Semiconductor final-test scheduling under setup operator constraints[†]

Dominik Kress[a], David Müller[b,*]

[a] *Helmut Schmidt University - University of the Federal Armed Forces Hamburg, Business Administration, especially Procurement and Production, Friedrich-Ebert-Damm 245, 22159 Hamburg, Germany*
[b] *University of Siegen, Management Information Science, Kohlbettstraße 15, 57068 Siegen, Germany*

Abstract

We consider a semiconductor final-test scheduling problem that aims at minimizing the 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 decomposition-based heuristic solution approaches and a mixed integer program. In a computational study based on real-world problem instances that mimic settings at our industry partner, we show that our heuristics clearly outperform 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.

Keywords: Scheduling, Flexible job shop, Semiconductor, Tabu search, Simulated annealing

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

*Corresponding author
Email addresses: dominik.kress@hsu-hh.de (Dominik Kress), david.mueller@uni-siegen.de (David Müller)

[†] This is an Accepted Manuscript of an article published by Elsevier in **Computers & Operations Research**, available online: https://doi.org/10.1016/j.cor.2021.105619

a scheduling problem in one of the company's final-test divisions, where packaged semiconductors are subjected to functional tests.

1.1. The semiconductor manufacturing process

The process of manufacturing semiconductor devices can be divided into *front-end* and *back-end* operations; see Figure 1, where optional operations are depicted by dash-dotted boxes. The process is

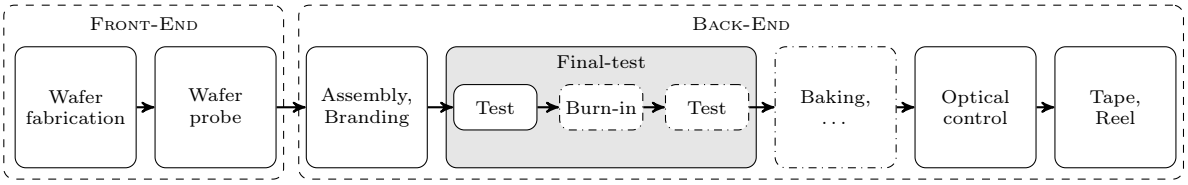

Figure 1: Semiconductor manufacturing process

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. First, *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 that aim to determine whether the device is operating at the required specifications and, thus, aim to guarantee defect free products. A device may, for example, be intended to operate at a specific frequency, which can be tested by stimulating the device accordingly. The functional tests are performed at different temperatures on specific test machines (see Section 1.2). 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*).

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). This is schematically illustrated in Figure 2. The *tester*

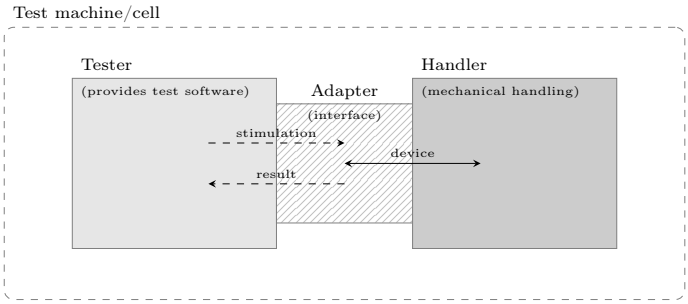

Figure 2: Schematic representation of a test cell

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*).

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.

1.3. Literature overview and contribution

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 and closely related fields 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 articles assumes labor to "not [being] a constraining factor" (Deng et al. 2010) and therefore deviates from our industry case. In this stream of research, Ovacik and Uzsoy (1992, 1994, 1995, 1996) develop decomposition-based solution 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). With respect to metaheuristic solution approaches, Deng et al. (2010) propose a greedy randomized adaptive search procedure. The vast majority of publications, however, focusses on population-based procedures. Genetic algorithms, for instance, are proposed by Wu et al. (2012), Wu and Chien (2008), and Herrmann et al. (1995). Wu and Chien (2008) additionally present an MIP for their specific problem setting. Wang et al. (2015) and Hao et al. (2014) introduce variants of the estimation of distribution algorithm. Other nature-inspired algorithms are proposed by Cao et al.

(2018), Sang et al. (2018), Wang and Wang (2015), 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 similar to our industry case. We are not aware of further articles that explicitly address the incorporation of setup operators into semiconductor final-test scheduling problems. Bard et al. (2012), however, consider a very specific industry case at Texas Instruments. They present procedures that are specifically designed for this setting and that make use of pre-computed (using the methods described in Deng et al. 2010) schedules that include decisions on machine setups for a very specific objective function found at Texas Instruments. These schedules are used as an input for a real-time control model that aims at prioritizing setup operations under limited crew availability and specifically targets performance measures at the industry partner. In contrast to the study by Bard et al. (2012), we consider a more general problem setting that uses an objective function that is more common in the scheduling literature and that directly integrates decisions on the setup of test machines and the assignment of setup operators when scheduling functional tests. This allows to derive managerial insights that can be utilized by a broader variety of companies.

