{w \in W} x^{m,w}_{i_j} \cdot p^{m,w}_{i_j} \leq t_{i_{j+1}} \qquad \forall\, i_j \in \bar{V} \text{with } j \leq q_i - 1, \tag{3.11}$$

$$t_{i_j} + p^{min}_{i_j} \leq t_{i_{j+1}} \qquad \forall\, i_j \in \bar{V} \text{with } j \leq q_i - 1, \tag{3.12}$$

$$\sum_{m \in M_{i_j}} x^{m,w}_{i_j} + \sum_{m \in M_{k_l}} x^{m,w}_{k_l} - 1 \leq x^w_{i_j,k_l} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, w \in W, \tag{3.13}$$

$$t_{i_j} + \sum_{m \in M_{i_j}} \sum_{w \in W} x^{m,w}_{i_j} \cdot p^{m,w}_{i_j} - t_{k_l} \leq B \cdot v_{i_j,k_l} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, \tag{3.14}$$

$$x^w_{i_j,k_l} \leq 2 - v_{i_j,k_l} - v_{k_l,i_j} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, w \in W, \tag{3.15}$$

$$\sum_{i_j \in \tilde{V}_{k_l} \text{ with } m \in M_{i_j}} y^m_{i_j,k_l} = \sum_{w \in W} x^{m,w}_{k_l} \qquad \forall\, k_l \in \bar{V}, m \in M_{k_l}, \tag{3.16}$$

$$y^m_{i_j,k_l} \in \{0,1\} \qquad \forall\, i_j \in V, k_l \in V_{i_j}, m \in M_{i_j,k_l}, \tag{3.17}$$

$$x^{m,w}_{i_j} \in \{0,1\} \qquad \forall\, i_j \in \bar{V}, m \in M_{i_j}, w \in W, \tag{3.18}$$

$$x^w_{i_j,k_l} \in \{0,1\} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, w \in W, \tag{3.19}$$

$$v_{i_j,k_l} \in \{0,1\} \qquad \forall\, i_j, k_l \in \bar{V}, i_j \neq k_l, \tag{3.20}$$

$$t_{i_j} \in \mathbb{N}^+_0 \qquad \forall\, i_j \in \bar{V}, \tag{3.21}$$

$$C_{max} \in \mathbb{N}^+_0. \tag{3.22}$$

The objective (3.5) minimizes the makespan, which is bounded from below by constraints (3.6). Constraints (3.7) ensure that each operation is assigned to exactly one of its eligible machines, while inequalities (3.8) guarantee that there is at most one operation that is processed first on each machine. Constraints (3.9) ensure that the variables (3.1) are set to their correct values with respect to the specific machine that processes an operation $i_j \in V$ as well as its preceding and succeeding operations (including the operation of the dummy job) on this very machine. Constraints (3.10) prevent an overlapping of two consecutive operations $i_j \in O_i$ of job $i \in I$ and $k_l \in O_k$ of job $k \in I$ processed on the same machine $m \in M_{i_j,k_l}$. The precedence relations between consecutive operations $i_j, i_{j+1} \in O_i$, $i \in I$, are taken account of in constraints (3.11). Constraints (3.12) are redundant to constraints (3.11), but have shown to improve the computational performance in our tests. Constraints (3.13) and (3.14) ensure that the variables (3.3) and (3.4) are set to one when needed. Based on these variables, constraints (3.15) prevent overlapping of operations that are assigned to the same worker. Constraints (3.16) connect the sequencing variables (3.1) with the worker variables (3.2). Finally, constraints (3.17)–(3.22) define the domains of the variables.

3.4 Constraint Programming Model

The IBM ILOG CPLEX CP Optimizer provides a CP engine which enables the modelling and solving of scheduling problems. In order to cover temporal dimensions, the optimizer provides *interval variables* and *sequence variables*. The former variables are used to model the start and the end of the processing of the operations, while the latter variables are used to represent the sequencing decisions, i.e., orderings of interval variables. Moreover, the optimizer provides several special constraint types, which we assume the reader to be familiar with for the sake of brevity. Details are given in Laborie et al. (2018), as well as in the online documentation (IBM, 2016a,b) that includes detailed examples that make use of IBM's Optimization Programming Language (OPL).

For WFJSP, we define interval and sequencing variables as illustrated in Table 3.2.

Table 3.2: Variables for the CP model

Variables	Definition
$I_{OP\{i_j\}}$	Interval variable for each operation, i.e., for all $i \in I$, $i_j \in O_i$
$I_{MO\{i_j,m,w,p\}}$	Interval variable for each *processing mode*, i.e., each eligible combination of an operation $i_j \in O_i$ of job $i \in I$, a machine, a worker and a (finite) processing time
$S_{\bar{m}}$	Sequence variable for each machine $\bar{m} \in M$; related to all interval variables $I_{MO\{i_j,m,w,p\}}$ with $m = \bar{m}$
$S_{\bar{w}}$	Sequence variable for each worker $\bar{w} \in W$; related to all interval variables $I_{MO\{i_j,m,w,p\}}$ with $w = \bar{w}$

Using the structures and notation provided by IBM's CP Optimizer, a compact formulation of the objective and the special constraint types for WFJSP is as follows.

$$\min \max_{i \in I}(endOf(I_{OP\{i_{q_i}\}})) \tag{3.23}$$

s.t.

$$endBeforeStart(I_{OP\{i_j\}}, I_{OP\{i_{j+1}\}}) \quad \forall\, i \in I, j \leq q_i - 1, \tag{3.24}$$

$$alternative(I_{OP\{i_j\}}, all\, I_{MO\{i_j,m,w,p\}}) \quad \forall\, i \in I, i_j \in O_i, \tag{3.25}$$

$$noOverlap(S_m) \quad \forall\, m \in M, \tag{3.26}$$

$$noOverlap(S_w) \quad \forall\, w \in W. \tag{3.27}$$

The objective function (3.23) represents the minimization of the makespan. Constraints (3.24) capture the precedence constraints among the operations of the jobs. Constraints (3.25) guarantee that an eligible processing mode is chosen for each operation. Constraints (3.26) and (3.27) ensure that each machine and each worker processes at most one operation at a time.

3.5 Computational Study

In order to compare the performance of CPLEX on the MIP model and the CP model, we performed computational tests on a PC with an Intel® Core™ i7-4770 CPU, running at 3.4 GHz, with 16 GB of RAM under a 64-bit version of Windows 8. The models were implemented in Java using Eclipse (Eclipse IDE for Java Developers (Oxygen 4.7)), where OPL was applied as a modelling language for the CP model. We used the MIP and CP solvers of IBM ILOG CPLEX in version 12.7 and the Java Runtime Environment (JRE) in version 1.8.0_191. If not stated otherwise, the time limit for each call of the CPLEX solvers was set to 3,600 seconds.

Our random testbed is composed of 16 instances sets, denoted by R1–R16 and summarized in Table 3.3. Each set features 10 instances with the number of jobs $|I|$, machines $|M|$, workers

Table 3.3: Random testbed

Set	$\|I\|$	$\|M\|$	$\|W\|$	q_i	$\|M_{i_j}\|$	$\|W_m\|$	Set	$\|I\|$	$\|M\|$	$\|W\|$	q_i	$\|M_{i_j}\|$	$\|W_m\|$
R1	3	2	2	[2, 4]	[1, 2]	2	R9	7	3	3	[2, 4]	[1, 2]	2
R2	3	2	2	[4, 8]	[1, 2]	2	R10	7	3	3	[4, 8]	[1, 2]	3
R3	5	2	2	[2, 4]	[1, 2]	2	R11	10	5	5	[4, 8]	[1, 3]	3
R4	5	2	2	[4, 8]	[1, 2]	2	R12	10	5	5	[2, 10]	[1, 3]	3
R5	5	3	3	[2, 4]	[1, 2]	2	R13	10	5	5	[5, 10]	[2, 3]	3
R6	5	3	3	[4, 8]	[1, 2]	3	R14	20	10	10	[5, 10]	[1, 3]	5
R7	7	2	2	[2, 4]	[1, 2]	2	R15	20	10	10	[5, 15]	[2, 3]	5
R8	7	2	2	[4, 8]	[1, 2]	2	R16	20	10	10	[10, 15]	[2, 3]	5

$|W|$, as well as the number of workers that can operate each machine (denoted by $|W_m|$, where W_m refers to the actual set of workers that is determined randomly) being fixed. The testbed was generated randomly. For each instance, the number of operations q_i, $i \in I$, and the number of eligible machines $|M_{i_j}|$ for operations $i_j \in O_i$, $i \in I$, were drawn from uniform distributions over the intervals given in Table 3.3. The process of generating the processing times of the operations was as follows (cf. Kress et al., 2019). First, auxiliary integer parameters p_{i_j} were drawn from a uniform distribution over $[10, 100]$ for all $i \in I$ and $i_j \in O_i$. Based on these parameters, we generated varying processing times over the corresponding eligible machines $m \in M_{i_j}$ by drawing auxiliary integer parameters $p_{i_j}^m$ from uniform distributions over the interval $[\lfloor 0.9 \cdot p_{i_j} \rfloor, \lfloor 1.1 \cdot p_{i_j} \rfloor]$.

Finally, we incorporated dependencies on workers $w \in W$ by drawing integer parameters $p_{i_j}^{m,w}$, $m \in M_w \cap M_{i_j}$, from uniform distributions over $[\lfloor 0.9 \cdot p_{i_j}^m \rfloor, \lfloor 1.1 \cdot p_{i_j}^m \rfloor]$. Here, M_w defines the set of machines that can be operated by worker $w \in W$. It can easily be constructed based on the sets W_m, $m \in M$.

In order to evaluate our results, we use a lower bound on the makespan introduced by Lei and Guo (2014), which we simplify to take account of the facts that all jobs are available at time zero and that $|M| \leq |I|$ and $|W| \leq |I|$ for all considered instances. For a given instance of WFJSP, the bound is defined as follows:

$$LB := \max \left(\max_{i \in I} \left(\sum_{i_j \in O_i} p_{i_j}^{min} \right), \left\lceil \frac{P}{|M|} \right\rceil, \left\lceil \frac{P}{|W|} \right\rceil \right).$$

Here, $P := \sum_{i \in I} \sum_{i_j \in O_i} p_{i_j}^{min}$. Note that, for the sake of brevity, we do not explicitly state the concrete instance in the definition of the bound.

Let $Inst$ be a given instance of WFJSP and denote by $C_{max}^{model}(Inst)$ the (not necessarily optimal) makespan returned by CPLEX for the CP or the MIP model ($model \in \{CP, MIP\}$) within the time limit. Then, we define the quality ratio

$$Q^{model}(Inst) := 100 \cdot \frac{C_{max}^{model}(Inst) - LB}{LB}$$

as a measure for the quality of the corresponding solution.

Table 3.4 presents the computational results for the instances of our random testbed. For each instance set, the table presents information about the average lower bound (column LB_{avg}), the number of test instances for which a feasible and optimal solution was obtained within the given time limit (columns "feas." and "opt."), the average quality ratios (columns "Q_{avg}^{model}", $model \in \{CP, MIP\}$) and the average runtimes (columns "t_{avg}"). Entries "tl" denote cases in which the time limit was reached for all instances of the set.

It can be seen that the CP model clearly outperforms the MIP model in its ability to determine feasible solutions. In fact, it returned a feasible solution for all considered instances. Moreover, CP is the clear winner with respect to determining optimal solutions, average quality ratios and runtimes for the small instances in the sets R1–R10. For the sets of large instances (R11–R16), only one instance could be solved to optimality by using the CP model.

In order to be able to analyze the influence of a varying *staffing level* (SL), defined as the ratio of the number of workers and the number of machines, on the performance of the CP solver for large instances, we ran additional tests. Here, we used modified instance sets R11–R16, where $|W|$ was decreased before generating the instances to achieve staffing levels of 60% and 80%. The corresponding results are illustrated in Figure 3.1.

We observe that the average quality ratios decrease for smaller staffing levels.

The above results indicate that heuristic approaches will need to be evaluated and compared against CP approaches. As an example, we will now analyze one of the most recent metaheuristic

Table 3.4: Performance of MIP and CP models on random testbed

		MIP				CP			
Set	LB_{avg}	feas.	opt.	Q_{avg}^{MIP}	t_{avg} [s]	feas.	opt.	Q_{avg}^{CP}	t_{avg} [s]
R1	204.2	10	10	9.48	0.21	10	10	9.48	0.07
R2	453.2	10	9	7.57	484.23	10	10	7.57	2.77
R3	344.7	10	7	3.84	1415.75	10	10	3.74	5.05
R4	712.2	10	0	6.26	tl	10	3	2.72	2837.9
R5	233.5	10	10	21.26	21.32	10	10	21.26	0.68
R6	473.4	10	1	11.42	3308.13	10	8	7.53	1348.27
R7	485.1	10	0	3.65	tl	10	7	1.93	1328.93
R8	1029.1	3	0	11.32	tl	10	0	2	tl
R9	323.4	10	1	14.19	3264.19	10	9	13.05	366.9
R10	668.2	8	0	23.72	tl	10	0	6.48	tl
R11	563.6	4	0	55.42	tl	10	0	11.01	tl
R12	570.6	1	0	40.63	tl	10	1	12.72	3260.93
R13	664.1	0	0	-	-	10	0	11.06	tl
R14	669.5	0	0	-	-	10	0	18.55	tl
R15	861.8	0	0	-	-	10	0	20.5	tl
R16	1075	0	0	-	-	10	0	21.28	tl

approaches, i.e., the knowledge-guided fruit fly optimization algorithm (denoted by KF) by Zheng and Wang (2016), that the authors find to be "more effective than the existing algorithms." The study of Zheng and Wang (2016) is based on two sets of FJSP benchmark instances from the literature, MK1–M10 (Brandimarte, 1993) and DP1–DP12 (Dauzère-Pérès and Paulli, 1997), that have been adapted to include worker information by Lei and Guo (2014) by providing the sets W and M_w. Unfortunately, the generation of the processing times is not clearly described in Lei and Guo (2014), so that (based on the information given by Lei and Guo, 2014) we propose to draw the processing times $p_{i_j}^{m,w}$, $m \in M_w \cap M_{i_j}$, from a uniform distribution over the interval $[\bar{p}_{i_j}^m, \bar{p}_{i_j}^m + \delta_{i_j}]$ for all $w \in W$, $i \in I$ and $i_j \in O_i$. Here, $\bar{p}_{i_j}^m$ is the processing time stated for the literature instances, and δ_{i_j} is drawn from a uniform distribution over the interval $[2, 8]$.

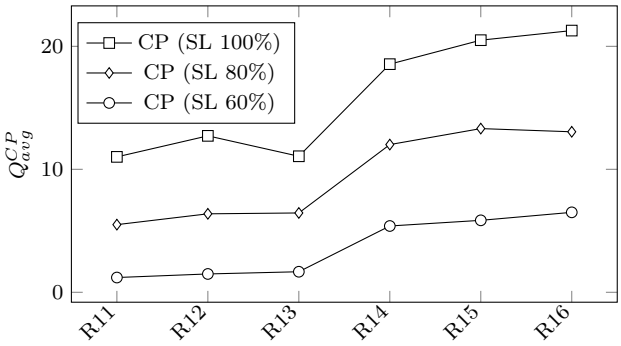


Figure 3.1: Staffing level impact (CP model)

For the sake of comparability, we set the time limit for the CP solver to the average (as the authors initiate multiple runs of their algorithm) runtime of the KF approach as stated in Zheng and Wang (2016) for each instance. Moreover, we deactivated the parallel processing mode in the CPLEX CP Optimizer. The computational results are presented in Table 3.5. For each instance, the table presents information about the lower bound (column LB), the quality ratio

Table 3.5: Performance of CP model for literature instances

Inst.	LB	Q^{CP}	Q^{KF}_{avg}	t^{KF}_{avg} $[s]$	Inst.	LB	Q^{CP}	Q^{KF}_{avg}	t^{KF}_{avg} $[s]$
MK1	66	19.7	8.2	3.14	DP1	2881	6.77	11.18	35.23
MK2	65	20	7.69	3.23	DP2	2881	1.8	9.94	35.33
MK3	182	39.01	41	13.44	DP3	2881	1.7	10.65	35.64
MK4	80	35	26.28	5.28	DP4	2862	5.45	13.11	35.78
MK5	295	6.78	9.21	15.11	DP5	2832	2.72	11.43	35.12
MK6	78	62.82	36.9	11.23	DP6	2799	3.93	10.19	34.44
MK7	213	12.68	13.38	10.47	DP7	2843	4.82	27.01	52.31
MK8	488	28.48	19.03	59.22	DP8	2835	1.83	25.22	52.12
MK9	443	20.54	28.16	52.86	DP9	2824	2.51	27.32	52.35
MK10	289	20.42	37.1	49.43	DP10	2840	5.53	26.87	52.56
					DP11	2786	4.45	31.21	52.98
					DP12	2723	5.77	30.09	53.15

resulting from the CP solver (column "Q^{CP}") and the average quality ratio (column "Q^{KF}_{avg}") as well as the average runtime (column "t^{KF}_{avg}") of the KF approach as stated by Zheng and Wang (2016) for the corresponding similar instance. The results indicate that the CP solver tends to outperform KF for most instances. This effect is particularly pronounced for the DP instances.

3.6 Conclusion and Future Research

In this paper, we have addressed a flexible job shop scheduling problem that aims to minimize the makespan and takes account of machine operators with differing skills. We have provided an overview of the related research and have presented a MIP model and a CP model that we have then compared by using the standard solvers provided by CPLEX. We found that the CP solver clearly outperforms the MIP solver for the considered modelling approaches. The CP solver tends to provide high quality solutions within reasonable time. It was especially interesting to see that it also tends to outperform a state-of-the-art metaheuristic approach. For future research, one will therefore have to provide (meta-)heuristics and exact approaches that prove to be competitive when compared with the use of standard CP solvers. Moreover, detailed benchmark sets will need to be published so that fair comparisons become possible.

Acknowledgements
This work has been supported by the European Union and the state North Rhine-Westphalia through the European Fund for Regional Development (EFRD). It has been conducted as part of the project "EKPLO: Echtzeitnahes kollaboratives Planen und Optimieren" (EFRE-0800463).

Bibliography

Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., and Węglarz, J. (2007). *Handbook on Scheduling: From Theory to Applications*. Springer, Berlin.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183.

Brucker, P. and Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375.

Chaudhry, I. A. and Khan, A. A. (2016). A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3):551–591.

Dauzère-Pérès, S. and Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306.

Gong, G., Deng, Q., Gong, X., Liu, W., and Ren, Q. (2018a). A new double flexible job-shop scheduling problem integrating processing time, green production, and human factor indicators. *Journal of Cleaner Production*, 174:560–576.

Gong, X., Deng, Q., Gong, G., Liu, W., and Ren, Q. (2018b). A memetic algorithm for multi-objective flexible job-shop problem with worker flexibility. *International Journal of Production Research*, 56(7):2506–2522.

IBM (2016a). IBM ILOG CPLEX optimization studio 12.7.0: Online documentation. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html`. Last accessed 2018-12-11.

IBM (2016b). IBM ILOG CPLEX optimization studio 12.7.0: Scheduling examples. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.ide.help/OPL_Studio/usroplexamples/topics/opl_cp_examples_scheduling.html`. Last accessed 2018-12-11.

Kress, D., Müller, D., and Nossack, J. (2019). A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, 41(1):179–217.

Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250.

Lang, M. and Li, H. (2011). Research on dual-resource multi-objective flexible job shop scheduling under uncertainty. In *Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce*, pages 1375–1378. IEEE.

Lei, D. and Guo, X. (2014). Variable neighbourhood search for dual-resource constrained flexible job shop scheduling. *International Journal of Production Research*, 52(9):2519–2529.

Lei, D. and Tan, X. (2016). Local search with controlled deterioration for multi-objective scheduling in dual-resource constrained flexible job shop. In *Proceedings of the 28th Chinese Control and Decision Conference*, pages 4921–4926. IEEE.

Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140.

Liu, X. X., Liu, C. B., and Tao, Z. (2011). Research on bi-objective scheduling of dual-resource constrained flexible job shop. *Advanced Materials Research*, 211–212:1091–1095.

Paksi, A. B. N. and Ma'ruf, A. (2016). Flexible job-shop scheduling with dual-resource constraints to minimize tardiness using genetic algorithm. In *Proceedings of the 2nd International Manufacturing Engineering Conference and 3rd Asia-Pacific Conference on Manufacturing Systems*, page 012060. IOP Publishing.

Peng, C., Fang, Y., Lou, P., and Yan, J. (2018). Analysis of double-resource flexible job shop scheduling problem based on genetic algorithm. In *Proceedings of the 15th International Conference on Networking, Sensing and Control*, pages 1–6. IEEE.

Puget, J. F. (2013). Solving flexible job shop scheduling problems. `https://www.ibm.com/developerworks/community/blogs/jfp/entry/solving_flexible_job_shop_scheduling_problems?lang=en`. Last accessed 2019-07-26.

Rossi, F., van Beek, P., and Walsh, T., editors (2006). *Handbook of Constraint Programming*. Elsevier, Amsterdam.

Treleven, M. (1989). A review of the dual resource constrained system research. *IIE Transactions*, 21(3):279–287.

Vallikavungal Devassia, J., Salazar-Aguilar, M. A., and Boyer, V. (2018). Flexible job-shop scheduling problem with resource recovery constraints. *International Journal of Production Research*, 56(9):3326–3343.

Wu, R., Li, Y., Guo, S., and Xu, W. (2018). Solving the dual-resource constrained flexible job shop scheduling problem with learning effect by a hybrid genetic algorithm. *Advances in Mechanical Engineering*, 10(10):1–14.

Xianzhou, C. and Zhenhe, Y. (2011). An improved genetic algorithm for dual-resource constrained flexible job shop scheduling. In *Proceedings of the 4th International Conference on Intelligent Computation Technology and Automation*, pages 42–45. IEEE.

Xu, J., Xu, X., and Xie, S. Q. (2011). Recent developments in dual resource constrained (DRC) system research. *European Journal of Operational Research*, 215(2):309–318.

Yazdani, M., Zandieh, M., Tavakkoli-Moghaddam, R., and Jolai, F. (2015). Two meta-heuristic algorithms for the dual-resource constrained flexible job-shop scheduling problem. *Scientia Iranica - Transactions E*, 22(3):1242–1257.

Zhang, J., Wang, W., and Xu, X. (2015). A hybrid discrete particle swarm optimization for dual-resource constrained job shop scheduling with resource flexibility. *Journal of Intelligent Manufacturing*, 28(8):1961–1972.

Zhang, J., Wang, W., Xu, X., and Jie, J. (2013). A multi-objective particle swarm optimization for dual-resource constrained shop scheduling with resource flexibility. In *Proceedings of the IEEE Symposium on Computational Intelligence for Engineering Solutions*, pages 29–34. IEEE.

Zheng, X.-L. and Wang, L. (2016). A knowledge-guided fruit fly optimization algorithm for dual resource constrained flexible job-shop scheduling problem. *International Journal of Production Research*, 54(18):5554–5566.

Chapter 4

Filter-and-Fan Approaches for Scheduling Flexible Job Shops under Workforce Constraints

Abstract

This paper addresses a flexible job shop scheduling problem that takes account of heterogeneous machine operators and aims to minimize the makespan. We propose to decompose the problem in order to be able to adapt a neighborhood structure that has formerly shown to perform well when machine operator restrictions are not considered. Based on this adaption, we develop different variants of filter-and-fan based heuristic solution approaches. These methods combine a local search procedure with a tree search procedure. The former procedure is used to obtain local optima, while the latter procedure generates compound transitions in order to explore larger neighborhoods to overcome these locally optimal solutions. In a computational study, we show that our solution approaches tend to outperform existing metaheuristics and that they are competitive when compared with the use of a standard constraint programming solver.

Authors	David Müller[a] david.mueller@uni-siegen.de Dominik Kress[a,b] dominik.kress@hsu-hh.de [a] *University of Siegen, Management Information Science, Kohlbettstraße 15, 57068 Siegen, Germany* [b] *Helmut Schmidt University - University of the Federal Armed Forces Hamburg, Business Administration, especially Procurement and Production, Friedrich-Ebert-Damm 245, 22159 Hamburg, Germany*
Full citation	Müller, D. and Kress, D. (2019). Filter-and-fan approaches for scheduling flexible job shops under workforce constraints. Working Paper. University of Siegen.

4.1 Introduction

The *job shop scheduling problem* (JSP) is a well-known scheduling setting that has attracted a lot of attention in the literature (see, e.g., Błażewicz et al., 2007, for an overview). It arises in traditional manufacturing systems and is composed of a set of jobs and a set of machines. Each job consists of a set of operations that have to be processed in a predefined order to complete the job. Moreover, each operation is associated to a machine that must be used for its processing as well as a corresponding processing time. A machine can process only one operation at a time and preemption of operations is not permitted. Given these restrictions, the problem is to sequence the operations on the machines so that all jobs are completed and some performance measure is optimized. When considering the minimization of the makespan, the JSP is known to be strongly NP-hard (Lenstra and Rinnooy Kan, 1979).

In the face of large product varieties, short product life cycles, and demand fluctuations, many manufacturing companies implement manufacturing systems that allow for a quick response to market changes. These companies oftentimes make use of multi-purpose machines that are able to process different types of operations (see, e.g., Beach et al., 2000; Jain et al., 2013). This is taken account of in a generalization of the JSP, which is commonly referred to as the *flexible job shop scheduling problem* (FJSP). It was originally introduced by Brucker and Schlie (1990) and assumes that each operation is associated to a set of *eligible machines*, so that a feasible schedule must specify the machines that are used for processing the operations. A recent survey on solution techniques for the FJSP is provided by Chaudhry and Khan (2016). Usually, a machine needs to be operated by some *machine operator* (*worker*). As a result of the manufacturing flexibility (and complexity) induced by the use of multi-purpose machines, workers are oftentimes not qualified for operating all machines or processing all manufacturing operations, so that one has to take account of a heterogeneous workforce (De Bruecker et al., 2015). We will refer to FJSP settings that explicitly incorporate workers by WFJSPs. Note, however, that these settings are sometimes also referred to as dual-resource constrained systems (see, e.g., Treleven, 1989; Xu et al., 2011). Usually, differing qualifications of workers are embedded into FJSPs by making use of worker dependent processing times.

4.1.1 Literature Overview

As pointed out by Kress and Müller (2019), most of the literature on WFJSPs is concerned with metaheuristic solution approaches. When considering makespan minimization, this includes Xianzhou and Zhenhe (2011) and Peng et al. (2018) (genetic algorithms), Lei and Guo (2014) and Vallikavungal Devassia et al. (2018) (variable neighborhood search), Yazdani et al. (2015) (simulated annealing and vibration damping optimization), Zhang et al. (2015) (particle swarm optimization), Zheng and Wang (2016) (knowledge-guided fruit fly optimization), and Wu et al. (2018) (hybrid genetic algorithm). Within this stream of research, Vallikavungal Devassia et al. (2018) and Wu et al. (2018) consider generalized settings with resource recovery constraints and learning effects of workers, respectively. Paksi and Ma'ruf (2016) analyse the objective

Table 4.1: Literature overview: basic WFJSPs aiming at makespan minimization

Publication	Objective	Approach
Kress and Müller (2019)	Makespan	MIP, CP
Lei and Guo (2014)	Makespan	Variable neighborhood search
Peng et al. (2018)	Makespan	Genetic algorithm
Xianzhou and Zhenhe (2011)	Makespan	Genetic algorithm
Yazdani et al. (2015)	Makespan	Simulated annealing and vibration damping optimization, MIP
Zhang et al. (2015)	Makespan	Particle swarm optimization
Zheng and Wang (2016)	Makespan	Knowledge-guided fruit fly optimization

Table 4.2: Literature overview: WFJSPs with objectives differing from pure makespan minimization or under additional constraints

Publication	Objective	Approach
Gong et al. (2018a)	Makespan, total worker cost and green-production factors	Hybrid genetic algorithm
Gong et al. (2018b)	Makespan, maximum workload of machines and total workload of all machines	Memetic algorithm
Kress et al. (2019b)[a]	Makespan; Total tardiness	Branch-and-cut algorithm, MIP, Decomposition based heuristic approaches
Lang and Li (2011)	Delivery satisfaction, process cost, energy consumption, and noise pollution	Genetic algorithm
Lei and Tan (2016)	Makespan and total tardiness	Local searchn
Liu et al. (2011)	Makespan and production cost	Hybrid genetic algorithm
Paksi and Ma'ruf (2016)	Total tardiness	Genetic algorithm
Vallikavungal Devassia et al. (2018)[b]	Makespan	Variable neighborhood search, MIP
Wu et al. (2018)[c]	Makespan	Hybrid genetic algorithm
Zhang et al. (2013)	Makespan and production cost	Hybrid discrete particle swarm optimization

[a] : Incorporation of sequence-dependent setup times.
[b] : Consideration of resource recovery constraints.
[c] : Consideration of learning effects of workers.

of minimising the total tardiness and apply a genetic algorithm. Kress et al. (2019b) address a WFJSP that takes account of sequence-dependent setup times. The authors analyse two objectives, minimising the makespan and minimising the total tardiness. They propose exact and heuristic decomposition based solution approaches. A few researchers address multiple objectives for WFJSPs. Examples include Lang and Li (2011), Liu et al. (2011), Lei and Tan (2016), Zhang et al. (2013), Gong et al. (2018a), and Gong et al. (2018b). Exact approaches or mixed-integer programming (MIP) or constraint programming (CP) models that are actually evaluated in computational tests are proposed by only a few authors (Kress and Müller, 2019; Kress et al., 2019b; Vallikavungal Devassia et al., 2018; Yazdani et al., 2015).

Overviews of the relevant literature are presented in Table 4.1 (basic WFJSPs aiming at makespan minimization; directly related to the paper at hand) and Table 4.2 (WFJSPs with objectives other than makespan minimization or under additional constraints).

4.1.2 Contribution and Overview

In this paper, we address a WFJSP (hereafter referred to as *the* WFJSP for the sake of simplicity) with the objective of minimising the makespan. This problem is strongly NP-hard as it extends the JSP. According to the classical three-field notation by Graham et al. (1979) that was adapted by Błażewicz et al. (1983) to include additional resource constraints, WFJSP falls into the category $FJ|res1 \cdot 1|C_{max}$.

Some foundations of this paper are provided in Kress and Müller (2019), where we present a MIP model as well as a CP based formulation of WFJSP and compare these models by using the standard solvers provided by IBM ILOG CPLEX. We find "that the CP solver clearly outperforms the MIP solver for the considered modelling approaches [and that] it also tends to outperform a state-of-the-art metaheuristic approach" by Zheng and Wang (2016), who claim that their metaheuristic approach is "more effective than the existing algorithms." In the paper at hand, we therefore aim to provide a (meta-)heuristic approach that proves to be competitive when compared with the use of the standard CP solver provided by CPLEX.

The success of metaheuristic approaches depends on an effective exploration of the solution space. The use of compound neighborhood structures, as, for example, generated by *filter-and-fan* (F&F) methods, has shown to be advantageous for many optimization problems (see Glover, 1998; Rego and Glover, 2002, 2010). Rego and Glover (2010) find that F&F methods "have been notable for providing robust methods that produce solutions that match or come very close to matching those produced by the best available methods for the problem classes to which they have been applied, while requiring solution times that are significantly [...] reduced by comparison to competing methods." This has motivated us to develop F&F solution approaches for WFJSP, which we present in this paper. For the case of the JSP, Rego and Duarte (2009) provide a F&F approach that performs remarkably well and is solely outperformed by a specific tabu search (TS) method. We therefore additionally provide a TS benchmark heuristic. Our solution approaches make use of neighborhood structures that have proven to be successful for the FJSP (see Mastrolilli and Gambardella, 2000) and a decomposition of WFJSP into a machine allocation and sequencing component and a worker assignment component (see Kress et al., 2019b).

The remainder of this paper is structured as follows. In Section 4.2, we provide a formal definition of WFJSP and we introduce the concept of the solution graph, which we use to represent solutions of WFJSP. In order to keep this paper self-contained, we then summarize the CP formulation of Kress and Müller (2019) in Section 4.3. Next, in Section 4.4, we describe our F&F approaches in detail. An extensive computational study that includes the TS approach is subject of Section 4.5. The paper closes with a summary in Section 4.6.

4.2 Problem Definition and Representation of Feasible Solutions

Our solution approaches make use of neighborhood functions that are an adaption of the one presented by Mastrolilli and Gambardella (2000) for the FJSP. The latter function is based on

the representation of solutions by so called solution graphs. The concept of these solution graphs is introduced in Section 4.2.2, after having formally defined the WFJSP and the notation used throughout this article in Section 4.2.1.

4.2.1 Notation and Problem Definition

The WFJSP is defined as follows. A set of I of *jobs*, $|I| = n$, a set M of *machines*, and a set W of *workers* are given. Each job $i \in I$ is associated with a set of q_i operations $O_i = \{i_1, \ldots, i_{q_i}\}$. The sets O_i are assumed to be ordered for all $i \in I$, which relates to the fact that for any pair of operations $i_j, i_k \in O_i$ with $j < k$, i_j must be completed before the processing of i_k may start. Each operation $i_j \in O_i$, $i \in I$, must be processed on exactly one machine out of a non-empty set of *eligible machines* $M_{i_j} \subseteq M$. Moreover, an operation $i_j \in O_i$ of a job $i \in I$ can only be processed on a machine, if exactly one worker out of a non-empty set of *eligible workers* $W_{i_j} \subseteq W$ is assigned to the operation for the entire processing time. Processing times are assumed to depend on worker and machine assignments. The processing time of an operation $i_j \in O_i$ of a job $i \in I$ assigned to worker $w \in W_{i_j}$ on machine $m \in M_{i_j}$ is denoted by $p_{i_j}^{m,w} \in \mathbb{N}^+ \cup \{\infty\}$. For each job $i \in I$, each operation $i_j \in O_i$, and each eligible machine $m \in M_{i_j}$, we assume that there exists at least one eligible worker $w \in W_{i_j}$ with a finite processing time $p_{i_j}^{m,w}$. Similarly, for each eligible worker, we assume that there exists at least one eligible machine with a finite processing time. The completion time of an operation $i_j \in O_i$ of job $i \in I$ is denoted by C_{i_j}. The completion time of job $i \in I$ is denoted by C_i. A job is completed if all of its operations are completed. Hence, $C_i = C_{i_{q_i}}$ for all $i \in I$.

We assume that all jobs, machines, and workers are available at time zero. The processing of operations may not be preempted. Furthermore, each machine and each worker can process at most one operation at a time. The problem is to find a schedule, i.e. an allocation of operations to machines and workers as well as corresponding sequences and starting times of the operations on the allocated machines and workers, such that the makespan $C_{max} := \max_{i \in I} C_i$ is minimized subject to the above constraints. We restrict our attention to *left-justified* schedules (see, e.g., Sprecher et al., 1995). That is, whenever considering feasible solutions in the remainder of this paper, we assume that each operation is started to be processed as early as possible when taking the allocation and sequencing decisions as given.

4.2.2 Solution Graph

The *disjunctive graph* model is a well-known and commonly used representation of scheduling problems and their solutions (see, e.g., Błażewicz et al., 2007). Mastrolilli and Gambardella (2000) make use of the underlying idea of this model to represent solutions of the FJSP by means of the so called *solution graph*, which – in the presence of workers as an additional resource – can be augmented in a straightforward manner:

- For all jobs $i \in I$, each operation $i_j \in O_i$ defines a vertex. We denote the resulting vertex set by V, i.e. $V := \bigcup_{i \in I} O_i$.

- Additional dummy vertices, denoted by 0_1 and $(n+1)_1$, represent the beginning and end of a schedule. We define $D := \{0_1\} \cup \{(n+1)_1\}$.

- Precedence relations among the operations of the jobs are represented by directed edges of the set A_1. For each job $i \in I$ and all pais i_j, i_{j+1} with $j \in \{1, \ldots, q_i - 1\}$, A_1 includes the directed edge (i_j, i_{j+1}). Additionally, A_1 includes dummy edges $(0_1, i_1)$ and $(i_{q_i}, (n+1)_1)$ for all $i \in I$. The elements of A_1 are referred to as *precedence edges*.

- Based on the given solution of the considered instance of WFJSP, the set A_2 of directed edges includes an edge (i_j, k_l), if and only if i_j is processed immediately before k_l on some machine $m \in M$. For each $m \in M$, A_2 additionally includes dummy edges from vertex 0_1 to the vertex that corresponds to the first operation that is processed on m and from the vertex that corresponds to the last operation that is processed on m to vertex $(n+1)_1$. For each machine $m \in M$ that processes no operation, A_2 includes a dummy edge from vertex 0_1 to vertex $(n+1)_1$. The elements of the set A_2 are referred to as *machine edges*.

- Similarly, the set A_3 includes a directed edge (i_j, k_l), if and only if i_j is processed immediately before k_l by some worker $w \in W$. Additional dummy edges are defined in line with their definition for machine edges. The elements of the set A_3 are referred to as *worker edges*.

Given some solution of an instance of WFJSP, we denote the corresponding solution graph by $G = (V \cup D, A_1 \cup A_2 \cup A_3, \mu)$, where $\mu : V \cup D \to \mathbb{N}$ defines a weight for each vertex of the graph. Note that, to ease the notation, we do not explicitly refer to the concrete instance and solution when denoting this graph. Furthermore, note that the solution is infeasible if the corresponding solution graph contains a cycle. The weight of each dummy vertex of the set D is 0, while the weights of the other vertices are defined by the processing times of the corresponding operations according to the machine and worker allocation of the solution. Given an integer $\delta \in \{1, 2, 3\}$, a solution graph G, and an operation $i_j \in V$, we denote the unique operation $k_l \in V \cup D$ with $(k_l, i_j) \in A_\delta$ ($(i_j, k_l) \in A_\delta$) by $P_\delta(i_j)$ ($S_\delta(i_j)$).

It is easy to see that the makespan of a solution of an instance of WFJSP corresponds to the length of some longest path, also referred as a *critical path*, from 0_1 to $(n+1)_1$ in the corresponding solution graph. Here, the length of a path is defined as the sum of the vertex weights of the vertices on the path. Operations that belong to a critical path are referred to as *critical operations*. For each operation $i_j \in V \cup D$, we define a *starting time* s_{i_j} and a *tail time* q_{i_j} in analogy to Mastrolilli and Gambardella (2000). s_{i_j} corresponds to the time instant at which i_j is started to be processed in the solution and equals the length of a longest path from vertex 0_1 to i_j when excluding the vertex weight of operation i_j. q_{i_j} corresponds to the length of a longest path from i_j to $(n+1)_1$ without the vertex weight of operation i_j. The makespan, as well as the starting times and tail times of all vertices of the solution graph can easily be computed in $O(|V \cup D|)$ time by a straightforward variation of Bellman's labeling algorithm as proposed by Taillard (1994) for the JSP. An operation $i_j \in V$ is critical if and only if $s_{i_j} + \mu(i_j) + q_{i_j} = C_{max}$.

4.3 Constraint Programming Formulation

As outlined above, Kress and Müller (2019) present a CP formulation for WFJSP. It is based on variables and constraint types provided by the IBM ILOG CPLEX CP Optimizer (see IBM, 2016; Laborie et al., 2018, for an introduction). The CP formulation uses *interval variables* to model the start and the end of the processing of the operations. *Sequence variables* represent the sequencing decisions, i.e. orderings of interval variables. An overview is given in Table 4.3. Based

Table 4.3: Variables for the CP model as introduced by Kress and Müller (2019)

Variables	Definition
$I_{OP\{i_j\}}$	Interval variable for each operation, i.e. for all $i \in I$, $i_j \in O_i$
$I_{MO\{i_j,m,w,p\}}$	Interval variable for each *processing mode*, i.e. each eligible combination of an operation $i_j \in O_i$ of job $i \in I$, a machine, a worker and a (finite) processing time
$S_{\bar{m}}$	Sequence variable for each machine $\bar{m} \in M$; related to all interval variables $I_{MO\{i_j,m,w,p\}}$ with $m = \bar{m}$
$S_{\bar{w}}$	Sequence variable for each worker $\bar{w} \in W$; related to all interval variables $I_{MO\{i_j,m,w,p\}}$ with $w = \bar{w}$

on these variables and the structures and notation provided by IBM's CP Optimizer, which we assume the reader to be familiar with, the compact formulation of WFJSP as presented in Kress and Müller (2019) is as follows.

$$\min \max_{i \in I}(endOf(I_{OP\{i_{q_i}\}})) \tag{4.1}$$

s.t.

$$endBeforeStart(I_{OP\{i_j\}}, I_{OP\{i_{j+1}\}}) \quad \forall\, i \in I, j \leq q_i - 1, \tag{4.2}$$

$$alternative(I_{OP\{i_j\}}, all\, I_{MO\{i_j,m,w,p\}}) \quad \forall\, i \in I, i_j \in O_i, \tag{4.3}$$

$$noOverlap(S_m) \quad \forall\, m \in M, \tag{4.4}$$

$$noOverlap(S_w) \quad \forall\, w \in W. \tag{4.5}$$

The objective function (4.1) represents the minimization of the makespan. Constraints (4.2) capture the precedence constraints among the operations of the jobs. Constraints (4.3) guarantee that an eligible processing mode is chosen for each operation. Constraints (4.4) and (4.5) ensure that each machine and each worker processes at most one operation at a time.

4.4 Filter-and-Fan Approaches

The historical development and basic functionality of F&F approaches is excellently summarized by Rego and Glover (2010). They characterize F&F methods as multi-stream neighborhood search strategies that date back to Glover (1998) and were extended by Rego and Glover (2002) as a method of creating efficient and robust combined neighborhood search strategies. In this sense, they can be seen as a complement to ejection chain procedures (see also Dorndorf et al., 2008;

Glover, 1996; Kress et al., 2017, 2019a; Pesch and Glover, 1997, for an introduction to ejection chain approaches and their applications). On their most general level, F&F approaches combine two fundamental search strategies that are applied in an alternating manner (see Figure 4.1). A *local search* procedure is used to obtain local optima, while a *tree search* procedure generates

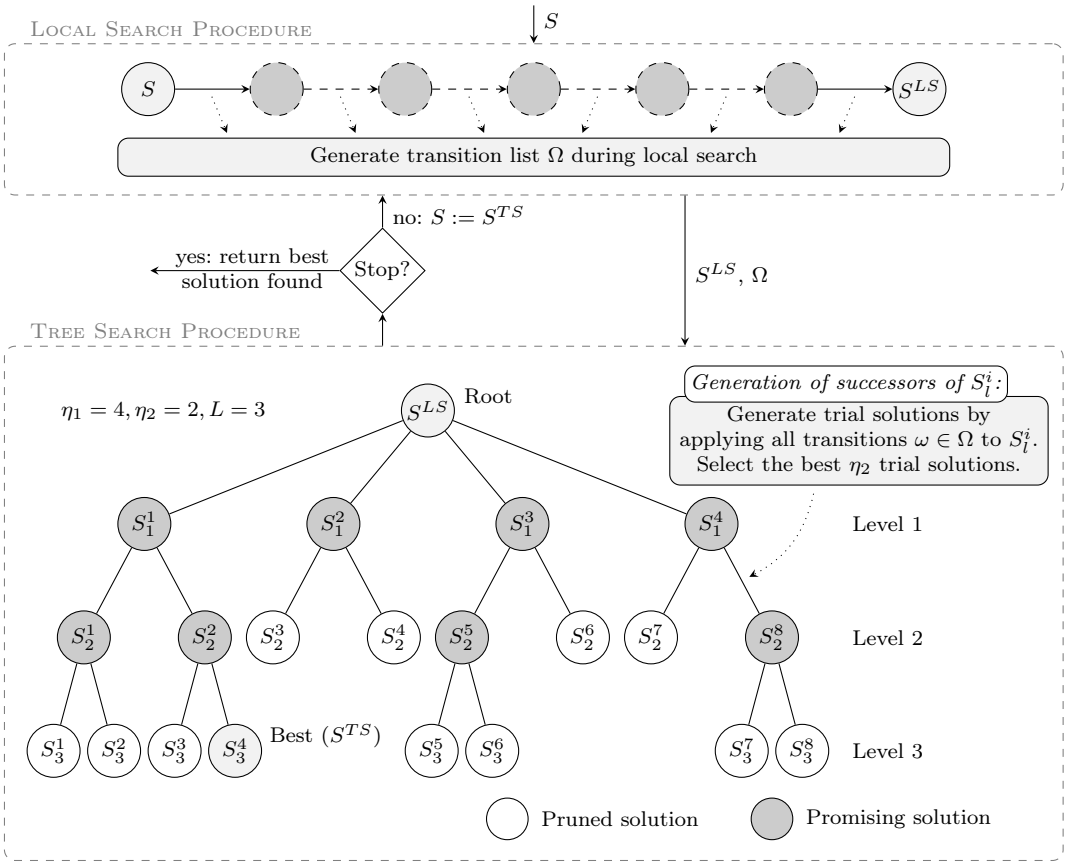


Figure 4.1: Alternating structure of F&F approaches

compound transitions in order to explore larger neighborhoods to overcome these locally optimal solutions. The tree search procedure essentially corresponds to a beam search approach that generates multiple paths using a breadth-first search strategy. A F&F method initiates with the local search procedure that is called on an input solution S and returns a local optimum S^{LS} as well as a list Ω of transitions (moves or meta-information on moves) associated with the "best" η_0 solutions evaluated within the local search. Upon termination of the local search, the F&F method switches to the tree search procedure. The nodes of the corresponding search tree in Figure 4.1 represent feasible solutions that are established by performing compound transitions to S^{LS} (the root node) based on the transition list Ω. In order to construct the first level of the tree, the best η_1 transitions returned by the local search procedure are applied to S^{LS}. This results in η_1 solutions $S_1^1, \ldots, S_1^{\eta_1}$. All other levels $l' = 2, \ldots, L$ of the tree are constructed by first selecting (marking) the best η_1 solutions on the preceding level $l = l' - 1$ (*filter candidate list* strategy). For each corresponding solution S_l^i, the procedure then generates a set of trial solutions by applying all transitions included in Ω and then selecting the best η_2 trial solutions to become elements of level l' (*fan candidate list* strategy). This results in a total of $\eta_1\eta_2$ solutions

$S_l^1, \ldots, S_l^{\eta_1 \eta_2}$ on each level $l > 1$. The tree search terminates as soon as one of the solutions on some level of the tree (referred to as S^{TS}) is better than S^{LS} or when the maximum number L of levels has been traversed without having found such a solution. In the latter case, the overall procedure terminates. Otherwise, the local search procedure is called on S^{TS}.

4.4.1 Neighborhood Structure

Mastrolilli and Gambardella (2000) construct a neighbor of a given solution of FJSP by *moving* an operation, i.e. deleting it from its current machine sequence and inserting it in some other feasible position on a corresponding (not necessarily different) eligible machine. By making use of the concept of solution graphs (see Section 4.2.2), they show that the resulting set of potential neighbors of a solution can be reduced to a specific subset that is guaranteed to include a neighbor with the lowest makespan. Unfortunately, this result does not immediately carry over to the case of the WFJSP because the processing times (and therefore the vertex weights of the solution graph) in the latter problem depend on both the machine and worker allocation. Nevertheless, we make use of this method as it guarantees the construction of feasible solutions, i.e. solutions with acyclic solution graphs, when adapted appropriately.

In order to handle the interdependencies between machine and worker allocations, we follow the main ideas of the hierarchical (decomposition based) approach introduced by Kress et al. (2019b) for a WFJSP with sequence-dependent setup times. In a first step, their approach solely takes account of the allocation of operations to eligible machines and the sequencing of these operations on the machines. The second step then determines a corresponding assignment of operations to eligible workers as well as the sequences of these operations for each worker. Given the solution graph G of some feasible solution of WFJSP and an operation a_b that we want to move, we adapt the underlying hierarchical idea as shown in Algorithm 4.1, where we first delete either all machine edges or all worker edges of the graph (construction of G' in line 2). We are left with a graph that solely considers one of the two resources, so that we can directly apply the ideas of Mastrolilli and Gambardella (2000) in order to construct feasible (with respect to the resource that has not been deleted as well as the precedence constraints) neighbors and select a promising candidate (line 4). Finally, we recompute a feasible allocation and sequencing decision for the remaining resource and update the solution graph accordingly (line 5). This routine is executed for all eligible machines or workers of operation a_b (line 3). It is important to note that line 4 of Algorithm 4.1 is based on the vertex weights of the input graph, even though one resource is neglected. Nevertheless, the feasibility of the solutions that correspond to the graphs constructed in line 5 is implied by the feasibility results presented by Mastrolilli and Gambardella (2000) because the assignment of workers to operations given an allocation and sequencing decision for the machines (and similarly an assignment of machines to operations given the worker decisions) will only cause temporal shifts of the operations on the machines (or in the worker sequences). Details of Algorithm 4.1 are given in the following sections.

Input: Solution graph $G = (V \cup D, A_1 \cup A_2 \cup A_3, \mu)$ with starting and tail times, operation $a_b \in V$
Output: Set N of neighboring solution graphs, including starting and tail times

1 Initialize $N := \emptyset$;
2 Determine $\gamma \in \{2, 3\}$ (Algorithm 4.2). Set $G' := (V \cup D, A_1 \cup A_\gamma, \mu)$. The starting times and tail times of the vertices of G' are set to the ones of the vertices of G;
3 **forall** $m \in M_{a_b}$ *(in case of* $\gamma = 2$*) or* $w \in W_{a_b}$ *(in case of* $\gamma = 3$*)* **do**
4 Apply an adapted version of the procedure presented by Mastrolilli and Gambardella (2000) on G' to determine a set of neighboring solution graphs resulting from moving a_b to machine m (worker w) and select a most promising candidate $\hat{G} = (V \cup D, A_1 \cup \hat{A}_\gamma, \mu)$ (Algorithm 4.3);
5 Recompute (feasible) set of machine edges (in case of $\gamma = 3$) or worker edges (in case of $\gamma = 2$) and add it to $\hat{G}$ (Algorithm 4.4). While doing so, redefine vertex weights μ and compute starting times and tail times of the vertices of $\hat{G}$ based on the corresponding worker and machine assignment;
6 Set $N := N \cup \{\hat{G}\}$;
7 **end**

Algorithm 4.1 Generate a set of neighboring solution graphs

4.4.1.1 Determine Set of Edges to be Deleted

In order to determine the set of edges that is deleted from G in line 2 of Algorithm 4.1, we analyse the completion times of the predecessor operations of the input operation a_b with respect to the edge sets A_2 and A_3, i.e. operations $P_2(a_b)$ and $P_3(a_b)$, as stated in Algorithm 4.2. The basic

Input: Solution graph $G = (V \cup D, A_1 \cup A_2 \cup A_3, \mu)$ with starting and tail times, operation $a_b \in V$
Output: Integer γ

1 **if** $s_{P_2(a_b)} + \mu(P_2(a_b)) > s_{P_3(a_b)} + \mu(P_3(a_b))$ **then** $\gamma = 2$;
2 **else if** $s_{P_2(a_b)} + \mu(P_2(a_b)) < s_{P_3(a_b)} + \mu(P_3(a_b))$ **then** $\gamma = 3$;
3 **else** randomly select $\gamma \in \{2, 3\}$;

Algorithm 4.2 Determine γ

idea is to determine the resource that has the strongest effect on the "delayed" start of a_b due to the sequencing decisions and later delete the edge set that corresponds to the other resource.

Consider an exemplary feasible solution of WFJSP as illustrated in Figure 4.2 as a Gantt chart. In case of $a_b = 2_2$ (processed on machine 1 by worker 3) we have $P_2(a_b) = 3_1$ and $P_3(a_b) = 4_1$, so that Algorithm 4.2 will return $\gamma = 2$. Similarly, for $a_b = 4_2$, the algorithm will return $\gamma = 3$. In case of $a_b = 3_2$, the algorithm will randomly determine $\gamma \in \{2, 3\}$.

4.4.1.2 Determine Neighboring Solution Graph

After having deleted either all machine (if $\gamma = 3$) or all worker (if $\gamma = 2$) edges from G in line 2 of Algorithm 4.1, we are left with a graph $G' = (V \cup D, A_1 \cup A_\gamma, \mu)$. Given operation a_b and some machine $m \in M_{a_b}$ (if $\gamma = 2$) or worker $w \in W_{a_b}$ (if $\gamma = 3$) as selected in line 3 of Algorithm 4.1, our adaption of the procedure presented by Mastrolilli and Gambardella (2000) is presented in

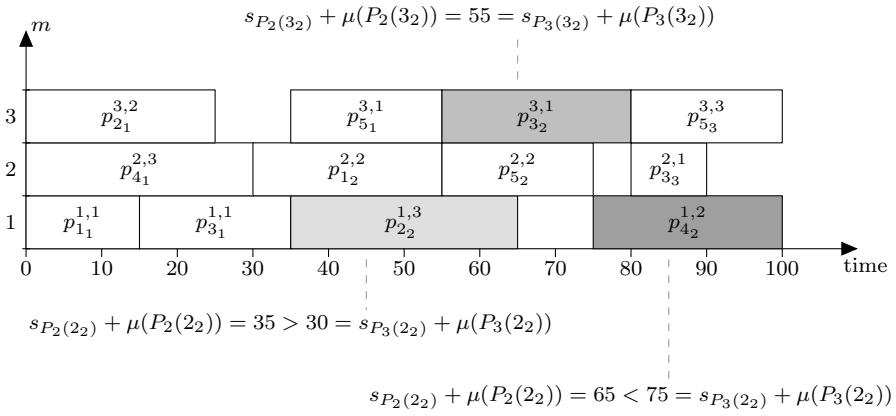


Figure 4.2: Exemplary solution of an instance of WFJSP

Algorithm 4.3. For the sake of notational convenience, we denote the resource under evaluation in the algorithm by θ, i.e. we set $\theta = m$ if $\gamma = 2$ and $\theta = w$ if $\gamma = 3$. The algorithm starts by deleting a_b from its current machine (worker) sequence in lines 1–2. The starting time and tail time of a_b is updated accordingly (line 3). Next, in lines 4–12, the algorithm checks potential trial moves of operation a_b to the operation sequence of resource θ. Denote the set of operations that is processed by resource θ by Q_θ (note that this will not include a_b, as this operation has been removed from its sequence) and assume that the elements of this set are ordered in non-decreasing order of their starting times. The algorithm computes the sets $L_\theta := \{i_j \in Q_\theta \mid \mu(i_j) + q_{i_j} > q_{a_b}\}$ and $R_\theta := \{i_j \in Q_\theta \mid s_{i_j} + \mu(i_j) > s_{a_b}\}$ (line 4). As shown by Mastrolilli and Gambardella (2000) for the FJSP, all insertions of a_b after the operations of the set $L_\theta \setminus R_\theta$ and before the operations of the set $R_\theta \setminus L_\theta$ result in feasible solutions. As indicated above, this result immediately carries over to the case of the WFJSP, so that Algorithm 4.3 restricts the construction of potential neighbors to these insertions. Let $\xi_\theta := Q_\theta \setminus (L_\theta \setminus R_\theta) \setminus (R_\theta \setminus L_\theta)$ and assume that the elements of this set are ordered in non-decreasing order of their starting times (see Figure 4.3). Furthermore, denote

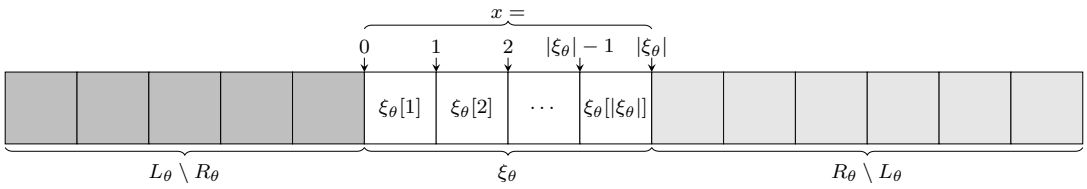


Figure 4.3: Illustration of Q_θ

the i-th element of this set by $\xi_\theta[i]$ and define $\xi_\theta[|\xi_\theta|+1]$ to be an operation with smallest starting time in the set $R_\theta \setminus L_\theta$ or, if this set is empty, the dummy operation $(n+1)_1$. The algorithm approximates the length of the longest paths from 0_1 to $(n+1)_1$ that include operation a_b and that result from inserting a_b at positions $x = 0$ (immediately before the first operation of ξ_θ) to

Input: Solution graph $G' := (V \cup D, A_1 \cup A_\gamma, \mu)$ with starting and tail times (as determined in line 2 of Algorithm 4.1), operation $a_b \in V$, eligible machine $m \in M_{a_b}$ or worker $w \in W_{a_b}$ (denoted by θ)
Output: Solution graph $\hat{G} = (V \cup D, A_1 \cup \hat{A}_\gamma, \mu)$

1 Initialize $\hat{A}_\gamma := A_\gamma$ and $\hat{G} := (V \cup D, A_1 \cup \hat{A}_\gamma, \mu)$ with starting and tail times identical to the ones of G' (all following deletion, adding, and updating operations are performed on $\hat{G}$);
2 Delete $(P_\gamma(a_b), a_b)$ and $(a_b, S_\gamma(a_b))$ from $\hat{A}_\gamma$;
3 Set $s_{a_b} := s_{P_1}(a_b) + \mu(P_1(a_b))$ and $q_{a_b} := \mu(S_1(a_b)) + q_{S_1}(a_b)$;
4 Compute L_θ and R_θ;
5 Initialize $LP^* := \infty$ and $x^* := 0$;
6 **forall** *potential positions* $x \in \{0, \ldots, |\xi_\theta|\}$ *after* $L_\theta \setminus R_\theta$ *and before* $R_\theta \setminus L_\theta$ **do**
7 Compute $LP(a_b, \theta, x)$;
8 **if** $LP(a_b, \theta, x) < LP^*$ **then**
9 $LP^* = LP(a_b, \theta, x)$;
10 $x^* = x$;
11 **end**
12 **end**
13 **if** $x^* = 0$ **then**
14 Delete an edge $(P_\gamma(\xi_\theta[1]), \xi_\theta[1])$ from $\hat{A}_\gamma$;
15 Add $(P_\gamma(\xi_\theta[1]), a_b)$ and $(a_b, \xi_\theta[1])$ to $\hat{A}_\gamma$;
16 **end**
17 **else if** $x^* = |\xi_\theta|$ **then**
18 Delete $(\xi_\theta[x^*], S_\gamma(\xi_\theta[x^*]))$ from $\hat{A}_\gamma$;
19 Add $(\xi_\theta[x^*], a_b)$ and $(a_b, S_\gamma(\xi_\theta[x^*]))$ to $\hat{A}_\gamma$;
20 **end**
21 **else**
22 Delete $(\xi_\theta[x^*], \xi_\theta[x^* + 1])$ from $\hat{A}_\gamma$;
23 Add $(\xi_\theta[x^*], a_b)$ and $(a_b, \xi_\theta[x^* + 1])$ to $\hat{A}_\gamma$;
24 **end**

Algorithm 4.3 Determine neighboring solution graph

$x = |\xi_\theta|$ (immediately after the last operation of ξ_θ) as follows:

$$LP(a_b, \theta, x) := p_{a_b}^{\theta,\min} + \begin{cases} [1.5]s_{a_b} + \max(p_{\xi_\theta[1]}^{\theta,\min} + q_{\xi_\theta[1]}, q_{a_b}), & \text{if } x = 0 \\ \max(s_{\xi_\theta[x]} + p_{\xi_\theta[x]}^{\theta,\min}, s_{a_b}) + & \\ \quad \max(p_{\xi_\theta[x+1]}^{\theta,\min} + q_{\xi_\theta[x+1]}, q_{a_b}), & \text{if } 1 \leq x < |\xi_\theta| \\ \max(s_{\xi_\theta[x]} + p_{\xi_\theta[x]}^{\theta,\min}, s_{a_b}) + q_{a_b}, & \text{if } x = |\xi_\theta| \text{ and } |\xi_\theta| > 0. \end{cases}$$

Here, $p_{i_j}^{m,\min} := \min_{w \in W_{i_j}} p_{i_j}^{m,w}$ for all $i_j \in V$ and $m \in M_{i_j}$. Similarly $p_{i_j}^{w,\min} := \min_{m \in M_{i_j}} p_{i_j}^{m,w}$ for all $i_j \in V$ and $w \in W_{i_j}$. Algorithm 4.3 selects a position x^* that results in the shortest approximate length (lines 5–12) and terminates after having updated the corresponding solution graph $\hat{G}$ (lines 13–24).

4.4.1.3 Recompute Missing Edge Set

Once a neighboring solution graph $\hat{G}$ has been selected in line 4 of Algorithm 4.1, we are left with having to recompute the edge set that has previously been deleted. To do so, we follow a simple greedy approach that is illustrated in Algorithm 4.4. Given the machine (in case of $\gamma = 2$)

Input: Solution graph $\hat{G} = (V \cup D, A_1 \cup \hat{A}_\gamma, \mu)$
Output: Modified solution graph $\hat{G}$ with starting times and tail times, values C_{max} and $\bar{C}_{max}$

1 Initialize $\hat{A}_{\bar{\gamma}} := \emptyset$, where $\bar{\gamma} \in \{2, 3\} \setminus \gamma$;
2 Add edge set $\hat{A}_{\bar{\gamma}}$ to $\hat{G}$ (all following adding and updating operations are performed on $\hat{G}$);
3 Set $\mu(i_j) := p_{i_j}^{\theta(i_j),\min}$, where $\theta(i_j)$ refers to the resource (specific machine, if $\gamma = 2$, or worker, if $\gamma = 3$) that operation i_j is currently allocated to, for all $i_j \in V$ in solution graph $\hat{G}$;
4 Recompute starting times s_{i_j} of vertices $i_j \in V \cup D$ in $\hat{G}$ (see Section 4.2.2) and set $\bar{C}_{max} := s_{(n+1)_1}$;
5 Assign an eligible worker $w \in W_{i_j}$ (if $\gamma = 2$) or machine $m \in M_{i_j}$ (if $\gamma = 3$) to each operation $i_j \in V$ in non-decreasing order of the starting times determined in line 4 in a greedy manner. While doing so, update the corresponding vertex weight $\mu(i_j)$ and starting time s_{i_j}. Additionally, define a corresponding worker or machine edge and add it to $\hat{A}_{\bar{\gamma}}$;
6 Recompute tail times q_{i_j} of vertices $i_j \in V \cup D$ in $\hat{G}$ (see Section 4.2.2) and set $C_{max} := s_{(n+1)_1}$;

Algorithm 4.4 Recompute missing edge set

or worker allocation (in case of $\gamma = 3$) of the input solution graph $\hat{G}$, it first updates the weights of the vertices $i_j \in V$ to the values $p_{i_j}^{\theta(i_j),\min}$ (line 3), where $\theta(i_j)$ refers to the specific resource (a machine, if $\gamma = 2$, or a worker, if $\gamma = 3$) that operation i_j is currently allocated to. The starting times of the vertices are recomputed accordingly (line 4). The vertices $i_j \in V$ are then traversed in non-decreasing order of these starting times and are allocated to an eligible resource (a worker, if $\gamma = 2$, or a machine, if $\gamma = 3$) in a greedy manner (line 5). That is, the resource is chosen such that the resulting completion time of the corresponding operation is as small as possible, given all previous resource allocation decisions. Based on the resource allocation, the vertex weight and starting time of the corresponding operation may change, so that these values are updated to their correct values. The worker or machine edge that corresponds to the allocation decision

is added to the edge set of the solution graph. Finally, we are left with having to compute the tail times of the vertices of the solution graph (line 6).

4.4.1.4 Neighborhood Definitions

Based on the above deliberations, we define three different neighborhoods of some solution graph G. The first neighborhood, denoted by $N_1(G)$, corresponds to the union of the sets returned by Algorithm 4.1 when being called for all critical operations of G. Similarly, $N_2(G)$ is constructed by calling the algorithm for all non-critical operations. Finally, we define a neighborhood $N_3(G)$ that is somewhat similar to a neighborhood generated by traditional swap moves. It is constructed by calling a modification of Algorithm 4.1 on all critical operations $a_b \in V$ of G. This modification executes the loop of lines 3–7 for all eligible machines or workers that are not identical to the one, say resource θ, that processes the input operation a_b in the input solution graph. After having executed line 5 and thus (potentially) having moved a_b to some other resource, say θ', the modified algorithm once more deletes all machine or worker edges as in line 2, i.e. without recomputing γ, and then constructs all neighboring solution graphs that result from moving some operation $k_l \neq a_b$ on θ' to resource θ (if eligible) in analogy to lines 4 and 5. All these graphs are then added to the set N as in line 6 of Algorithm 4.1.

4.4.2 Details of the Filter-and-Fan Algorithms

In this section, we present the details of all elements of our F&F approaches. As indicated above, these approaches make use of multiple neighborhood definitions. Within the approaches, as also suggested by He et al. (2016) and Rego and Duarte (2009), these neighborhoods will successively be locked and unlocked (*neighborhood switching*).

To ease the notation in the remainder of this section, we will denote the makespan of a solution S by using an additional label, i.e. $\ddot{S}$. Furthermore, we will sometimes refer to a solution graph by its corresponding solution and vice versa.

4.4.2.1 Constructive Procedure

In line with our deliberations in Section 4.4.1, we make use of the hierarchical approach of Kress et al. (2019b) in order to construct a first feasible solution S (and its solution graph) of a given instance of WFJSP. Hence, we first allocate all operations to eligible machines and make the corresponding sequencing decisions without considering the workers. Here, we apply a priority-rule based heuristic proposed by Kress et al. (2019b) (see therein for details; setup times can easily be neglected) that follows an algorithmic idea of Giffler and Thompson (1960) for the classical JSP. Basically, this heuristic iteratively allocates operations that can start being processed at the respective point of time with respect to all corresponding precedence constraints. Among all operations that compete for the same machine in some iteration, exactly one operation is chosen based on the *most work remaining* (MWKR) priority rule. Given the corresponding solution graph that solely includes precedence and machine edges, our constructive procedure

then proceeds in a greedy manner as in Algorithm 4.4 in order to generate the missing worker edges.

4.4.2.2 Transition List

The transition list Ω generated within the local search procedure is a crucial component of any F&F approach. As defined above, it contains information regarding the "best" η_0 solutions evaluated within the local search (or all solutions, if less than η_0 solutions have been evaluated). Usually, the list contains concrete moves that have been used to generate these solutions, so that it is referred to as the move list (see, e.g., Rego and Duarte, 2009). In our case, however, we store meta-information rather than concrete moves, because the interdependency of machine and worker allocations in the WFJSP causes classical moves to be not applicable or result in infeasible solutions within the tree search procedure more often than the use of meta-information based transitions.

We define a transition ω to include information on the operation $a_b \in V$ that serves as an input of Algorithm 4.1 when generating a neighboring solution, the corresponding value of γ determined in line 2 and the resource θ that the operation a_b is moved to in line 4, a boolean value *swap* that indicates whether or not the transition refers to a solution generated when using neighborhood definition N_3 (one for yes, zero for no), as well as the objective function values C_{max} and $\bar{C}_{max}$ returned by the final call of Algorithm 4.4 in the course of generating the solution. Thus, we denote a transition ω by a tuple $(a_b, \gamma, \theta, swap, C_{max}, \bar{C}_{max})$. The value C_{max} defines the quality of the transition. $\bar{C}_{max}$ is used as a tie-breaker when comparing transitions with identical C_{max}.

In order to apply a transition ω to a solution within the tree search procedure, we call a modified version of Algorithm 4.1 with input operation a_b, where γ is fixed to the given value and where the loop 3–7 is executed solely for resource θ. In case of $swap = 1$, we additionally incorporate the modifications highlighted in Section 4.4.1.4, where k_l is additionally fixed to the operation with the smallest starting time that is processed on resource θ' and is eligible to be processed on resource θ. If no such operation exists, the transition is considered non-applicable. As in case of transitions, the corresponding values C_{max} and $\bar{C}_{max}$ are used as a quality measure of the solution.

4.4.2.3 Local Search

Our local search approach uses a best-fit strategy with a predefined neighborhood operator $\phi \in \{1, 2, 3\}$ (corresponding to neighborhood N_ϕ) on an input solution S. The transition list Ω of length η_0 (or less, if less neighbors have been evaluated) is generated during runtime of the procedure as illustrated in Algorithm 4.5.

4.4.2.4 Tree Search Procedure

Our tree search procedure is outlined in Algorithm 4.6. It follows the main principles of tree

Input: Solution S, parameters η_0 and ϕ
Output: Solution S^{LS}, transition list Ω

1 Initialize $\Omega := \emptyset$, $\Omega_{temp} := \emptyset$, and $S^{LS} := S$;
2 **forall** *critical (if* $\phi \in \{1, 3\}$*) or non-critical (if* $\phi = 2$*) operations* $a_b \in V$ *of the solution graph of* S **do**
3 Call (modified, if $\phi = 3$) Algorithm 4.1 to determine the set N of neighboring solution graphs;
4 Insert the transitions corresponding to the elements of N into Ω_{temp};
5 **if** *one of the elements of* N *has a smaller makespan than* S^{LS} **then** update S^{LS} to the best solution corresponding to these elements;
6 **end**
7 **if** $\ddot{S}^{LS} < \ddot{S}$ **then** set $S := S^{LS}$ and go to line 2;
8 Insert the best $\max\{\eta_0, |\Omega_{temp}|\}$ transitions of Ω_{temp} into Ω;

Algorithm 4.5 Local search procedure

Input: Solution S^{LS}, transition list Ω, parameters η_0, η_1, η_2, L, and ϕ
Output: Solution S^{TS}

▷ `Create first level of the tree` ($l = 1$)
1 Initialize $l := 1$ and an empty tabu list;
2 Construct solutions of the first level of the tree by applying (if applicable) the η_1 best (or all, if there exists less than η_1) transitions included in Ω to S^{LS}. Mark all of these solutions;
3 If no solution was generated in line 1, call Algorithm 4.7 to generate at most η_1 alternative solutions (and update Ω accordingly). Mark all of these solutions. If no solution is generated, terminate the procedure and return $S^{TS} := S^{LS}$;
4 Initialize S^{TS} with the best solution generated in lines 2 and 3;
5 **if** $\ddot{S}^{TS} < \ddot{S}^{LS}$ **then** terminate the procedure;
▷ `Create further levels of the tree` ($1 < l \leq L$)
6 **forall** *marked solutions on the current level* l **do**
7 Apply all transitions in Ω (if applicable) to obtain potential trial solutions for the next level $l + 1$. Among these solutions, discard all but the best η_2 candidates, the transitions of which are non-tabu or the makespan of which is smaller than $\ddot{S}^{LS}$ (aspiration criterion). If there are less than η_2 corresponding candidates, call Algorithm 4.7 to generate additional solutions (and update Ω accordingly), until a total of at most η_2 solutions has been constructed;
8 **end**
9 Initialize S^*_{l+1} with the best solution generated in loop 6–8. If no solution has been generated, terminate the procedure;
10 **if** $\ddot{S}^*_{l+1} < \ddot{S}^{TS}$ **then** set $S^{TS} := S^*_{l+1}$;
11 **if** $\ddot{S}^{TS} < \ddot{S}^{LS}$ **then** terminate the procedure;
12 Mark the best η_1 (or all, if there exists less than η_1) trial solutions on level $l + 1$;
13 Update the tabu list;
14 Delete the worst $\max\{0, |\Omega| - \eta_0\}$ transitions from Ω;
15 **if** $l < L$ **then** set $l := l + 1$ and go to line 6;
16 **else** exit the procedure;

Algorithm 4.6 Tree search procedure

search procedures within F&F approaches described above and uses a neighborhood operator $\phi \in \{1, 2, 3\}$ as an input parameter. The procedure is such that it generates all solutions of a given level, before it potentially terminates (lines 5, 11, and 15). Additionally, note that it uses a global *tabu list* (starting with the generation of the second level) that solely contains operations $i_j \in V$. If the tuple that defines some transition ω includes an operation of the tabu list and if the aspiration criterion (new best solution) is not met, the corresponding solution is discarded in line 7. The tabu list is updated in line 13. Here, the operations that correspond to the transitions that were used when generating the η_1 trial solutions marked in line 12 are added to the list. Moreover, if the length of the tabu list exceeds some threshold (tabu length), the algorithm removes entries of the list in a first-in-first-out manner, until the tabu length is met. The tabu length is dynamically updated within the tree search. When generating level $l > 1$, it is set to the average number of critical (in case of $\phi \in \{1, 3\}$) or non-critical (in case of $\phi = 2$) operations of all marked solutions of the previous level $l - 1$.

Algorithm 4.6 includes details on how to handle situations where less than η_1 or η_2 transitions or solutions are available in the corresponding steps of a F&F approach. The algorithm, for instance, calls Algorithm 4.7 in lines 3 and 7 to potentially generate additional solutions as well as the corresponding transitions based on alternative neighborhood operators. Note that

Input: Solution S, transition list Ω, parameters λ (number of solutions to be determined) and ϕ
Output: Set $\bar{N}$ of solutions including the corresponding transitions, transition list Ω

1 Initialize $\bar{N} := \emptyset$ and $\tilde{N} := \emptyset$;
2 Initialize alternative neighborhood operators $\Phi := \{1, 2, 3\} \setminus \{\phi\}$;
3 Randomly select operator $\bar{\phi} \in \Phi$. Set $\Phi := \Phi \setminus \bar{\phi}$;
4 **forall** *critical (if* $\bar{\phi} \in \{1, 3\}$*) or non-critical (if* $\bar{\phi} = 2$*) operations* $a_b \in V$ *of the solution graph of* S **do**
5 Call (modified, if $\bar{\phi} = 3$) Algorithm 4.1 to determine the set N of neighboring solution graphs. Add all of the corresponding solutions to the set $\tilde{N}$;
6 **end**
7 Select the best $\min\{\lambda, |\tilde{N}|\}$ solutions from $\tilde{N}$, add these solutions to $\bar{N}$, add the corresponding transitions to Ω;
8 **if** $|\bar{N}| < \lambda$ **then**
9 set $\lambda := \lambda - |\bar{N}|$, $\tilde{N} := \emptyset$;
10 **if** $|\Phi| > 0$ **then** go to line 3;
11 **else** exit the procedure;
12 **end**
13 **else** exit the procedure;

Algorithm 4.7 Generate alternative neighbors

Algorithm 4.7 potentially alters the transition list Ω, so that Algorithm 4.6 updates this list in line 14 in order to balance the computational effort and to only keep the most promising transitions.

4.4.2.5 Filter-and-Fan Framework

Our overall F&F framework including neighborhood switching is presented in Algorithm 4.8. It consists of two phases, the *initialization phase* and the *filter-and-fan procedure*. The former

Input: Instance $Inst$ of WFJSP, parameters η_0, η_1, η_2, and L
Output: Solution S^*

▷ `Initialization phase`
1 Determine a feasible solution S of $Inst$ with the constructive procedure described in Section 4.4.2.1 and initialize $S^* := S$;
2 Unlock all neighborhood operators in the specific order $\{1, 3, 2\}$;
▷ `Filter-and-fan procedure`
3 Get the first unlocked neighborhood operator ϕ;
4 Determine S^{LS} and Ω by calling Algorithm 4.5 on solution S with operator ϕ. ; ▷ `Local search`
5 **if** $\ddot{S}^{LS} < \ddot{S}^*$ **then** set $S^* := S^{LS}$ and unlock all neighborhood operators in the specific order $\{1, 3, 2\}$;
6 **if** $|\Omega| < \eta_0$ **then** modify Ω by calling Algorithm 4.7 on S^{LS} with $\lambda := \eta_0 - |\Omega|$;
7 Determine S^{TS} by calling Algorithm 4.6 on S^{LS} with transition list Ω and operator ϕ. ; ▷ `Tree search`
8 **if** $\ddot{S}^{TS} < \ddot{S}^*$ **then** set $S^* := S^{TS}$ and unlock all neighborhood operators in the specific order $\{1, 3, 2\}$;
9 **else if** $\ddot{S}^{TS} \geq \ddot{S}^{LS}$ **then** lock current neighborhood operator ϕ;
10 **if** *there is at least one unlocked neighborhood operator* **then** set $S := S^{TS}$ and go to line 3;
11 **else** exit the procedure;

Algorithm 4.8 Filter-and-fan framework

phase makes use of the constructive procedure described in Section 4.4.2.1 (line 1) to determine a feasible solution. It furthermore unlocks all neighborhood operators (line 2) in a specific order. It is important to note that the corresponding get-procedure (line 3) will always consider this ordering. An operator is locked, if a call of the tree search procedure does not improve its input solution (line 9). Similarly, all operators are unlocked, whenever the overall best solution S^* is improved (lines 5 and 8). The filter-and-fan procedure calls the local search procedure (line 4) and the tree search procedure (line 7) in an alternating manner. This process is repeated until all neighborhood operators are locked (line 10). If the transition list Ω includes less than η_0 elements before executing a tree search, the framework calls Algorithm 4.7 to potentially determine more transitions.

4.4.2.6 A Modified Filter-and-Fan Procedure

As described above, F&F procedures rely on transition lists that are generated during calls of a local search procedure and that are later globally applied in a tree search procedure. We will additionally consider a variant of our F&F framework that makes local decisions within the tree search procedure. Our corresponding modifications are as follows. Algorithm 4.5 (local search) does not compute a transition list. It solely returns a local optimum. Therefore, in Algorithm 4.6 (tree search), whenever generating the successors of some solution S with solution graph G in the tree, we randomly select η_1 (adaption of line 2) or η_2 (adaption of line 7) solutions from

the neighborhood $N_1(G)$ (the other neighborhood operators are not used in the tree search) instead of using a transition list. Hence, we make use of the concrete characteristics, i.e. the critical operations, of S, rather than relying on information generated during local search. We additionally do not make use of Algorithm 4.7 in our modified F&F framework.

4.5 Computational Study

In order to assess the performance of our F&F approaches, we conducted extensive computational tests. They were performed on a PC with an Intel® Core™ i7-4770 CPU, running at 3.4 GHz, with 16 GB of RAM under a 64-bit version of Windows 8. All algorithms were implemented in Java (JRE 1.8.0_191), using Eclipse (Eclipse IDE for Java Developers, Oxygen 4.7). We used IBM's Optimization Programming Language (OPL) to implement the CP model and applied the ILOG CPLEX CP Optimizer in version 12.7 as a CP solver.

In total, we implemented four approaches as illustrated in Table 4.4. Our TS benchmark

Table 4.4: Algorithms

Abbreviation	Algorithm	Reference
FaF	F&F framework	Section 4.4.2.5
FaFM	Modified F&F framework	Section 4.4.2.6
TS	Tabu search procedure	Section 4.5
CPA	IBM's CP solver on the CP model of WFJSP	Section 4.3

heuristic is presented in Algorithm 4.9. All elements of Algorithm 4.9 are in line with the setup of our F&F framework. It uses neighborhood switching and the structure of the tabu list (line 11) is identical to the one used in Algorithm 4.6. The tabu length is dynamically updated. It is set to the average number of critical (in case of $\phi \in \{1, 3\}$) or non-critical (in case of $\phi = 2$) operations of the current solution S. The input parameter τ specifies the termination criterion of the subroutine of lines 4–13.

The setup of the algorithms' parameters was derived in preliminary computational tests on two sets (small and large) of randomly generated test instances with a maximum number of 10 (small) or 30 (large) jobs, and a maximum number of 5 (small) or 25 (large) operations per job. In order to balance the trade-off between the solution quality and runtime over all instances, we set $\eta_0 = 40$, $\eta_1 = 20$, $\eta_2 = 10$ and $L = 15$ for FaF, $\eta_1 = 15$, $\eta_2 = 25$ and $L = 300$ for FaFM, and $\tau = 50$ for TS.