SFTPS is a variant of the *flexible job shop scheduling problem* (FJSP), which itself is a generalization of the *job shop scheduling problem* (JSP; see, e.g., Blazewicz et al. 2019). It is well known that JSP is strongly NP-hard for minimizing total tardiness (Graham et al. 1979; Lenstra and Rinnooy Kan 1979). Hence, SFTPS is strongly NP-hard as well.

Comprehensive surveys on scheduling problems with setup considerations, including the JSP and its generalizations, are presented by Allahverdi (2015), Allahverdi et al. (1999), Allahverdi et al. (2008), and Zhu and Wilhelm (2006). Additionally, Chaudhry and Khan (2016) have recently surveyed literature on solution approaches for the FJSP and its variants. According to their study, population-based metaheuristics are the most popular approaches in the literature, followed by quite a few variants of tabu search approaches. The predominance of population-based approaches can also be observed for recently published articles on the FJSP that are not surveyed by Chaudhry and Khan (2016). Examples include Defersha and Rooyani (2020) and Ahmadi et al. (2016), who present genetic algorithms, Luo et al. (2020) and Gong et al. (2018), who propose memetic algorithms, or Mihoubi et al. (2020), who embed a genetic algorithm into their solution approach. Many researches combine population-based metaheuristics with other algorithms. Chen et al. (2020), for instance, combine a genetic algorithm with a reinforcement learning method. Kato et al. (2018) propose a particle swarm optimization approach that incorporates a random-restart hill climbing procedure. Other researchers incorporate tabu search techniques into population-based approaches, both for intensification and diversification purposes. Examples include Nouri et al. (2018) and Li and Gao (2016) (genetic algorithms) or Li et al. (2017) (artificial bee colony algorithm). Lunardi et al. (2021) propose an iterated local search heuristic and two population-based metaheuristics (a genetic algorithm and a differential evolution method). They combine the differential evolution method with a tabu search approach and find that this combined approach outperforms the

other procedures. Tabu search procedures have recently been proposed by Lunardi et al. (2021), Shen et al. (2018) and Aschauer et al. (2017).

Summing up, 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. With respect to solution approaches, most of the promising and recently published articles present population-based procedures. The development and success of these approaches, however, oftentimes requires intricate guiding strategies, non-adaptive stopping criteria, and complex parameter adjustments (see, e.g., He et al. 2016). Especially the adjustment of the parameters can be a very time consuming task, in particular when facing frequently changing problem instances (see, e.g., Burke and Kendall 2014). Hence, from the perspective of our industry partner, these approaches are less attractive than classical and well-established local search approaches that tackle the aforementioned drawbacks and are additionally easy to implement. In the paper at hand, we therefore focus on the development of two of the most popular classical metaheuristic procedures, namely tabu search procedures, that, as shown above, are frequently applied for FJSP settings, and simulated annealing approaches, that have been successfully applied to various optimization problems (see, e.g., Burke and Kendall 2014; Delahaye et al. 2019). In order to be competitive with respect to solution quality, these approaches have to make use of strategies that allow an effective exploration of the solution space. Hence, our approaches make use of neighborhood structures that have proven successful for the FJSP (see Mastrolilli and Gambardella 2000). In order to adapt these neighborhood structures, we decompose SFTPS into a FJSP with sequence-dependent setup times and an assignment problem. While the former problem focusses on the allocation of operations to testers and the sequencing of the operations, the latter problem considers the assignment of the additional resources, i.e., handlers, adapters, and setup operators.

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.

The remainder of this article is structured as follows. In Section 2, we define SFTPS in detail. Next, in Section 3, we introduce our decomposition-based heuristic solution approaches. They are evaluated in an extensive computational study in Section 4, where we derive managerial implications on the

application of our heuristics in dynamic environments. The paper closes with a conclusion in Section 5.

2. Detailed problem statement

In this section, we define the notation and provide a formal definition of SFTPS. A corresponding MIP is presented in Appendix A. For the sake of notational convenience, we will usually make use of index sets in order to address specific problem elements (jobs, resource types, resource classes, etc.). The concrete entity that we refer to will always become clear from the context.

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 (see Figure 2), each test cell is a combination of entities belonging to three *resource types*: one tester, one handler, and one adapter. We denote the set of these types by $K = \{1, 2, 3\}$. There exist different *classes* of each resource type (see Table 1). Each class represents a specification of a resource type, e.g., a

Table 1: Overview of notation related to hardware resources

Resource type	Tester ($k = 1$)	Handler ($k = 2$)	Adapter ($k = 3$)
Resource classes	$R^1 = \{1, \ldots, r^1\}$	$R^2 = \{1, \ldots, r^2\}$	$R^3 = \{1, \ldots, r^3\}$
Number of copies	One copy of each class ($q_i^1 = 1 \, \forall \, i \in R^1$)	Arbitrary number of copies of each class ($q_i^2 \in \mathbb{N}$, $i \in R^2$)	Arbitrary number of copies of each class ($q_i^3 \in \mathbb{N}$, $i \in R^3$)

specific form of a 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 $m[k]$. For the sake of brevity in our textual descriptions

in the remainder of this paper, we will only explicitly differentiate between resource classes and specific copies/entities if this is needed in the given context.