Thr remainder of section is split into two parts. In the first part, we analyse the performance of our F&F approaches (FaF and FaFM) on randomly generated test instances when compared with TS (as a benchmark heuristic using the same neighborhood structure) and CPA (as a standard solver benchmark). In the second part, we make use of benchmark instances from the literature in oder to compare the performance of our heuristics with existing metaheuristics from the literature. All test instances are available in supplementary files that accompany this paper (`https://doi.org/10.6084/m9.figshare.8082059`).

Input: Instance $Inst$ of WFJSP, parameter τ
Output: Solution S^*

▷ `Initialization phase`

1 Determine a feasible solution S of $Inst$ with the constructive procedure described in Section 4.4.2.1 and initialize $S^* := S$ and $S^{TBS} := S$;
2 Unlock all neighborhood operators in the specific order $\{1, 3, 2\}$ and initialize an empty tabu list;

▷ `Tabu search phase`

3 Get the first unlocked neighborhood operator ϕ, clear tabu list, and set $NoImprCounter := 0$;
4 **forall** *critical (if* $\phi \in \{1, 3\}$*) or non-critical (if* $\phi = 2$*) operations* $a_b \in V$ *of the solution graph of* S **do**
5 Call (modified, if $\phi = 3$) Algorithm 4.1 to determine the set N of neighboring solution graphs;
6 Discard all elements of N, where the corresponding transition from S is tabu if their makespan is not smaller than $\ddot{S}^*$ (aspiration criterion);
7 Select the best solution S^{NBS} among the remaining solutions in N. If no solution remains, i.e. if $N = \emptyset$, lock current neighborhood operator ϕ and go to line 16;
8 **end**
9 **if** $\ddot{S}^{NBS} < \ddot{S}^{TBS}$ **then** set $S^{TBS} := S^{NBS}$ and $NoImprCounter := 0$;
10 **else** set $NoImprCounter := NoImprCounter + 1$;
11 Update the tabu list;
12 Set $S := S^{NBS}$;
13 **if** $NoImprCounter < \tau$ **then** go to line 4;
14 **else if** $\ddot{S}^{TBS} < \ddot{S}^*$ **then** set $S^* := S^{TBS}$ and unlock all neighborhood operators in the specific order $\{1, 3, 2\}$;
15 **else** lock current neighborhood operator ϕ;
16 **if** *there is at least one unlocked neighborhood operator* **then** set $S := S^{TBS}$ and go to line 3;
17 **else** exit the procedure;

Algorithm 4.9 Tabu search heuristic

4.5.1 Random Testbed

Our random testbed is composed of 15 instances sets, denoted by wfjsp1–wfjsp15. Each set represents a different scenario and features 10 randomly generated instances with the parameter ranges illustrated in Table 4.5. The numbers of jobs, machines, and workers are fixed for the

Table 4.5: Parameters of random testbed

Inst. set	$\|I\|$	$\|M\|$	$\|W\|$	q_i	$\|M_{i_j}\|$	$\|W_m\|$
wfjsp1	5	3	3	$[2,4]$	$[1,2]$	2
wfjsp2	7	3	3	$[2,4]$	$[1,2]$	2
wfjsp3	10	5	5	$[2,4]$	$[1,3]$	3
wfjsp4	10	5	5	$[2,10]$	$[1,3]$	3
wfjsp5	15	5	5	$[5,10]$	$[2,3]$	3
wfjsp6	15	5	5	$[8,10]$	$[2,3]$	3
wfjsp7	20	10	10	$[5,10]$	$[1,3]$	5
wfjsp8	20	10	10	$[5,15]$	$[2,3]$	5
wfjsp9	30	15	15	$[5,10]$	$[2,5]$	8
wfjsp10	30	15	15	$[10,15]$	$[3,5]$	8
wfjsp11	15	5	4	$[8,10]$	$[2,3]$	3
wfjsp12	20	10	8	$[5,10]$	$[1,3]$	5
wfjsp13	20	10	8	$[5,15]$	$[2,3]$	5
wfjsp14	30	15	12	$[5,10]$	$[2,5]$	8
wfjsp15	30	15	12	$[10,15]$	$[3,5]$	8

instances of each set. The number of operations q_i was drawn from uniform distributions over the intervals given in the table for all jobs $i \in I$. Similarly, the eligible machines M_{i_j} for operations $i_j \in O_i$ of jobs $i \in I$ were randomly determined based on a random generation of their number $|M_{i_j}|$. The sets of eligible workers for each operation were generated indirectly by iterating over all machines $m \in M$ and randomly generating a subset of workers (denoted by W_m in Table 4.5) of a given size that is assumed to be able to process the respective machines. The processing times were then generated as follows (cf. Kress et al., 2019b). First, auxiliary integer parameters p_{i_j} for all jobs $i \in I$ and operations $i_j \in O_i$ were drawn from uniform distributions over the interval $[10, 100]$. Based on these parameters, we constructed varying processing times over the corresponding eligible machines $m \in M_{i_j}$ by drawing integer values $p_{i_j}^m$ from uniform distributions over $[\lfloor 0.9 \cdot p_{i_j} \rfloor, \lceil 1.1 \cdot p_{i_j} \rceil]$. In the last step, we incorporated dependencies on the workers $w \in W$ by drawing integer values $p_{i_j}^{m,w}$ from uniform distributions over $[\lfloor 0.9 \cdot p_{i_j}^m \rfloor, \lceil 1.1 \cdot p_{i_j}^m \rceil]$ for all $m \in M_w \cap M_{i_j}$, where M_w denotes the set of machines that worker $w \in W$ may process. It follows directly from the sets W_m for all $m \in M$. All remaining processing times were set to infinity. Note that the basic parameters of instance sets wfjsp6–wfjsp10 and wfjsp11–wfjsp15 differ solely in the staffing levels, i.e. in the ratio of the number of workers and the number of machines. The staffing level is 1 for sets wfjsp6–wfjsp10 and 0.8 for sets wfjsp11–wfjsp15.

Due to the non-deterministic elements of FaF, FaFM and TS, we ran these algorithms five times on each instance. CPA was run once on each instance with a time limit 3,600 seconds. All calls of each algorithm returned a feasible solution. For some given instance and some run of algorithm FaF, FaFM, or TS, we measure the quality of the solution returned by the algorithm with the quality ratio $100 \cdot (C_{\max} - C_{\max}^{\text{CPA}})/C_{\max}^{\text{CPA}}$, where $C_{\max}$ and $C_{\max}^{\text{CPA}}$ denote the makespan of the solution determined by the heuristic and CPA, respectively.

The computational results are presented in Table 4.6. For each instance set, the table

Table 4.6: Performance of the heuristic approaches on random testbed

	CPA		FaF			FaFM			TS		
Inst. set	opt. [%]	t_{avg} [s]	Q_{best}	Q_{avg}	t_{avg} [s]	Q_{best}	Q_{avg}	t_{avg} [s]	Q_{best}	Q_{avg}	t_{avg} [s]
wfjsp1	100	0.91	0.22	0.7	0.16	**0.11**	**0.64**	0.25	0.35	1.32	0.02
wfjsp2	90	374.27	0.78	1.38	0.29	**0.35**	**0.84**	0.35	0.59	1.42	0.03
wfjsp3	90	1034.58	2.21	3.93	0.57	**1.52**	**2.38**	0.69	2.57	4.13	0.06
wfjsp4	10	3272.8	3.88	5.61	2.85	**2.33**	**3.37**	3.6	3.74	5.49	0.66
wfjsp5	-	tl	**1.51**	**2.91**	24.81	2.51	3.27	17.68	2.54	3.44	7.42
wfjsp6	-	tl	**1.46**	**2.38**	41.41	1.97	2.57	22.93	1.85	2.76	10.5
wfjsp7	-	tl	6.6	8.77	19.18	**1.5**	**3.02**	18.3	4.9	7.45	7.36
wfjsp8	-	tl	6.67	8.21	45.39	**2.98**	**4.2**	34.28	6.18	7.74	21.79
wfjsp9	-	tl	3.61	4.67	70.8	**1.45**	**2.22**	33.82	3.53	4.48	32.19
wfjsp10	-	tl	3.51	4.03	266.23	**1.63**	**2.33**	99.22	3.19	3.77	136.94
wfjsp11	-	tl	**0.61**	1.55	56.57	0.83	**1.34**	31.01	1.12	2.38	17.86
wfjsp12	-	tl	2.48	3.97	22.52	**1.24**	**1.92**	22.34	3.49	4.27	10.34
wfjsp13	-	tl	2.95	3.81	49.82	**1.46**	**2.11**	46.87	3.2	3.81	31.84
wfjsp14	-	tl	2.27	2.88	84.04	**1.81**	**2.32**	45.48	2.95	3.43	47.91
wfjsp15	-	tl	2.05	2.64	349.33	**1.38**	**2.05**	116.04	2.4	2.81	295.86

presents information about the percentage of test instances that were solved to optimality with CPA within the time limit (column "opt."), the average quality ratios of the heuristic approaches over all runs and all instances of the set (columns "Q_{avg}"), the average values of the best quality ratios among the five runs of the heuristics for each instance of the set (columns "Q_{best}"), as well as the average runtimes (columns "t_{avg}") of the algorithms. Entries "tl" indicate that the time limit was reached by CPA for all instances of the set. Bold entries highlight the best heuristic approaches with respect to the quality ratios.

We find that CPA is able to determine optimal solutions only for the small instances of the sets wfjsp1–wfjsp4 within the time limit. For instance sets wfjsp1–wfjsp3, CPA solves the majority of instances to optimality. Here, all heuristics provide high quality solutions. In general, however, both F&F approaches tend to outperform TS. Moreover, it can be concluded that FaFM provides the overall best performance among the heuristics with respect to the solution quality. This effect is particularly pronounced for instances of larger size. The average computational times of the F&F approaches are quite similar, except for the largest instance sets (wfjsp9–wfjsp10 and wfjsp14–wfjsp15), where FaFM terminates faster than FaF. Based on these results, it can be concluded that it certainly pays off to locally decide on the transitions applied within the tree search procedure. Finally, when comparing the results for instances wfjsp11–wfjsp15 and wfjsp6–wfjsp10, we observe that smaller staffing levels tend to result in smaller quality ratios.

It is interesting to additionally compare the performance of CPA when the time limit for some instance is set to the average time needed by the five corresponding calls of FaFM and when the parallel processing mode in the CPLEX CP Optimizer is deactivated for the sake of comparability. This is subject of Table 4.7, where quality ratios are determined in relation to the previous CPA calls. We observe that in this setting, i.e. when time becomes a limiting factor as it is, for example, often the case in rolling horizon based planning approaches, the standard CP solver provides feasible solutions for all considered instances. However, FaFM clearly outperforms the standard solver. This provides first evidence for the fact that FaFM may also outperform

Table 4.7: Comparison of FaFM and CPA with FaFM-based time limit

	CPA	FaFM		
Inst. set	Q_{avg}	Q_{best}	Q_{avg}	t_{avg} [s]
wfjsp1	0.03	0.11	0.64	0.25
wfjsp2	0.89	0.35	0.84	0.35
wfjsp3	4.94	1.52	2.38	0.69
wfjsp4	5.95	2.33	3.37	3.6
wfjsp5	4.32	2.51	3.27	17.68
wfjsp6	3.48	1.97	2.57	22.93
wfjsp7	7.46	1.5	3.02	18.3
wfjsp8	3.63	2.98	4.2	34.28
wfjsp9	3.79	1.45	2.22	33.82
wfjsp10	0.93	1.63	2.33	99.22
wfjsp11	3.58	0.83	1.34	31.01
wfjsp12	4.28	1.24	1.92	22.34
wfjsp13	3.41	1.46	2.11	46.87
wfjsp14	5.17	1.81	2.32	45.48
wfjsp15	2.14	1.38	2.05	116.04

state-of-the-art metaheuristics from the literature.

4.5.2 Literature Instances

In order to compare the performance of our approaches with previously published algorithms, we make use of benchmark instances provided by Lei and Guo (2014), who extend two FJSP benchmark sets with worker related information. This includes ten instances (MK1–MK10) of the set by Brandimarte (1993) with 10 to 20 jobs, 4 to 15 machines, 3 to 15 operations per job, 2 to 6 eligible machines for each operation, and processing times that range from 1 to 20 time units. Moreover, this includes twelve instances (DP1–DP12) of the set by Dauzère-Pérès and Paulli (1997) with 10 to 15 jobs, 5 to 8 machines, and 15 to 25 operations per job. The set of eligible machines of the operations of the instances within this latter set has been randomly constructed, by assuming that each machine is eligible with a 0.1 to 0.5 probability. The processing times range from 10 to 100 time units. The worker related information of these instances provided by Lei and Guo (2014) is summarized in Table 4.8. In order to keep the

Table 4.8: Worker related information of the literature instances (Lei and Guo, 2014)

Literature instance	$\|W\|$	M_w
MK1–MK2	4	$M_1 = \{1,3,5\}$, $M_2 = \{2,4,5\}$, $M_3 = \{1,4,6\}$, $M_4 = \{2,3,6\}$
MK3–MK4; DP7–DP12	6	$M_1 = \{1,5\}$, $M_2 = \{2,4\}$, $M_3 = \{1,4,6\}$, $M_4 = \{2,3,6,7\}$, $M_5 = \{6,7,8\}$, $M_6 = \{5,8\}$
MK5	3	$M_1 = \{1,3,4\}$, $M_2 = \{2,4\}$, $M_3 = \{1,2,3\}$
MK6,MK10	8	$M_1 = \{1,8,10\}$, $M_2 = \{2,7,11\}$, $M_3 = \{3,4,6,11\}$, $M_4 = \{2,9,12,13\}$, $M_5 = \{6,7,8,15\}$, $M_6 = \{5,8,10\}$, $M_7 = \{4,9,14,15\}$, $M_8 = \{1,3,10,14\}$
MK7; DP1–DP6	4	$M_1 = \{1,3,5\}$, $M_2 = \{2,4\}$, $M_3 = \{3,4\}$, $M_4 = \{1,2,5\}$
MK8–MK9	6	$M_1 = \{1,3,5\}$, $M_2 = \{2,4,9\}$, $M_3 = \{3,4,8,10\}$, $M_4 = \{1,7,9\}$, $M_5 = \{5,6,7\}$, $M_6 = \{2,4,8,10\}$

notation simple, we will not change the labels of the augmented instance sets, and refer to them by MK1–MK10 and DP1–DP12. Table 4.8 includes information on the number of workers and the sets of machines that can be operated by the workers. Unfortunately, the generation of the processing times is only very briefly summarized by Lei and Guo (2014). Moreover, the instances

are not available publicly. Hence, based on the related information given by Lei and Guo (2014), we made use of the following procedure. For all workers $w \in W$, all operations $i_j \in O_i$ of jobs $i \in I$, and all $m \in M_w \cap M_{i_j}$, we drew the processing times from a uniform distributions over the intervals $[\bar{p}^m_{i_j}, \bar{p}^m_{i_j} + \delta_{i_j}]$, where $\bar{p}^m_{i_j}$ are the original processing time stated by Brandimarte (1993) or Dauzère-Pérès and Paulli (1997), and the values δ_{i_j} were drawn from a uniform distributions over the interval $[2, 8]$. All remaining processing times were set to infinity.

With respect to the relevant metaheuristic algorithms that have been presented in the literature (see Section 4.1.1, Table 4.1), Peng et al. (2018) and Xianzhou and Zhenhe (2011) provide only basic case studies or examples in order to evaluate their approaches. Zhang et al. (2015) and Yazdani et al. (2015) do not provide detailed information on their test instances. Thus, we can only compare our approaches with the variable neighborhood search approach (denoted by VNS) by Lei and Guo (2014) and the knowledge-guided fruit fly optimization algorithm (denoted by KF) by Zheng and Wang (2016). As Zheng and Wang (2016) claim that KF outperforms "existing algorithms" (including VNS), this is a comparison with the state-of-the-art. Both, Lei and Guo (2014) and Zheng and Wang (2016), make use of the above benchmark instances (with potentially slightly different processing times) and present detailed computational results derived on a computer comparable to ours.

In order to evaluate our results, we use a lower bound on the makespan introduced by Lei and Guo (2014), which we simplify to take account of the facts that all jobs are available at time zero and that $|M| \leq |I|$ and $|W| \leq |I|$ for all considered instances. For a given instance of WFJSP, the bound is defined as follows:

$$LB := \max \left(\max_{i \in I} \left(\sum_{i_j \in O_i} p^{\min}_{i_j} \right), \left\lceil \frac{P}{|M|} \right\rceil, \left\lceil \frac{P}{|W|} \right\rceil \right).$$

Here, $p^{\min}_{i_j} := \min_{m \in M_{i_j}} p^{m,\min}_{i_j}$ (see Section 4.4.1.2) for all $i \in I$, $i_j \in O_i$, and $P := \sum_{i \in I} \sum_{i_j \in O_i} p^{\min}_{i_j}$. Note that, for the sake of brevity, we do not explicitly state the concrete instance in the definition of the bound.

As in the previous section, we initiated five runs of the heuristics FaF, FaFM, and TS on each instance. Moreover, we ran CPA with the FaFM-based time limit as introduced in the context of Table 4.7. All calls of the algorithms returned feasible solutions. As before, we measure the quality of a solution with makespan $C_{\max}$ with the corresponding quality ratio $100 \cdot (C_{\max} - LB)/LB$. The computational results are presented in Table 4.9. The table includes information on the best (columns "Q_{best}") and the average (columns "Q_{avg}") quality ratios over all calls of some algorithm on the respective instance, as well as the average runtime of the heuristic approaches (columns "t_{avg}"). As mentioned above, the values for KF and VNS are based on the values (i.e. lower bounds, makespan, runtime) reported by the respective authors. Nevertheless, as our process of generating processing times mimics their procedure as close as possible, the lower bounds stated by these authors only slightly differ from the ones in the table, so that our results can be utilised for detecting tendencies and first insights on the relative performance of

Table 4.9: Performance of heuristic approaches on literature instances

		CPA	FaF			FaFM			TS			KF		VNS	
Inst.	LB	Q_{best}	Q_{best}	Q_{avg}	t_{avg} [s]	Q_{best}	Q_{avg}	t_{avg} [s]	Q_{best}	Q_{avg}	t_{avg} [s]	Q_{avg}	t_{avg} [s]	Q_{avg}	t_{avg} [s]
MK1	66	18.2	10.6	12.7	2.9	10.6	11.8	4.2	12.1	15.5	0.5	8.2	3.1	17.3	4.3
MK2	65	10.8	7.7	11.1	6.4	10.8	12.6	5.4	10.8	12.6	3	7.7	3.2	22.6	4.4
MK3	182	37.9	34.6	37.6	25	34.1	34.7	22	34.6	36.6	11.1	41	13.4	48.8	31
MK4	80	32.5	26.3	31	11.3	22.5	25.8	11.5	30	32.5	1.1	26.3	5.3	32.5	5.4
MK5	295	5.8	3.4	5.7	31.8	3.4	4.2	37.5	4.4	5.6	13.1	9.2	15.1	12.7	34.8
MK6	78	41	64.1	71.8	31.1	44.9	51.3	29.1	55.1	65.4	13	36.9	11.2	49.7	22.2
MK7	213	11.7	8.9	12.4	26.2	9.9	12.3	18.5	11.3	13.1	8.3	13.4	10.5	24.7	21.9
MK8	488	28.9	29.1	31	72.6	28.1	28.5	51.1	29.1	30.6	11.8	19	59.2	22.9	114
MK9	443	20.3	12.9	14.1	292	11.1	12.8	74.8	12.6	14.1	133	28.2	52.9	35.1	117
MK10	289	19.7	18	23.6	339.3	13.8	16.1	73.4	20.1	25.9	72.7	37.1	49.4	42.2	89.5
DP1	2881	5.1	8.1	9.7	31.4	4.6	5.2	64.1	7.7	9	8.6	11.2	35.2	13.5	61.4
DP2	2881	1.8	2.3	3.4	60.9	1.9	2.4	32.5	2.3	3	19.9	9.9	35.3	12.9	62.1
DP3	2881	1.5	1.7	3.1	74.1	2.1	2.4	42	1.8	3.4	41.4	10.7	35.6	10.6	61.6
DP4	2862	3.7	7.7	9.9	38.6	3.7	4.7	64.9	7.4	8.4	8.5	13.1	35.8	15.3	60.8
DP5	2832	2.7	3	3.6	65.3	3	3.6	41.1	3.1	4.4	17.6	11.4	35.1	13.2	60.2
DP6	2799	3.6	3.3	4.9	106.8	3.8	4.1	53.7	3.2	4.1	73.6	10.2	34.4	13.8	60.8
DP7	2843	3.8	7.7	11.1	78.4	4.4	5.4	88.3	6.5	9.7	17.7	27	52.3	29.3	169
DP8	2835	1.8	2	3.3	316.6	2.6	3.1	62	2.2	4	64.1	25.2	52.1	29.5	164
DP9	2824	2.5	1.8	3.8	484.2	2.4	3.7	78.3	2.2	5	104.8	27.3	52.4	29.2	165
DP10	2840	5.2	9.4	11.1	89.1	6.5	7.7	67.6	7.5	10.2	16	26.9	52.6	31.1	164
DP11	2786	4.1	3.4	4	379.1	4.1	4.3	76.5	3.7	5.2	64.5	31.2	53	34.3	166
DP12	2723	5.8	3.8	5.8	771.2	5.2	5.5	82.4	4.6	6.7	171	30.1	53.2	32.6	168

the approaches. Figure 4.4 additionally plots the values given in Table 4.9 for the approaches introduced in this paper.

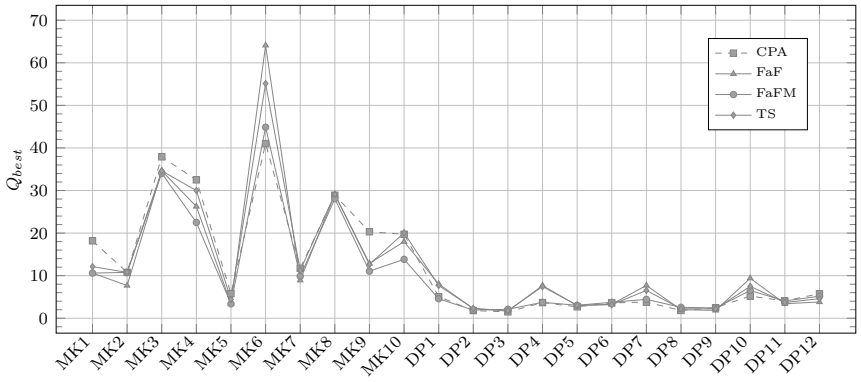


Figure 4.4: Performance of heuristic approaches on literature instances

As in case of the random testbed, FaFM tends to outperform FaF and TS. We furthermore observe that, while especially KF uses sightly less computational time than FaFM on average, FaFM tends to outperform KF and VNS for most instances with respect to solution quality, especially in case of the DP instances. Moreover, the runtimes of FaFM are in ranges that allow its usage in real-world scenarios, where time usually is a limiting factor. FaF, FaFM, and TS also prove to be competitive with the standard solver (with FaFM-based time limit) for the MK instances, which is in line with our results for the random testbed. In case of the DP instances, CPA sometimes also performs slightly better than our F&F approaches. We believe that this is a result of the small average numbers of eligible machines for the operations, that allow the CP solver to quickly determine promising allocation decisions of machines to operations. However,

the difference in the performance of the standard solver and our F&F heuristics is not nearly as large as in case of KF ans VNS. We can therefore conclude that, overall, FaFM tends to outperform KF and tends to be competitive when compared with the use of the standard CP solver provided by CPLEX when runtime becomes a limiting factor.

4.6 Conclusion

In this paper, we have addressed a flexible job shop scheduling problem that aims at makespan minimization and takes account of a heterogeneous workforce by making use of processing times that do not only depend on the machine, but also on the specific worker that operates the machine while it processes some operation. We have developed two filter-and-fan based heuristic solution approaches. These methods combine a local search procedure with a tree search procedure that generates compound transitions in order to explore larger neighborhoods to overcome locally optimal solutions. They make use of a decomposition of the problem that allows to make use of a neighborhood structure that has formerly shown to perform well when machine operator restrictions are not considered. In a computational study, we have shown that our heuristic approaches tend to outperform existing metaheuristic approaches from the literature. They have also proven competitive when compared with a standard constraint programming solver. With respect to the setup of the filter-and-fan framework, we found that it pays off to make local decisions when generating solutions within the tree search procedure instead of relying on transitions lists that are generated during local search.

Acknowledgements
This work has been supported by the European Union and the state North Rhine-Westphalia through the European Fund for Regional Development (EFRD). It has been conducted as part of the project "EKPLO: Echtzeitnahes kollaboratives Planen und Optimieren" (EFRE-0800463).

Bibliography

Beach, R., Muhlemann, A. P., Price, D. H. R., Paterson, A., and Sharp, J. A. (2000). A review of manufacturing flexibility. *European Journal of Operational Research*, 122(1):41–57.

Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., and Węglarz, J. (2007). *Handbook on Scheduling: From Theory to Applications*. Springer, Berlin.

Błażewicz, J., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183.

Brucker, P. and Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375.

Chaudhry, I. A. and Khan, A. A. (2016). A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3):551–591.

Dauzère-Pérès, S. and Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306.

De Bruecker, P., Van den Bergh, J., Beliën, J., and Demeulemeester, E. (2015). Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243(1):1–16.

Dorndorf, U., Jaehn, F., and Pesch, E. (2008). Modelling robust flight-gate scheduling as a clique partitioning problem. *Transportation Science*, 42(3):292–301.

Giffler, B. and Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503.

Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1):223–253.

Glover, F. (1998). A template for scatter search and path relinking. In Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., and Snyers, D., editors, *Artificial Evolution*, pages 1–51, Berlin, Heidelberg. Springer.

Gong, G., Deng, Q., Gong, X., Liu, W., and Ren, Q. (2018a). A new double flexible job-shop scheduling problem integrating processing time, green production, and human factor indicators. *Journal of Cleaner Production*, 174:560–576.

Gong, X., Deng, Q., Gong, G., Liu, W., and Ren, Q. (2018b). A memetic algorithm for multi-objective flexible job-shop problem with worker flexibility. *International Journal of Production Research*, 56(7):2506–2522.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

He, J., Chen, X., and Chen, X. (2016). A filter-and-fan approach with adaptive neighborhood switching for resource-constrained project scheduling. *Computers & Operations Research*, 71:71–81.

IBM (2016). IBM ILOG CPLEX optimization studio 12.7.0: Scheduling examples. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.ide.help/OPL_Studio/usroplexamples/topics/opl_cp_examples_scheduling.html`. Last accessed 2018-12-11.

Jain, A., Jain, P. K., Chan, F. T. S., and Singh, S. (2013). A review on manufacturing flexibility. *International Journal of Production Research*, 51(19):5946–5970.

Kress, D., Boysen, N., and Pesch, E. (2017). Which items should be stored together? A basic partition problem to assign storage space in group-based storage systems. *IISE Transactions*, 49(1):13–30.

Kress, D., Meiswinkel, S., and Pesch, E. (2019a). Straddle carrier routing at seaport container terminals in the presence of short term quay crane buffer areas. *European Journal of Operational Research*, 279(3):732–750.

Kress, D. and Müller, D. (2019). Mathematical models for a flexible job shop scheduling problem with machine operator constraints. *IFAC-PapersOnLine*, 52(13):94–99.

Kress, D., Müller, D., and Nossack, J. (2019b). A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, 41(1):179–217.

Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250.

Lang, M. and Li, H. (2011). Research on dual-resource multi-objective flexible job shop scheduling under uncertainty. In *Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce*, pages 1375–1378. IEEE.

Lei, D. and Guo, X. (2014). Variable neighbourhood search for dual-resource constrained flexible job shop scheduling. *International Journal of Production Research*, 52(9):2519–2529.

Lei, D. and Tan, X. (2016). Local search with controlled deterioration for multi-objective scheduling in dual-resource constrained flexible job shop. In *Proceedings of the 28th Chinese Control and Decision Conference*, pages 4921–4926. IEEE.

Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140.

Liu, X. X., Liu, C. B., and Tao, Z. (2011). Research on bi-objective scheduling of dual-resource constrained flexible job shop. *Advanced Materials Research*, 211–212:1091–1095.

Mastrolilli, M. and Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20.

Paksi, A. B. N. and Ma'ruf, A. (2016). Flexible job-shop scheduling with dual-resource constraints to minimize tardiness using genetic algorithm. In *Proceedings of the 2nd International Manufacturing Engineering Conference and 3rd Asia-Pacific Conference on Manufacturing Systems*, page 012060. IOP Publishing.

Peng, C., Fang, Y., Lou, P., and Yan, J. (2018). Analysis of double-resource flexible job shop scheduling problem based on genetic algorithm. In *Proceedings of the 15th International Conference on Networking, Sensing and Control*, pages 1–6. IEEE.

Pesch, E. and Glover, F. (1997). TSP ejection chains. *Discrete Applied Mathematics*, 76(1):165–181.

Rego, C. and Duarte, R. (2009). A filter-and-fan approach to the job shop scheduling problem. *European Journal of Operational Research*, 194(3):650–662.

Rego, C. and Glover, F. (2002). Local search and metaheuristics for the traveling salesman problem. In Gutin, G. and Punnen, A., editors, *The traveling salesman problem and its variations*, pages 309–368, Boston. Springer.

Rego, C. and Glover, F. (2010). Ejection chain and filter-and-fan methods in combinatorial optimization. *Annals of Operations Research*, 175(1):77–105.

Sprecher, A., Kolisch, R., and Drexl, A. (1995). Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80(1):94–102.

Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117.

Treleven, M. (1989). A review of the dual resource constrained system research. *IIE Transactions*, 21(3):279–287.

Vallikavungal Devassia, J., Salazar-Aguilar, M. A., and Boyer, V. (2018). Flexible job-shop scheduling problem with resource recovery constraints. *International Journal of Production Research*, 56(9):3326–3343.

Wu, R., Li, Y., Guo, S., and Xu, W. (2018). Solving the dual-resource constrained flexible job shop scheduling problem with learning effect by a hybrid genetic algorithm. *Advances in Mechanical Engineering*, 10(10):1–14.

Xianzhou, C. and Zhenhe, Y. (2011). An improved genetic algorithm for dual-resource constrained flexible job shop scheduling. In *Proceedings of the 4th International Conference on Intelligent Computation Technology and Automation*, pages 42–45. IEEE.

Xu, J., Xu, X., and Xie, S. Q. (2011). Recent developments in dual resource constrained (DRC) system research. *European Journal of Operational Research*, 215(2):309–318.

Yazdani, M., Zandieh, M., Tavakkoli-Moghaddam, R., and Jolai, F. (2015). Two meta-heuristic algorithms for the dual-resource constrained flexible job-shop scheduling problem. *Scientia Iranica - Transactions E*, 22(3):1242–1257.

Zhang, J., Wang, W., and Xu, X. (2015). A hybrid discrete particle swarm optimization for dual-resource constrained job shop scheduling with resource flexibility. *Journal of Intelligent Manufacturing*, 28(8):1961–1972.

Zhang, J., Wang, W., Xu, X., and Jie, J. (2013). A multi-objective particle swarm optimization for dual-resource constrained shop scheduling with resource flexibility. In *Proceedings of the IEEE Symposium on Computational Intelligence for Engineering Solutions*, pages 29–34. IEEE.

Zheng, X.-L. and Wang, L. (2016). A knowledge-guided fruit fly optimization algorithm for dual resource constrained flexible job-shop scheduling problem. *International Journal of Production Research*, 54(18):5554–5566.

Chapter 5

Semiconductor Final-Test Scheduling under Setup Operator Constraints

Abstract

We consider a semiconductor final-test scheduling problem that aims at minimizing total weighted tardiness. In contrast to previous studies on this problem, we explicitly take account of the need to assign human operators to setup operations. We present a mixed integer programming model and a decomposition based heuristic solution approach. In an extensive computational study based on real-world problem instances that mimic settings at our industry partner, we show that the latter approach clearly outperforms a standard solver when computational time is limited. Based on this result, we provide decision support for managers by analyzing the capability and effect of rescheduling jobs in the presence of a highly dynamic environment with frequently changing customer requests and common test machine failures.

Authors	Dominik Kress[a,b] dominik.kress@hsu-hh.de David Müller[a] david.mueller@uni-siegen.de [a] *University of Siegen, Management Information Science, Kohlbettstraße 15, 57068 Siegen, Germany* [b] *Helmut Schmidt University - University of the Federal Armed Forces Hamburg, Business Administration, especially Procurement and Production, Friedrich-Ebert-Damm 245, 22159 Hamburg, Germany*
Full citation	Kress, D. and Müller, D. (2020). Semiconductor final-test scheduling under setup operator constraints. Working Paper. University of Siegen

5.1 Introduction and Overview

Today, semiconductor components are omnipresent in a wide range of products. They are essential elements of, for example, smartphones, tablets, flat-screen monitors, sophisticated cars and aircrafts, and many medical devices (PwC, 2012). Semiconductor companies focusing on the automotive industry are facing especially strong competition and high customer expectations (PwC, 2013). It is therefore not surprising that operational aspects of semiconductor manufacturing are becoming increasingly important to these companies (Deng et al., 2010; Uzsoy et al., 1992a).

During an industry project with a developer and manufacturer of semiconductor-based system solutions in North Rhine-Westphalia (Germany), an optimization problem in the above context was brought to our attention. The company's customers are mostly automotive manufacturers with extremely high requirements regarding the quality and flawless functionality as well as on-time delivery of the semiconductor devices. It is therefore of major importance to carefully test every single device based on a schedule that allows on-time delivery of customer orders in the presence of scarce test machine and labor resources in a dynamic environment with frequently changing customer requests and common test machine failures. Hence, the focus of our industry project, as well as the scope of this paper, was on a scheduling problem in one of the company's final-test divisions, where packaged semiconductors are subjected to functional tests.

5.1.1 The Semiconductor Manufacturing Process

The process of manufacturing semiconductor devices can be divided into *front-end* and *back-end* operations; see Figure 5.1, where optional operations are depicted by dash-dotted boxes. The

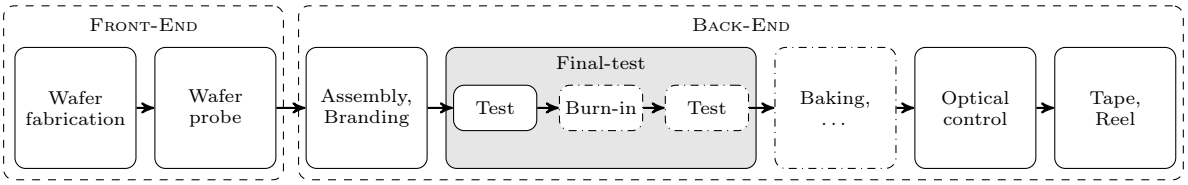


Figure 5.1: Semiconductor manufacturing process

process is presented in detail by Uzsoy et al. (1992a) and Lee et al. (1992) and may, of course, slightly deviate among different companies. We refer the interested reader to these articles and restrict ourselves to briefly summarizing the process with a focus on the specific setting at our industry partner.

Front-end operations include two major steps. Fist, *wafer fabrication* develops the actual integrated circuits on silicon wafers. Each wafer may contain hundreds or even thousands of circuits. Second, in *wafer probe*, the wafers are subjected to tests in order to detect defective circuits. The wafers are then cut into individual circuits. Non-defect circuits are packaged into branded plastic or ceramic cases with attached leads. These latter operations are referred to as *assembly and branding* and form the first steps in the back-end. The packaged semiconductors

are then forwarded to the *final-test* stage in lots of varying sizes. We will henceforth refer to these lots of semiconductor packages as *jobs*. In the final-test, each device is subjected to multiple functional tests at different temperatures on specific test machines in order to guarantee defect free products. If required by a job's customer, an optional subsequent *burn-in* operation followed by additional functional tests is performed. During the burn-in operation, the circuits are loaded into ovens and are subjected to thermal stress for multiple hours or even days in order to be able to detect latent defects. Some devices then go through optional steps, e.g., in order to remove moisture (*baking*). Finally, the devices are inspected and rotated in a predefined position in an *optical control* stage, and are then packed onto reels with a carrier tape for transportation to the customers (*tape and reel*).

5.1.2 Final-Test Scheduling

In this article, we focus on the final-test stage of semiconductor processing. While, as mentioned above, final-test operations of a job may include a burn-in operation and additional subsequent tests, the broad majority of jobs at our industry partner solely undergoes (multiple) functional testing operations. We were therefore asked to restrict ourselves to the scheduling of jobs that are exclusively composed of these latter operations. With respect to the practical applicability of our model, this seems to be a reasonable assumption because burn-in takes place at large (non-bottleneck) ovens with relatively long processing times when compared to the processing times of functional tests, such that they can be considered in a higher-level scheduling problem (as, for instance, presented by Kim et al., 2011; Lee et al., 1992) that determines partial jobs, being solely composed of functional testing operations, that can then be included into our (lower-level) optimization problem.

In order for a test machine (also referred to as a test cell) to be eligible for performing a specific functional test for a given device, it must be a feasible combination of three hardware components (see also Hao et al., 2014; Uzsoy et al., 1992b). The *tester* provides the logic needed to test a device (test software) and is able to stimulate the device in order to decide whether or not it is defect. The *handler* mechanically handles the device at the required temperature for a specific test. Further accessories, especially the *adapter*, define the interface between tester and device. Some handler-adapter combinations are able to handle multiple devices in parallel (*multisite testing*). At our industry partner, some combinations allow multisite testing of up to eight devices.

As mentioned above, a device is usually subjected to multiple functional tests at different temperatures on different test cells. The time needed to prepare a test cell for the execution of a functional test is referred to as a *setup time*. It can, for example, result from the need to change the adapter or handler, load a new software, or wait for bringing the handler to the required temperature. Setup times are *sequence-dependent*. That is, they do not only depend on the functional test to be performed on a specific test cell configuration but also on the preceding functional test and configuration. Setups are executed by human *setup operators* and may take a significant amount of time in the range of multiple hours. At our industry partner, setup

operators are considered to be a scarce resource, so that a detailed planning of setup operations is required.

The testing of semiconductor devices takes place in a *dynamic environment*, in which changes regarding the parameters of the final-test scheduling problem occur regularly. Most important, customers fairly frequently request a change of the due dates of their orders (so called "emergency orders", cf. also Bard et al., 2012). Furthermore, as also observed by Freed et al. (2007), "machine failures are common and unpredictable." Typical issues at our industry partner are defect handlers or adapters. Defect devices must be replaced on their current test cells, so that the testing process can continue while the defect device is repaired by a technician. This may take multiple hours or even days. As the purchase of additional devices is a main cost driver while meeting the customer due dates is highly important with respect to service quality, it is necessary to carefully determine the required number of devices at a testing facility.

The main goal of our industry partner is on-time delivery of customer orders. We therefore consider the objective of minimizing the total weighted tardiness. We refer to the problem under consideration as the *semiconductor final test scheduling problem with setup operator constraints* and denote it by SFTPS.

5.1.3 Related Literature

SFTPS is a generalization of the well-known *job shop scheduling problem*, which we assume the reader to be familiar with (see, e.g., Blazewicz et al., 2019). For comprehensive surveys on scheduling problems with setup considerations, including job shop settings and their generalizations, we refer the reader to Allahverdi (2015), Allahverdi et al. (1999), Allahverdi et al. (2008), and Zhu and Wilhelm (2006).

There exists a multitude of articles that focus on scheduling problems in the field of semiconductor manufacturing. Most of this literature focusses on front-end operations (Bard et al., 2012). Detailed reviews and overviews are given by Freed et al. (2002), Gupta and Sivakumar (2006), Mathirajan and Sivakumar (2006), Mönch et al. (2011), and Uzsoy et al. (1992a, 1994), so that we restrict ourselves to giving a brief overview of research on semiconductor final-test scheduling in the remainder of this section.

On a fairly general level, Freed et al. (2007) discuss trade-offs between in-house development of a scheduling system and buying a software solution for scheduling semiconductor testing operations.

With respect to concrete variants of semiconductor final-test scheduling problems, the broad majority of existing approaches assumes labor to "not [being] a constraining factor" (Deng et al., 2010) and therefore deviates from our industry case, where setup operators are considered to be a scarce resource. In this stream of research, Ovacik and Uzsoy (1992, 1994, 1995, 1996) develop decomposition based methods and rolling horizon procedures. Uzsoy et al. (1992b) and Uzsoy et al. (1991) describe and refine an approximation methodology based on the shifting bottleneck approach of Adams et al. (1988). Zhang et al. (2011) present a machine learning approach. Zhang et al. (2006) develop and make use of a mixed integer program (MIP) to

analyze capacity planning issues at Intel Shanghai. Chen et al. (1995) and Chen and Hsia (1997) describe Lagrangean relaxation approaches. A Petri net based approach is presented by Xiong and Zhou (1998). Pearn et al. (2004) adapt network algorithms designed for the vehicle routing problem. A simulation study is performed by Lin et al. (2004). Finally, metaheuristic approaches are, amongst others, presented by Deng et al. (2010), Herrmann et al. (1995), Hao et al. (2014), Sang et al. (2018), Wang and Wang (2015), Wang et al. (2015), Wu and Chien (2008), Wu et al. (2012), and Zheng et al. (2014).

In contrast to the aforementioned articles, Bard et al. (2012) state that "in practice, one of the biggest obstacles [...] is crew availability," and thus consider a setting which is in line with our industry case. In a real-time control model, the authors develop two procedures for prioritizing choices whenever a setup operation is possible. However, the considered model addresses a fairly specific industry case at Texas Instruments. We are not aware of further articles that explicitly address the incorporation of setup operators into semiconductor final-test scheduling problems.

5.1.4 Contribution and Article Overview

As illustrated in Section 5.1.3, the vast majority of existing approaches for semiconductor final-test scheduling does not explicitly address the incorporation of setup operators, which is the main research gap that our study aims to address. As for the incorporation of the dynamic environment, it is of utmost importance for our industry partner to be able to make use of the expert knowledge of the planners when making rescheduling decisions. Hence, we decided against internalizing these decisions into our model or to make use of stochastic processing times or due dates. We rather assume that lots of devices are non-separable (as usually done in the literature; see, e.g., Lee et al., 1992; Uzsoy et al., 1991) and that due dates and processing times are deterministic. However, our solution approaches are designed to be applied in a rolling horizon planning approach (see also Ovacik and Uzsoy, 1994, 1995), which allows to manually initiate a rescheduling of jobs, e.g., with manually split lots that allow parallel testing on multiple machines in order to take account of changing due dates, or with adapted processing times or a restricted set of resources in case of hardware failures. In this context, note that it is well known that the job shop scheduling problem is strongly NP-hard for minimizing the total tardiness (Graham et al., 1979; Lenstra and Rinnooy Kan, 1979). Hence, SFTPS is strongly NP-hard as well. Given the fact that we will have to be able to quickly compute solutions for large problem instances to ensure real-world applicability in a rolling horizon approach, exact solution approaches do not seem to be a promising research direction. Therefore, we propose a tabu search approach.

The remainder of this article is structured as follows. In Section 5.2, we define SFTPS in detail and present a corresponding MIP. Next, in Section 5.3, we introduce our tabu search approach. It is evaluated in an extensive computational study in Section 5.4, where we derive managerial implications on the application of our heuristic in dynamic environments. The paper closes with a conclusion in Section 5.5.

5.2 Detailed Problem Statement

In the following Section 5.2.1, we define the notation and provide a formal definition of SFTPS. A corresponding MIP is presented in Section 5.2.2.

5.2.1 Notation and Definition of the Problem

We assume that the *planning horizon* of length T is divided into a finite number of intervals (time slots) $[t-1, t]$, $t = 1, \ldots, T$, of equal length and refer to the length of a time interval as a time unit. All time parameters that are introduced below, i.e., due dates, processing times, and setup times, are assumed to be integral multiples of a time unit and can therefore be specified by natural numbers.

We are given a set $J = \{1, \ldots, n\}$ of *jobs*. Each job $j \in J$ corresponds to a non-separable lot of devices and is associated with a *weight* $w_j \in \mathbb{N}$ and a *due date* $d_j \in \mathbb{N}$. Furthermore, each job $j \in J$ is associated with a set of q_j *operations* $O_j = \{j_1, \ldots, j_{q_j}\}$ that have to be assigned to and sequenced on eligible test cells. The processing of operations may not be preempted. The sets O_j are assumed to be ordered for all $j \in J$, which relates to the fact that, for any pair of operations $j_i, j_l \in O_j$ with $i < l$, j_i must be completed before the processing of j_l may start. We define $O = \bigcup_{j \in J} O_j$.

Each operation $j_i \in O$ must be processed by exactly one *test cell*. As described above, each test cell is a combination of three *resource types* (tester, handler, adapter). We denote the set of these types by $K = \{1, 2, 3\}$. There exist different *classes* of each resource type. Each class represents a specification of a resource type, e.g., a specific type of handler. The set of classes of resource type $k \in K$ is denoted by $R^k = \{1, \ldots, r^k\}$, $|R^k| = r^k$. The number of identical copies of resource class $i \in R^k$ of type $k \in K$ is denoted by q_i^k. Hence, a feasible schedule uses at most q_i^k entities of this resource class at a given point in time. Each test cell combines resources of all types and is therefore also referred to as a *machine configuration*. At our industry partner, each tester is associated to a unique location in the testing facility, which we implement by assuming $q_i^1 = 1$ for all $i \in R^1$. Thus, in a feasible schedule, each machine configuration can process at most one operation at a time. The set of all potential machine configurations is denoted by $M = R^1 \times R^2 \times R^3$. The set of machine configurations that require resource class $i \in R^k$ of type $k \in K$ is denoted by $M_i^k \subseteq M$. Furthermore, for some $k \in K$, we denote the element of the set R^k that is used in machine configuration $m \in M$ by $m[k]$.

Each operation $j_i \in O$ is associated to a set $M_{j_i} \subseteq M$ of *eligible machine configurations*, on which it can be processed in order to be completed. Its *processing time* $p_{j_i}^m \in \mathbb{N}$ does not only depend on the operation itself, but also on the configuration $m \in M_{j_i}$, which allows the incorporation of multisite testing into our model (see, e.g., Freed and Leachman, 1999).

In order to take account of machine configurations at $t = 0$ and operations that are being processed at the beginning of the planning horizon, we define *dummy operations* $0_1, \ldots, 0_{r^1}$ and set $\hat{O} = O \cup \{0_1, \ldots, 0_{r^1}\}$. The set M_{0_i} of eligible machines of some dummy operation 0_i, $i \in \{1, \ldots, r^1\}$, solely includes the machine configuration which is "active" at $t = 0$ and

that includes resource $i \in R^1$. Furthermore, $p^m_{0_i} = 0$ for all $m \in M_{0_i}$. Incomplete machine configurations (arising, for example, because of setup operations that are not finished at $t = 0$ or testers that are currently unused) are then easily modelled by defining *dummy resource types* that are included in the sets R^k, $k \in \{2, 3\}$.

The notation that has been introduced so far is summarized in Table 5.1.

Table 5.1: Notation used throughout the paper

J	Set of jobs	$J = \{1, \ldots, n\}$, $\lvert J\rvert = n$
O_j	Set of operations of job $j \in J$	$O_j = \{j_1, \ldots, j_{q_j}\}$, $\lvert O_j\rvert = q_j$
O	Set of all operations of jobs $j \in J$	$O = \bigcup_{j \in J} O_j$
$\hat{O}$	Set of all operations including the dummy operations $\{0_1, \ldots, 0_{r^1}\}$	$\hat{O} = O \cup \{0_1, \ldots, 0_{r^1}\}$
K	Set of resource types (tester, handler, adapter)	$K = \{1, 2, 3\}$
R^k	Set of classes of resource type $k \in K$	$R^k = \{1, \ldots, r^k\}$, $\lvert R^k\rvert = r^k$
M	Set of machine configurations	$M = R^1 \times R^2 \times R^3$
M_{j_i}	Set of machine configurations eligible for operation $j_i \in \hat{O}$	$M_{j_i} \subseteq M$
M^k_i	Set of machine configurations that require resource class $i \in R^k$ of type $k \in K$	$M^k_i \subseteq M$
$m[k]$	Element of the set R^k of type $k \in K$ that is used in machine configuration $m \in M$	
w_j	Weight of job $j \in J$	$w_j \in \mathbb{N}$
d_j	Due date of job $j \in J$	$d_j \in \mathbb{N}$
$p^m_{j_i}$	Processing time of operation $j_i \in \hat{O}$ on machine $m \in M_{j_i}$	$p^m_{j_i} \in \mathbb{N}$
q^k_i	Number of identical copies of resource class $i \in R^k$ of type $k \in K$	$q^k_i \in \mathbb{N}$, $q^1_i = 1 \,\forall\, i \in R^1$
T	Length of the planning horizon	$T \in \mathbb{N}$

We assume that *sequence-dependent setup times* occur when an operation $j_i \in \hat{O}$ is processed on machine configuration $m \in M_{j_i}$ and is the direct predecessor of some operation $g_h \in O$ that is processed on $m' \in M_{g_h} \cap M^1_{m[1]}$. There are three types of setups (see Table 5.2). First, the

Table 5.2: Sequence-dependent setup times ($j_i \in \hat{O}$ on $m \in M_{j_i}$ to $g_h \in O$ on $m' \in M_{g_h} \cap M^1_{m[1]}$)

Setup type	Disassembly	Assembly
Type 1: maintain machine configuration	-	s_{j_i,g_h}
Type 2: change adapter	$\bar{s}^m_{out}$	$\bar{s}^{m'}_{in} + \bar{s}_{j_i,g_h}$
Type 3: change handler (and adapter)	$\bar{s}^m_{out} + \hat{s}^m_{out}$	$\hat{s}^{m'}_{in} + \bar{s}^{m'}_{in} + \hat{s}_{j_i,g_h}$

machine configuration may remain unchanged. In this case, we assume the setup time to solely depend on the sequence of operations and denote this component by $s_{j_i,g_h} \in \mathbb{N}_0$. Second, m' may result from m by solely changing the adapter. In this case, one will have to disassemble m by removing adapter $m[3]$, which we assume to take $\bar{s}^m_{out} \in \mathbb{N}_0$ time units, and install adapter $m'[3]$ for a period of $\bar{s}^{m'}_{in} \in \mathbb{N}_0$ time units. The operation-specific component for this case is referred to by $\bar{s}_{j_i,g_h} \in \mathbb{N}_0$ and may deviate from s_{j_i,g_h}. Third, one may want to change the handler. We assume that a handler can only be disassembled after the adapter of the corresponding machine configuration has been removed. We denote the time needed to remove handler $m[2]$ from m by $\hat{s}^m_{out} \in \mathbb{N}_0$ and the time needed to install handler $m'[2]$ for machine configuration m' by $\hat{s}^{m'}_{in} \in \mathbb{N}_0$. The operation-specific component for this case is referred to by $\hat{s}_{j_i,g_h} \in \mathbb{N}_0$. The times needed to insert and remove adapters and handlers as well as the operation-specific components

add to sequence-dependent setup time as illustrated in Table 5.2. We allow idle times between disassembling handlers and adapters of machine configurations. However, in order to take account of the dynamic environment and the scarce machine resources, idle times are not allowed when assembling a machine configuration, i.e., a required configuration is completely assembled directly before the processing of the corresponding operation starts. All parts (handler, adapter) needed for an assembly must be available during the entire setup time. The choice of setup times that involve operations 0_i, $i \in \{1, \ldots, r^1\}$ and configurations $m \in M_{0_i}$ allow a flexible modelling of the start of the planning horizon. All components of the setup operations require the assignment of a *setup operator* during the entire setup time. We assume that there are h equally skilled setup operators. Of course, each setup operator can execute at most one setup operation at a time.

A job is *completed* when all of its operations are completed. The completion time of an operation $j_i \in \hat{O}$ is denoted by C_{j_i}. The completion time of job $j \in J$ is denoted by C_j. Obviously, $C_j = C_{j_{q_j}}$ for all $j \in J$. The objective is to minimize the total weighted tardiness, $\sum_{j \in J} w_j T_j$, of jobs, where the tardiness T_j of job $j \in J$ is defined as $T_j := \max\{C_j - d_j, 0\}$. This is summarized in Table 5.3.

Table 5.3: Remaining notation used throughout the paper

h	Number of setup operators	$h \in \mathbb{N}$
C_{j_i}	Completion time of operation $j_i \in \hat{O}$	
C_j	Completion time of job $j \in J$	$C_j = C_{j_{q_j}}$
T_j	Tardiness of job $j \in J$	$T_j := \max\{C_j - d_j, 0\}$

5.2.2 A Mixed Integer Program

In order to facilitate the understanding of the problem setting and in order to be able to compute benchmark solutions for small problem instances in our computational tests, we present a MIP for SFTPS in this section. It uses elements of the models presented by Wu and Chien (2008) and Wu et al. (2012) but, to the best of our knowledge, is the first model that explicitly takes account of the specific setting regarding disassembly and assembly operations introduced in this paper. The model makes use of a large positive integer B that has to be chosen appropriately.

5.2.2.1 Graph Representation, Time Variables, Allocation and Sequencing Variables

We represent the potential allocation, sequencing and setup decisions by a directed (multi-) graph, which we refer to as the *allocation, sequencing and setup graph* (ASAS graph). Its structure is illustrated in Figure 5.2. It is inspired by graph representations of vehicle routing problems that are known to have multiple similarities with machine scheduling problems (see, e.g., Bigras et al., 2008; Kress et al., 2019). Each operation $j_i \in \hat{O}$ defines three vertices, j_i, $\bar{j}_i$, and $\hat{j}_i$, of the graph. The latter two vertices represent states, in which the adapter or handler of the machine configuration that is assigned to j_i have been removed after completing j_i. Each vertex j_i is

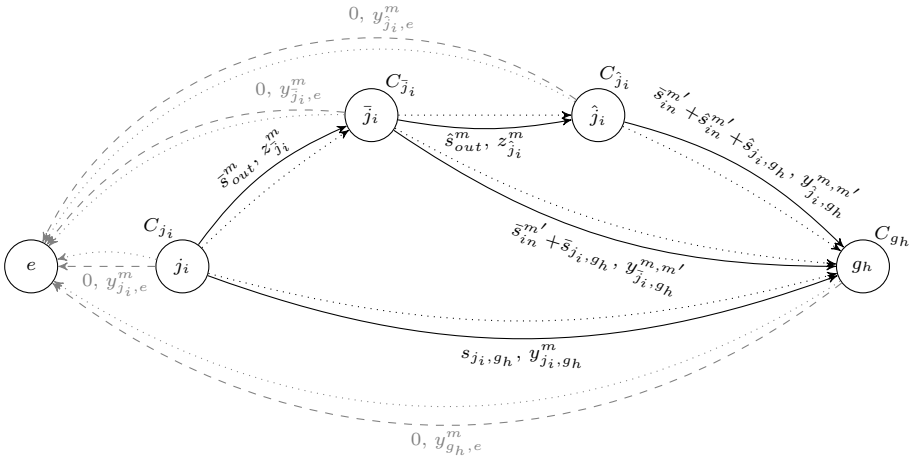


Figure 5.2: Illustration of the ASAS graph, $j_i, g_h \in \hat{O}$, $j_i \neq g_h$

associated to a completion time variable $C_{j_i} \in \mathbb{R}_0^+$ as defined in Section 5.2.1. Additionally, we define variables $C_{\bar{j}_i}, C_{\hat{j}_i} \in \mathbb{R}_0^+$ for all $j_i \in \hat{O}$, and set $C_{0_i} = 0$ for all $i \in \{1, \ldots, r^1\}$. An additional vertex e serves as a sink of the graph and represents the final states of the machine configurations associated to the testers at the end of the planning horizon. All allocation, sequencing and setup decisions are represented by directed edges that are weighted with the following variables:

$$y^m_{j_i,g_h} := \begin{cases} 1 & \text{if } g_h \text{ directly follows } j_i \text{ on } m \text{ without change of handler or adapter,} \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, m \in M_{j_i} \cap M_{g_h}, \tag{5.1}$$

$$y^{m,m'}_{\bar{j}_i,g_h} := \begin{cases} 1 & \text{if } g_h \text{ is processed on } m' \text{ and directly follows } j_i \text{ on } m \text{ after having disassembled adapter } m[3] \text{ and assembled adapter } m'[3], \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, m \in M_{j_i}, m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}, \tag{5.2}$$

$$y^{m,m'}_{\hat{j}_i,g_h} := \begin{cases} 1 & \text{if } g_h \text{ is processed on } m' \text{ and directly follows } j_i \text{ on } m \text{ after having disassembled handler } m[2] \text{ and assembled handler } m'[2], \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, m \in M_{j_i}, m' \in M_{g_h} \cap M^1_{m[1]}, \tag{5.3}$$

$$z^m_{\bar{j}_i} := \begin{cases} 1 & \text{if adapter } m[3] \text{ is disassembled after processing } j_i \text{ on } m, \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{5.4}$$

$$z^m_{\hat{j}_i} := \begin{cases} 1 & \text{if handler } m[2] \text{ is disassembled after processing } j_i \text{ on } m, \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}. \tag{5.5}$$

In line with the definition of these variables, parallel edges in the graph represent the fact that multiple machine configurations may be used when processing (succeeding) operations (dottes edges in Figure 5.2). Additional binary variables $y^m_{j_i,e}$, $y^m_{\bar{j}_i,e}$, and $y^m_{\hat{j}_i,e}$ for all operations $j_i \in \hat{O}$ are

used for modelling the end of the planning horizon. They are defined and represented in analogy to the variables (5.1)–(5.3).

The concrete allocation, sequencing and setup decisions of a solution to an instance of SFTPS are represented by r^1 edge-disjoint paths in the corresponding ASAS graph. Each path starts at a distinct vertex 0_i associated to a dummy operation $i \in \{1, \ldots, r^1\}$ (start of the planning horizon) and ends at vertex e (end of the planning horizon). It represents the machine configurations that include the corresponding tester, the allocation of operations to the corresponding configurations, as well as the sequencing and setup decisions. In a feasible solution, each vertex $j_i \in \hat{O}$ is included exactly once in exactly one path.

As defined in Section 5.2.1, the tardiness of each job $j \in J$ is represented by the variable $T_j \in \mathbb{R}_0^+$.

5.2.2.2 Objective, Processing Time Related Constraints, and Precedence Constraints

The objective

$$\min \sum_{j \in J} w_j T_j \tag{5.6}$$

minimizes the total weighted tardiness, where the correct tardiness values are enforced by the following constraints:

$$T_j \geq C_{j_{q_j}} - d_j \qquad \forall\, j \in J. \tag{5.7}$$

The following five sets of conditions establish the correct differences of the completion times of succeeding operations when taking account of setup and processing times:

$$C_{j_i} + s_{j_i,g_h} + p^m_{g_h} - C_{g_h} \leq (1 - y^m_{j_i,g_h})B \qquad \forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, m \in M_{j_i} \cap M_{g_h}, \tag{5.8}$$

$$C_{\bar{j}_i} + \bar{s}^{m'}_{in} + \bar{s}_{j_i,g_h} + p^{m'}_{g_h} - C_{g_h} \leq (1 - y^{m,m'}_{\bar{j}_i,g_h})B \qquad \begin{array}{l}\forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, \\ m \in M_{j_i}, m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]},\end{array} \tag{5.9}$$

$$C_{\hat{j}_i} + \bar{s}^{m'}_{in} + \hat{s}^{m'}_{in} + \hat{s}_{j_i,g_h} + p^{m'}_{g_h} - C_{g_h} \leq (1 - y^{m,m'}_{\hat{j}_i,g_h})B \qquad \begin{array}{l}\forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, \\ m \in M_{j_i}, m' \in M_{g_h} \cap M^1_{m[1]},\end{array} \tag{5.10}$$

$$C_{j_i} + \bar{s}^m_{out} z^m_{\bar{j}_i} \leq C_{\bar{j}_i} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{5.11}$$

$$C_{\bar{j}_i} + \hat{s}^m_{out} z^m_{\hat{j}_i} \leq C_{\hat{j}_i} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}. \tag{5.12}$$

The restrictions

$$\begin{aligned} C_{j_i} \leq C_{j_{i+1}} &- \sum_{g_h \in \hat{O} \setminus \{j_{i+1}\}} \sum_{m \in M_{g_h}} \sum_{m' \in M_{j_{i+1}} \cap M^1_{m[1]}} y^{m,m'}_{\bar{g}_h,j_{i+1}} p^{m'}_{j_{i+1}} \\ &- \sum_{g_h \in \hat{O} \setminus \{j_{i+1}\}} \sum_{m \in M_{g_h}} \sum_{m' \in M_{j_{i+1}} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m,m'}_{\bar{g}_h,j_{i+1}} p^{m'}_{j_{i+1}} \\ &- \sum_{g_h \in \hat{O} \setminus \{j_{i+1}\}} \sum_{m \in M_{g_h} \cap M_{j_{i+1}}} y^m_{g_h,j_{i+1}} p^m_{j_{i+1}} \qquad \forall\, j_i \in O \text{ with } i \leq q_j - 1 \end{aligned} \tag{5.13}$$

enforce the precedence relations among the operations of each job.

5.2.2.3 Allocation and Sequencing Constraints

Constraints (5.14) take account of the fact that each operation $g_h \in O$ must be processed by exactly one eligible machine configuration, i.e. that exactly one incoming edge is chosen for the corresponding vertex in the ASAS graph:

$$\begin{aligned}\sum_{j_i \in \hat{O}\setminus\{g_h\}} \sum_{m \in M_{j_i} \cap M_{g_h}} y^{m}_{\bar{j}_i,g_h} + \sum_{j_i \in \hat{O}\setminus\{g_h\}} \sum_{m \in M_{j_i}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m,m'}_{\bar{j}_i,g_h} \\ + \sum_{j_i \in \hat{O}\setminus\{g_h\}} \sum_{m \in M_{j_i}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m,m'}_{\hat{j}_i,g_h} = 1 \qquad \forall\, g_h \in O.\end{aligned} \tag{5.14}$$

Similarly, after completing the processing of an operation or at the beginning of the planning horizon, the adapter or the handler may have to be disassembled from the associated machine configuration, i.e. at most one incoming edge may be chosen for all vertices $\bar{j}_i$ and $\hat{j}_i$, $j_i \in \hat{O}$:

$$\sum_{m \in M_{j_i}} z^{m}_{\bar{j}_i} \leq 1 \qquad \forall\, j_i \in \hat{O}, \tag{5.15}$$

$$\sum_{m \in M_{j_i}} z^{m}_{\hat{j}_i} \leq 1 \qquad \forall\, j_i \in \hat{O}. \tag{5.16}$$

With respect to the sequencing of operations, each operation must have exactly one successor, which is modelled by appropriately selecting outgoing edges of the vertices of the ASAS graph:

$$\sum_{g_h \in O\setminus\{j_i\}} \sum_{m \in M_{j_i} \cap M_{g_h}} y^{m}_{j_i,g_h} + \sum_{m \in M_{j_i}} z^{m}_{\bar{j}_i} + \sum_{m \in M_{j_i}} y^{m}_{j_i,e} = 1 \qquad \forall\, j_i \in \hat{O}, \tag{5.17}$$

For all vertices $\bar{j}_i$ and $\hat{j}_i$, $j_i \in \hat{O}$, we may only select an outgoing edge if the adapter or handler is actually disassembled, so that

$$\sum_{g_h \in O\setminus\{j_i\}} \sum_{m \in M_{j_i}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m,m'}_{\bar{j}_i,g_h} + \sum_{m \in M_{j_i}} z^{m}_{\bar{j}_i} + \sum_{m \in M_{j_i}} y^{m}_{\bar{j}_i,e} \leq 1 \qquad \forall\, j_i \in \hat{O}, \tag{5.18}$$

$$\sum_{g_h \in O\setminus\{j_i\}} \sum_{m \in M_{j_i}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m,m'}_{\hat{j}_i,g_h} + \sum_{m \in M_{j_i}} y^{m}_{\hat{j}_i,e} \leq 1 \qquad \forall\, j_i \in \hat{O}. \tag{5.19}$$

Additionally, the machine configurations chosen for processing the operations must be consistent (flow conservation constraints):

$$\begin{aligned}\sum_{g_h \in \hat{O}\setminus\{j_i\}} \sum_{m' \in \{m\} \cap M_{g_h}} y^{m'}_{g_h,j_i} + \sum_{g_h \in \hat{O}\setminus\{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m',m}_{\bar{g}_h,j_i} \\ + \sum_{g_h \in \hat{O}\setminus\{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m',m}_{\hat{g}_h,j_i} \\ = \sum_{g_h \in O\setminus\{j_i\}} \sum_{m' \in \{m\} \cap M_{g_h}} y^{m'}_{j_i,g_h} + z^{m}_{\bar{j}_i} + y^{m}_{j_i,e} \qquad \forall\, j_i \in O, m \in M_{j_i},\end{aligned} \tag{5.20}$$

$$z^{m}_{\bar{j}_i} = \sum_{g_h \in O\setminus\{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m,m'}_{\bar{j}_i,g_h} + z^{m}_{\hat{j}_i} + y^{m}_{\bar{j}_i,e} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{5.21}$$

$$z^{m}_{\hat{j}_i} = \sum_{g_h \in O\setminus\{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m,m'}_{\hat{j}_i,g_h} + y^{m}_{\hat{j}_i,e} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}. \tag{5.22}$$

5.2.2.4 Resource Variables and Constraints: Handler and Adapter

In order to be able to model the resource constraints (availability of handlers and adapters), we define the following binary variables:

$$u^m_{j_i,t} := \begin{cases} 1 & \text{if the setup for processing } j_i \text{ on } m \text{ starts at time instant } t-1, \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T\}, \tag{5.23}$$

$$\bar{v}^{m,k}_{j_i,t} := \begin{cases} 1 & \text{if the use of resource class } m[k] \text{ of type } k \text{ in machine configuration } m \text{ for operation } j_i \text{ finishes at time instant } t, \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{0,\dots,T\}, k \in \{2,3\}, \tag{5.24}$$

$$x^{m,k}_{j_i,t} := \begin{cases} 1 & \text{if resource class } m[k] \text{ of type } k \text{ is being used in machine configuration } m \text{ for operation } j_i \text{ in time slot } [t-1,t], \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T\}, k \in \{2,3\}. \tag{5.25}$$

Furthermore, we define $x^{m,k}_{j_i,0} = x^{m,k}_{j_i,T+1} = 0$ for all $j_i \in \hat{O}$, $m \in M_{j_i}$, and $k \in \{1,2\}$, as well as $u^m_{j_i,T+1} = 0$ for all $j_i \in \hat{O}$ and $m \in M_{j_i}$.

The variables (5.23), i.e. the start of the setups that precede the processing of operations, are handled by the following constraints:

$$\sum_{m \in M_{j_i}} \sum_{t \in \{1,\dots,T\}} u^m_{j_i,t} = 1 \qquad \forall\, j_i \in \hat{O}, \tag{5.26}$$

$$\sum_{m \in M_{0_i}} u^m_{0_i,1} = 1 \qquad \forall\, i \in \{1,\dots,r^1\}, \tag{5.27}$$

$$\begin{aligned} \sum_{m \in M_{j_i}} \sum_{t \in \{1,\dots,T\}} (t-1) u^m_{j_i,t} = C_{j_i} &- \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m \in M_{j_i} \cap M_{g_h}} y^m_{g_h,j_i} (s_{g_h,j_i} + p^m_{j_i}) \\ &- \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h}} \sum_{m \in M_{j_i} \cap M^1_{m'[1]} \cap M^2_{m'[2]}} y^{m',m}_{\bar{g}_h,j_i} (\bar{s}^m_{in} + \bar{s}_{g_h,j_i} + p^m_{j_i}) \\ &- \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h}} \sum_{m \in M_{j_i} \cap M^1_{m'[1]}} y^{m',m}_{\hat{g}_h,j_i} (\bar{s}^m_{in} + \hat{s}^m_{in} + \hat{s}_{g_h,j_i} + p^m_{j_i}) \qquad \forall\, j_i \in O, \end{aligned} \tag{5.28}$$

$$\begin{aligned} \sum_{t \in \{1,\dots,T\}} u^m_{j_i,t} \leq &\sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in \{m\} \cap M_{g_h}} y^{m'}_{g_h,j_i} + \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m',m}_{\bar{g}_h,j_i} \\ &+ \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m',m}_{\hat{g}_h,j_i} \qquad \forall\, j_i \in O, m \in M_{j_i}. \end{aligned} \tag{5.29}$$

Here, conditions (5.26) and (5.27) guarantee that each operation needs a setup. The dummy operations take a special role as their setups have been started before or at the beginning of the planning horizon. The actual setup times as presented in Table 5.2 are taken account of in restrictions (5.28). Conditions (5.29) ensure that the machine configurations are consistent.

In line with the system of restrictions (5.26)–(5.29), the following seven sets of conditions handle the end of the usage of resources used for processing the operations:

$$\sum_{t\in\{0,\dots,T\}} \bar{v}^{m,k}_{j_i,t} = \sum_{t\in\{1,\dots,T\}} u^{m}_{j_i,t} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, k \in \{2,3\}, \tag{5.30}$$

$$\sum_{t\in\{0,\dots,T\}} t\bar{v}^{m,k}_{j_i,t} - C_{g_h} + p^{m}_{g_h} + s_{j_i,g_h} \geq (y^{m}_{j_i,g_h} - 1)B \qquad \begin{array}{l}\forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h,\\ m \in M_{j_i} \cap M_{g_h}, k \in \{2,3\},\end{array} \tag{5.31}$$

$$\bar{v}^{m,k}_{j_i,T} \geq y^{m}_{j_i,e} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, k \in \{2,3\}, \tag{5.32}$$

$$\bar{v}^{m,2}_{j_i,T} \geq y^{m}_{\bar{j}_i,e} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{5.33}$$

$$\sum_{t\in\{0,\dots,T\}} t\bar{v}^{m,2}_{j_i,t} - C_{g_h} + p^{m'}_{g_h} + \bar{s}^{m'}_{in} + \bar{s}_{j_i,g_h} \geq (y^{m,m'}_{\bar{j}_i,g_h} - 1)B \quad \begin{array}{l}\forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, m \in\\ M_{j_i}, m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]},\end{array} \tag{5.34}$$

$$\sum_{m\in M_{j_i}} \sum_{t\in\{0,\dots,T\}} t\bar{v}^{m,3}_{j_i,t} \geq C_{\bar{j}_i} \qquad \forall\, j_i \in \hat{O}, \tag{5.35}$$

$$\sum_{m\in M_{j_i}} \sum_{t\in\{0,\dots,T\}} t\bar{v}^{m,2}_{j_i,t} \geq C_{\hat{j}_i} \qquad \forall\, j_i \in \hat{O}. \tag{5.36}$$

Conditions (5.30) ensure that the use of some resource class must end if it has previously been setup. Depending on the allocation and sequencing decisions (Section 5.2.2.1), constraints (5.31)–(5.36) then set the variables (5.24) to their correct values.

Now, based on the choice of the variables (5.23) and (5.24), conditions (5.37) fix the variables (5.25) that model the actual resource usage over all time slots of the planning horizon:

$$x^{m,k}_{j_i,t} - x^{m,k}_{j_i,t-1} = u^{m}_{j_i,t} - \bar{v}^{m,k}_{j_i,t-1} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T+1\}, k \in \{2,3\}. \tag{5.37}$$

Finally, constraints (5.38) restrict the usage of the handler and adapter classes based on their availability:

$$\sum_{j_i\in\hat{O}} \sum_{m\in M_{j_i}\cap M^k_r} x^{m,k}_{j_i,t} \leq q^k_r \qquad \forall\, t \in \{1,\dots,T\}, k \in \{2,3\}, r \in R^k. \tag{5.38}$$

5.2.2.5 Setup Operator Variables and Constraints

The setup operators are handled in analogy to the hardware resources, so that we define the following variables:

$$\bar{u}^{m}_{j_i,t} := \begin{cases} 1 & \text{if the setup for processing } j_i \text{ on } m \text{ finishes at time instant } t,\\ 0 & \text{else,}\end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{0,\dots,T\}, \tag{5.39}$$

$$v^{m,k}_{j_i,t} := \begin{cases} 1 & \text{if the procedure of disassembling resource class } m[k] \text{ of type } k \text{ from machine configuration } m \text{ after processing operation } j_i \text{ starts at time instant } t-1,\\ 0 & \text{else,}\end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T\}, k \in \{2,3\}, \tag{5.40}$$

$$w^{m}_{j_i,t} := \begin{cases} 1 & \text{if a setup operator is required for the setup of machine configuration } m \text{ for operation } j_i \text{ in time slot } [t-1,t],\\ 0 & \text{else,}\end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T\}, \tag{5.41}$$

$$\tilde{w}^{m,k}_{j_i,t} := \begin{cases} 1 & \text{if a setup operator is required for disassembling resource class } m[k] \text{ of type } k \text{ from machine configuration } m \text{ for operation } j_i \text{ in time slot } [t-1,t], \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T\}, k \in \{2,3\}. \tag{5.42}$$

While the variables (5.39) model the end of the setup operations, variables (5.40) are used to identify time instants at which handlers and adapters are started to be disassembled. The variables (5.41) and (5.42) are used to indicate the need for setup operators for setup or disassembly operations. As in Section 5.2.2.4, we define $w^m_{j_i,0} = w^m_{j_i,T+1} = 0$ for all $j_i \in \hat{O}$ and $m \in M_{j_i}$, as well as $\tilde{w}^{m,k}_{j_i,0} = \tilde{w}^{m,k}_{j_i,T+1} = v^{m,k}_{j_i,T+1} = 0$ for all $j_i \in \hat{O}$, $m \in M_{j_i}$, and $k \in \{2,3\}$.

The handling of the above variables is in line with our deliberations in Section 5.2.2.4. Conditions (5.43)–(5.45) relate to variables (5.39), while constraints (5.46)–(5.48) relate to variables (5.40):

$$\sum_{t\in\{0,\dots,T\}} \bar{u}^m_{j_i,t} = \sum_{t\in\{1,\dots,T\}} u^m_{j_i,t} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{5.43}$$

$$\sum_{m\in M_{0_i}} \bar{u}^m_{0_i,0} = 1 \qquad \forall\, i \in \{1,\dots,r^1\}, \tag{5.44}$$

$$\begin{aligned} \sum_{m\in M_{j_i}} \sum_{t\in\{0,\dots,T\}} t\bar{u}^m_{j_i,t} = C_{j_i} &- \sum_{g_h\in\hat{O}\setminus\{j_i\}} \sum_{m\in M_{j_i}\cap M_{g_h}} y^m_{g_h,j_i} p^m_{j_i} \\ &- \sum_{g_h\in\hat{O}\setminus\{j_i\}} \sum_{m'\in M_{g_h}} \sum_{m\in M_{j_i}\cap M^1_{m'[1]}\cap M^2_{m'[2]}} y^{m',m}_{\bar{g}_h,j_i} p^m_{j_i} \\ &- \sum_{g_h\in\hat{O}\setminus\{j_i\}} \sum_{m'\in M_{g_h}} \sum_{m\in M_{j_i}\cap M^1_{m'[1]}} y^{m',m}_{\hat{g}_h,j_i} p^m_{j_i} \qquad \forall\, j_i \in O, \end{aligned} \tag{5.45}$$

$$\sum_{t\in\{1,\dots,T\}} v^{m,k}_{j_i,t} = \sum_{t\in\{1,\dots,T\}} u^m_{j_i,t} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, k \in \{2,3\}, \tag{5.46}$$

$$\sum_{m\in M_{j_i}} \sum_{t\in\{1,\dots,T\}} (t-1) v^{m,3}_{j_i,t} = C_{\bar{j}_i} - \sum_{m\in M_{j_i}} z^m_{\bar{j}_i} \bar{s}^m_{out} \qquad \forall\, j_i \in \hat{O}, \tag{5.47}$$

$$\sum_{m\in M_{j_i}} \sum_{t\in\{1,\dots,T\}} (t-1) v^{m,2}_{j_i,t} = C_{\hat{j}_i} - \sum_{m\in M_{j_i}} z^m_{\hat{j}_i} \hat{s}^m_{out} \qquad \forall\, j_i \in \hat{O}. \tag{5.48}$$

The need for setup operators over all time slots of the planning horizon as well as the availability of the setup operators is then taken account of in the following conditions (as in Section 5.2.2.4):

$$w^m_{j_i,t} - w^m_{j_i,t-1} = u^m_{j_i,t} - \bar{u}^m_{j_i,t-1} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T+1\}, \tag{5.49}$$

$$\tilde{w}^{m,k}_{j_i,t} - \tilde{w}^{m,k}_{j_i,t-1} = v^{m,k}_{j_i,t} - \bar{v}^{m,k}_{j_i,t-1} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T+1\}, k \in \{2,3\} \tag{5.50}$$

$$\sum_{j_i\in\hat{O}} \sum_{m\in M_{j_i}} (w^m_{j_i,t} + \tilde{w}^{m,2}_{j_i,t} + \tilde{w}^{m,3}_{j_i,t} \le h \qquad \forall\, t \in \{1,\dots,T\}. \tag{5.51}$$

5.3 Tabu Search Approach

In this section, we present a tabu search approach for solving SFTPS. In order to handle the interdependencies between all relevant hardware resources and the allocation of setup operators,

we follow the main ideas of the decomposition based approaches introduced by Kress et al. (2019) and Müller and Kress (2019), i.e., we decompose SFTPS into a flexible job shop scheduling problem with sequence-dependent setup times that considers the assignment of operations to eligible testers and the sequencing of these operations (master problem), and an assignment problem that considers handlers, adapters, and setup operators (subproblem).

5.3.1 Master Problem, Solution Graph, Neighborhood Structure

Within the master problem, we make use of lower bounds for the processing and setup times. The former values are defined as $\min_{m \in M_{j_i} \cap M_r^1} p_{j_i}^m$ for operations $j_i \in \hat{O}$ and testers $r \in R^1$. With respect to the setup times between two operations on some specific eligible tester, we use the minimum assembly time for all potential machine configurations over the three corresponding setup types presented in Table 5.2.

In order to represent feasible solutions of the master problem and in order to define a neighborhood structure, we make use of the concept of the *solution graph* as introduced by Mastrolilli and Gambardella (2000) and later adapted by Müller and Kress (2019). This concept can be generalized to take account of setup times in a straightforward manner, so that we will restrict ourselves to solely presenting its essential ideas in this paper. For details, we refer to the latter two articles. Basically, the solution graph contains a vertex for each operation as well as dummy vertices that represent the start and the end of the schedule. The vertices are weighted with the corresponding lower bounds on the processing times according to the tester allocation of the solution. The graph furthermore includes two sets of directed edges. The first set E_1 includes an edge for each direct precedence relation among the operations of the jobs as well as corresponding dummy edges for the start and end of the schedule. The second set E_2 directly represents the tester allocation and sequencing decisions of the given solution in the same manner. The edges of this set are weighted with the respective lower bounds on the setup times. Based on this weighted graph and when assuming that the bounds represent the correct processing and setup times, one can easily compute the time instants at which the operations are started to be processed (starting times) on their respective testers in polynomial time. Given these values, the corresponding objective function value follows readily. It defines a lower bound on the total weighted tardiness of the allocation and sequencing decision of the solution when taking account of the remaining hardware resources and the setup restriction.

Based on the solution graph of some feasible solution of the master problem, we construct a neighboring solution graph by moving an operation a_b to some eligible tester r. To do so, we remove a_b from its current tester sequence by deleting the corresponding edges from E_2. We then recompute the starting time of operation a_b and determine a specific set of feasible insertion positions in the sequence of tester r as described in Mastrolilli and Gambardella (2000) and Müller and Kress (2019). For each of these positions, we construct the corresponding solution graph (update the edges in E_2) and recompute the objective function value. Out of these candidates, we then select the most promising solution.