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* $p^m_{j_i} \in \mathbb{N}$ 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). For all resource types and operations, each resource class that is included in at least one eligible machine configuration is said to be eligible for this specific operation.

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 machine configurations 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 class $i \in R^1$. 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 classes* that are included in the sets R^k, $k \in \{2, 3\}$.

Table 2 summarizes the notation that has been introduced so far. It complements Table 1.

Table 2: 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)	$K = \{1, 2, 3\}$
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 $m \in M_{j_i}$	$p^m_{j_i} \in \mathbb{N}$
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 3). First, the machine configuration may remain unchanged with the adapter and handler entities remaining installed. In this case, we assume that the setup time solely depends 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 first disassembling and afterwards assembling an adapter. Note that this includes the case $m' = m$, which becomes practically relevant when there are very few copies of some frequently needed adapter class. In this case, one will have to disassemble m by

removing the adapter entity of class $m[3]$, which we assume to take $\bar{s}^m_{out} \in \mathbb{N}_0$ time units, and install an adapter entity of class $m'[3]$ 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 modify the handler. Again, this includes disassembling a handler entity of some specific class and later installing an entity of the same class. 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 a handler entity of class $m[2]$ from m by $\hat{s}^m_{out} \in \mathbb{N}_0$ and the time needed to install a handler entity of class $m'[2]$ 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 3. We

Table 3: 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)	$\bar{s}^m_{out} + \hat{s}^m_{out}$	$\hat{s}^{m'}_{in} + \bar{s}^{m'}_{in} + \hat{s}_{j_i,g_h}$

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 entities 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. Finally, given three operations $j_i \in \hat{O}$ and $u_v, k_l \in O$ and a tester that is eligible for processing all of these operations, we assume that the smallest possible overall setup time needed when processing k_l immediately after j_i on eligible machine configurations that include this tester can never be larger than the smallest possible overall setup time needed when processing u_v in between j_i and k_l on eligible machine configurations that include the tester.

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 4.

We close this section by presenting an example for disassembling and assembling a test cell on a specific tester class (tester class index 1) by means of a partial Gantt-chart in Figure 3. It assumes that an operation 4_2 is, at some point in time, started to be processed on machine configuration $(1, 3, 5)$.

Table 4: 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 $j \in J$	$T_j := \max\{C_j - d_j, 0\}$

Afterwards the machine configuration is modified by disassembling the currently installed adapter and handler entities and assembling entities of other eligible handler and adapter classes in order to process operation 3_1 on machine configuration $(1, 4, 2)$.

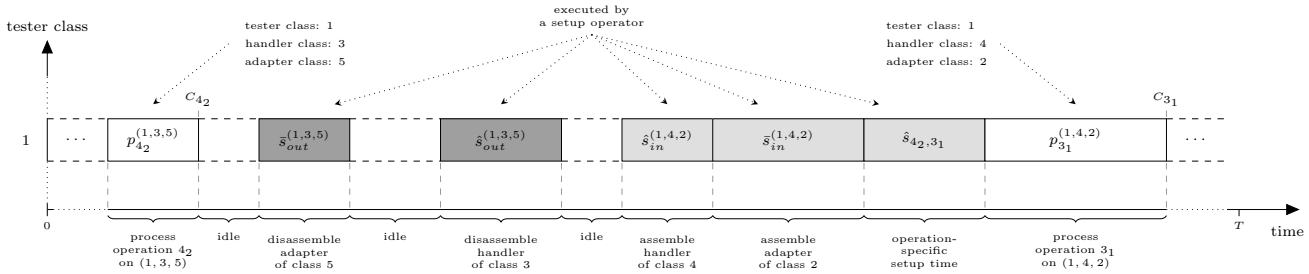

Figure 3: Exemplary illustration of setup operations

3. Heuristic approaches

In this section, we present heuristic approaches 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 (2021), i.e., we decompose SFTPS into a FJSP with sequence-dependent setup times that considers the assignment of operations to eligible testers and the sequencing of these operations (master problem, Section 3.1), and an assignment problem that considers handlers, adapters, and setup operators (subproblem, Section 3.2).

3.1. Master problem

In the master problem, we solely focus on assigning the operations to eligible testers and on sequencing these operations on their assigned testers. In line with the overall objective function, we aim at minimizing the total weighted tardiness. However, as we neglect handlers, adapters and setup operators, we make use of modified processing times and setup times. With respect to the processing times of operations $j_i \in O$ on testers $r \in R^1$, we use lower bounds $\min_{m \in M_{j_i} \cap M_r^1} p_{j_i}^m$ (*master processing times*). Similarly, the setup times between succeeding operations on some specific eligible tester are represented by the smallest possible total setup times (disassembly, assembly, and operation-specific components) that can arise over all corresponding eligible machine configurations that include the tester. The master problem is a variant of the FJSP. It is therefore strongly NP-hard (see Section 1.3).

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 (2021). In the following, we will generalize this concept in order to be able to take account of setup times.