5.3.2 Subproblem

Given a solution to the master problem, we construct a solution to the subproblem and, therefore, to the corresponding instance of SFTPS with a greedy approach. Based on the solution graph, it iteratively assigns handler and adapter entities as well as setup operators to the operations. It considers the operations in non-decreasing order of their starting times in the solution of the master problem and corrects (i.e., shifts) these starting times based on the selected resource assignment. Details are presented in Algorithm 5.1. During runtime, the algorithm keeps track of

Input: Solution graph G of master problem solution
Output: Solution S of SFTPS with objective function value WT

▷ `Initialization phase`
1 Initialize load of testers $L_i := 0 \, \forall i \in R^1$;
2 Initialize resource information of adapter ($k = 3$) and handler ($k = 2$) entities $q \in \{1, \ldots, q_i^k\}$ for all $i \in R^k$;
3 Initialize release times of setup operators $i \in \{1, \ldots, h\}$;
4 **forall** *dummy operations* $0_i, i \in \{1, \ldots, r^1\}$, **do**
5 Get $m \in M_{0_i}$, select an adapter entity $m[3]$ and a handler entity $m[2]$ and update their corresponding resource information;
6 **end**
7 Set the initial state of solution S;
▷ `Machine configuration evaluation phase`
8 **forall** *operations* $j_i \in O$ *in non-decreasing order of their starting times in* G **do**
9 Initialize $C^*_{j_i} := \infty$ and $m^* := \emptyset$;
10 Get the tester i' assigned to j_i from G;
11 Get the last assigned machine configuration $\hat{m}$ that includes tester i';
12 Set $S^* := S$;
13 **forall** *eligible machine configurations* $m \in M_{j_i} \cap M^1_{i'}$ **do**
14 Set $\hat{S} := S$;
15 Evaluate machine configuration m (details below);
16 Compute the completion time $\hat{C}_{j_i}$ according to the above evaluation;
17 **if** $\hat{C}_{j_i} < C^*_{j_i}$ **then**
18 $C^*_{j_i} := \hat{C}_{j_i}$;
19 $m^* := m, S^* := \hat{S}$;
20 **end**
21 **end**
22 Assign the most promising machine configuration m^* to operation j_i;
23 Update resource information and release times of setup operators;
24 Set $S := S^*$;
25 **end**
26 Compute objective function value WT;

Algorithm 5.1 Determine a feasible allocation of handlers, adapters and setup operators

the load of the testers, i.e., the time instants at which the last operations that have been assigned to the testers are completed, and information on the current use of all adapter and handler entities as well as the setup operators. These values are initialized based on the machine configurations at time $t = 0$ in lines 1–7 of the algorithm. In the second phase of the algorithm (lines 8–26), the iterative assignment of resources to the operations takes place. For each operation, the algorithm considers every eligible machine configuration based on the fixed tester decision of the given

solution graph (loop 13–21). When evaluating the machine configuration (line 15), it considers setup operator constraints and disassembly as well as assembly operations to later select the most promising configuration based on the completion time of the considered operation in a greedy manner (lines 17–20). More specifically, the evaluation proceeds as follows. Based on the setup type (see Table 5.2) and the current resource information, it first considers necessary disassembly operations on the selected tester. Here, the algorithm selects setup operators that can complete the operations as quick as possible. The resource information as well as the operator release times are updated accordingly. In the next step, the machine configuration is assembled. Here, the algorithm prioritizes handler and adapter entities that are currently unused. If all entities are currently in use, it schedules disassembly operations as described above, so that the needed entities are available as early as possible. Hereafter, the assembly operations are scheduled in a similar greedy manner, the resource information as well as the operator release times are updated, and the completion time of the operation is computed. Finally, after assigning each operation to a promising machine configuration, the objective function value of the constructed solution of SFTPS is obtained (line 26).

5.3.3 Generating an Initial Solution

We generate an initial solution by making use of a constructive procedure composed of two steps. First, the approach considers the master problem, i.e., it allocates operations to eligible testers and decides on the sequences of the operations on the testers based on the lower bounds of the processing times and the sequence-dependent setup times. For these allocation and sequencing decisions, we adapt a priority-rule based heuristic introduced by Kress et al. (2019). It follows an algorithmic idea of Giffler and Thompson (1960) for the classical job shop scheduling problem. In general, the algorithm iteratively allocates operations that can start being processed at the respective point of time when considering the corresponding precedence constraints. Among all eligible testers that can be used for these operations, it then selects a tester, the use of which results in the smallest possible completion time for one of the considered operations. Among all operations that compete for this selected tester, exactly one operation is chosen based on a priority rule. In our case, we take account of the due dates as well as the weights of the corresponding jobs and select an operation j_i with smallest value d_j/w_j. In the second step, given the corresponding solution graph of the master problem, we apply the greedy procedure as presented in Section 5.3.2.

5.3.4 Heuristic Framework

Our resulting overall best-fit tabu search framework is presented in Algorithm 5.2. To ease the notation, we will refer to the objective function value of a solution S by using an additional label, i.e., $\ddot{S}$. The framework consists of two phases, the *initialization phase* and the *tabu search phase*. In the former phase, a first feasible solution is determined by making use of the constructive procedure described in Section 5.3.3 (line 1) and the tabu search parameters are initialized (line 2). In the *tabu search phase*, we first compute a set of critical operations (lines 3–6). An operation

Input: Instance $Inst$ of SFTPS, parameter τ
Output: Solution S^*

▷ `Initialization phase`
1 Determine a feasible solution S of $Inst$ with the constructive procedure described in Section 5.3.3 and initialize $S^* := S$;
2 Initialize an empty tabu list and set $\lambda := 0$;
▷ `Tabu search phase`
3 Initialize the set of critical operations $O^c := \emptyset$;
4 Insert all operations of all jobs that complete after their due date in S into O^c;
5 Initialize the master problem solution graph G of S;
6 **if** $O^c = \emptyset$ **then** exit the procedure;
7 Initialize the set of feasible neighboring solutions $N := \emptyset$;
8 **forall** *critical operations* $a_b \in O^c$ **do**
9 Determine the set $\bar{R}^1$ of eligible testers of operation a_b;
10 **forall** $r \in \bar{R}^1$ **do**
11 Determine a neighbor G^N of G as illustrated in Section 5.3.1;
12 Apply Algorithm 5.1 on G^N to determine a neighboring solution S^N;
13 Set $N := N \cup S^N$;
14 **end**
15 **end**
16 Discard all elements S^N of N where the corresponding move from S is tabu if $\ddot{S}^N \geq \ddot{S}^*$ (aspiration criterion);
17 Select the best solution S^{NBS} among the remaining solutions in N. If no solution remains, i.e. if $N = \emptyset$, exit the procedure;
18 **if** $\ddot{S}^{NBS} < \ddot{S}^*$ **then** set $S^* := S^{NBS}$ and $\lambda := 0$;
19 **else** set $\lambda := \lambda + 1$;
20 Update the tabu list;
21 Set $S := S^{NBS}$;
22 **if** $\lambda < \tau$ **then** go to line 3;
23 **else** exit the procedure;

Algorithm 5.2 Tabu search framework

is critical, if its corresponding job completes after its due date. For each of these operations, we make use of the solution graph of the corresponding master problem to compute a neighboring solution graph as described in Section 5.3.1. For each of these graphs, we compute a solution of the input instance by calling Algorithm 5.1. This results in a set N of neighboring solutions (lines 8–15). In the next step (line 16), neighboring solutions are discarded if their corresponding move (operation to tester) is tabu and if they do not represent a new overall best solution (aspiration criterion). The tabu list is updated in line 20. It is fed with the pair $(a_b, \bar{r})$ for the move of operation a_b from tester $\bar{r}$ to tester r in order to generate the current solution S^{NBS}. The length of the tabu list is dynamically updated. It is set to the current number of critical operations. If the length of the tabu list exceeds the tabu length threshold, the procedure removes entries of the list in a first-in-first-out manner, until the tabu length is met. Whenever the overall best solution S^* is improved, the counter λ is set to 0, otherwise the value is incremented by 1. The tabu search phase executes when no improvement is found for τ iterations.

5.4 Computational Study and Managerial Implications

In this section, we analyze the applicability and performance of our heuristic framework in real-world industry settings in order to provide decision support for managers. We analyze the effectiveness of rescheduling jobs in case of changing customer requests and we explore the impact of handler or adapter failures. Before presenting these managerial insights in Section 5.4.5, we present an overview of the considered solution approaches (Section 5.4.1) and the generation of the instance sets that our study is based upon (Section 5.4.2). The general question of whether or not our heuristic framework is suited for drawing managerial conclusions is answered in Sections 5.4.3 and 5.4.4.

Our computational study was performed on a PC with an Intel® Core™ i7-4770 CPU, running at 3.4 GHz, with 16 GB of RAM under a 64-bit version of Windows 8. All algorithms were implemented in Java (JRE 1.8.0_221), using Eclipse (Eclipse IDE for Java Developers, Oxygen 4.7). We used IBM ILOG CPLEX in version 12.9 as a MIP solver.

5.4.1 Overview of Solution Approaches

We implemented five solution approaches for our computational study. First, in order to provide benchmarks for evaluating our heuristics, we make use of CPLEX in its standard settings with a time limit of 3,600 seconds on the MIP presented in Section 5.2.2. Additionally, we implemented the constructive procedure outlined in Section 5.3.3. It is referred to as PR ("priority rule") and mimics the status quo scheduling approach at our industry partner. With respect to our heuristic framework (Section 5.3.4), we implemented Algorithm 5.2 as presented above (referred to as TS-FI, "fully-integrated") as well as two variations of Algorithm 5.2, the main ideas of which are briefly summarized as follows:

1. TS-PI ("partially-integrated"): The procedure of evaluating machine configurations in line

15 of Algorithm 5.1 does not take account of the availability of setup operators. Instead, setup operators are taken account of only once in an additional call of Algorithm 5.1 at the very end of Algorithm 5.2. The algorithm then compares the resulting solution with the solution determined in the initialization phase and selects the best solution.

2. TS-HIER ("hierarchical"): Line 12 of Algorithm 5.2 is not executed at all, so that the tabu search phase is executed solely on the master problem. As in TS-PI, Algorithm 5.1 is called only once at the very end of Algorithm 5.2.

Within Algorithm 5.2 and its adaptions, we set $\tau := \sum_{j \in J} q_j$.

5.4.2 Instance Generation

Our test instances are based on resource scenarios that imitate the resource pool at the testing facility of our industry partner. While the small problem instances, that we make use of to evaluate the basic performance of our heuristic framework, feature only two testers (Table 5.4), the

Table 5.4: Resource scenario of small problem instances

r^1	q_i^1	r^2	q_i^2	$r^2_{R^1}$	r^3	q_i^3	$r^3_{R^1}$	h
2	1	2	1	$[1,2]$	2	1	$[1,2]$	2

large problem instances are based on eight different scenarios (Table 5.5). The tables indicate

Table 5.5: Resource scenarios of large problem instances

Scenario	r^1	q_i^1	r^2	q_i^2	$r^2_{R^1}$	r^3	q_i^3	$r^3_{R^1}$	h
A	10	1	10	$[1,2]$	$[3,5]$	15	$[1,2]$	$[3,5]$	4
B	10	1	10	$[2,3]$	$[3,5]$	15	$[2,3]$	$[3,5]$	4
C	20	1	15	$[1,2]$	$[3,5]$	20	$[1,2]$	$[3,5]$	8
D	20	1	15	$[2,3]$	$[3,5]$	20	$[2,3]$	$[3,5]$	8
E	30	1	20	$[1,2]$	$[3,5]$	25	$[1,2]$	$[3,5]$	12
F	30	1	20	$[2,3]$	$[3,5]$	25	$[2,3]$	$[3,5]$	12
G	40	1	25	$[1,2]$	$[3,5]$	35	$[1,2]$	$[3,5]$	16
H	40	1	25	$[2,3]$	$[3,5]$	35	$[2,3]$	$[3,5]$	16

whether the integer parameters were fixed or drawn from uniform distributions over the given intervals. We restricted the interoperability of testers, handlers and adapters, by generating eligible machine configurations: For each tester, we randomly generated a subset of eligible handler and adapter types of given size $r^2_{R^1}$ and $r^3_{R^1}$, that can be combined arbitrarily. With respect to the beginning of the planning horizon, we randomly selected eligible machine configurations and potentially generated incomplete configurations by "removing" adapters and handlers (and formally introducing dummy resource types). The number h of setup operators is fixed in each scenario. For the large problem instances, this results in a *staffing level*, defined as the ratio of the number of setup operators and the number of testers, of 40%.

The small instances are grouped into five instances sets, small1–small5. For each set, we randomly generated 10 instances based on the values and intervals given in Table 5.6. As

Table 5.6: Parameters of small problem instances

Inst. set	$\|J\|$	q_j	$\|M_{j_i}\|$
small1	2	$[1,2]$	$[1,2]$
small2	2	$[2,2]$	$[1,2]$
small3	3	$[2,2]$	$[1,2]$
small4	5	$[1,2]$	$[1,2]$
small5	5	$[2,2]$	$[2,2]$

indicated in the table, the number of jobs is fixed for each instance within a group, while the number of operations q_j was drawn randomly for each job j. Similarly, we randomly determined the number of eligible machine configurations and, in a next step, the configurations themselves. For the large instances, the corresponding parameter values are given in Table 5.7. For each

Table 5.7: Parameters of real-world problem instances

$\|J\|$	20, 30, 40, 50
q_j	$[2,3]$, $[3,5]$, $[4,6]$
$\|M_{j_i}\|$	$[1,2]$, $[2,3]$, $[3,5]$

combination and each resource scenario (A-H), we randomly generated 20 test instances.

In order to generate the processing times, we first drew auxiliary integer parameters p_{j_i} for all operations $j_i \in O$ from uniform distributions over $[1, 100]$. Then, in order to construct varying processing times over the eligible machine configurations $m \in M_{j_i}$, we drew integer values $p^m_{j_i}$ from uniform distributions over $[\max\{\lfloor 0.9 \cdot p_{j_i} \rfloor, 1\}, \lfloor 1.1 \cdot p_{j_i} \rfloor]$.

All relevant operation-specific setup components (s_{j_i,g_h}, $\bar{s}_{j_i,g_h}$, $\hat{s}_{j_i,g_h}$) were drawn independently from uniform distributions over $[1, 5]$. Similarly, the setup times needed to remove or install adapters ($\bar{s}^m_{out}$, $\bar{s}^{m'}_{in}$) and handlers ($\hat{s}^m_{out}$, $\hat{s}^{m'}_{in}$) were drawn separately from uniform distributions over $[3, 10]$.

Define $p^{max}_{j_i} := \max\{p^m_{j_i} | m \in M_{j_i}\}$, and $\tilde{s}^{max}_{j_i} := \max\{s^{max}_{j_i}, \bar{s}^{max}_{j_i}, \hat{s}^{max}_{j_i}\}$, for all $j_i \in O$. Furthermore, set $s^{max}_{j_i} := \max\{s_{g_h,j_i} | g_h \in \hat{O}\}$, $\bar{s}^{max}_{j_i} := \max\{\bar{s}^m_{out} + \bar{s}^m_{in} | m \in M_{j_i}\} + \max\{\bar{s}_{g_h,j_i} | g_h \in \hat{O}\}$, and $\hat{s}^{max}_{j_i} := \max\{\bar{s}^m_{out} + \hat{s}^m_{out} + \bar{s}^m_{in} + \hat{s}^m_{in} | m \in M_{j_i}\} + \max\{\hat{s}_{g_h,j_i} | g_h \in \hat{O}\}$, for all $j_i \in O$. Based on these values, we drew the due dates d_j of jobs $j \in J$ from uniform distributions over $[0, \lfloor 1.5 \cdot T^{max}/r^1 \rfloor]$. Here, $T^{max} := \sum_{j_i \in O}(p^{max}_{j_i} + \tilde{s}^{max}_{j_i})$. Similarly, the weights w_j of jobs $j \in J$ were randomly generated based on the interval $[1, 5]$.

5.4.3 Small Instances: Basic Evaluation of the Heuristic Framework

The computational results of the heuristic approaches PR, TS-FI, and TS-HIER are presented in Table 5.8. Furthermore, the table includes information on the performance of CPLEX. With respect to CPLEX, the table presents information on the number of instances for which a feasible or optimal solution was obtained within the time limit (columns "feas." and "opt."), the corresponding average objective function value (column "WT_{avg}"), as well as the average runtime over all runs that resulted in a feasible solution. For the heuristic approaches, the table presents

Table 5.8: Performance of the heuristic approaches on small problem instances

	CPLEX				PR	TS-FI	TS-PI	TS-HIER
Inst. set	feas.	opt.	WT_{avg}	t_{avg} [s]	WT_{avg}	WT_{avg}	WT_{avg}	WT_{avg}
small1	10	10	281.7	271.42	411.1	328.8	328.8	343.2
small2	10	10	442.3	361.45	627.9	509	509	560
small3	6	2	496.67	3559.53	857	580.3	580.3	724.4
small4	4	0	1156	3600	1241.5	715.2	715.2	1078.3
small5	0	0	-	-	918.9	421.6	421.6	719.6

the average objective function values. Note that all calls of the heuristics resulted in a feasible solution.

As to be expected, CPLEX returns feasible or optimal solutions solely for the smallest instances and it is therefore not a reasonable choice when facing instances of larger size in practice. However, the few resulting benchmarks certainly allow to draw first conclusions on the performance of the heuristic approaches. Specifically, we observe that the solution quality of our heuristic approaches TS-FI, TS-PI and TS-HIER is competitive when compared with the optimal solutions for the sets small1 and small2. TS-FI and TS-PI tend to provide the best solutions and there is a clear improvement with respect to PR, being the current status quo scheduling approach at our industry partner. The average improvement of the solution quality when applying the tabu-search heuristics is up to 37.02%. Moreover, when comparing TS-FI and TS-PI with TS-HIER, we find that it clearly pays off to integrate the solution of subproblems into the tabu search phase. The average runtimes of the heuristic approaches over all considered instances range between 0.1 and 9.5 milliseconds.

5.4.4 Large Instances: Applicability of the Heuristic Framework

As can be concluded from the previous subsection, it is not reasonable use CPLEX to determine benchmark solutions for the large problem instances. Instead, we compare the objective function value returned by the tabu search framework with the real-world solution approach at our industry partner (PR). Thus, for some given instance and some run of algorithm TS-FI, TS-PI and TS-HIER, we measure the quality of the solution returned by the algorithm with the quality ratio $100 \cdot (WT^{\mathrm{PR}} - WT)/WT^{\mathrm{PR}}$, where WT and WT^{PR} denote the total weighted tardiness of the solution determined by the tabu search heuristic and PR, respectively.

Table 5.9 summarizes the computational results for the large problem instances grouped by the eight scenarios and the number of jobs. The table includes information about the average quality ratio over all instances of the corresponding sets (columns "Q_{avg}"), as well as the average runtimes (columns "t_{avg}") of the algorithms. Figure 5.3 complements Table 5.9 by illustrating the average quality ratios (Q_{avg}) as well as the average runtimes (t_{avg}) over all resource scenarios. It can be seen that the use of the tabu search framework significantly improves the solutions that are currently implemented in practice (and used as initial solutions within our framework). TS-FI tends to provide the best solutions at the cost of the largest average runtimes. Certainly, with respect to the runtimes, all approaches are applicable in practice. As to be expected based

Table 5.9: Performance of the heuristic approaches on large problem instances

		TS-FI		TS-PI		TS-HIER				TS-FI		TS-PI		TS-HIER	
Sce.	$\|J\|$	Q_{avg}	t_{avg} $[s]$	Q_{avg}	t_{avg} $[s]$	Q_{avg}	t_{avg} $[s]$	Sce.	$\|J\|$	Q_{avg}	t_{avg} $[s]$	Q_{avg}	t_{avg} $[s]$	Q_{avg}	t_{avg} $[s]$
A	20	66.19	5.36	54.89	4.61	5.61	0.18	B	20	48.33	4.22	33.21	3.33	9.43	0.18
	30	68.63	17.84	56.69	15.72	3.84	0.68		30	52.87	13.79	32.31	10.35	6.05	0.72
	40	67.86	42.31	53.64	35.06	2.94	1.84		40	55.16	30.75	36.61	25.1	6.14	1.8
	50	69.12	85.6	53.83	66.9	3.33	4.07		50	58.25	59.91	36.63	48.07	4.95	4.05
C	20	59.21	8.98	53.03	8.54	4.92	0.24	D	20	40.41	8.3	31.63	6.95	9.74	0.24
	30	65.39	31.09	56.83	28.58	3.75	0.75		30	48.32	25.71	37.35	20.77	6.67	0.76
	40	67.6	75.3	56.83	69.36	3.46	1.83		40	54.05	55.67	40.17	43.13	5.33	1.86
	50	67.87	153.48	56.51	126.26	2.69	4.22		50	56.02	108.29	41.98	86.41	3.62	4.26
E	20	57.02	11.62	53.62	11.59	6.06	0.25	F	20	35.33	11.63	30.88	10.52	10.2	0.26
	30	62.7	41.95	57.68	39.15	4.42	0.87		30	44.52	34.31	37.29	31.47	7.15	0.85
	40	65.25	103.15	57.45	92.55	2.52	2.14		40	50.16	78.59	41.3	70.88	4.82	2.08
	50	65.63	206.21	57.13	183.37	3.11	4.23		50	53.36	149.74	44.07	131.09	4.31	4.48
G	20	50.3	13.9	48.42	13.17	7.85	0.26	H	20	29.03	14.69	26.53	13.39	12.17	0.28
	30	60.23	51.1	55.95	46.74	6.05	0.96		30	39.49	47.35	34.11	41.16	8.52	0.92
	40	63.7	119.92	57.79	118.26	4.38	2.21		40	44.76	106.46	37.92	90.02	6.66	2.22
	50	65.89	253.62	58.77	234.47	2.98	4.8		50	49.7	217.09	41.09	162	5.89	4.32

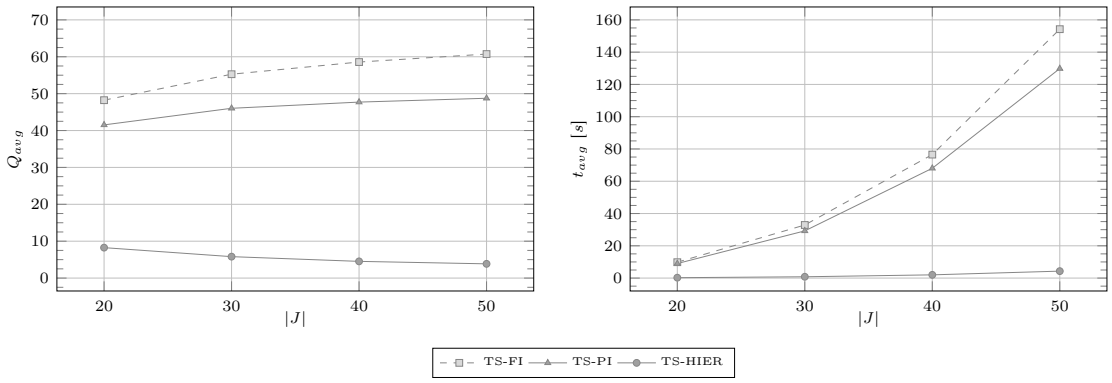


Figure 5.3: Quality ratios for large problem instances over all resource scenarios

on the number of testers and handlers, we observe that the average quality ratios of TS-FI are smaller in scenarios B, D, F, and H when compared with scenarios A, C, E, and G. Furthermore, as also indicated above, the incorporation of computing solutions for the subproblems within the tabu search phase (TS-FI, TS-PI) clearly pays off with respect to the solution quality, so that a hierarchical planning approach has significant drawbacks. Moreover, when comparing TS-FI and TS-PI, we observe that the consideration of setup operators when computing solutions of the corresponding subproblems on average improves the solution quality at the cost of only slightly increased runtimes.

5.4.5 Managerial Implications

Based on the above results we conclude that, in general, our heuristic framework is an adequate method for deriving managerial insights. It furthermore seems reasonable to focus on the use of the most promising setup TS-FI. Additionally, with regard to the test instances, we restrict our attention to representative scenarios that feature instances with an expected makespan of about one working-day, namely the resource scenarios C and D with parameter values $|J| = 30$, $q_j \in [3, 5]$, and $|M_{ji}| \in [3, 5]$ (see Table 5.7). We will, however, adjust selected parameters of these instances in the following.

We first analyze the impact of the number of copies of handler (adapter) classes q_i^2 (q_i^3) on the objective function value. Therefore, for each test instance, we modify the values of q_i^2 and q_i^3, such that they take all integer values from 1 to 6 for all i in all possible combinations. Figure 5.4 illustrates the corresponding computational results over increasing values of q_i^2 and q_i^3. Naturally, we find that increasing the number of copies of handler and adapter classes has a

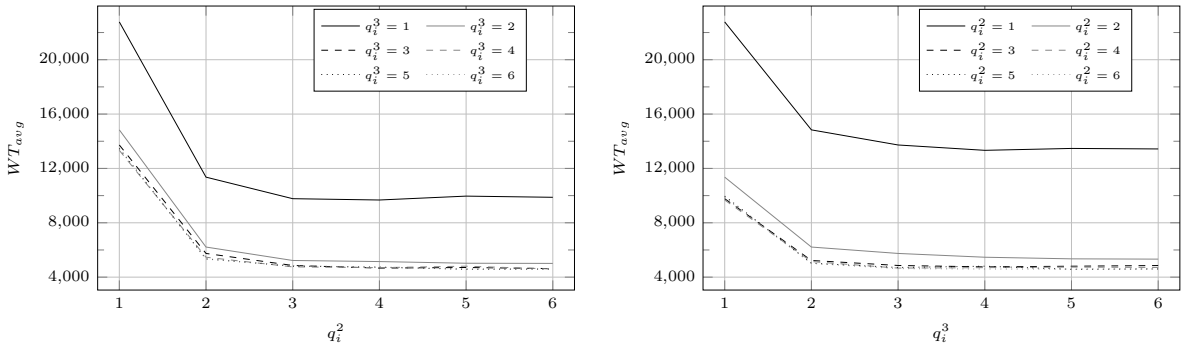


Figure 5.4: Impact of increasing the number of copies of handler and adapter classes

positive impact on the objective function value. This positive effect, however, quickly diminishes when the values become relatively large. Moreover, the number of handlers has a larger impact on the objective function value than the number of adapters. Essentially, this is induced by the structure of the setup times for assembly and disassembly operations as described in Section 5.2. Thus, when investing in the test infrastructure, one should focus on a sufficiently large number of handler copies before increasing the number of adapter copies.

Next, in order to evaluate the impact of the staffing level on the objective function value,

we adjust the value of h for each test instance in order to achieve staffing levels between 20% and 100%. Figure 5.5 plots the resulting increase of the total weighted tardiness in comparison to a staffing level of 100% over the different staffing levels for both resource scenarios. As can be

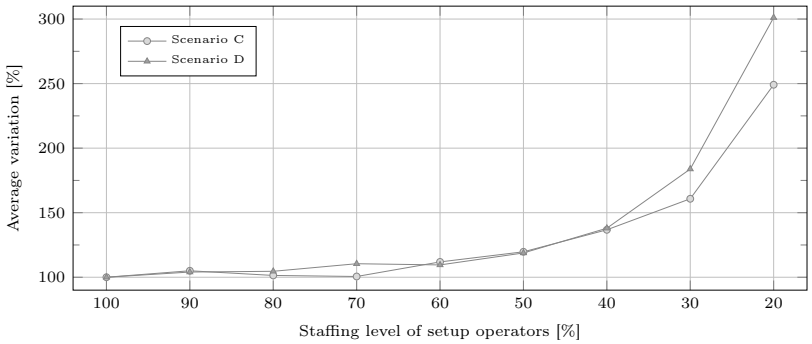


Figure 5.5: Analysis of staffing level (Setup operators)

seen, the total weighted tardiness remains relatively stable for staffing levels between 100% and 70%. Only below this threshold of 70%, we find a significant deterioration of the total weighted tardiness.

As mentioned above, customers at our industry partner fairly frequently request a change of the due dates of their orders. Hence, we now turn our attention to analyzing the effectiveness of rescheduling jobs with our solution approach. To do so, we consider an idealized situation where, at a given point in time (breakpoint), a rescheduling based on the initial schedule is manually initiated. At this point in time, the test cells are associated to some machine configuration and some of the jobs are labelled as emergency orders/jobs (see Section 5.1.2). For the sake of simplicity, we set the corresponding jobs' due dates and weights to the time instant associated with the breakpoint and five, respectively. All operations that are currently processed at the breakpoint define new operations with adapted processing times, so that their processing can potentially be preempted. Other than that, the instance remains unchanged. The initial schedule S_{Init} is determined by the heuristic approach TS-FI. We compute the makespan C_{max} of this solution and randomly determine the breakpoint in the interval $[\lfloor 0.2 \cdot C_{max} \rfloor, \lfloor 0.5 \cdot C_{max} \rfloor]$. Given the set of emergency jobs, we then modify the problem instance in accordance with the remaining planning scenario as described above and then call TS-FI on this modified instance to compute a solution S_{Res} that also takes account of the jobs that have been scheduled before the breakpoint. With respect to selecting the emergency jobs, we make use of different techniques that are based on specifying some fixed percentage of emergency jobs. Our first technique models real-world scenarios by randomly selecting the corresponding orders. The other techniques aim at analyzing the influence of the jobs' parameters. Here, we sort the relevant jobs in the order of non-decreasing (non-increasing) values d_j/w_j and select the emergency jobs based on this ordering.

In order to assess the effectiveness of rescheduling by making use of TS-FI, we determine the total weighted tardiness over all emergency jobs for the solutions S_{Init} and S_{Res} based on the modified parameters and denote these values by $WT_{S_{Init}}$ and $WT_{S_{Res}}$, respectively. We measure the quality of the rescheduling solution with the quality ratio $WT_{S_{Res}}/WT_{S_{Init}}$. For each test

instance, each technique for selecting emergency jobs, and varying percentages δ of emergency jobs, we generated ten random breakpoints. The resulting average quality ratios (columns "Q_{avg}") are presented in Table 5.10. We observe that rescheduling jobs with our solution approach has a

Table 5.10: Rescheduling effectiveness

		Selection of emergency jobs		
		Randomly	Smallest d_j/w_j	Largest d_j/w_j
Scenario	δ [%]	Q_{avg}	Q_{avg}	Q_{avg}
C	4	0.87	1.04	0.59
	6	0.83	1.02	0.61
	8	0.8	1.02	0.62
	10	0.79	1.03	0.63
	12	0.8	1.03	0.64
	14	0.81	1.03	0.64
	16	0.82	1.03	0.65
	18	0.81	1.02	0.65
	20	0.81	1.02	0.67
	25	0.81	1.02	0.71
D	4	0.79	1.02	0.58
	6	0.78	1.02	0.57
	8	0.77	1.01	0.58
	10	0.77	1.01	0.6
	12	0.78	1.01	0.61
	14	0.77	1.01	0.63
	16	0.77	1.01	0.65
	18	0.76	1.01	0.66
	20	0.76	1	0.68
	25	0.77	0.98	0.7

significant positive effect, at least when the emergency orders correspond to jobs j that originally have a relatively large ratio d_j/w_j. This positive effect is relevant even if the percentage of emergency orders is relatively small.

Finally, we analyze the impact of defect hardware resources. We construct idealized scenarios where either only adapters or only handlers are defect. The corresponding failure rate is referred to as ϵ. As above, we first compute an initial solution S_{Init} by means of TS-FI, determine a breakpoint at which the handler or adapter failures occur, and then apply TS-FI as a rescheduling heuristic to determine S_{Res}. The set of defect handlers or adapters is randomly selected, whilst ensuring at least one remaining eligible machine configuration for each operation. We define a quality ratio $100 \cdot (WT_{S_{Res}} - WT_{S_{Init}})/WT_{S_{Init}}$ based on the total weighted tardiness $WT_{S_{Init}}$ and $WT_{S_{Res}}$ over all jobs in the corresponding solutions. As above, we compute ten random breakpoints for each instance and various failure rates for the case of handler or adapter defects. The results are presented in Figure 5.6. It depicts the average quality ratios ("Q_{avg}") as well as the average percentage of unavailable machine configurations ("UMC_{avg}", in comparison to the original instances) over different failure rates for scenarios C and D and the cases of handler or adapter defects. As already indicated by the above results on varying numbers of handler and adapter copies, the defect of handlers has a stronger negative effect on the objective function value than the defect to adapters. This is interesting in light of the fact that the corresponding

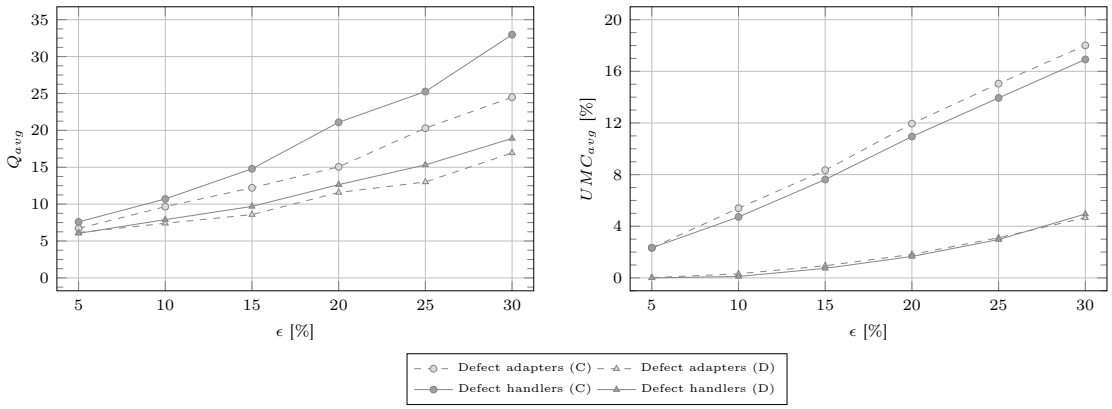


Figure 5.6: Impact of defect hardware resources

difference of the percentage of unavailable machine configurations is relatively small. This effect is particularly pronounced for large failure rates in scenario C. It is a result of the fact that a defect handler requires the disassembly of both adapter and handler. Furthermore, when increasing failure rates, the total weighted tardiness tends to increase slower in scenario D than in scenario C which reflects the fact that managers should carefully determine the number of available copies of the relevant adapter and handler classes in accordance to the investment costs and the time needed to repair or replace defect entities. While these tests have not explicitly targeted the case of multisite testing, it is obvious that the results will carry over in a straightforward manner.

5.5 Conclusion

In this article, we have addressed a semiconductor final test scheduling problem that takes account of setup operator restrictions and aims to minimize the total weighted tardiness. We have introduced a MIP and proposed heuristic tabu search approaches. In a computational study, we have shown that our heuristics are competitive when compared with the performance of a standard solver on the MIP on small problem instances. They have furthermore shown to clearly outperform an as-is solution procedure at our industry partner on large instances that mimic real-world settings within reasonable time and have thus proven to be well suited for daily usage at our industry partner. When setting up the tabu search framework, we found that it pays off to fully integrate the allocation of the setup operators. From a managerial perspective, our results can be summarized by the following simple take-home messages:

- Integrating setup operators into heuristic final-test scheduling approaches pays off at the cost of only slightly increased computational runtimes. When labor is considered to be a constraining factor, corresponding approaches should thus be taken into account by managers.

- There exists a threshold value, above which an increase of the staffing level results in relatively small improvements with respect to on-time delivery of customer orders. Managers should therefore carefully determine a reasonable staffing level for their specific setting.

- With respect to the number of hardware resources at a testing facility, managers should invest in a sufficiently large number of handler copies before increasing the number of adapter copies. The decisions must reflect the investment cost as well as the time needed to repair or replace defect entities.
- Rescheduling should be manually initiated whenever due dates change significantly. This remains true even if only few jobs are affected and when rescheduling decisions induce a significant setup effort.

Bibliography

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401.

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

Allahverdi, A., Gupta, J. N. D., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239.

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

Bard, J. F., Gao, Z., Chacon, R., and Stuber, J. (2012). Real-time decision support for assembly and test operations in semiconductor manufacturing. *IIE Transactions*, 44(12):1083–1099.

Bigras, L.-P., Gamache, M., and Savard, G. (2008). The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, 5(4):685–699.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Sterna, M., and Weglarz, J. (2019). *Handbook on Scheduling: From Theory to Practice*. Springer, Berlin, 2nd edition.

Chen, T.-R., Chang, T.-S., Chen, C.-W., and Kao, J. (1995). Scheduling for IC sort and test with preemptiveness via Lagrangian relaxation. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(8):1249–1256.

Chen, T.-R. and Hsia, T. C. (1997). Scheduling for IC sort and test facilities with precedence constraints via Lagrangian relaxation. *Journal of Manufacturing Systems*, 16(2):117–128.

Deng, Y., Bard, J. F., Chacon, G. R., and Stuber, J. (2010). Scheduling back-end operations in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 23(2):210–220.

Freed, T., Doerr, K. H., and Chang, T. (2007). In-house development of scheduling decision support systems: case study for scheduling semiconductor device test operations. *International Journal of Production Research*, 45(21):5075–5093.

Freed, T. and Leachman, R. C. (1999). Scheduling semiconductor device test operations on multihead testers. *IEEE Transactions on Semiconductor Manufacturing*, 12(4):523–530.

Freed, T., Leachman, R. C., and Doerr, K. H. (2002). A taxonomy of scheduling problems in semiconductor device test operations. In *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing*, pages 252–259. IEEE.

Giffler, B. and Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Gupta, A. K. and Sivakumar, A. I. (2006). Job shop scheduling techniques in semiconductor manufacturing. *International Journal of Advanced Manufacturing Technology*, 27(11–12):1163–1169.

Hao, X.-C., Wu, J.-Z., Chien, C.-F., and Gen, M. (2014). The cooperative estimation of distribution algorithm: a novel approach for semiconductor final test scheduling problems. *Journal of Intelligent Manufacturing*, 25(5):867–879.

Herrmann, J. W., Lee, C.-Y., and Hinchman, J. (1995). Global job shop scheduling with a genetic algorithm. *Production and Operations Management*, 4(1):30–45.

Kim, Y.-D., Kang, J.-H., Lee, G.-E., and Lim, S.-K. (2011). Scheduling algorithms for minimizing tardiness of orders at the burn-in workstation in a semiconductor manufacturing system. *IEEE Transactions on Semiconductor Manufacturing*, 24(1):14–26.

Kress, D., Müller, D., and Nossack, J. (2019). A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, 41(1):179–217.

Lee, C.-Y., Uzsoy, R., and Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775.

Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140.

Lin, J. T., Wang, F. K., and Lee, W. T. (2004). Capacity-constrained scheduling for a logic IC final test facility. *International Journal of Production Research*, 42(1):79–99.

Mastrolilli, M. and Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20.

Mathirajan, M. and Sivakumar, A. I. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *International Journal of Advanced Manufacturing Technology*, 29(9–10):990–1001.

Mönch, L., Fowler, J. W., Dauzère-Pérès, S., Mason, S. J., and Rose, O. (2011). A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14(6):583–599.

Müller, D. and Kress, D. (2019). Filter-and-fan approaches for scheduling flexible job shops under workforce constraints. Working Paper. University of Siegen.

Ovacik, I. M. and Uzsoy, R. (1992). A shifting bottleneck algorithm for scheduling semiconductor testing operations. *Journal of Electronics Manufacturing*, 2(3):119–134.

Ovacik, I. M. and Uzsoy, R. (1994). Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 32(6):1243–1263.

Ovacik, I. M. and Uzsoy, R. (1995). Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. *International Journal of Production Research*, 33(11):3173–3192.