The solution graph contains a distinct vertex for each operation (excluding the dummy operations). We denote the resulting vertex set by V, so that $V := O$. The vertices of this set are weighted with the master processing times according to the tester allocation of the solution. Additional dummy vertices with zero weights, denoted by $(n+1)_0$ and $(n+1)_1$, represent the start and the end of the schedule. We define $D := \{(n+1)_0\} \cup \{(n+1)_1\}$. The graph furthermore includes two sets of directed edges. The first set, denoted by E_1, includes an edge for each direct precedence relation among the operations of the jobs. Hence, for each job $j \in J$ and all pairs j_i, j_{i+1} with $i \in \{1, \ldots, q_j - 1\}$, the set E_1 includes the directed edge (j_i, j_{i+1}). Additionally, dummy edges $((n+1)_0, j_1)$ and $(j_{q_j}, (n+1)_1)$ for all $j \in J$ are added to E_1. The second set, denoted by E_2, represents the tester allocation and sequencing decisions of the given solution in the same manner. The edges of this set are weighted with the master setup times. An edge of this set that originates in the dummy vertex $(n+1)_0$ ends in the first operation that is processed on the respective tester in the given solution and is thus weighted with the master setup time that relates to the corresponding dummy operation as defined in Section 2. For each tester that processes no operation, a dummy edge with zero weight from vertex $(n+1)_0$ to vertex $(n+1)_1$ is included in E_2.

Figure 4 illustrates the solution graph and the corresponding Gantt-chart of a feasible solution of the master problem for an example instance of SFTPS with three jobs and two testers. The solid edges represent the edges of the set E_1 (precedence constraints). Dashed and dotted edges are included in the set E_2 and represent the tester allocation as well as the sequencing decisions for testers 1 and 2, respectively. The master processing times of the current solution are depicted for all operations. In order to keep the example simple, only two of the relevant master setup times are assumed to be positive (1_1 to 2_2 on tester 1 and 3_1 to 3_2 on tester 2). The figure only depicts these non-zero edge weights. In the depicted solution, tester 1 processes operations 1_1, 2_2, and 3_3. Tester 2 processes operations 3_1, 3_2, 2_1, 1_2, and 1_3. As a result of the precedence constraints, tester 1 is idle from time instant 1 to time instant 4.

Each vertex $j_i \in V \cup D$ is associated to a starting time and a tail time. The *starting time*, denoted by $start_{j_i}$, corresponds to the time instant at which the corresponding operation is started to be processed on its respective tester when assuming that the master processing times and the master setup times actually occur. It corresponds to the length of a longest path from $(n+1)_0$ to vertex j_i. Here, the length of a path is measured in terms of the sum of vertex and edge weights on the path without taking the last vertex weight into account. Similarly, the *tail time*, denoted by $tail_{j_i}$, relates to the length of a longest path from j_i to $(n+1)_1$ when excluding the vertex weight of operation j_i. The starting

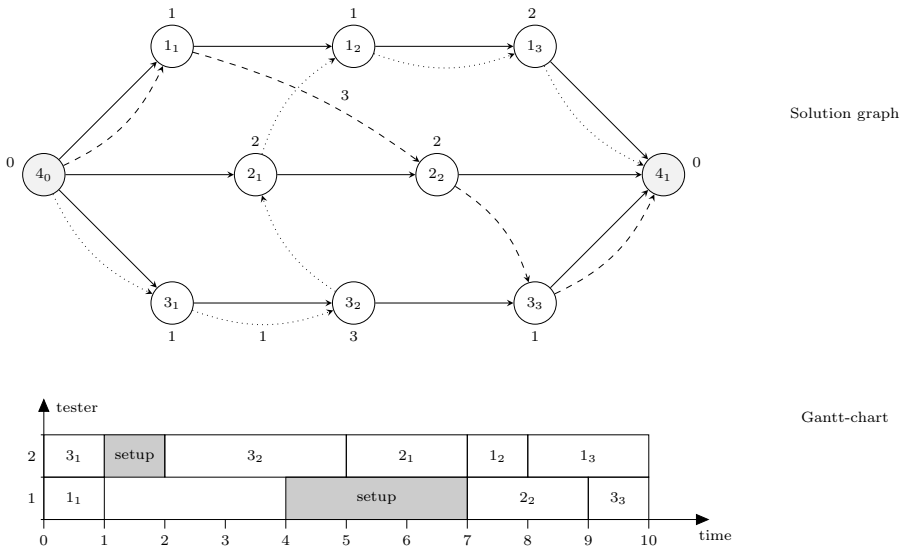

Figure 4: Exemplary illustration of the solution graph

and tail times of all vertices can easily be computed in polynomial time by a straightforward labelling algorithm. Given these values, the objective function value of the solution follows readily. It defines a lower bound on the total weighted tardiness of the solution after additionally taking account of the remaining hardware resources and setup restrictions.

Table 5 illustrates the staring times and tail times for the solution graph of Figure 4.

Table 5: Starting times and tail times for the exemplary solution graph

j_i	4_0	1_1	1_2	1_3	2_1	2_2	3_1	3_2	3_3	4_1
$start_{j_i}$	0	0	7	8	5	7	0	2	9	10
$tail_{j_i}$	10	6	2	0	3	1	9	5	0	0

We are now ready to define a neighborhood structure on the solution graph in line with Mastrolilli and Gambardella (2000). Given a solution graph G, and an operation $j_i \in V$, we denote the unique operation $u_v \in V \cup D$ with $(u_v, j_i) \in E_1$ ($(j_i, u_v) \in E_1$) by $P(j_i)$ ($S(j_i)$). Furthermore, we denote the mapping of the vertex weights of the solution graph by $\mu : V \cup D \to \mathbb{N}$. Generally speaking, we construct a neighboring solution graph by moving a selected operation a_b to one of its eligible testers r. To do so, we remove a_b from its current tester sequence by modifying edge set E_2 accordingly. We then update the starting time and tail time of operation a_b, i.e., we set $start_{a_b} := start_{P(a_b)} + \mu(P(a_b))$ and $tail_{a_b} := \mu(S(a_b)) + tail_{S(a_b)}$. Note that, due to our assumptions on the setup times in Section 2, the starting times and the tail times of the remaining vertices of the graph are guaranteed to not increase due to this modification. Next, we determine potential trial moves of operation a_b to the operation sequence of tester r by making use of feasibility results by Mastrolilli and Gambardella (2000). Denote the set of operations that is processed by tester r by Q_r and assume that the elements of this set are ordered in non-decreasing order of their starting times. Next, compute the sets $L_r := \{j_i \in Q_r \mid \mu(j_i) + tail_{j_i} > tail_{a_b}\}$ and $R_r := \{j_i \in Q_r \mid start_{j_i} + \mu(j_i) > start_{a_b}\}$. As shown by Mastrolilli and Gambardella (2000), all insertions of a_b after the operations of the set $L_r \setminus R_r$ and before the operations of the set $R_r \setminus L_r$ result

in feasible solutions of the master problem. In the corresponding proof, the authors show that these insertion operations do not induce a cycle in the solution graph, which directly relates to feasibility. This remains true for the case at hand in spite of the fact that, in contrast to the setting considered by Mastrolilli and Gambardella (2000), the weight of vertex a_b will potentially be altered after the insertion and the fact that new weighted edges are introduced. This is essentially because these modifications solely cause temporal shifts of operations on testers.

Our neighborhood structure is such that, for a given operation a_b and tester r, we construct the solution graphs for all of the aforementioned feasible insertions (without considering the feasible insertion at the original position of the operation), recompute the corresponding starting times and tail times as well as the objective function values, and then select the most promising candidate. It is embedded into our heuristic frameworks as described in detail in Section 3.4.1.

Consider the example solution graph of Figure 4 and assume that we consider the insertion of operation $a_b = 2_1$ on tester 1. According to the above procedure, we first remove operation 2_1 from its current tester sequence by modifying edge set E_2 and updating the starting and tail time of operation 2_1. We compute $start_{2_1} = 0$, $tail_{2_1} = 3$, $Q_1 = \{4_0, 1_1, 2_2, 3_3, 4_1\}$, $L_1 = \{4_0, 1_1\}$, and $R_1 = \{1_1, 2_2, 3_3, 4_1\}$. Hence, all insertions of 2_1 after operations of the set $L_1 \setminus R_1 = \{4_0\}$ and before operations of the set $R_1 \setminus L_1 = \{2_2, 3_3, 4_1\}$ result in feasible neighboring solution graphs. Figure 5 illustrates the case of inserting 2_1 after operation 1_1.

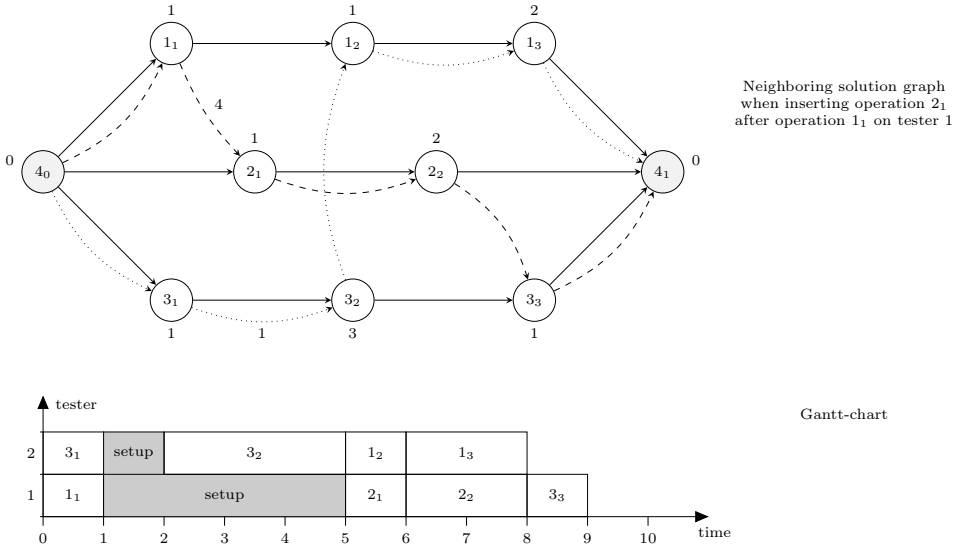

Figure 5: Exemplary illustration of a neighboring solution graph

3.2. Subproblem

The subproblem is to determine a feasible solution of SFTPS by assigning handler and adapter entities to the operations that have been allocated to testers within the master problem so that they are processed on eligible machine configurations. Moreover, the subproblem has to decide on the assignment on setup operators to the resulting setup operations. Again, the problem aims at the minimization of

the total weighted tardiness.