Ovacik, I. M. and Uzsoy, R. (1996). Decomposition methods for scheduling semiconductor testing facilities. *International Journal of Flexible Manufacturing Systems*, 8(4):357–387.

Pearn, W. L., Chung, S. H., Chen, A. Y., and Yang, M. H. (2004). A case study on the multistage IC final testing scheduling problem with reentry. *International Journal of Production Economics*, 88(3):257–267.

PwC (2012). Faster, greener, smarter – reaching beyond the horizon in the world of semiconductors. `http://www.pwc.com/gx/en/technology/publications/assets/pwc-faster-greener-smarter.pdf`. PricewaterhouseCoopers AG.

PwC (2013). Spotlight on automotive – pwc semiconductor report. `https://www.pwc.com/gx/en/technology/publications/assets/pwc-semiconductor-survey-interactive.pdf`. PricewaterhouseCoopers AG.

Sang, H.-Y., Duan, P.-Y., and Li, J.-Q. (2018). An effective invasive weed optimization algorithm for scheduling semiconductor final testing problem. *Swarm and Evolutionary Computation*, 38:42–53.

Uzsoy, R., Lee, C.-Y., and Martin-Vega, L. A. (1992a). A review of production planning and scheduling models in the semiconductor industry part I: system characteristics, performance evaluation and production planning. *IIE Transactions*, 24(4):47–60.

Uzsoy, R., Lee, C.-Y., and Martin-Vega, L. A. (1992b). Scheduling semiconductor test operations: minimizing maximum lateness and number of tardy jobs on a single machine. *Naval Research Logistics*, 39(3):369–388.

Uzsoy, R., Lee, C.-Y., and Martin-Vega, L. A. (1994). A review of production planning and scheduling models in the semiconductor industry part II: Shop-floor control. *IIE Transactions*, 26(5):44–55.

Uzsoy, R., Martin-Vega, L. A., Lee, C.-Y., and Leonard, P. A. (1991). Production scheduling algorithms for a semiconductor test facility. *IEEE Transactions on Semiconductor Manufacturing*, 4(4):270–280.

Wang, S. and Wang, L. (2015). A knowledge-based multi-agent evolutionary algorithm for semiconductor final testing scheduling problem. *Knowledge-Based Systems*, 84:1–9.

Wang, S., Wang, L., Liu, M., and Xu, Y. (2015). A hybrid estimation of distribution algorithm for the semiconductor final testing scheduling problem. *Journal of Intelligent Manufacturing*, 26(5):861–871.

Wu, J.-Z. and Chien, C.-F. (2008). Modeling semiconductor testing job scheduling and dynamic testing machine configuration. *Expert Systems with Applications*, 35(1–2):485–496.

Wu, J.-Z., Hao, X.-C., Chien, C.-F., and Gen, M. (2012). A novel bi-vector encoding genetic algorithm for the simultaneous multiple resources scheduling problem. *Journal of Intelligent Manufacturing*, 23(6):2255–2270.

Xiong, H. H. and Zhou, M. (1998). Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search. *IEEE Transactions on Semiconductor Manufacturing*, 11(3):384–393.

Zhang, Z., Zhang, M. T., Niu, S., and Zheng, L. (2006). Capacity planning with reconfigurable kits in semiconductor test manufacturing. *International Journal of Production Research*, 44(13):2625–2644.

Zhang, Z., Zheng, L., Hou, F., and Li, N. (2011). Semiconductor final test scheduling with Sarsa(λ, k) algorithm. *European Journal of Operational Research*, 215(2):446–458.

Zheng, X.-L., Wang, L., and Wang, S.-Y. (2014). A novel fruit fly optimization algorithm for the semiconductor final testing scheduling problem. *Knowledge-Based Systems*, 57:95–103.

Zhu, X. and Wilhelm, W. E. (2006). Scheduling and lot sizing with sequence-dependent setup: A literature review. *IIE Transactions*, 38(11):987–1007.

Chapter 6

An Algorithm Selection Approach for the Flexible Job Shop Scheduling Problem: Choosing Constraint Programming Solvers through Machine Learning

Abstract

Standard constraint programming solvers are known to perform remarkably well for most scheduling problems. However, when comparing the performance of different available solvers, there is usually no clear winner over all relevant problem instances. This gives rise to the question of how to select a promising solver when knowing the concrete instance to be solved. In this article, we aim to provide first insights into this question for the flexible job shop scheduling problem. We investigate relative performance differences among state-of-the-art commercial and non-commercial constraint programming solvers on problem instances taken from the literature as well as randomly generated problem instances. We then leverage the resulting performance complementarity to propose algorithm selection approaches that predict the best solver for a given problem instance based on instance features or parameters. The approaches are based on two machine learning techniques, decision trees and deep neural networks, in various variants. In a computational study, we analyze the performance of the resulting algorithm selection models and show that our approaches outperform the use of single solvers and should thus be considered as a relevant tool by decision makers in practice.

Authors	David Müller[a] david.mueller@uni-siegen.de Marcus G. Müller[b] marcus.mueller@dlr.de Dominik Kress[a,c] dominik.kress@hsu-hh.de Erwin Pesch[a,d] erwin.pesch@uni-siegen.de [a] *University of Siegen, Management Information Science, Kohlbettstraße 15, 57068 Siegen, Germany* [b] *German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Münchner Straße 20, 82234 Wessling, Germany* [c] *Helmut Schmidt University - University of the Federal Armed Forces Hamburg, Business Administration, especially Procurement and Production, Friedrich-Ebert-Damm 245, 22159 Hamburg, Germany* [d] *HHL Leipzig, Center for Advanced Studies in Management, Jahnallee 59, 04109 Leipzig, Germany*
Full citation	Müller, D., Müller, M. G., Kress, D., and Pesch, E. (2021). An algorithm selection approach for the flexible job shop scheduling problem: choosing constraint programming solvers through machine learning. Working Paper. University of Siegen.

6.1 Introduction

Most real-world manufacturing environments are highly complex systems that feature a variety of constraints and characteristics that correspond to fairly specific settings found at the related companies. Nevertheless, they usually have similarities that can be modelled by means of generic problem formulations that reduce the complex settings to their very core. In some cases, solutions based on these problem formulations can be implemented directly. Alternatively, they may serve as building blocks or elementary subproblems in more specific optimization approaches. They are thus of major importance not only from a theoretical perspective.

Certainly, in the scheduling context, one of the most practically relevant but rather compact problem settings is the *flexible job shop scheduling problem* (FJSP). It generalizes the well-known *job shop scheduling problem* (JSP; see, e.g., the survey by Błażewicz et al. 1996) and is composed of a set of jobs and a set of machines. Each job consists of a set of operations that have to be processed non-preemptively in a predefined sequence without overlapping in time in order to complete the job. Each operation is associated to a specific machine for its processing. The FJSP (see Blazewicz et al., 2019; Brucker and Schlie, 1990; Chaudhry and Khan, 2016) generalizes the latter assumption in order to take account of the fact that manufacturing systems oftentimes feature multiple machines of the same type as well as multi-purpose machines that are able to process different types of operations. More specifically, it assumes that each operation must be processed by exactly one machine out of a set of *eligible machines*. In both variants, the problem is to allocate the operations to the (eligible) machines and to sequence the operations on the machines so that some performance measure is optimized, whilst ensuring that, at any time, each machine processes at most one operation and each operation is processed on at most one machine.

In practice, manufacturing companies typically compute feasible schedules in rolling horizon based planning approaches that, for example, allow to dynamically take account of varying customer orders or maintenance issues arising on the shop floor. In order to guarantee smooth production processes, these approaches rely on the ability to (re-)compute feasible schedules with an acceptable solution quality in a short amount of time. In this regard, *constraint programming* (CP) approaches have been successfully applied to many scheduling problems and, in particular, to shop scheduling problems where disjunctive constraints are prevalent as it is the case for JSP and FJSP (Dorndorf et al., 2000, 2002). Especially, the standard CP solver provided by IBM ILOG CPLEX has shown to provide an excellent performance (see, e.g., Lunardi et al. 2020, Wari and Zhu 2019, Laborie et al. 2018, Ham and Cakici 2016, and, for the specific case of the FJSP, Kress and Müller 2019, Puget 2013). Of course, there exist alternative, non-commercial, CP solvers. One of them is included in OR-Tools, an open source software suite developed by Google. It won most of the gold medals in the 2020 MiniZinc Challenge (see MiniZinc, 2020), an annual competition that compares different CP solvers on a variety of combinatorial optimization problems (Stuckey et al., 2014). A performance evaluation between Google's OR-Tools and the IBM ILOG CPLEX CP Optimizer was recently conducted by Da Col and Teppan (2019a,b) for the JSP. The paper at hand complements this study by focussing on the FJSP.

We present a thorough computational study that analyzes the relative performance of non-commercial CP solvers when compared with the CPLEX CP Optimizer for solving well-known benchmark instances taken from the literature as well as randomly generated test instances. We focus on practically relevant industry settings, where, as outlined above, the solvers must be able to quickly compute feasible solutions. To the best of our knowledge, there exist only one corresponding study as part of the 2013 MiniZinc Challenge (MiniZinc, 2013) for the FJSP. In the face of the rapid development of CP solvers, however, this study is rather outdated. Additionally, it solely considers 5 instances.

In their aforementioned study, Da Col and Teppan (2019a,b) find that, on average, CPLEX performs only slightly better than OR-Tools on small-sized problem instances of JSP. On large-scale problem instances, CPLEX outperforms OR-Tools more significantly. However, when looking at the results individually, there exist quite a few instances, where OR-Tools is able to determine better solutions or proves optimality faster than CPLEX. Similar results can be observed for many optimization problems when comparing the performance of different algorithms. Typically, there exists no single best algorithm that outperforms all other approaches on all problem instances. While some algorithm may perform best on some set of instances, another algorithm may perform better on some other instance set. The problem of selecting the most promising algorithm out of a given set of algorithms for a previously unseen instance of some specific problem is also referred to as the *algorithm selection problem*. It was formally introduced by Rice (1976). In order to predict the best algorithm, algorithm selection approaches generally make use of features describing the given instance. Algorithm selection approaches can be categorized into *classification methods* and *regression methods*. While the former methods directly predict an algorithm for a given instance, the latter methods predict continuous algorithm performance measures, e.g., regarding the computational time, which are then used for the selection of an algorithm.

Recently, algorithm selection approaches have been successfully applied for different problem domains. For the sake of brevity, we refer to the surveys by Kerschke et al. (2019) and Smith-Miles (2009) that cover the methodology and challenges of building an algorithm selector in general and summarize practical algorithm selection approaches for various optimization problems. One of the most prominent examples in the field of combinatorial optimization is concerned with the propositional satisfiability problem (SAT). SATzilla (Xu et al., 2012, 2008), a corresponding algorithm selection approach, has won several medals in multiple SAT competitions. Another relevant study is presented by Kerschke et al. (2018), who analyze the complementary performance of five state-of-the-art inexact solvers for solving the traveling salesman problem (TSP) on various benchmark instances and utilize this knowledge to build a very successful algorithm selector. In the domain of scheduling problems, a successful implementation of an algorithm selection approach is presented by Messelis and De Causmaecker (2014) for the multi-mode resource-constrained project scheduling problem (MRCPSP).

The success of algorithm selection approaches for the above problem domains has motivated us to develop algorithm selection approaches for the FJSP in order to leverage the performance

complementarity of the CP solvers analyzed in our computational study. Our algorithm selection approaches fall into the category of classification methods. In order to predict the best solver for a given instance, we make use of two machine learning techniques. First, we make use of decision trees, which are commonly used in the aforementioned studies. Second, we make use of deep neural networks that have - to the best of the authors' knowledge - barely been used for implementing algorithm selection approaches, while they have recently proven very successful in other applications like speech or image recognition. A similar observation is due to Kraus et al. (2020), who find that the Operations Research (OR) community "is still in its infancy with regard to adopting" the deep learning technology. In an extensive literature review of papers published between October 2018 and September 2019 across the major OR journals, the authors find only three papers that explicitly make use of deep learning techniques. A directly related survey on the possibilities of leveraging machine learning to solve combinatorial optimization problems (rather than selecting from existing algorithms) is presented by (Bengio et al., 2021).

It is important to note that the set of features that is used for characterizing an instance is highly relevant for the performance of an algorithm selection approach. Therefore, we compute a diverse set of features based on instance characteristics. However, the necessary a priori knowledge that is needed for selecting appropriate features is not always available in practice. Therefore, we additionally analyze the case of providing unprocessed instance data, i.e., the processing times of operations on their eligible machines, that do not result from pre-computing specific features. In an extensive computational study, we analyze the performance of our algorithm selection approaches and show that they perform better than a single solver.

Our research is also related to the research fields of algorithm configuration and hyper-heuristics. Algorithm configuration refers to the problem of obtaining parameter settings of a given algorithm to optimize the performance on given problem instances (see, e.g., Hutter et al., 2014). Hyper-heuristics are approaches that select or generate heuristics for solving a given problem based on a given set of heuristics or components (see Burke et al., 2013). In the context of production scheduling, several approaches have been proposed by, e.g., Dorndorf and Pesch (1995); Hart et al. (1998); Vázquez-Rodríguez and Petrovic (2010).

Summing up, the contribution of this paper is twofold. First, we provide a computational study on the performance of commercial and non-commercial state-of-the-art CP solvers on a wide variety of benchmark instances of FJSP. Second, we develop multiple variants of an algorithm selection approach that aims to leverage the performance complementarity of the CP solvers. The overall setup is such that these approaches are applicable for practitioners in the sense that the computational times are in ranges that allow their usage in rolling horizon based scheduling approaches that require comparatively quick computations of feasible schedules.

The remainder of this paper is structured as follows. First, in Section 6.2, we provide a formal definition of the FJSP. The computational study on the performance of the considered CP solvers is provided in Section 6.3. Next, in Section 6.4, we develop our algorithm selection approaches in detail, and then evaluate their performance in Section 6.5. We close the paper with a summary in Section 6.6.

6.2 Definition of the Flexible Job Shop Scheduling Problem

The FJSP is defined as follows. Given is a set $J = \{J_1, \ldots, J_n\}$ of n *jobs*. Each job $J_i \in J$ consists of a set $O_i = \{i_1, \ldots, i_{q_i}\}$ of q_i operations that have to be processed on a set $M = \{M_1, \ldots, M_m\}$ of m *machines*. The processing of operations must not be preempted. The sets O_i are assumed to be linearly ordered for all $i \in \{1, \ldots, n\}$. This relates to the fact that, for any pair of operations $i_j, i_{j'} \in O_i$ with $j < j'$, $i_{j'}$ may only start to be processed after the processing of i_j has completed. Each operation $i_j \in O_i$, $i \in \{1, \ldots, n\}$, must be processed on exactly one machine out of a non-empty set of *eligible machines* $M_{i_j} \subseteq M$. Each machine can process only one operation at a time and each operation can be processed by at most one machine at a time. The processing time of an operation $i_j \in O_i$ of a job $J_i \in J$ on an eligible machine $M_k \in M_{i_j}$ is denoted by $p^k_{i_j} \in \mathbb{N}^+$. We assume that all jobs and machines are available at the beginning of the planning horizon. The problem is to find a schedule, i.e., an assignment of operations to eligible machines and a sequencing of the operations on the machines, that is feasible with respect to the constraints stated above and that optimizes some performance measure. The completion time of an operation $i_j \in O_i$ of job $J_i \in J$ in a schedule is denoted by C_{i_j} and the completion time of job $J_i \in J$ is denoted by C_i. A job is completed if all of its operations are completed, i.e., $C_i = C_{i_{q_i}}$ for all $i \in \{1, \ldots, n\}$. We restrict our attention to the objective of minimizing the makespan $C_{max} = \max_{i \in \{1,\ldots,n\}} C_i$. The resulting problem is strongly NP-hard as it generalizes the JSP, which is well known to be NP-hard in the strong sense when aiming to minimize the makespan (Lenstra and Rinnooy Kan, 1979).

6.3 Performance Evaluation of Constraint Programming Solvers

In order to assess the performance of relevant CP solvers (see Section 6.3.1), we conducted extensive computational tests on a variety of test instances (see Section 6.3.2). The tests were performed on a PC with an Intel® Core™ i7-4770 CPU, running at 3.4 GHz, with 16 GB of RAM under a 64-bit version of Windows 8. The results are presented in Section 6.3.3.

6.3.1 Constraint Programming Solvers

The CP solvers analyzed in our computational study are listed in Table 6.1. Note that CPLEX

Table 6.1: Overview of CP solvers

Solver	Version	Modelling language	Reference
Choco	4.0.4	MiniZinc	Prud'homme et al. (2016)
Chuffed	0.10.4	MiniZinc	Chu et al. (2019)
IBM ILOG CPLEX	12.9.0	OPL	IBM (2019a)
Gecode	6.2.0	MiniZinc	Schulte et al. (2019)
Google OR-Tools	7.2	MiniZinc and directly via API	Google (2019a)

and OR-Tools are optimization suites that provide not only CP solvers. In our tests, however, we solely focussed on the respective CP solvers.

As the performance of a solver depends on the modelling of a problem, we used a solver-independent modelling language called MiniZinc. The MiniZinc problem specification separates the definition of the general model and the data that defines some particular instance. To ensure readability by a solver, the MiniZinc specifications are translated to a low-level solver input language, called FlatZinc, by considering solver dependent constraint specifications. For further details, we refer to Nethercote et al. (2007). For our tests, we made use of a MiniZinc model for the FJSP that was part of the MiniZinc Challenge 2013 (Stuckey et al., 2014). This model includes *search annotations* that define search strategies for finding feasible solutions. Below, we will also consider the pure model without these annotations. The FlatZinc format is supported by all solvers listed in Table 6.1 but CPLEX. For the latter CP solver, we therefore used a FJSP model provided in the corresponding online documentation (IBM, 2019b) which makes use of IBM's Optimization Programming Language (OPL) and a set of modelling features that allow covering temporal dimensions that are especially relevant in scheduling problems (details are given in IBM, 2019a; Laborie et al., 2018). Google's OR-Tools provide similar features (variable and constraint types) for modelling scheduling problems (Google, 2019b), which we made use of in an additional implementation of the FJSP in Java using the OR-Tools library via its API.

In line with the requirements in real-world manufacturing systems outlined in Section 6.1, we set the time limit for each solver to rather small values. We considered two settings, 300 seconds and 30 seconds. Moreover, we activated the parallel processing mode in each solver.

6.3.2 Test Instances

We considered benchmark instances from the literature as well as randomly generated test instances. All literature instances are available online via the FJSP instance collection (Mastrolilli, 2020). We restrict ourselves to summarizing the main features of the instances sets in this section. More details are summarized by Behnke and Geiger (2012) and Mastrolilli and Gambardella (2000).

- Instance set BR (Brandimarte, 1993): This set is composed of 10 (available) instances with 10 to 20 jobs, 4 to 15 machines, and 3 to 15 operations per job. The maximum number of eligible machines of each operation ranges from 2 to 6. The processing times range from 1 to 20 time units.

- Instance set CH (Chambers and Barnes, 1996): The set consists of 21 instances that were generated from three challenging JSP problem instances, introduced by Fisher and Thompson (1963) and Lawrence (1984), by "replicating" machines. The processing times of operations are identical on all of their eligible machines and correspond to the one of the original problem instance. The set features instances with 10 to 15 jobs and 11 to 17 machines. For all instances, the number of operations per job equals the corresponding number of machines.

- Instance set DA (Dauzère-Pérès and Paulli, 1997): The set consists of 18 instances with 10 to 20 jobs, 5 to 10 machines, and 15 to 25 operations per job. The set of eligible machines

of each operation was constructed randomly by assuming that each machine is eligible with a 10 to 50 percent probability. The processing times range from 10 to 100 time units.

- Instance set HU (Hurink et al., 1994; Mastrolilli, 2020): These benchmark instances are based on three JSP problem instances by Fisher and Thompson (1963) and, according to Mastrolilli and Gambardella (2000) and Behnke and Geiger (2012), 40 JSP problem instances by Lawrence (1984). The set of eligible machines of each operation consists of the associated machine of the original problem instance and, in addition, any of the other machines with a given probability. By considering different values for these probabilities, Hurink et al. (1994) generated three different instance sets, denoted by edata, rdata, and vdata. The average number of eligible machines per operation is 1.15 in edata, 2 in rdata and $m/2$ in vdata, so that the degree of flexibility is lowest in the instances of the first set. The processing times of operations are identical on all of their eligible machines and correspond to the ones of the original problem instance. According to Behnke and Geiger (2012), the remaining instances associated to Hurink et al. (1994) in the database available via Mastrolilli (2020) are based on benchmark instances by Adams et al. (1988), Carlier and Pinson (1989) and Applegate and Cook (1991) and were generated in the same manner. This results in a total of 198 HU instances.

Our random testbed, denoted by RA, is composed of three classes of instance sets with 10 jobs (small instances), 20 jobs (medium instances) or 30 to 40 jobs (large instances), respectively. Each class consists of five sets (sfjsp1–sfjsp5, mfjsp1–mfjsp5, lfjsp1–lfjsp5) with different settings regarding the remaining parameters. Each set features ten randomly generated test instances with the parameter ranges illustrated in Table 6.2. Hence, there are 150 instances in our random

Table 6.2: Parameters of random testbed

small instances					medium instances					large instances				
set	n	m	q_i	$\lvert M_{i_j}\rvert$	set	n	m	q_i	$\lvert M_{i_j}\rvert$	set	n	m	q_i	$\lvert M_{i_j}\rvert$
sfjsp1	10	2	$[1,3]$	$[1,2]$	mfjsp1	20	2	$[1,3]$	$[1,2]$	lfjsp1	30	3	$[2,3]$	$[1,2]$
sfjsp2	10	2	$[2,3]$	$[1,2]$	mfjsp2	20	3	$[2,3]$	$[1,2]$	lfjsp2	30	5	$[2,5]$	$[1,3]$
sfjsp3	10	3	$[1,4]$	$[1,2]$	mfjsp3	20	3	$[2,5]$	$[1,2]$	lfjsp3	40	5	$[2,5]$	$[2,3]$
sfjsp4	10	3	$[2,4]$	$[1,3]$	mfjsp4	20	4	$[5,8]$	$[2,3]$	lfjsp4	40	8	$[5,12]$	$[4,5]$
sfjsp5	10	3	$[3,5]$	$[1,3]$	mfjsp5	20	6	$[8,10]$	$[1,3]$	lfjsp5	40	10	$[8,12]$	$[4,5]$

testbed. While n and m are fixed for the instances of each set. The number of operations per job and the number of eligible machines for each operation were drawn from uniform distributions over the intervals given in the table. Based on this data, the set of eligible machines was randomly determined for all operations. Here, each machine was selected with the same probability. The processing times were then generated as follows. First, auxiliary integer parameters p_{i_j} for all jobs $J_i \in J$ and operations $i_j \in O_i$ were drawn from uniform distributions over the interval $[10, 100]$. Based on these parameters, we constructed varying processing times over the corresponding eligible machines by drawing integer values $p_{i_j}^k$ from uniform distributions over the interval $[\lfloor(1-p)\cdot p_{i_j}\rfloor, \lfloor(1+p)\cdot p_{i_j}\rfloor]$, where $p = 0.1$. All test instances of the RA set are available in supplementary files that accompany this paper (Müller et al., 2021).

6.3.3 Computational Results

Define $p_{i_j}^{\min} = \min_{M_k \in M_{i_j}} p_{i_j}^k$ for all $J_i \in J$ and $i_j \in O_i$, and set $P = \sum_{i=1}^{n} \sum_{i_j \in O_i} p_{i_j}^{\min}$. Then, a simple lower bound on the makespan of a given instance of FJSP is as follows:

$$LB = \max \left\{ \max_{i \in \{1,\ldots,n\}} \sum_{i_j \in O_i} p_{i_j}^{\min}, \left\lceil \frac{P}{m} \right\rceil \right\}.$$

Note that, for the sake of brevity, we do not explicitly state the concrete instance in the definition of the bound. We make use of this bound to measure the quality of a (not necessarily optimal) solution with makespan $C_{\max}$ returned by one of the considered CP solvers within the time limit with the *quality ratio* $100 \cdot (C_{\max} - LB)/LB$. In addition, we introduce a scoring system inspired by the one used in MiniZinc (2013). For a given instance, we assign one point to a solver, if it proves optimality faster or finds a better solution than all of the other solvers. If the solvers are indistinguishable, e.g., if the makespan of the two best solutions is equal without having proven optimality, no point is assigned.

The computational results over the entire set of test instances are presented in Table 6.3. For each solver and each time limit, it presents information about the percentage of instances for

Table 6.3: Performance of CP solvers

	Time limit: 300 seconds					Time limit: 30 seconds				
CP Solver	feas. [%]	opt. [%]	Q_{avg}	t_{avg} [s]	score	feas. [%]	opt. [%]	Q_{avg}	t_{avg} [s]	score
Choco*	99.24	23.68	39.39	240.5	0	99.24	13.85	88.2	27.7	0
Choco	99.24	23.43	47.16	242.27	0	99.24	13.6	108.43	27.67	0
Chuffed*	100	2.02	96.22	296.15	0	100	1.01	97.18	29.81	0
Chuffed	42.82	4.53	267.9	278.39	0	27.96	1.51	280.31	28.66	0
CPLEX	100	46.85	8.17	164.94	132	100	42.32	8.44	18.83	150
Geocode*	99.24	0.25	103.33	299.37	0	99.24	0.25	103.73	30.02	0
Geocode	99.24	0.5	113.51	298.56	0	99.24	0.25	113.88	29.97	0
OR-Tools (API)	100	44.58	13.5	178.53	26	100	33.5	31.99	21.49	17
OR-Tools*	99.75	51.64	8.67	151.59	135	99.75	46.6	13.47	17.42	138
OR-Tools	89.67	49.87	10.5	140.77	42	83.38	44.33	12.14	15.8	32

which a feasible or optimal solution was obtained within the given time limit (columns "feas." and "opt."), the average quality ratios (columns "Q_{avg}"), the average runtimes (columns "t_{avg}"), and the sum of scoring points (columns "score"). The asterisk symbol ($*$) in the solver column indicates that the underlying MiniZinc model includes the aforementioned search annotations.

It can be seen that, with the exception of Chuffed on the MiniZinc model without search annotations, the solvers are competitive with respect to their ability to determine feasible solutions. Note, however, that some solvers had conversion errors for three instances of the BR set and one instance of the HU set. The search annotations specified in the MiniZinc model for the FJSP have a positive impact on the performance of the solvers (Chuffed, Gecode and OR-Tools). Moreover, it can be concluded that CPLEX provides the overall best performance with respect to the solution quality. A more detailed analysis shows that this effect is particularly pronounced for instances of larger size and when the time limit is set to a lower value.

However, when analyzing the percentage of instances for which an optimal solution was obtained, CPLEX is narrowly beaten by OR-Tools on MiniZinc with search annotations. This result is also reflected in the distribution of the scoring points. CPLEX and OR-Tools are the only solvers that received scoring points. For 62 (60) instances no clear winner regarding our scoring system (solvers indistinguishable) was found for a time limit of 300 (30) seconds. These results are in line with the computational study conducted by Da Col and Teppan (2019a,b), which demonstrates the strengths of CPLEX especially on large-scale problem instances of JSP, but also shows that OR-Tools is competitive on classical benchmark instances.

With respect to the different OR-Tools modes, most points were obtained when using MiniZinc with search annotations. The two other modes (API mode and MiniZinc without search annotations) obtained the majority of their points only because of determining optimal solutions faster than MiniZinc with search annotations. However, the corresponding time differences are in the range of a few milliseconds, so that the aforementioned benefit of using OR-Tools when compared with CPLEX is even larger than indicated in Table 6.3. This is highlighted in Table 6.4, where we present the distribution of scoring points when restricting the analysis to CPLEX and OR-Tools on MiniZinc with search annotations. Apparently, OR-Tools on MiniZinc

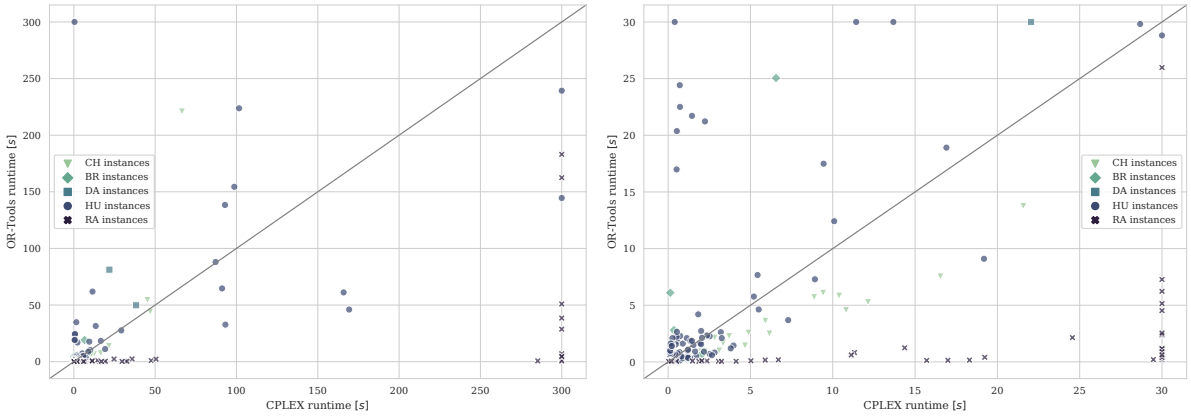
Table 6.4: Distribution of scoring points when restricting the analysis to CPLEX and OR-Tools*

	Time limit: 300 seconds	Time limit: 30 seconds
CPLEX	149	165
OR-Tools*	210	210

with search annotations and CPLEX tend to be competitive in terms of performance, i.e., one solver outperforms the other on some instances, while it is beaten by the other solver on some other instances. This can be quantified when using the *competitiveness ratio* introduced by Messelis and De Causmaecker (2014). Let T be the set of all 397 test instances. Furthermore, denote by A the set of instances for which some specific algorithm X outperforms another algorithm Y with respect to some criterion. Set B is defined analogously as the set of instances on which X is beaten by Y. Then, the competitiveness ratio c is defined by $c = 2 \cdot \min\{\frac{|A|}{|T|}, \frac{|B|}{|T|}\}$. Obviously, a ratio close to one indicates that X and Y perform similarly well. In our case, based on the scoring points presented in Table 6.4, the competitiveness ratio of CPLEX and OR-Tools on MiniZinc with search annotations is 0.751 for the 300 seconds time limit and 0.831 for the 30 seconds time limit. Note that, if we assigned a point to both solvers in case of a tie, these values were even larger. Hence, we will restrict our attention to these two settings in the remainder of this paper. For the sake of brevity, we will not explicitly specify the fact that we are making use of the MiniZinc search annotations and we will omit the asterisk when referring to OR-Tools.

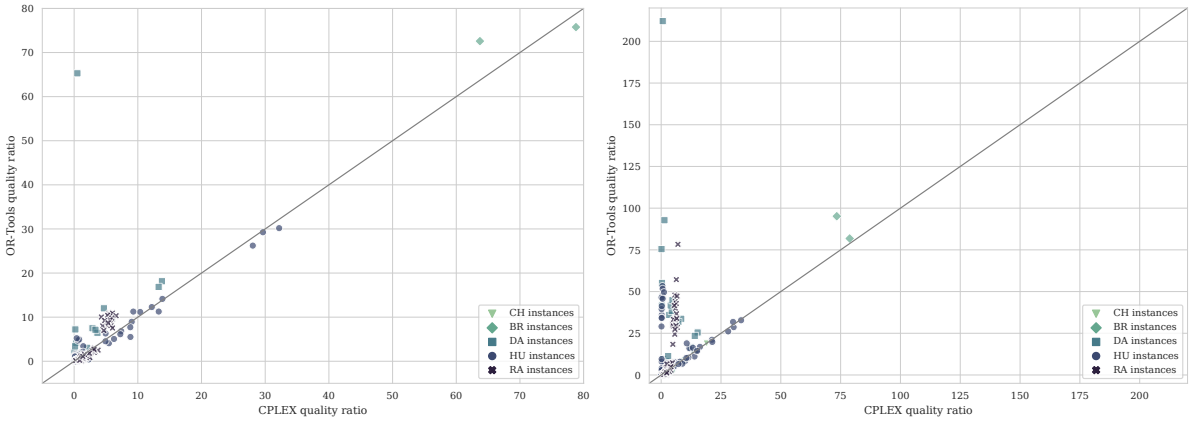
In order to gain further insights into the competitiveness of both solvers, a more detailed analysis that takes account of the different sets of test instances is presented in Figures 6.1 and 6.2.

Figure 6.1 illustrates the runtimes of CPLEX and OR-Tools when restricting the attention to the instances for which at least one of the two solvers was able to determine an optimal solution,



(a) Time limit: 300 seconds (b) Time limit: 30 seconds

Figure 6.1: Performance of CPLEX and OR-Tools - ability to determine provable optimal solutions



(a) Time limit: 300 seconds (b) Time limit: 30 seconds

Figure 6.2: Performance of CPLEX and OR-Tools - quality ratios

including the proof of optimality. The data is represented by means of scatter plots, with each data point representing a specific test instance. The maximum values of the runtimes depicted in the plots represent the time limits of 300 and 30 seconds, respectively. Hence, data points at the very top or right of the plots represent instances on which one of the solvers was not able to determine an optimal solution within the time limit. If both solvers have determined an optimal solution for a specific instance within the same time, the respective data point lies on the diagonal lines in the plots. As can be seen, there exist quite a few instances that were solved to optimality by both solvers within milliseconds or a few seconds. However, for some instances, the performance of the solvers is significantly different, which is in line with the findings of Pesch and Tetzlaff (1996). As an example, we observe that (at least for the smaller time limit), CPLEX seems to perform better than OR-Tools on the HU instances. For the RA and CH instances, on the other hand, OR-Tools seems to perform better.

Figure 6.2 focusses on the quality ratios of the computed solutions. For reasons of clarity, we omit data points for which the quality ratio of both solvers is identical. As already observed in Table 6.3, CPLEX outperforms OR-Tools on most instances, especially when the time limit is set to only 30 seconds. However, on a few instances, OR-Tools was able to compute better solutions.

In summary, we conclude that there clearly is a potential payoff when using an algorithm selection approach based on the CP solvers CPLEX and OR-Tools for the FJSP. Hence, the remainder of this paper aims at developing such an approach.

6.4 Algorithm Selection Approaches

Rice (1976) introduces an abstract model for the algorithm selection problem. It consists of the following elements:

- A *problem space* X, that consists of a set of instances of a problem. In our case, the FJSP represents the (optimization) problem under consideration.
- An *algorithm space* A, that includes a set of algorithms for (potentially heuristically) solving the problem instances. Here, based on the computational results presented in Section 6.3, A is composed of the CP solvers CPLEX and OR-Tools with specified time limits.
- A *performance measure* y, which is used to quantify the performance of an algorithm on a given problem instance. We make use of the scoring system introduced in Section 6.3.3.

The algorithm selection problem is to find a mapping $S : X \to A$ that maps each problem instance to an algorithm that maximizes the performance measure y. Usually, in order to (heuristically) obtain a mapping, one defines a *feature set* F to characterize the problem instances and then, for example, applies machine learning techniques to select an appropriate algorithm based on the concrete features of an instance.

This section proceeds as follows. In Section 6.4.1, we define an appropriate feature set for FJSP. Hereafter, in Section 6.4.2, we randomly construct a problem space that aims at providing

a sufficiently large amount of instances to later construct and evaluate our algorithm selection approaches based on machine learning techniques. Our procedure of generating the problem space is such that some of the instances mimic the ones of the sets introduced in Section 6.3.2. After a preliminary analysis on the competitiveness of the CP solvers on the problem space in Section 6.4.3, our algorithm selection approaches are introduced in detail in Section 6.4.4.

6.4.1 Feature Set

The definition of the feature set is a crucial part of any algorithm selection approach as it is used to characterize the instances and then link these characteristics to the performance of the algorithms. According to Kerschke et al. (2019), the features should be informative, interpretable, cheaply computable, generally applicable, and complementary. Following these principles, we define a total of 151 features, which are grouped into four categories that are presented in the following subsections.

Given k real values $x_1, \ldots, x_k$, we will make use of the corrected standard deviation, which is defined as

$$\sqrt{\frac{1}{k-1}\sum_{i=1}^{k}(x_i-\mu)^2},$$

where μ is the arithmetic mean of the values x_i, $i = 1, \ldots, k$.

6.4.1.1 Instance Size Related Features

With respect to the size of an instance, we consider a total of 28 features. This includes the following standard features:

- Number of jobs n
- Number of machines m
- Total number of operations $\sum_{i=1}^{n} q_i$

Additionally, we consider a few statistical measures:

- With respect to the number of operations of the jobs, we consider the minimum value, the maximum value, the arithmetic mean, the corrected standard deviation, and the nine deciles (as nine separate features).
- Similarly, with respect to the number of operations of the jobs divided by the corresponding arithmetic mean, we consider the minimum value, the maximum value, the corrected standard deviation, and the nine deciles.

6.4.1.2 Flexibility Related Features

The degree of flexibility within an instance is captured by 50 features. First, we consider the minimum value, the maximum value, the arithmetic mean, the corrected standard deviation, and

the nine deciles of the following attributes:

- Number of eligible machines for each operation
- Number of eligible operations of each machine, being defined as the number of operations that include the respective machine in their sets of eligible machines

Additionally, as in case of the instance size related features, we consider the minimum value, the maximum value, the corrected standard deviation, and the nine deciles of these attributes after having divided them by their corresponding arithmetic means.

6.4.1.3 Processing Time Related Features

The third category of features relates to the processing times of operations. This gives rise to 37 features. Specifically, we consider the minimum value, the maximum value, the arithmetic mean, the corrected standard deviation, and the nine deciles of the average processing time of operations over their eligible machines. As before, we additionally use this latter attribute divided by its corresponding arithmetic mean to define features that correspond to the associated minimum, maximum, corrected standard deviation, and the nine deciles. Finally, we compute the corrected standard deviation of the average processing times of the operations over their eligible machines. The corresponding minimum, maximum, corrected standard deviation, and nine deciles define the final 12 features in this category.

6.4.1.4 Priority Rule Related Features

The forth category of features is structurally different from the three prior categories, as it relies on heuristically computing feasible schedules in order to determine the feature values (see Mirshekarian and Šormaz, 2016, for a similar approach). To do so, we apply a priority rule based approach proposed by Kress et al. (2019) that follows an algorithmic idea of Giffler and Thompson (1960). In each iteration, this heuristic selects time instant and a machine, and appends an eligible operation that can start being processed at the respective point of time with respect to the precedence constraints at the end of the machine's operation sequence. Among all potential operations (referred to as candidate operations), exactly one operation is chosen based on a priority rule. We implemented four different priority rules, namely *shortest processing time*, *longest processing time*, *least work remaining*, and *most work remaining* (for details, see, e.g., Dorndorf and Pesch, 1995; Haupt, 1989).

For each of the four resulting schedules, we compute the following features:

- Arithmetic mean of job completion times
- Minimum, maximum, and corrected standard deviation of the job completion times divided by the corresponding arithmetic mean
- Arithmetic mean of machine loads, where the load of a machine is defined as the completion time of the operation that is scheduled last on the machine

- Minimum, maximum, and corrected standard deviation of the machine loads divided by the corresponding arithmetic mean

Moreover, during runtime of the heuristic, we count the total number of candidate operations and define the final count divided by the total number of operations as another feature. This results in a total of 36 features.