Even though the sequences of the operations on their associated testers are fixed, the subproblem turns out to be strongly NP-hard, even for the case when all setup times are zero and when only one of the hardware resources (either handlers or adapters) is scarce. This can be seen when considering the flow shop scheduling problem with machine operators (FSPO) as introduced by Benkalai et al. (2019). Here, a set J of n jobs and a set M of m machines is given. Each job $j \in J$ has to be processed on each machine in fixed order of the machine indices. It therefore has exactly m operations, denoted by $O_{1j}, \ldots, O_{mj}$. The processing of operation O_{ij} takes $p_{ij} \in \mathbb{N}$ time units and may not be preempted. The processing of an operation requires the presence of one of $k \leq m$ machine operators for the entire processing time. Benkalai et al. (2019) show that this problem is strongly NP-hard for minimizing the makespan when the job sequence is fixed. As illustrated by Graham et al. (1979), this implies strong NP-hardness when instead aiming to minimize the total weighted tardiness. It is now straightforward to show that our subproblem with zero setup times and only one scarce hardware resource is a generalization of this setting (the scarce resource can be interpreted as a machine operator in the FSPO setting).

When designing a heuristic approach for SFTPS based on the proposed decomposition, it is especially important to effectively explore the solution space. We will therefore have to evaluate a large amount of subproblems in the course of the algorithm (see Section 4.3 for an associated analysis). Thus, in light of the subproblem's computational complexity, a heuristic algorithm for solving the subproblem is needed. In order to achieve overall runtimes that are acceptable for practical applications, a greedy algorithm is a reasonable choice. Given the solution graph of a solution to the master problem, we therefore solve the subproblem by iteratively assigning handler and adapter entities as well as setup operators in a greedy manner.

The details of our greedy approach are presented in Algorithm 1. It essentially iteratively considers the operations, i.e., the vertices of the solution graph, in an order that guarantees the construction of a feasible solution. Feasibility is achieved by solely selecting *available operations* in line 9. An operation is available, if all of its direct predecessors (with respect to both edge sets) in the solution graph have been selected in a previous iteration of the algorithm. The dummy vertex $(n+1)_0$ is initially defined to be a vertex that has previously been selected. The concrete strategy of selecting available operations must be specified when calling the algorithm (see Section 4.3 for more details). Next, the algorithm corrects (i.e., shifts) the starting time of the selected operation based on a resource assignment that is constructed in a greedy manner and the associated correct processing and setup times as well as the availability of setup operators (lines 10–25). During runtime, the algorithm keeps track of 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. For a selected available operation, the algorithm considers every eligible

Algorithm 1. Determine a feasible allocation of handlers, adapters and setup operators

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 **while** *vertex set* V *of* G *contains at least one available operation that has not yet been selected* **do**
9 Select an available operation j_i that has not yet been selected;
10 Initialize $C^*_{j_i} := \infty$ and $m^* := \emptyset$;
11 Get the tester class i' assigned to j_i from G;
12 Get the last assigned machine configuration $\hat{m}$ that includes tester i';
13 Set $S^* := S$;
14 **forall** *eligible machine configurations* $m \in M_{j_i} \cap M^1_{i'}$ **do**
15 Set $\hat{S} := S$;
16 Evaluate machine configuration m (details in the text);
17 Compute the completion time $\hat{C}_{j_i}$ according to the above evaluation;
18 **if** $\hat{C}_{j_i} < C^*_{j_i}$ **then**
19 $C^*_{j_i} := \hat{C}_{j_i}$;
20 $m^* := m, S^* := \hat{S}$;
21 **end**
22 **end**
23 Assign the most promising machine configuration m^* to operation j_i;
24 Update resource information and release times of setup operators;
25 Set $S := S^*$;
26 **end**
27 Compute objective function value WT;

machine configuration based on the fixed tester decision of the given solution graph (loop 14–22). When evaluating the machine configuration (line 16), 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 18–21). More specifically, the evaluation proceeds as follows. Based on the setup type (see Table 3) 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 quickly 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 27).

3.3. Constructive procedure

We generate an initial solution by making use of a constructive procedure composed of two steps. First, the approach considers the master problem. For the corresponding 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 solve the subproblem as outlined in Section 3.2.

3.4. Improvement procedures

As motivated in Section 1.3, we implemented a tabu search heuristic as well as a simulated annealing procedure that we outline in this section. To ease the notation, we will refer to the objective function value of a solution S by using an additional label, i.e., $\ddot{S}$.