6.4.2 Problem Space Generation

The development and evaluation of algorithm selection approaches depends on a sufficiently large problem space, which we generated randomly. The underlying procedure is identical to the one for the RA set in Section 6.3.2. It is used to generate instances with 10 to 40 jobs and 2 to 8 machines. In order to take account of our results in Section 6.3.3, the remaining parameter ranges are such that the generation procedure is likely to result in a set that includes instances in accordance with the literature sets introduced in Section 6.3.2. They are illustrated in Table 6.5. For each reasonable combination of parameter values, we generated 25 test instances. Recall, that

Table 6.5: Parameters used for generating the problem space

n	10, 20, 30, 40
m	2, 3, 4, 5, 6, 8
q_i	$[1,3]$, $[1,4]$, $[2,4]$, $[2,6]$, $[2,8]$, $[4,10]$, $[6,12]$, $[10,15]$
$\lvert M_{i_j}\rvert$	$[1,2]$, $[2,3]$, $[4,5]$
p_{i_j}	$[10,100]$
p	0, 0.1

the parameter p is used for generating the processing times based on values p_{i_j}, $i \in \{1,\dots,n\}$, $i_j \in O_i$. For $p = 0$, the processing times of operations are identical on all of their eligible machines, which mimics the instances of the CH set and the HU set. In total, the problem space is composed of 22400 instances.

6.4.3 Preliminary Processing and Analysis

In line with the computational results presented in Section 6.3.3, we first analyze the competitiveness of the considered solvers on the problem space. As we focus on practically relevant industry settings, we fix the time limit to 30 seconds.

Both solvers were able to determine a feasible solution for all instances. For 3799 instances, they perform identically with respect to the above scoring system, so that a clear winner cannot be identified. We deleted these instances from the problem space. Hence, the remainder of this paper focusses on the remaining 18601 instances. In a next step, we randomly partitioned this set of instances into a *training set* (80 %, 14881 instances) used for the training of our algorithm selection approaches and a *validation set* (20 %, 3720 instances) used for the pre-evaluation of our approaches (see Section 6.4.4). Afterwards, for the final validation of our approaches, our algorithm selection approaches were tested on a separate *test set* composed of previously unseen instances (see Section 6.5). With respect to the distribution of scoring points achieved by the

solvers on the training set, we find that both solvers show a complementary performance with CPLEX receiving 7674 scoring points and OR-Tools receiving 7207 scoring points, resulting in a competitiveness ratio $c = 0.969$. In line with our results in Section 6.3.3, this allows to conclude that there exists a potential payoff when using algorithm selection approaches.

Figure 6.3 allows to take a closer look on the competitiveness of the solvers on the instances of the training set. For exemplary features of the feature set, the scatter plots illustrate the

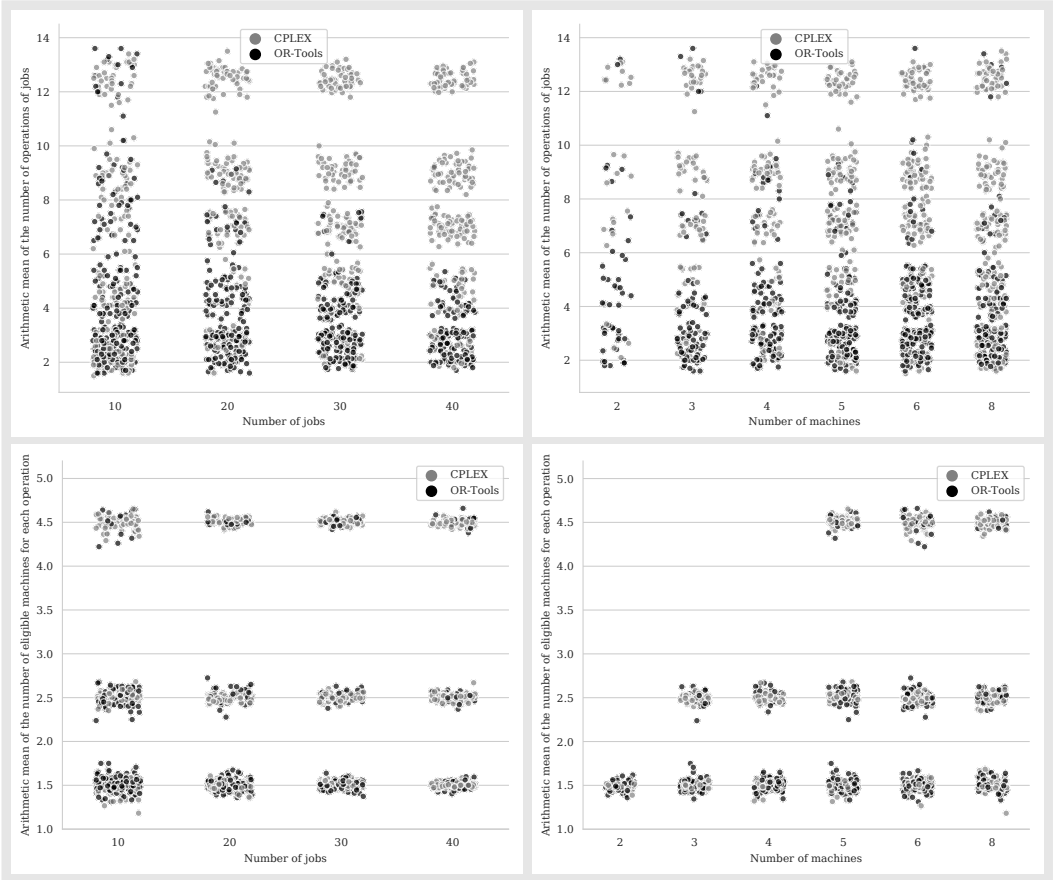


Figure 6.3: Scatter plots illustrating the performance competitiveness between CPLEX and OR-Tools on the training set

strengths and weaknesses of the solvers. Each data point represents an instance of the training set. The grayscale of the point highlights the algorithm that performs better on the respective instance. For the sake of clarity, the plots focus on a representative subset of training instances. As can be seen, OR-Tools tends to perform better than CPLEX when the arithmetic mean of the number of operations of jobs is rather small. This effect decreases for an increasing number of jobs or machines. Similarly, an increase of the arithmetic mean of the number of eligible machines for each operation is in favor of CPLEX, which seems to be independent of the number of jobs or machines.

Based on the plots in Figure 6.3, it is almost impossible for the human eye to identify instance clusters that are expressive enough to allow a substantiated choice of a solver when being confronted with a previously unseen instance. We therefore make use of machine learning

techniques to tackle this task. This allows to implicitly gain knowledge about a high quality mapping S.

6.4.4 Algorithm Selection Models

Since the desired output of our algorithm selection approaches, i.e., a prediction of the best solver, is categorial, our approaches fall into the category of classification methods. The instances of the problem space are labelled with binary labels according to our preliminary analysis of Section 6.4.3, i.e., each instance is associated with a value that indicates the solver that received the respective scoring point. We make use of two technologies: decision trees (Section 6.4.4.1), which are frequently used in the field of algorithm selection (see, e.g., Kerschke et al., 2018; Messelis and De Causmaecker, 2014), and deep neural networks (Section 6.4.4.2), which are less common in the OR literature (Kraus et al., 2020). In order to prevent overfitting and in order to specify the details, e.g., the network architecture, of our models, we pre-evaluate our models on the validation set. A model's performance is measured in terms of its *accuracy*, i.e., the percentage of correct classifications on the validation set.

6.4.4.1 Decision Trees

Decisions trees can be graphically illustrated by flowchart-like tree structures, in which each node (except the leaf nodes) represents a test on a feature, each edge represents a result of the test, and each leaf node represents a final prediction. There exists a broad variety of decision tree techniques and algorithms. For the sake of brevity, instead of summarizing well-known results, we refer the reader to Han et al. (2011).

For building our decision tree models, we used the Waikato Environment for Knowledge Analysis, WEKA, in version 3.8.3 (Frank et al., 2016; Hall et al., 2009). WEKA provides a broad collection of machine learning techniques. Amongst others, it offers a variety of different decision tree techniques and algorithms that can be utilized to construct an algorithm selector:

- Decision stump (DS): method for constructing a one-level decision tree using a single feature (Iba and Langley, 1992)
- Hoeffding tree (HT): incremental decision tree algorithm (Hulten et al., 2001)
- J48: generates a C4.5 decision tree (Quinlan, 1993)
- Logistic model tree (LMT): algorithm for building decision trees with logistic regression functions at the leaves (Landwehr et al., 2005; Sumner et al., 2005)
- Random tree (RT): constructs a tree that considers a set of randomly chosen features at each node (Frank et al., 2016)
- Random forest (RF): method to construct a variety of random trees (Breiman, 2001)

- REP tree (REPT): builds a tree using information gain and reduced-error pruning (Frank et al., 2016)

We made use of these algorithms with the software's default parameters on our training set. The results of the pre-evaluation of the resulting decision tree models on the validation set are presented in Table 6.6. For each of the models, the table presents the accuracy. Based on these

Table 6.6: Pre-evaluation of decision tree models on the validation set

Decision tree algorithm/model	DS	HT	J48	LMT	RF	RT	REPT
Accuracy [%]	76.24	82.07	83.41	85.19	86.34	81.21	84.22

results, we decided to select the best three variants, RF, LMT, and REPT, for evaluation on the test set in Section 6.5.

6.4.4.2 Deep Neural Networks

With recent advances in machine learning, deep learning has become an increasingly popular technique (Bengio et al., 2007; Hinton et al., 2006; Kraus et al., 2020; Schmidhuber, 2015). It refers to the use of deep neural networks that are inspired by neuroscience and date back to McCulloch and Pitts (1943). These networks can be represented by graphs. A respective graph is structured into multiple layers (see, e.g., Goodfellow et al., 2016; Han et al., 2011): an input layer, one or more hidden layers, and an output layer. Each layer consists of a collection of units, also referred to as neurons or nodes. These units receive input from other units, or - in case of the input layer - are fed with given data (in our case, instance feature values), compute some value as a function (typically a weighted sum) of the inputs and, based on this value, compute an output via a potentially nonlinear activation function, which is then passed on to connected units or serves as an output of the network. The number of units of the input layer therefore represents the number of elements of an input data tuple, while the units in the output layer are associated with the output values, e.g., the class labels. Usually, the layers are sequentially connected and, thus, build up a feed-forward structure. If, additionally, each unit of a layer is connected with each unit in the succeeding layer, the network is referred to as a *fully connected neural network*. After having defined a network topology, the weights of the network (as well as bias values) are computed and adjusted during a learning process on the training set, e.g., by making use of backpropagation, in order to minimize a loss function that measures the error between the predicted values and true observations.

In our case, as we solely consider the choice among two solvers, we restrict ourselves to network topologies with a single output unit that computes a real value in the range $(0, 1)$. This value represents the probability that a specific solver, say OR-Tools, is the best choice for the considered instance. A value close to zero then predicts CPLEX to perform better, while a value close to one predicts OR-Tools to be the best choice. We make use of the binary cross-entropy loss function. Denote by K some set of instances of the problem space and, for some $k \in K$, let $\tilde{y}_k \in (0, 1)$ be the output of the network and $y_k \in \{0, 1\}$ be the binary label, representing the

best solver for this instance. Then, this loss function is defined as

$$L = -\frac{1}{|K|} \sum_{k=1}^{K} y_k \log(\tilde{y}_k) + (1 - y_k) \log(1 - \tilde{y}_k). \tag{6.1}$$

For implementing our deep neural networks, we used the open source machine learning framework PyTorch in version 1.6.0 (PyTorch, 2020) using Python. In order to determine a final network architecture that does not tend to overfit and that has an acceptable accuracy, we implemented and trained several network architectures with varying parameter settings and pre-evaluated these networks on the validation set. Throughout this empirical evaluation, we found the following fully connected neural network, denoted by FCNN, to perform best (accuracy: 86.53%):

- Input layer: 151 units according to the number of features introduced in Section 6.4.1
- Three hidden layers with 256, 128, and 64 units with relu activation functions (see, e.g., Goodfellow et al., 2016; Han et al., 2011),
- Output layer: one unit using a sigmoid activation function (see, e.g., Goodfellow et al., 2016; Han et al., 2011)

The learning process was performed with Adam (Kingma and Ba, 2014), a popular adaptive learning rate optimization algorithm, with default parameters as suggested by the authors. The number of epochs, specifying the number of learning phases over the entire training set, was set to 10. The batch size, i.e., the number of instances used in a learning iteration, was set to 100. Moreover, for better generalization and robustness of the learning process, we normalized the input data to values between 0 and 1.

There exist advanced network architectures that target at specific data structures (see, e.g., Goodfellow et al., 2016; Kraus et al., 2020). *Convolutional neural networks*, for example, aim to exploit spatial dependencies in grid-like data, e.g., among neighboring pixels in images. These networks provide special filtering and pooling techniques to reduce the dimension and to identify certain patterns within the data. In our case, grid-like structures arise, when providing unprocessed instance data in form of the processing times of operations on their eligible machines to a neural network. Hence, for this case, we additionally trained a convolutional neural network, denoted by CNN. It is illustrated in Figure 6.4. The (normalized) processing times are represented by a 600×8 matrix (a maximum of 40 jobs with a maximum of 15 operations on a maximum of 8 machines) with 4800 elements. A zero within this matrix indicates that the corresponding operation cannot be processed on the respective machine (hence, it does not exist if all elements within a row are zero), while a one is associated to the largest possible processing time value (in our case 110). As before, the learning process was performed with Adam (learning rate 0.005, all other parameters with default values, 5 epochs, batch size 100). Based on an empirical evaluation on the validation set, we decided to make use of the following structure. In a convolutional layer, the information included in this matrix is aggregated using a kernel size of 15×8 (a maximum

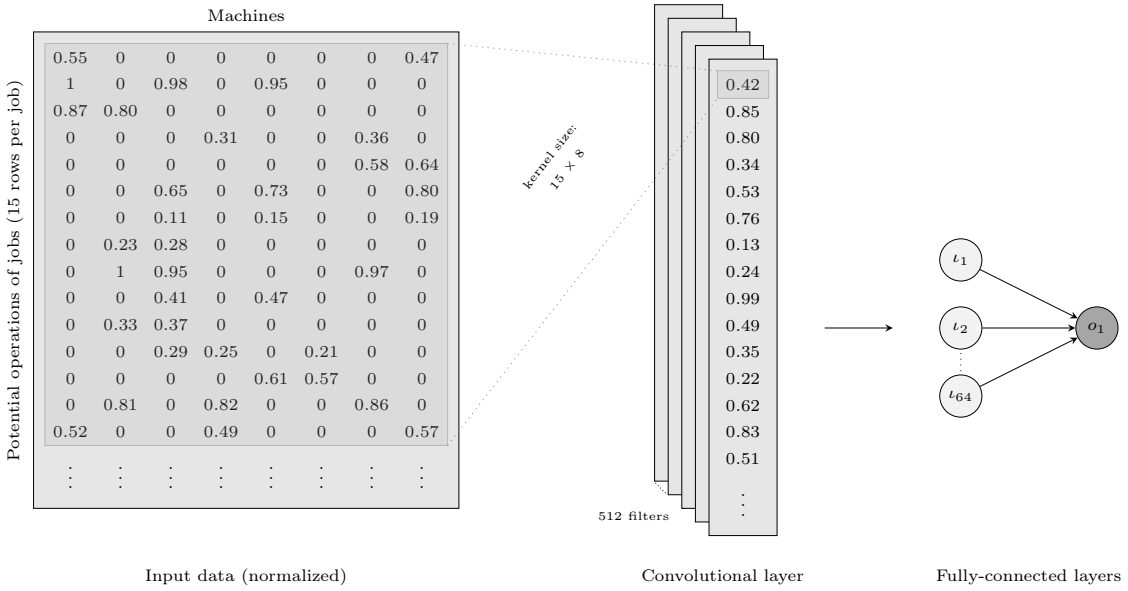


Figure 6.4: Illustration of the convolutional neural network

of 15 operations per job on a maximum of 8 machines) and a stride of 15, in order to aggregate the information for each job separately. We use a total of 512 filters. The resulting output is passed through fully connected layers consisting of a 64 units layer with relu activation functions and an output layer composed of one unit with a sigmoid activation function. On the validation set, CNN has an accuracy of 83.97%.

Finally, in order to be able to evaluate the effect of the convolutional layer, we trained an additional fully connected neural network (Adam setup as in case of CNN), denoted by FCNN', that uses the above processing time matrix as input data but does not feature a convolutional layer. As in case of FCNN, it has three hidden layers with 256, 128, and 64 units with relu activation functions. The input layer is adjusted according to the modified input data, while the output layer is identical to the one in FCNN. FCNN' has an accuracy of 82.58% on the validation set. At first glance, this indicates that the use of a convolutional layer pays off when using unprocessed instance data. A detailed analysis follows in Section 6.5.

6.4.4.3 Overview of Resulting Models

Before finally validating our algorithm selection models on a randomly constructed test set, we summarize all of our models in Table 6.7. It includes the computational times needed for building/training the models. The computation of the feature set of an instance takes only a few milliseconds, so that it is of minor interest.

6.5 Computational Results and Evaluation

For the final validation of our algorithm selection models, we generated a separate test set. The generation procedure and size of the instances is in line with Section 6.4.2. In order to generate

Table 6.7: Overview of algorithm selection models

Abbreviation	Method	Model	Input data	Build/ Training time [s]
LMT	Decision Tree	Logistic model tree	Features	100.22
RF	Decision Tree	RandomForest	Features	14.78
REPT	Decision Tree	REPTree	Features	4.17
FCNN	Neural network	Fully connected neural network	Features	134.13
FCNN'	Neural network	Fully connected neural network	Proc. time parameters	189.38
CNN	Neural network	Convolutional neural network	Proc. time parameters	228.47

previously unseen instances, we used new random seeds. For each reasonable combination of parameter values (see Table 6.5), we generated 5 test instances. For the resulting 4480 instances, we proceeded as described in Section 6.4.3, i.e., we called both solvers, CPLEX and OR-Tools, with a time limit of 30 (300) seconds and discarded 768 (1157) instances as there was no clear winner. Both solvers returned a feasible solution for all instances and time limits. We denote the remaining 3712 (3323) instances as *test set A* (*test set B*). Note that, even though our algorithm selection models were trained with a time limit of 30 seconds, test set B allows to analyze the performance of our approaches when increasing the time limit for the selected solver to 300 seconds.

As an additional performance indicator, we used *k-fold cross-validation* on the training set (see, e.g., Han et al., 2011) with a time limit of 30 seconds. To do so, we randomly partitioned the training set into $k = 10$ data folds $F_1, \ldots, F_{10}$ with $|F_i| - |F_j| \leq 1$ for all $i, j \in \{1, 2, \ldots, 10\}$. In ten iterations, $i = 1, \ldots, 10$, the models were trained with the union of all data folds but F_i. Each of the resulting iterations resulted in specific model variants, that were then validated with data fold F_i. As a performance indicator, we computed the average accuracy over all iterations.

Our overall approach is illustrated in Figure 6.5.

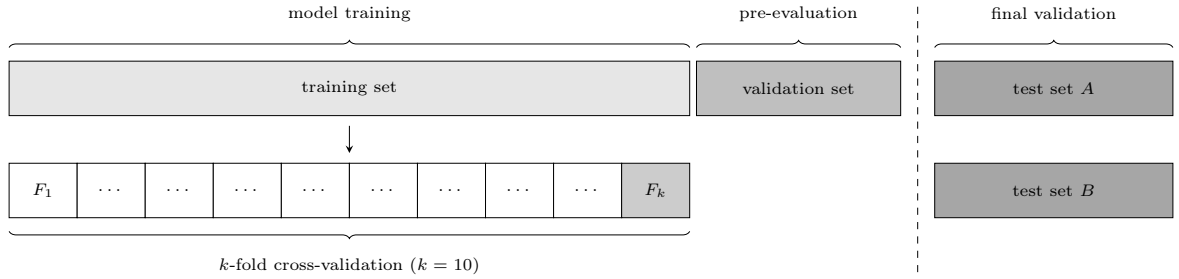


Figure 6.5: Illustration of training, validation, and test sets as well as k-fold cross-validation

6.5.1 Evaluation of the Algorithm Selection Models

The results of the final evaluation on test sets A and B are presented in Tables 6.8 and 6.9, respectively. The tables illustrate the performance of our algorithm selection models, the use of the individual solvers CPLEX and OR-Tools, as well as the use of the *virtual best solver* (VBS), i.e., the use of an oracle that reveals the best solver for each instance. They present information on the percentage of instances for which a provably optimal solution was found with the selected

Table 6.8: Final validation on test set A

	CPLEX	OR-Tools	LMT	RF	REPT	FCNN	FCNN'	CNN	VBS
Accuracy [%]	-	-	85.91	86.77	84.29	86.02	80.87	83.19	100
opt. [%]	22.58	27.13	27.05	27.1	27.02	27.07	26.94	26.97	27.16
Q_{avg}	2.94	12.53	2.84	2.82	2.84	2.83	2.84	2.87	2.76
t_{avg} $[s]$	23.83	22.42	22.41	22.4	22.41	22.41	22.44	22.42	22.38
score	1926	1786	3189	3221	3129	3193	3002	3088	3712

Table 6.9: Final validation on test set B

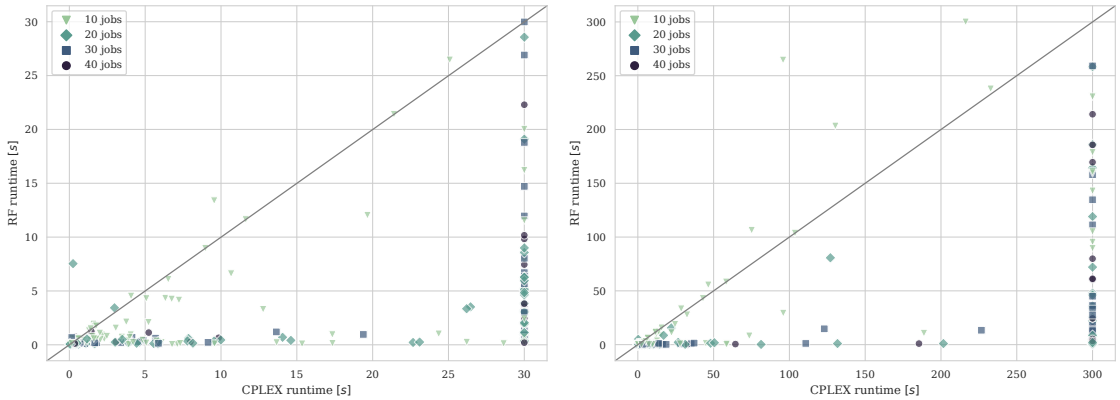
	CPLEX	OR-Tools	LMT	RF	REPT	FCNN	FCNN'	CNN	VBS
Accuracy [%]	-	-	74.99	75.23	74.9	75.77	69.49	72.62	100
opt. [%]	27.87	33.85	33.85	33.85	33.79	33.82	33.58	33.79	33.92
Q_{avg}	2.94	4.61	2.86	2.86	2.85	2.85	2.88	2.87	2.74
t_{avg} $[s]$	219.51	203.29	203.32	203.3	203.43	203.25	203.85	203.39	202.98
score	1349	1974	2492	2500	2489	2518	2309	2413	3323

solver within the given time limit (row "opt."), the average quality ratios (row "Q_{avg}"), the average runtimes (row "t_{avg}"), and the sum of scoring points (row "score") over all instances. Moreover, they include information on the accuracy of the selectors (row "Accuracy"). Again, the time needed to compute the feature values of an instance is negligible.

We observe that all of our algorithm selection models perform similar or better than the individual solvers with respect to all performance indicators for both test sets. As to be expected, the positive effect of using our selectors is more significant for test set A. The accuracy values are in similar ranges for all models but FCNN', that suffers from the absence of the convolutional layer when compared with CNN. RF and FCNN tend to provide the best overall performance on the test sets A and B, respectively. Moreover, the selectors that make use of the pre-calculated features outperform the other selectors. However, this effect is rather small when a convolutional layer is used (CNN). Hence, in practice, planners can abstain from the selection and computation of these features at little cost.

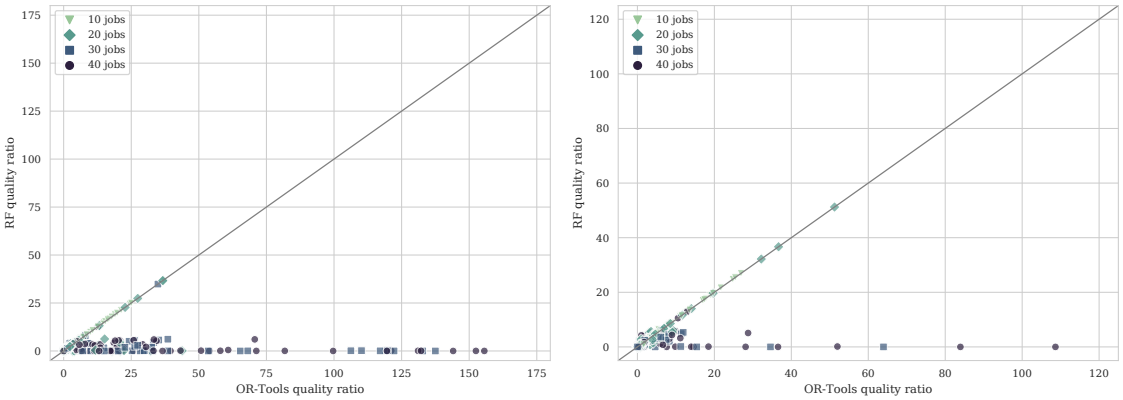
As already observed in Sections 6.3 and 6.4, CPLEX and OR-Tools show a performance complementary. For the test set, CPLEX receives 1926 (1349) and OR-Tools receives 1786 (1974) scoring points for test set A (B). Again, CPLEX tends to provide better quality ratios while OR-Tools wins when considering the ability to quickly determine optimal solutions. As exemplarily illustrated for RF in Figures 6.6 and 6.7, these effects are leveraged by our algorithm selection approaches. The scatter plots presented in these figures are constructed as the ones in Section 6.3.3. Figure 6.6 illustrates the runtimes on the instances for which CPLEX or the solver selected by RF was able to determine an optimal solution. Figure 6.7 presents the quality ratios of the solutions returned by OR-Tools and the solver selected by RF. For the sake of clarity, the plots focus on a representative subset of instances.

We observe that RF outperforms CPLEX with respect to the runtimes needed to compute optimal solutions. Additionally, the use of RF results in better quality ratios than the individual use of OR-Tools for the majority of test instances. Hence, RF incorporates the performance



(a) Test set A (b) Test set B

Figure 6.6: Comparison of CPLEX and RF for instances for which at least one approach resulted in an optimal solution



(a) Test set A (b) Test set B

Figure 6.7: Comparison of quality ratios when using OR-Tools and RF

benefits of both solvers. This effect can be observed for both test sets, while it is larger on test set A.

The average accuracy values resulting from the 10-fold cross-validation on the training set are presented in Table 6.10. We find that the average accuracy values are in similar ranges for

Table 6.10: Results for 10-fold cross-validation on the training set

	LMT	RF	REPT	FCNN	FCNN'	CNN
Avg. accuracy [%]	86.32	87.71	85.13	86.41	83.43	83.58

all models, with RF being the overall best choice.

6.5.2 Feature Subset Selection

In Section 6.4.1, we introduced 151 features that were used to train the models. In order to detect a subset of the most relevant features and allow shorter training times, we applied a correlation-based feature subset selection procedure (see Hall, 1998) using a best first search strategy with a bi-directional search provided by WEKA. The procedure was performed on the training set. It selected four instance size related features, five flexibility related features, one processing time related feature, and five priority rule related features. These 15 features were then used for training the three selected decision tree variants and the fully connected neural network as illustrated in Sections 6.4.4.1 and 6.4.4.2. The resulting models are referred to with an additional index $\circ$: LMT°, RF°, REPT° and FCNN°. The performance of these algorithm selection models on the test sets is presented in Table 6.11. We observe that the use of the reduced

Table 6.11: Performance of the selectors when trained with the reduced feature set

	Test set A				Test set B			
	LMT°	RF°	REPT°	FCNN°	LMT°	RF°	REPT°	FCNN°
Accuracy [%]	85.48	85.7	84.78	84.7	75.08	75.35	75.32	76.02
opt. [%]	27.07	27.07	27.05	27.07	33.82	33.89	33.82	33.82
Q_{avg}	2.83	2.83	2.84	2.83	2.86	2.86	2.86	2.85
t_{avg} [s]	22.4	22.4	22.41	22.41	203.29	203.22	203.36	203.36
score	3173	3181	3147	3144	2495	2504	2503	2526

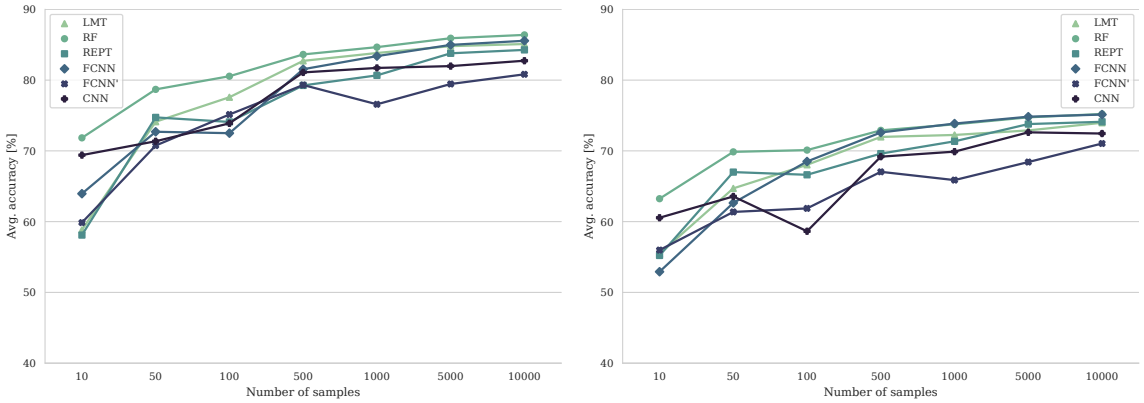
feature set does not significantly impair the performance of the algorithm selection models. In case of test set B, there are even slight improvements.

6.5.3 Reducing the Size of the Training Set

In this section, we analyze the impact of a reduced training set size on the performance of our selectors. This is of major importance for decision makers in practice, as it specifies the number of samples needed in order to obtain adequate results.

We randomly selected subsets of instances of the training set with 10, 50, 100, 500, 1000, 5000, and 10000 instances, respectively. These reduced training sets were then used to train our

models. This was done 10 times for each combination of training set size and model. The average performance of the resulting selectors on test sets A and B is presented in Figure 6.8. The plots



(a) Test set A (b) Test set B

Figure 6.8: Performance of algorithm selection approaches when using sparse training data

show the average accuracy of the different selectors over the size of the training set that was used to train the models. For the sake of readability, the sample size values are arranged equidistantly on the abscissa.

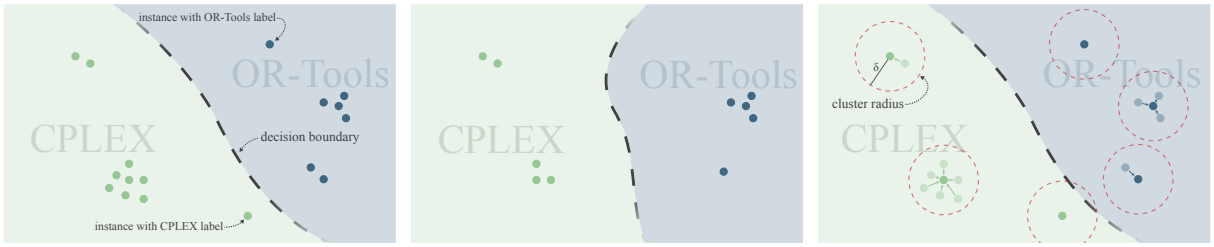
As can be seen, while an increasing sample size has a positive effect on the average performance of the selectors, most selectors perform quite well even for very small sample sizes. The positive effect of increasing the sample size tends to diminish when exceeding an amount of 1000 instances. Overall, RF tends to perform best across the varying samples sizes. Hence, in practice, varying characteristics of representative instances, e.g., due to the availability of new machines or the production of new products, can be successfully incorporated into the selectors based on only a few data samples.

6.5.4 Reducing the Training Set to Relevant Instances

The previous section has demonstrated that it is possible to train the selectors with relatively few instances. However, in some cases, larger training sets are available or it may simply be beneficial to take account of larger training sets. In this section, we therefore analyze the question of whether these datasets can be reduced more effectively than by applying pure random subsampling (see Figure 6.9). We restrict or attention to FCNN and CNN. When subsampling randomly, chances are high that isolated samples are deleted even though they may be relevant for the training process as they have a major impact on the final model and its prediction accuracy. Reducing the training set in a strategic way, on the other hand, may allow to effectively train and evaluate more models.

In order to reduce the size of the training set in a more strategic manner, we developed a simple procedure that is based on dividing the training set into two subsets based on the binary instance labels (indicating the best solvers for the instances, see Figure 6.9a) and then detecting

clusters of similar instances within these subsets. To do so, each instance is associated to a point



(a) Instance subsets based on labels (b) Random subsampling (c) Instance clusters

Figure 6.9: Exemplary illustration the construction of a reduced training set

in $\mathbb{R}^5$ that corresponds to the normalized values of the following five features (see Section 6.4.1): number of jobs, number of machines, arithmetic mean of the number of operations of the jobs, arithmetic mean of the number of eligible machines for each operation, corrected standard deviation of the corrected standard deviation of the average processing times of the operations. These features reflect the attributes used for generating the instances (see Section 6.4.2). In order to measure the degree of similarity of two instances, we compute the Euclidian distance between the corresponding points. Two instances are defined to be similar, if this distance is not larger than a given threshold δ. Based on random orderings of the instances of the two subsets of the training set (see Figure 6.9a), we then construct clusters of similar instances as follows. For each ordering, we initialize a first cluster with the first instance of the ordering and add all similar instances of the ordering, i.e., all instances of the corresponding instance subset that lie within a hypersphere of radius δ around this instance (see Figure 6.9c). All instances that have been clustered are then deleted from the respective ordering and the process repeats until all instances are associated to a cluster. We then randomly select exactly one instance of each cluster to become an element of the reduced training set. It is associated to a weight that is set to the number of instances that are included in the cluster divided by the total number of instances in the original training set. As a result, larger values of δ result in less samples in the reduced training set. This process can be interpreted as randomly merging similar instances to a single sample (see Figure 6.9c). The weights compensate for this reduction. Hence, isolated samples are not deleted from the training set as it is potentially done in case of random subsampling (see Figure 6.9b). Instead, isolated samples are taken account of with a small weight. The loss function (6.1) is adapted accordingly:

$$L' = -\frac{1}{|K|}\sum_{k=1}^{|K|}(1+\alpha\lambda_k)[y_k \log(\tilde{y}_k) + (1-y_k)\log(1-\tilde{y}_k)] \tag{6.2}$$

Here, λ_k represents the weight of instance k in the reduced training set and α is an additional parameter to regularize the impact of this balancing factor.

We analyzed various parameter settings, $\delta \in 0.5, 0.4, 0.2$, for constructing the instance clusters. This resulted in a total of 62 ($\delta = 0.5$), 125 ($\delta = 0.4$) and 735 ($\delta = 0.2$) samples. As

the selection of the specific instances is random, we generated ten different reduced training sets for each threshold value. Then, we trained the models FCNN and CNN on the resulting training sets and determined the accuracy with respect to test set A. The results are presented in Table 6.12. It presents information about the average accuracy and the corrected standard

Table 6.12: Performance using a weighting sampling strategy

	FCNN		CNN	
	Avg. accuracy [%]	Std. deviation	Avg. accuracy [%]	Std. deviation
62 samples (weighted, $\delta = 0.5$)				
$\alpha = 0$	55.77	5.81	54.91	5.35
$\alpha = 100$	75.22	4.16	75.71	5.24
$\alpha = 200$	74.18	5.62	72.8	7.37
$\alpha = 300$	76.38	4.55	75.44	4.23
62 samples (random)	72.12	8.36	72.72	7.4
125 samples (weighted, $\delta = 0.4$)				
$\alpha = 0$	72.59	6.46	67.18	9.42
$\alpha = 100$	78.67	1.78	71.15	7.87
$\alpha = 200$	79.15	1.38	78.11	3.54
$\alpha = 300$	79.74	1.17	79.45	1.76
125 samples (random)	78.73	1.93	79.54	2.51
735 samples (weighted, $\delta = 0.2$)				
$\alpha = 0$	81.41	1.13	79.92	0.81
$\alpha = 100$	82.08	0.72	80.38	0.52
$\alpha = 200$	82.14	1.3	79.75	0.86
$\alpha = 300$	82.36	0.48	80.19	1.51
735 samples (random)	82.14	1.4	81.35	0.7

deviation of the accuracy over the ten runs. For comparison, we include results based on random instance subsets of the same size (again, for 10 runs). With respect to the parameter α, we used values $\alpha \in 0, 100, 200, 300$. For $\alpha = 0$, the balancing factors of the corresponding samples are not considered (see equation (6.2)).

We observe that our sampling strategy tends to outperform random subsampling. This effect is more pronounced when the number of samples is small, because random subsampling is less likely to delete relevant isolated samples for larger sizes of the reduced training sets. When ignoring the weighting parameters (case $\alpha = 0$), our sampling strategy fails for larger values of δ, as it does not take account of the actual distribution of the instances. Furthermore, in the considered range, our sampling strategy benefits from increasing values of α.

6.6 Summary

In this paper, we evaluated the performance of different CP solvers on the FJSP for test instances taken from the literature as well as randomly generated test instances. In a computational study, we introduced a scoring system that captures different performance measures (solution quality/optimality, runtime). When solely considering solution quality, the CP solver provided by CPLEX is the clear winner and outperforms the other solvers. However, when additionally considering the ability to quickly compute optimal solutions (as specified in our scoring system), the

overall picture changes, and the CP solver provided by OR-Tools, an open source software suite developed by Google, becomes competitive. We leveraged this performance complementarity by developing algorithm selection approaches that aim to select the best of the two solvers for a given instance. These selectors make use of two machine learning techniques, decision trees and deep neural networks. We trained and validated the resulting models on a large set of random test instances. In an extensive study, we demonstrated that our models outperform the use of a single solver when restricting the runtimes of the solvers to relatively small values that allow their usage in rolling horizon based planning approaches in practice. This allows to conclude that our approaches should be considered as a relevant tool by decision makers in practice. While all of our selectors perform similarly well, an advantage of the use of deep neural networks is the good performance on unprocessed instance data that does not result from precomputing specific features. We additionally showed that varying instance characteristics that, for example, result from the availability of new machines or the production of new products, can be successfully incorporated into the selectors based on only a few data samples. Finally, we have developed a method to reduce the size of the training set to a subset of the most relevant instances. In practice, this method allows to effectively train and evaluate more models.

Bibliography

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401.

Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156.

Behnke, D. and Geiger, M. J. (2012). Test instances for the flexible job shop scheduling problem with work centers. Technical Report RR-12-01-01, Helmut Schmidt University Hamburg.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J. C., and Hoffman, T., editors, *Proceeding of the 19th Conference on Neural Information Processing Systems*, pages 153–160, Cambridge. MIT Press.

Bengio, Y., Lodi, A., and Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421.

Błażewicz, J., Domschke, W., and Pesch, E. (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(1):1–33.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Sterna, M., and Weglarz, J. (2019). *Handbook on Scheduling: From Theory to Practice*. Springer, Berlin, 2nd edition.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Brucker, P. and Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724.

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176.

Chambers, J. B. and Barnes, J. W. (1996). Flexible job shop scheduling by tabu search. Technical Report ORP96-09, Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin.

Chaudhry, I. A. and Khan, A. A. (2016). A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3):551–591.

Chu, G., Stuckey, P. J., Schutt, A., Ehlers, T., Gange, G., and Francis, K. (2019). Chuffed solver documentation. `https://github.com/chuffed/chuffed`. Last accessed 2020-11-18.

Da Col, G. and Teppan, E. C. (2019a). Google vs IBM: a constraint solving challenge on the job-shop scheduling problem. arXiv preprint arXiv:1909.08247.