3.4.1. Neighborhood definition

Our improvement procedures make use of a neighborhood definition that relies on our deliberations in Section 3.1. Specifically, given some solution S of SFTPS, we compute a set of neighboring solutions, denoted by $N(S)$, as outlined in Algorithm 2. The procedure iterates over all operations that belong

Algorithm 2. Construction of the neighborhood $N(S)$

Input: Solution S of SFTPS
Output: Set $N(S)$ of neighboring solutions

Initialize the set of feasible neighboring solutions $N(S) := \emptyset$;
Initialize $O^t := \emptyset$;
Insert all operations of all jobs that complete after their due date in S into O^t;
Initialize the master problem solution graph G of S;
if $O^t = \emptyset$ **then** exit the procedure;
forall *operations* $a_b \in O^t$ **do**
    Determine the set $\bar{R}^1$ of eligible testers of operation a_b;
    forall $r \in \bar{R}^1$ **do**
        Determine a neighbor G^N of G as illustrated in Section 3.1;
        Apply Algorithm 1 on G^N to determine a neighboring solution S^N;
        Set $N(S) := N(S) \cup S^N$;
    end
end

to jobs that complete after their due date in the given solution. For each of these operations and each corresponding eligible tester, we make use of the solution graph of the master problem solution to

compute a neighboring solution graph as described in Section 3.1. For each of these graphs, we compute a solution of the SFTPS instance by calling Algorithm 1 and consider this solution as a neighbor of the input solution.

3.4.2. Tabu search framework

Our tabu search procedure applies a best-fit strategy and is presented in Algorithm 3. The framework

Algorithm 3. Tabu search framework

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 3.3 and initialize $S^* := S$;
2 Initialize an empty tabu list and set $\lambda := 0$;
▷ `Tabu search phase`
3 Construct set $N(S)$ of neighboring solutions by calling Algorithm 2;
4 Discard all elements S^N of $N(S)$ where the corresponding move from S is tabu if $\ddot{S}^N \geq \ddot{S}^*$ (aspiration criterion);
5 Select the best solution S^{NBS} among the remaining solutions in $N(S)$. If no solution remains, i.e., if $N(S) = \emptyset$, exit the procedure;
6 **if** $\ddot{S}^{NBS} < \ddot{S}^*$ **then** set $S^* := S^{NBS}$ and $\lambda := 0$;
7 **else** set $\lambda := \lambda + 1$;
8 Update the tabu list;
9 Set $S := S^{NBS}$;
10 **if** $\lambda < \tau$ **then** go to line 3;
11 **else** exit the procedure;

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 3.3 (line 1) and the tabu search parameters are initialized (line 2). In the *tabu search phase*, we first construct a set $N(S)$ of feasible neighboring solutions by applying Algorithm 2 (line 3). In the next step (line 4), 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 8. 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 operations included in the set O^t in solution S. 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.

3.4.3. Simulated annealing framework

Our simulate annealing framework is presented in Algorithm 4. The *initialization phase* is in analogy to the one of our tabu search heuristic. Here, the current temperature T_{cur} is initialized with the parameter T_{init} (line 2). In the *simulated annealing phase*, we randomly select a feasible neighboring solution S^N of the current solution S (lines 5–9). If this solution has a better objective function value

Algorithm 4. Simulated annealing framework

Input: Instance $Inst$ of SFTPS, parameters $T_{init}, T_{min}, \delta$, and ϕ
Output: Solution S^*

▷ `Initialization phase`
1 Determine a feasible solution S of $Inst$ with the constructive procedure described in Section 3.3 and initialize $S^* := S$;
2 Initialize the current temperature $T_{cur} := T_{init}$;
▷ `Simulated annealing phase`
3 Set $\lambda := 0$ and $N := \emptyset$;
4 Determine $N(S)$ by calling Algorithm 2;
5 **while** $\lambda < \delta$ **do**
6 | Set $\lambda := \lambda + 1$;
7 | **if** $N(S) \neq \emptyset$ **then**
8 | | Randomly select a neighboring solution S^N of $N(S)$;
9 | | Set $N := N \cup S^N$;
10 | | **if** $\ddot{S}^N < \ddot{S}$ **then**
11 | | | Set $S := S^N$ and update $N(S)$ by calling Algorithm 2;
12 | | **else**
13 | | | Draw a random number μ from a uniform distribution over the interval $[0, 1]$;
14 | | | **if** $\mu < \exp(\ddot{S} - \ddot{S}^N / T_{cur})$ **then** set $S := S^N$ and update $N(S)$ by calling Algorithm 2;
15 | | **end**
16 | **end**
17 **end**
18 **if** $N = \emptyset$ **then** exit the procedure;
19 **else** select the best solution S^{NBS} from N;
20 **if** $\ddot{S}^{NBS} < \ddot{S}^*$ **then** set $S^* := S^{NBS}$;
21 Set $S := S^{NBS}$;
22 Update the current temperature $T_{cur} = \phi \cdot T_{cur}$;
23 **if** $T_{min} < T_{cur}$ **then** go to line 4;
24 **else** exit the procedure;

than the current solution, S is updated accordingly (lines 11–12). If, otherwise, the objective function value does not improve, S is only potentially updated according to the classical acceptance criterion applied in most simulated annealing procedures (lines 13–16; see, e.g., Burke and Kendall 2014). This is repeated δ times (loop 6–18). Then, the best solution S^{NBS} among all generated solutions at the current temperature level is selected and the overall best solution S^* is potentially updated (lines 19–22). The current temperature is lowered according to the cooling parameter ϕ (line 23). The algorithm terminates if the resulting temperature T_{cur} is not larger than the one defined by parameter T_{min} (lines 4–25).