Da Col, G. and Teppan, E. C. (2019b). Industrial size job shop scheduling tackled by present day CP solvers. In Schiex, T. and de Givry, S., editors, *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming*, pages 144–160, Cham. Springer.

Dauzère-Pérès, S. and Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306.

Dorndorf, U. and Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1):25–40.

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2000). Constraint propagation techniques for disjunctive scheduling problems. *Artificial Intelligence*, 122:189–240.

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2002). Constraint propagation and problem decomposition: a preprocessing procedure for the job shop problem. *Annals of Operations Research*, 115(1–4):125–145.

Fisher, H. and Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In Muth, J. F. and Thompson, G. L., editors, *Industrial Scheduling*, pages 225–251, Englewood Cliffs. Prentice-Hall.

Frank, E., Hall, M. A., and Witten, I. H. (2016). *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Morgan Kaufmann, Cambridge, 4th edition.

Giffler, B. and Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Google (2019a). Google OR-Tools. `https://developers.google.com/optimization/`. Last accessed 2020-11-18.

Google (2019b). Google OR-Tools: Scheduling overview. `https://developers.google.com/optimization/scheduling`. Last accessed 2020-11-18.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18.

Hall, M. A. (1998). *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand.

Ham, A. M. and Cakici, E. (2016). Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches. *Computers & Industrial Engineering*, 102:160–165.

Han, J., Kamber, M., and Pei, J. (2011). *Data Mining: Concepts and Techniques*. Elsevier, Waltham, 3rd edition.

Hart, E., Ross, P., and Nelson, J. (1998). Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computation*, 6(1):61–80.

Haupt, R. (1989). A survey of priority rule-based scheduling. *OR Spektrum*, 11(1):3–16.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, New York. ACM.

Hurink, J., Jurisch, B., and Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, 15(4):205–215.

Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2014). AClib: a benchmark library for algorithm configuration. In Pardalos, P. M., Resende, M. G., Vogiatzis, C., and Walteros, J. L., editors, *Proceedings of the 8th International Conference on Learning and Intelligent Optimization*, pages 36–40, Cham. Springer.

Iba, W. and Langley, P. (1992). Induction of one-level decision trees. In Sleeman, D. and Edwards, P., editors, *Proceedings of the 9th International Conference on Machine Learning*, pages 233–240. Morgan Kaufmann, San Francisco.

IBM (2019a). IBM ILOG CPLEX optimization studio 12.9.0: Online documentation. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html`. Last accessed 2020-11-18.

IBM (2019b). IBM ILOG CPLEX optimization studio 12.9.0: Scheduling examples. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.ide.help/OPL_Studio/usroplexamples/topics/opl_cp_examples_scheduling.html`. Last accessed 2020-11-18.

Kerschke, P., Hoos, H. H., Neumann, F., and Trautmann, H. (2019). Automated algorithm selection: survey and perspectives. *Evolutionary Computation*, 27(1):3–45.

Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., and Trautmann, H. (2018). Leveraging TSP solver complementarity through machine learning. *Evolutionary Computation*, 26(4):597–620.

Kingma, D. P. and Ba, J. L. (2014). Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kraus, M., Feuerriegel, S., and Oztekin, A. (2020). Deep learning in business analytics and operations research: models, applications and managerial implications. *European Journal of Operational Research*, 281(3):628–641.

Kress, D. and Müller, D. (2019). Mathematical models for a flexible job shop scheduling problem with machine operator constraints. *IFAC-PapersOnLine*, 52(13):94–99.

Kress, D., Müller, D., and Nossack, J. (2019). A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, 41(1):179–217.

Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250.

Landwehr, N., Hall, M., and Frank, E. (2005). Logistic model trees. *Machine Learning*, 59(1-2):161–205.

Lawrence, S. (1984). Resouce constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University.

Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140.

Lunardi, W. T., Birgin, E. G., Laborie, P., Ronconi, D. P., and Voos, H. (2020). Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. *Computers & Operations Research*, 123:105020.

Mastrolilli, M. (2020). Flexible job shop problem. http://people.idsia.ch/~monaldo/fjsp.html. Last accessed 2020-12-08.

Mastrolilli, M. and Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

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

MiniZinc (2013). MiniZinc challenge 2013. https://www.minizinc.org/challenge2013/challenge.html. Last accessed 2020-11-18.

MiniZinc (2020). MiniZinc challenge 2020. https://www.minizinc.org/challenge2020/challenge.html. Last accessed 2020-11-18.

Mirshekarian, S. and Šormaz, D. N. (2016). Correlation of job-shop scheduling problem features with scheduling efficiency. *Expert Systems with Applications*, 62:131–147.

Müller, D., Müller, M. G., Kress, D., and Pesch, E. (2021). Test instances for the FJSP. https://doi.org/10.6084/m9.figshare.13522625.v1. Last accessed 2021-01-14.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). MiniZinc: towards a standard CP modelling language. In "Bessière, C., editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, pages 529–543, Berlin. Springer.

Pesch, E. and Tetzlaff, U. A. W. (1996). Constraint propagation based scheduling of job shops. *INFORMS Journal on Computing*, 8(2):144–157.

Prud'homme, C., Fages, J.-G., and Lorca, X. (2016). *Choco documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. https://choco-solver.org. Last accessed 2020-11-18.

Puget, J. F. (2013). Solving flexible job shop scheduling problems. https://www.ibm.com/developerworks/community/blogs/jfp/entry/solving_flexible_job_shop_scheduling_problems?lang=en. Last accessed 2019-07-26.

PyTorch (2020). PyTorch online documentation. https://pytorch.org/docs/stable/index.html. Last accessed 2020-11-18.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, 15:65–118.

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Networks*, 61:85–117.

Schulte, C., Tack, G., and Lagerkvist, M. Z. (2019). Modeling and programming with Gecode. `https://www.gecode.org/doc-latest/MPG.pdf`. Last accessed 2020-11-18.

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):1–25.

Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., and Fischer, J. (2014). The minizinc challenge 2008–2013. *AI Magazine*, 35(2):55–60.

Sumner, M., Frank, E., and Hall, M. (2005). Speeding up logistic model tree induction. In Jorge, A. M., Torgo, L., Brazdil, P., Camacho, R., and Gama, J., editors, *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 675–683, Berlin. Springer.

Vázquez-Rodríguez, J. A. and Petrovic, S. (2010). A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, 16(6):771–793.

Wari, E. and Zhu, W. (2019). A constraint programming model for food processing industry: a case for an ice cream processing facility. *International Journal of Production Research*, 57(21):6648–6664.

Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. (2012). Evaluating component solver contributions to portfolio-based algorithm selectors. In Cimatti, A. and Sebastiani, R., editors, *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, pages 228–241, Berlin. Springer.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606.

Chapter 7

Summary and Outlook

This thesis has investigated various flexible job shop scheduling problems incorporating workforce restrictions that arise in manufacturing systems. We have proposed different exact and heuristic solution approaches for scheduling the respective problems and analyzed the corresponding computational results. Moreover, we have introduced an algorithm selection approach for the classical flexible job shop scheduling problem. In the following, we summarize the main findings again with focusing on the research questions and discuss several interesting directions for future research.

The first research question was:

> *Which FJSP settings incorporating workforce constraints are relevant from a practical point of view and have attracted less attention in the literature?*

To this end, we have presented two flexible job shop problem variants, which have been motivated by real-world scheduling problems. In Chapter 2, we have analyzed a flexible job shop scheduling problem with sequence-dependent setup times that incorporates machine operator qualifications. The different machine operator qualifications have been taken into account by making use of machine- and operator-dependent processing times. Based on the conducted literature review, this problem variant has attracted little attention. We have analyzed two objective functions, minimizing the makespan and minimizing the total tardiness. Since the explicit incorporation of setup operators in flexible job shop settings is rarely addressed in the literature, Chapter 5 has been concerned with a flexible job shop setting with sequence-dependent setup times under setup operator constraints. This research has been motivated by a real-world scheduling problem encountered at a developer and manufacturer of semiconductor-based system solutions. We have analyzed the objective of minimizing the total weighted tardiness. Future research may focus on further problem variants regarding the flexible shop job problem. For instance, the simultaneous consideration of machine operators and setup operators, or the incorporation of timetabling restrictions would be interesting to analyze. Furthermore, the consideration of other objectives that are relevant in practice would be valuable.

Along with the first research question, the next research question was:

What are novel exact as well as heuristic solution approaches for scheduling flexible job shops with workforce restrictions?

With regard to the second research question, this thesis has proposed several novel exact as well as heuristic solution approaches for scheduling flexible job shops incorporating workforce restrictions. In order to evaluate the performance of our proposed solution approaches, we have conducted extensive computational tests using randomly generated instances as well as real-world problem instances that mimic settings at the considered manufacturing companies. In Chapter 2, we have presented exact and heuristic decomposition based solution approaches for a flexible job shop problem with sequence-dependent setup-times that incorporates machine operator qualifications. These approaches divide the problem into a vehicle routing problem with precedence constraints and a machine operator assignment problem. Our exact decomposition based solution approach has outperformed an integrated approach. Moreover, our heuristic solution approaches have shown to provide high-quality solutions within reasonable time and to be suitable for daily usage at the corresponding manufacturing company.

In the case of a flexible job shop scheduling problem that solely considers machine operator restrictions and aims to minimize the makespan, we have also proposed exact and heuristic solution approaches. With regard to the exact approaches, we have presented two mathematical models, a mixed-integer programming model and a constraint programming model, that have been evaluated by using the standard solvers provided by IBM ILOG CPLEX. We have shown, that the constraint programming solver is able to find quickly high-quality solutions. Moreover, it clearly outperforms the mixed-integer programming solver and interestingly also tends to outperform a state-of-the-art metaheuristic approach (see Chapter 3). Against this background, we have presented filter-and-fan based heuristic solution approaches in Chapter 4, which make use of the main ideas of our decomposition based approach by dividing the problem into a machine allocation and sequencing component as well as machine operator assignment component. We have shown that our heuristic approaches tend to outperform existing metaheuristic approaches from the literature and that they are competitive when compared with a standard constraint programming solver.

With respect to the semiconductor final-test scheduling problem incorporating setup operator restrictions, we have presented in Chapter 5 a mixed-integer programming model and a tabu search heuristic framework which applies the main ideas of our proposed decomposition based approach. We have shown that our proposed tabu search heuristic approaches clearly outperform a mixed-integer programming model using IBM ILOG CPLEX as solver and are able to quickly compute high-quality solutions to ensure real-world applicability.

Overall, we have shown that our different decomposition based heuristic solution approaches are promising candidates that have proven to be suitable in practice. As far as the trade-off between runtime and solution quality is concerned, an interesting direction for future research is to develop adaptive methods that adjust the hyperparameters of our heuristic solution approaches

depending on a given instance. Algorithm configuration approaches that make use of machine learning techniques offer promising research perspectives.

The third research question was:

> *How can an algorithm selection approach be designed that leverage the performance complementary of a given set of algorithms for solving the FJSP?*

Chapter 6 is concerned with an algorithm selection approach for the flexible job shop scheduling problem minimizing the makespan. Since standard constraint programming solvers have been applied to various scheduling problems and have shown to perform remarkably well, we have, as a first step, analyzed the performance differences among state-of-the-art commercial and non-commercial constraint programming solvers on problem instances available in the literature and randomly generated problem instances. Based on these results, we have leveraged the performance complementarity to propose an algorithm selection approach which predicts the best solver for a given problem instance based on features describing this given instance. In order to implement this approach, we have proposed several algorithm selection models by making use of different machine learning techniques. We have shown that our implemented algorithm selection models provide a better performance than the usage of a single solver. There remain interesting questions to be answered by future research. The level of generalization of machine learning methods depends on the available training set. Therefore, in practice, detecting new relevant data samples is crucial in order to adapt the algorithm selectors. Future research may focus on online learning techniques that react dynamically to changes in the problem characteristics. This can lead to a better performance and robustness. Furthermore, the set of features describing an instance is a crucial part for the performance of an algorithm selection approach. One of the advantages of deep neural networks is the capability to learn features based on the unprocessed instance input data. Based on their ability, it would be interesting to apply this technique to other problems even beyond the scheduling domain.

Bibliography

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401.

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

Allahverdi, A., Gupta, J. N. D., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239.

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

Allahverdi, A. and Soroush, H. M. (2008). The significance of reducing setup times/setup costs. *European Journal of Operational Research*, 187(3):978–984.

Andrade-Pineda, J. L., Canca, D., Gonzalez-R, P. L., and Calle, M. (2020). Scheduling a dual-resource flexible job shop with makespan and due date-related criteria. *Annals of Operations Research*, 291(1):5–35.

Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156.

Bagheri, A. and Zandieh, M. (2011). Bi-criteria flexible job-shop scheduling with sequence-dependent setup times—variable neighborhood search approach. *Journal of Manufacturing Systems*, 30(1):8–15.

Balas, E., Simonetti, N., and Vazacopoulos, A. (2008). Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, 11(4):253–262.

Bard, J. F., Gao, Z., Chacon, R., and Stuber, J. (2012). Real-time decision support for assembly and test operations in semiconductor manufacturing. *IIE Transactions*, 44(12):1083–1099.

Beach, R., Muhlemann, A. P., Price, D. H. R., Paterson, A., and Sharp, J. A. (2000). A review of manufacturing flexibility. *European Journal of Operational Research*, 122(1):41–57.

Behnamian, J. (2014). Scheduling and worker assignment problems on hybrid flowshop with cost-related objective function. *The International Journal of Advanced Manufacturing Technology*, 74(1-4):267–283.

Behnke, D. and Geiger, M. J. (2012). Test instances for the flexible job shop scheduling problem with work centers. Technical Report RR-12-01-01, Helmut Schmidt University Hamburg.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J. C., and Hoffman, T., editors, *Proceeding of the 19th Conference on Neural Information Processing Systems*, pages 153–160, Cambridge. MIT Press.

Bengio, Y., Lodi, A., and Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421.

Bigras, L.-P., Gamache, M., and Savard, G. (2008). The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, 5(4):685–699.

Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., and Węglarz, J. (2007). *Handbook on Scheduling: From Theory to Applications*. Springer, Berlin.

Błażewicz, J., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24.

Błażewicz, J., Domschke, W., and Pesch, E. (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(1):1–33.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Sterna, M., and Weglarz, J. (2019). *Handbook on Scheduling: From Theory to Practice*. Springer, Berlin, 2nd edition.

Boost (2018). The Boost Graph Library (BGL). `https://www.boost.org/doc/libs/1_67_0/libs/graph/doc/index.html`. Last accessed 2018-06-22.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Browne, J., Dubois, D., Rathmill, K., Sethi, S. P., and Stecke, K. E. (1984). Classification of flexible manufacturing systems. *The FMS Magazine*, 2(2):114–117.

Brucker, P. and Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724.

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176.

Chambers, J. B. and Barnes, J. W. (1996). Flexible job shop scheduling by tabu search. Technical Report ORP96-09, Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin.

Chaudhry, I. A. and Khan, A. A. (2016). A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3):551–591.

Chen, D., Luh, P. B., Thakur, L. S., and Moreno Jr, J. (2003). Optimization-based manufacturing scheduling with multiple resources, setup requirements, and transfer lots. *IIE Transactions*, 35(10):973–985.

Chen, T.-R., Chang, T.-S., Chen, C.-W., and Kao, J. (1995). Scheduling for IC sort and test with preemptiveness via Lagrangian relaxation. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(8):1249–1256.

Chen, T.-R. and Hsia, T. C. (1997). Scheduling for IC sort and test facilities with precedence constraints via Lagrangian relaxation. *Journal of Manufacturing Systems*, 16(2):117–128.

Chu, G., Stuckey, P. J., Schutt, A., Ehlers, T., Gange, G., and Francis, K. (2019). Chuffed solver documentation. `https://github.com/chuffed/chuffed`. Last accessed 2020-11-18.

Cunha, M. M., Viegas, J. L., Martins, M. S. E., Coito, T., Costigliola, A., Figueiredo, J., Sousa, J. M. C., and Vieira, S. M. (2019). Dual resource constrained scheduling for quality control laboratories. *IFAC-PapersOnLine*, 52(13):1421–1426.

Da Col, G. and Teppan, E. C. (2019a). Google vs IBM: a constraint solving challenge on the job-shop scheduling problem. arXiv preprint arXiv:1909.08247.

Da Col, G. and Teppan, E. C. (2019b). Industrial size job shop scheduling tackled by present day CP solvers. In Schiex, T. and de Givry, S., editors, *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming*, pages 144–160, Cham. Springer.

Dauzère-Pérès, S. and Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306.

De Bruecker, P., Van den Bergh, J., Beliën, J., and Demeulemeester, E. (2015). Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243(1):1–16.

Defersha, F. M. and Chen, M. (2010). A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. *The International Journal of Advanced Manufacturing Technology*, 49(1-4):263–279.

Della Croce, F., Grosso, A., and Salassa, F. (2014). A matheuristic approach for the two-machine total completion time flow shop problem. *Annals of Operations Research*, 213(1):67–78.

Deng, Y., Bard, J. F., Chacon, G. R., and Stuber, J. (2010). Scheduling back-end operations in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 23(2):210–220.

Dorndorf, U., Jaehn, F., and Pesch, E. (2008). Modelling robust flight-gate scheduling as a clique partitioning problem. *Transportation Science*, 42(3):292–301.

Dorndorf, U. and Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1):25–40.

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2000). Constraint propagation techniques for disjunctive scheduling problems. *Artificial Intelligence*, 122:189–240.

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2002). Constraint propagation and problem decomposition: a preprocessing procedure for the job shop problem. *Annals of Operations Research*, 115(1–4):125–145.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., and Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27.

Fisher, H. and Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In Muth, J. F. and Thompson, G. L., editors, *Industrial Scheduling*, pages 225–251, Englewood Cliffs. Prentice-Hall.

Frank, E., Hall, M. A., and Witten, I. H. (2016). *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Morgan Kaufmann, Cambridge, 4th edition.

Freed, T., Doerr, K. H., and Chang, T. (2007). In-house development of scheduling decision support systems: case study for scheduling semiconductor device test operations. *International Journal of Production Research*, 45(21):5075–5093.

Freed, T. and Leachman, R. C. (1999). Scheduling semiconductor device test operations on multihead testers. *IEEE Transactions on Semiconductor Manufacturing*, 12(4):523–530.

Freed, T., Leachman, R. C., and Doerr, K. H. (2002). A taxonomy of scheduling problems in semiconductor device test operations. In *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing*, pages 252–259. IEEE.

Giffler, B. and Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503.

Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1):223–253.

Glover, F. (1998). A template for scatter search and path relinking. In Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., and Snyers, D., editors, *Artificial Evolution*, pages 1–51, Berlin, Heidelberg. Springer.

Gong, G., Deng, Q., Gong, X., Liu, W., and Ren, Q. (2018a). A new double flexible job-shop scheduling problem integrating processing time, green production, and human factor indicators. *Journal of Cleaner Production*, 174:560–576.

Gong, X., Deng, Q., Gong, G., Liu, W., and Ren, Q. (2018b). A memetic algorithm for multi-objective flexible job-shop problem with worker flexibility. *International Journal of Production Research*, 56(7):2506–2522.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Google (2019a). Google OR-Tools. `https://developers.google.com/optimization/`. Last accessed 2020-11-18.

Google (2019b). Google OR-Tools: Scheduling overview. `https://developers.google.com/optimization/scheduling`. Last accessed 2020-11-18.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Günther, H.-O. and Lee, T. E. (2007). Scheduling and control of automated manufacturing systems. *OR Spectrum*, 29(3):373–374.

Gupta, A. K. and Sivakumar, A. I. (2006). Job shop scheduling techniques in semiconductor manufacturing. *International Journal of Advanced Manufacturing Technology*, 27(11–12):1163–1169.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18.

Hall, M. A. (1998). *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand.

Ham, A. M. and Cakici, E. (2016). Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches. *Computers & Industrial Engineering*, 102:160–165.

Han, J., Kamber, M., and Pei, J. (2011). *Data Mining: Concepts and Techniques*. Elsevier, Waltham, 3rd edition.

Hao, X.-C., Wu, J.-Z., Chien, C.-F., and Gen, M. (2014). The cooperative estimation of distribution algorithm: a novel approach for semiconductor final test scheduling problems. *Journal of Intelligent Manufacturing*, 25(5):867–879.

Hart, E., Ross, P., and Nelson, J. (1998). Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computation*, 6(1):61–80.

Haupt, R. (1989). A survey of priority rule-based scheduling. *OR Spektrum*, 11(1):3–16.

He, J., Chen, X., and Chen, X. (2016). A filter-and-fan approach with adaptive neighborhood switching for resource-constrained project scheduling. *Computers & Operations Research*, 71:71–81.

Herrmann, J. W., Lee, C.-Y., and Hinchman, J. (1995). Global job shop scheduling with a genetic algorithm. *Production and Operations Management*, 4(1):30–45.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Hooker, J. N. and Ottosson, G. (2003). Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, New York. ACM.

Hurink, J., Jurisch, B., and Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, 15(4):205–215.

Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2014). AClib: a benchmark library for algorithm configuration. In Pardalos, P. M., Resende, M. G., Vogiatzis, C., and Walteros, J. L., editors, *Proceedings of the 8th International Conference on Learning and Intelligent Optimization*, pages 36–40, Cham. Springer.

Iba, W. and Langley, P. (1992). Induction of one-level decision trees. In Sleeman, D. and Edwards, P., editors, *Proceedings of the 9th International Conference on Machine Learning*, pages 233–240. Morgan Kaufmann, San Francisco.

IBM (2016a). IBM ILOG CPLEX optimization studio 12.7.0: Online documentation. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html`. Last accessed 2018-12-11.

IBM (2016b). IBM ILOG CPLEX optimization studio 12.7.0: Scheduling examples. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.ide.help/OPL_Studio/usroplexamples/topics/opl_cp_examples_scheduling.html`. Last accessed 2018-12-11.

IBM (2019a). IBM ILOG CPLEX optimization studio 12.9.0: Online documentation. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html`. Last accessed 2020-11-18.

IBM (2019b). IBM ILOG CPLEX optimization studio 12.9.0: Scheduling examples. https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.ide.help/OPL_Studio/usroplexamples/topics/opl_cp_examples_scheduling.html. Last accessed 2020-11-18.

Jain, A., Jain, P. K., Chan, F. T. S., and Singh, S. (2013). A review on manufacturing flexibility. *International Journal of Production Research*, 51(19):5946–5970.

Kerschke, P., Hoos, H. H., Neumann, F., and Trautmann, H. (2019). Automated algorithm selection: survey and perspectives. *Evolutionary Computation*, 27(1):3–45.

Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., and Trautmann, H. (2018). Leveraging TSP solver complementarity through machine learning. *Evolutionary Computation*, 26(4):597–620.

Kim, Y.-D., Kang, J.-H., Lee, G.-E., and Lim, S.-K. (2011). Scheduling algorithms for minimizing tardiness of orders at the burn-in workstation in a semiconductor manufacturing system. *IEEE Transactions on Semiconductor Manufacturing*, 24(1):14–26.

Kingma, D. P. and Ba, J. L. (2014). Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kraus, M., Feuerriegel, S., and Oztekin, A. (2020). Deep learning in business analytics and operations research: models, applications and managerial implications. *European Journal of Operational Research*, 281(3):628–641.

Kress, D., Boysen, N., and Pesch, E. (2017). Which items should be stored together? A basic partition problem to assign storage space in group-based storage systems. *IISE Transactions*, 49(1):13–30.

Kress, D., Meiswinkel, S., and Pesch, E. (2019a). Straddle carrier routing at seaport container terminals in the presence of short term quay crane buffer areas. *European Journal of Operational Research*, 279(3):732–750.

Kress, D. and Müller, D. (2019). Mathematical models for a flexible job shop scheduling problem with machine operator constraints. *IFAC-PapersOnLine*, 52(13):94–99.

Kress, D. and Müller, D. (2020). Semiconductor final-test scheduling under setup operator constraints. Working Paper. University of Siegen.

Kress, D., Müller, D., and Nossack, J. (2019b). A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, 41(1):179–217.

Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250.

Landwehr, N., Hall, M., and Frank, E. (2005). Logistic model trees. *Machine Learning*, 59(1-2):161–205.

Lang, M. and Li, H. (2011). Research on dual-resource multi-objective flexible job shop scheduling under uncertainty. In *Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce*, pages 1375–1378. IEEE.

Lawrence, S. (1984). Resouce constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University.

Lee, C.-Y., Uzsoy, R., and Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775.

Lei, D. and Guo, X. (2014). Variable neighbourhood search for dual-resource constrained flexible job shop scheduling. *International Journal of Production Research*, 52(9):2519–2529.

Lei, D. and Tan, X. (2016). Local search with controlled deterioration for multi-objective scheduling in dual-resource constrained flexible job shop. In *Proceedings of the 28th Chinese Control and Decision Conference*, pages 4921–4926. IEEE.

Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140.

Li, Y., Carabelli, S., Fadda, E., Manerba, D., Tadei, R., and Terzo, O. (2020). Machine learning and optimization for production rescheduling in industry 4.0. *The International Journal of Advanced Manufacturing Technology*, 110(9):2445–2463.

Lin, J. T., Wang, F. K., and Lee, W. T. (2004). Capacity-constrained scheduling for a logic IC final test facility. *International Journal of Production Research*, 42(1):79–99.

Liu, X. X., Liu, C. B., and Tao, Z. (2011). Research on bi-objective scheduling of dual-resource constrained flexible job shop. *Advanced Materials Research*, 211–212:1091–1095.

Lowerre, B. T. (1976). *The HARPY speech recognition system*. PhD thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Lunardi, W. T., Birgin, E. G., Laborie, P., Ronconi, D. P., and Voos, H. (2020). Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. *Computers & Operations Research*, 123:105020.

Mastrolilli, M. (2020). Flexible job shop problem. `http://people.idsia.ch/~monaldo/fjsp.html`. Last accessed 2020-12-08.

Mastrolilli, M. and Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20.

Mathirajan, M. and Sivakumar, A. I. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *International Journal of Advanced Manufacturing Technology*, 29(9–10):990–1001.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Meng, L., Zhang, C., Zhang, B., and Ren, Y. (2019). Mathematical modeling and optimization of energy-conscious flexible job shop scheduling problem with worker flexibility. *IEEE Access*, 7:68043–68059.

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

MiniZinc (2013). MiniZinc challenge 2013. `https://www.minizinc.org/challenge2013/challenge.html`. Last accessed 2020-11-18.

MiniZinc (2020). MiniZinc challenge 2020. `https://www.minizinc.org/challenge2020/challenge.html`. Last accessed 2020-11-18.

Mirshekarian, S. and Šormaz, D. N. (2016). Correlation of job-shop scheduling problem features with scheduling efficiency. *Expert Systems with Applications*, 62:131–147.

Müller, D., Müller, M. G., Kress, D., and Pesch, E. (2021). Test instances for the FJSP. `https://doi.org/10.6084/m9.figshare.13522625.v1`. Last accessed 2021-01-14.

Mönch, L., Fowler, J. W., Dauzère-Pérès, S., Mason, S. J., and Rose, O. (2011). A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14(6):583–599.

Morinaga, Y., Nagao, M., and Sano, M. (2014). Optimization of flexible job-shop scheduling with weighted tardiness and setup-worker load balance in make-to-order manufacturing. In *Proceedings of the 7th International Conference on Soft Computing and Intelligent Systems and the 15th International Symposium on Advanced Intelligent Systems*, pages 87–94. IEEE.

Morinaga, Y., Nagao, M., and Sano, M. (2016). Balancing setup workers' load of flexible job shop scheduling using hybrid genetic algorithm with tabu search strategy. *International Journal of Decision Support Systems*, 2(1-3):71–90.

Mousakhani, M. (2013). Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness. *International Journal of Production Research*, 51(12):3476–3487.

Müller, D. and Kress, D. (2019). Filter-and-fan approaches for scheduling flexible job shops under workforce constraints. Working Paper. University of Siegen.

Müller, D., Müller, M. G., Kress, D., and Pesch, E. (2021). An algorithm selection approach for the flexible job shop scheduling problem: choosing constraint programming solvers through machine learning. Working Paper. University of Siegen.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). MiniZinc: towards a standard CP modelling language. In "Bessière, C., editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, pages 529–543, Berlin. Springer.

Nourali, S., Imanipour, N., and Shahriari, M. R. (2012). A mathematical model for integrated process planning and scheduling in flexible assembly job shop environment with sequence dependent setup times. *International Journal of Mathematical Analysis*, 6(41-44):2117–2132.

Ovacik, I. M. and Uzsoy, R. (1992). A shifting bottleneck algorithm for scheduling semiconductor testing operations. *Journal of Electronics Manufacturing*, 2(3):119–134.

Ovacik, I. M. and Uzsoy, R. (1994). Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 32(6):1243–1263.

Ovacik, I. M. and Uzsoy, R. (1995). Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. *International Journal of Production Research*, 33(11):3173–3192.

Ovacik, I. M. and Uzsoy, R. (1996). Decomposition methods for scheduling semiconductor testing facilities. *International Journal of Flexible Manufacturing Systems*, 8(4):357–387.

Özgüven, C., Yavuz, Y., and Özbakır, L. (2012). Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times. *Applied Mathematical Modelling*, 36(2):846–858.

Paksi, A. B. N. and Ma'ruf, A. (2016). Flexible job-shop scheduling with dual-resource constraints to minimize tardiness using genetic algorithm. In *Proceedings of the 2nd International Manufacturing Engineering Conference and 3rd Asia-Pacific Conference on Manufacturing Systems*, page 012060. IOP Publishing.

Pearn, W. L., Chung, S. H., Chen, A. Y., and Yang, M. H. (2004). A case study on the multistage IC final testing scheduling problem with reentry. *International Journal of Production Economics*, 88(3):257–267.

Peng, C., Fang, Y., Lou, P., and Yan, J. (2018). Analysis of double-resource flexible job shop scheduling problem based on genetic algorithm. In *Proceedings of the 15th International Conference on Networking, Sensing and Control*, pages 1–6. IEEE.

Pesch, E. and Glover, F. (1997). TSP ejection chains. *Discrete Applied Mathematics*, 76(1):165–181.

Pesch, E. and Tetzlaff, U. A. W. (1996). Constraint propagation based scheduling of job shops. *INFORMS Journal on Computing*, 8(2):144–157.

Prud'homme, C., Fages, J.-G., and Lorca, X. (2016). *Choco documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. `https://choco-solver.org`. Last accessed 2020-11-18.

Puget, J. F. (2013). Solving flexible job shop scheduling problems. `https://www.ibm.com/developerworks/community/blogs/jfp/entry/solving_flexible_job_shop_scheduling_problems?lang=en`. Last accessed 2019-07-26.

PwC (2012). Faster, greener, smarter – reaching beyond the horizon in the world of semiconductors. `http://www.pwc.com/gx/en/technology/publications/assets/pwc-faster-greener-smarter.pdf`. PricewaterhouseCoopers AG.

PwC (2013). Spotlight on automotive – pwc semiconductor report. `https://www.pwc.com/gx/en/technology/publications/assets/pwc-semiconductor-survey-interactive.pdf`. PricewaterhouseCoopers AG.

PyTorch (2020). PyTorch online documentation. `https://pytorch.org/docs/stable/index.html`. Last accessed 2020-11-18.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann.

Rego, C. and Duarte, R. (2009). A filter-and-fan approach to the job shop scheduling problem. *European Journal of Operational Research*, 194(3):650–662.

Rego, C. and Glover, F. (2002). Local search and metaheuristics for the traveling salesman problem. In Gutin, G. and Punnen, A., editors, *The traveling salesman problem and its variations*, pages 309–368, Boston. Springer.

Rego, C. and Glover, F. (2010). Ejection chain and filter-and-fan methods in combinatorial optimization. *Annals of Operations Research*, 175(1):77–105.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, 15:65–118.

Rossi, A. (2014). Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. *International Journal of Production Economics*, 153:253–267.

Rossi, F., van Beek, P., and Walsh, T., editors (2006). *Handbook of Constraint Programming.* Elsevier, Amsterdam.

Saidi-Mehrabad, M. and Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. *The International Journal of Advanced Manufacturing Technology*, 32(5-6):563–570.

Sang, H.-Y., Duan, P.-Y., and Li, J.-Q. (2018). An effective invasive weed optimization algorithm for scheduling semiconductor final testing problem. *Swarm and Evolutionary Computation*, 38:42–53.

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Networks*, 61:85–117.

Schulte, C., Tack, G., and Lagerkvist, M. Z. (2019). Modeling and programming with Gecode. `https://www.gecode.org/doc-latest/MPG.pdf`. Last accessed 2020-11-18.

Sels, V., Gheysen, N., and Vanhoucke, M. (2012). A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions. *International Journal of Production Research*, 50(15):4255–4270.

Shen, L., Dauzère-Pérès, S., and Neufeld, J. S. (2017). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265(2):503–516.

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):1–25.

Sprecher, A., Kolisch, R., and Drexl, A. (1995). Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80(1):94–102.

Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., and Fischer, J. (2014). The minizinc challenge 2008–2013. *AI Magazine*, 35(2):55–60.

Sumner, M., Frank, E., and Hall, M. (2005). Speeding up logistic model tree induction. In Jorge, A. M., Torgo, L., Brazdil, P., Camacho, R., and Gama, J., editors, *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 675–683, Berlin. Springer.

Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117.

Tran, T. T. and Beck, J. C. (2012). Logic-based Benders decomposition for alternative resource scheduling with sequence dependent setups. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 774–779. ACM.

Treleven, M. (1989). A review of the dual resource constrained system research. *IIE Transactions*, 21(3):279–287.

Uzsoy, R., Lee, C.-Y., and Martin-Vega, L. A. (1992a). A review of production planning and scheduling models in the semiconductor industry part I: system characteristics, performance evaluation and production planning. *IIE Transactions*, 24(4):47–60.

Uzsoy, R., Lee, C.-Y., and Martin-Vega, L. A. (1992b). Scheduling semiconductor test operations: minimizing maximum lateness and number of tardy jobs on a single machine. *Naval Research Logistics*, 39(3):369–388.

Uzsoy, R., Lee, C.-Y., and Martin-Vega, L. A. (1994). A review of production planning and scheduling models in the semiconductor industry part II: Shop-floor control. *IIE Transactions*, 26(5):44–55.

Uzsoy, R., Martin-Vega, L. A., Lee, C.-Y., and Leonard, P. A. (1991). Production scheduling algorithms for a semiconductor test facility. *IEEE Transactions on Semiconductor Manufacturing*, 4(4):270–280.

Vallikavungal Devassia, J., Salazar-Aguilar, M. A., and Boyer, V. (2018). Flexible job-shop scheduling problem with resource recovery constraints. *International Journal of Production Research*, 56(9):3326–3343.

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., and De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385.

Vázquez-Rodríguez, J. A. and Petrovic, S. (2010). A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, 16(6):771–793.

Venditti, L., Pacciarelli, D., and Meloni, C. (2010). A tabu search algorithm for scheduling pharmaceutical packaging operations. *European Journal of Operational Research*, 202(2):538–546.

Vilcot, G. and Billaut, J.-C. (2008). A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem. *European Journal of Operational Research*, 190(2):398–411.

Wang, S. and Wang, L. (2015). A knowledge-based multi-agent evolutionary algorithm for semiconductor final testing scheduling problem. *Knowledge-Based Systems*, 84:1–9.

Wang, S., Wang, L., Liu, M., and Xu, Y. (2015). A hybrid estimation of distribution algorithm for the semiconductor final testing scheduling problem. *Journal of Intelligent Manufacturing*, 26(5):861–871.

Wari, E. and Zhu, W. (2019). A constraint programming model for food processing industry: a case for an ice cream processing facility. *International Journal of Production Research*, 57(21):6648–6664.

Wu, J.-Z. and Chien, C.-F. (2008). Modeling semiconductor testing job scheduling and dynamic testing machine configuration. *Expert Systems with Applications*, 35(1–2):485–496.

Wu, J.-Z., Hao, X.-C., Chien, C.-F., and Gen, M. (2012). A novel bi-vector encoding genetic algorithm for the simultaneous multiple resources scheduling problem. *Journal of Intelligent Manufacturing*, 23(6):2255–2270.

Wu, R., Li, Y., Guo, S., and Xu, W. (2018). Solving the dual-resource constrained flexible job shop scheduling problem with learning effect by a hybrid genetic algorithm. *Advances in Mechanical Engineering*, 10(10):1–14.

Wu, X., Peng, J., Xiao, X., and Wu, S. (2020). An effective approach for the dual-resource flexible job shop scheduling problem considering loading and unloading. *Journal of Intelligent Manufacturing*, pages 1–22.

Xianzhou, C. and Zhenhe, Y. (2011). An improved genetic algorithm for dual-resource constrained flexible job shop scheduling. In *Proceedings of the 4th International Conference on Intelligent Computation Technology and Automation*, pages 42–45. IEEE.

Xiong, H. H. and Zhou, M. (1998). Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search. *IEEE Transactions on Semiconductor Manufacturing*, 11(3):384–393.

Xu, J., Xu, X., and Xie, S. Q. (2011). Recent developments in dual resource constrained (DRC) system research. *European Journal of Operational Research*, 215(2):309–318.

Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. (2012). Evaluating component solver contributions to portfolio-based algorithm selectors. In Cimatti, A. and Sebastiani, R., editors, *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, pages 228–241, Berlin. Springer.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606.

Yang, G., Chung, B. D., and Lee, S. J. (2019). Limited search space-based algorithm for dual resource constrained scheduling problem with multilevel product structure. *Applied Sciences*, 9(19):4005.

Yazdani, M., Zandieh, M., and Tavakkoli-Moghaddam, R. (2019). Evolutionary algorithms for multi-objective dual-resource constrained flexible job-shop scheduling problem. *OPSEARCH*, 56(3):983–1006.

Yazdani, M., Zandieh, M., Tavakkoli-Moghaddam, R., and Jolai, F. (2015). Two meta-heuristic algorithms for the dual-resource constrained flexible job-shop scheduling problem. *Scientia Iranica - Transactions E*, 22(3):1242–1257.

Zhang, J. and Liu, G. B. (2012). Hybrid ant colony algorithm for job shop schedule with unrelated parallel machines. In *Proceedings of the Conference on Frontiers of Advanced Materials and Engineering Technology*, pages 905–908. Trans Tech Publications.

Zhang, J., Wang, W., and Xu, X. (2015). A hybrid discrete particle swarm optimization for dual-resource constrained job shop scheduling with resource flexibility. *Journal of Intelligent Manufacturing*, 28(8):1961–1972.

Zhang, J., Wang, W., Xu, X., and Jie, J. (2013). A multi-objective particle swarm optimization for dual-resource constrained shop scheduling with resource flexibility. In *Proceedings of the IEEE Symposium on Computational Intelligence for Engineering Solutions*, pages 29–34. IEEE.

Zhang, Z., Zhang, M. T., Niu, S., and Zheng, L. (2006). Capacity planning with reconfigurable kits in semiconductor test manufacturing. *International Journal of Production Research*, 44(13):2625–2644.

Zhang, Z., Zheng, L., Hou, F., and Li, N. (2011). Semiconductor final test scheduling with Sarsa(λ, k) algorithm. *European Journal of Operational Research*, 215(2):446–458.

Zheng, X.-L. and Wang, L. (2016). A knowledge-guided fruit fly optimization algorithm for dual resource constrained flexible job-shop scheduling problem. *International Journal of Production Research*, 54(18):5554–5566.

Zheng, X.-L., Wang, L., and Wang, S.-Y. (2014). A novel fruit fly optimization algorithm for the semiconductor final testing scheduling problem. *Knowledge-Based Systems*, 57:95–103.

Zhu, H., Deng, Q., Zhang, L., Hu, X., and Lin, W. (2020). Low carbon flexible job shop scheduling problem considering worker learning using a memetic algorithm. *Optimization and Engineering*, 21(4):1691–1716.

Zhu, X. and Wilhelm, W. E. (2006). Scheduling and lot sizing with sequence-dependent setup: A literature review. *IIE Transactions*, 38(11):987–1007.