4. Computational study and managerial implications

In this section, we analyze the applicability and performance of our heuristic solution approaches 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 4.6, we present an overview of the considered solution approaches (Section 4.1) and the generation of the instance sets that our study is based upon (Section 4.2). In Section 4.3, we perform a pre-evaluation regarding the use and setup of Algorithm 1. The general question of whether or not our heuristics are suited for drawing

managerial conclusions is answered in Sections 4.4 and 4.5.

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 an MIP solver.

4.1. Overview of solution approaches

We implemented four 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 3600 seconds on the MIP presented in Appendix A. Additionally, we implemented the priority rule based constructive procedure (referred to as H-PR, see Section 3.3). It mimics the status quo scheduling approach at our industry partner. Furthermore, we implemented the tabu search procedure (referred to as H-TS, see Section 3.4.2) and the simulated annealing heuristic (H-SA, see Section 3.4.3). With respect to the parameters of the latter two approaches, we set $\tau := \sum_{j \in J} q_j$ for H-TS, and $T_{init} := 5 \cdot |J|$, $T_{min} := 1$, $\delta := 5 \cdot \sum_{j \in J} q_j$, $\phi := 0.95$ for H-SA.

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 6), the large problem

Table 6: Resource scenario of small problem instances

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

instances are based on eight different scenarios (Table 7). The tables indicate whether the integer

Table 7: Resource scenarios of large problem instances

Scenario	r^1	q_i^1	r^2	q_i^2	$r_{R^1}^2$	r^3	q_i^3	$r_{R^1}^3$	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

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 classes of given size $r_{R^1}^2$ and $r_{R^1}^3$, 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 classes). 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 20 instances based on the values and intervals given in Table 8. As indicated in the table, the

Table 8: 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	$[2, 2]$	$[2, 2]$

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 9. For each combination and each resource scenario (A-H), we

Table 9: Parameters of real-world problem instances

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

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 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 from uniform distributions over $[3, 10]$. The overall process is such that the assumptions outlined in Section 2 are met.

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 $[1, 5]$.

4.3. Pre-evaluation of the use and setup of Algorithm 1

Before turning our attention to a detailed analysis of H-TS and H-SA, this section aims to evaluate the use and setup of Algorithm 1. We focus on 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 $|M_{j_i}| \in [3, 5]$ (see Table 9).

4.3.1. General effect of computing subproblem solutions

First, we analyze the question of whether or not it pays off to frequently compute solutions to the subproblems within H-TS and H-SA. We implemented two variants of the heuristics:

1. Hierarchical (HIER): The procedures are solely executed on the master problem, i.e., line 11 of Algorithm 2 is not executed at all. As in PI, Algorithm 1 is called only once at the very end of the procedures.
2. Partial integration (PI): The procedure of evaluating machine configurations in line 16 of Algorithm 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 1 at the very end of the procedures. The algorithm then compares the resulting solution with the solution determined in the initialization phase and selects the best solution.

The computational results are presented in Table 10. For H-TS and H-SA, the table illustrates the performance of the standard setup and the two variants. It presents average objective function values over all instances of the two scenarios (columns 'WT_{avg}') as well as average runtimes (columns 't_{avg}'). Due to the non-deterministic elements of H-SA, we ran all of its variants three times on each instance.

Table 10: Analysis of the incorporation of computing solutions for the subproblems within the heuristics

	H-TS						H-SA					
	Standard		PI		HIER		Standard		PI		HIER	
Sce.	WT_{avg}	t_{avg} [s]	WT_{avg}	t_{avg} [s]	WT_{avg}	t_{avg} [s]	WT_{avg}	t_{avg} [s]	WT_{avg}	t_{avg} [s]	WT_{avg}	t_{avg} [s]
C	11280.1	43.66	14597.7	36.83	43384.85	0.91	9302.52	23.7	12799.98	23.28	42458.63	1.54
D	5138.95	33.73	6626.65	23.36	10715.15	0.95	4501.35	24.37	6201.5	24.01	10912.55	1.54

We observe that a frequent consideration of the subproblem clearly pays off with respect to the solution quality. It is furthermore worthwhile to not only take account of the hardware resources when computing solutions to the subproblems within H-TS and H-SA. The additional consideration of setup operators comes at the cost of only slightly increased runtimes while tending to improve the average solution quality.

In the standard setup, H-TS and H-SA considered an average number of 84560 and 52392 subproblems after the initialization phase, respectively. As the procedures are designed to be applied in a rolling horizon manner at our industry partner, the average runtimes reported in Table 10 therefore support the appropriateness of our choice of a greedy approach used within Algorithm 1.

4.3.2. Selecting available operations within Algorithm 1

As outlined in Section 3.2, we have to specify the concrete strategy of selecting available operations whenever calling Algorithm 1 within our heuristics. We implemented three variants that aim at a quick selection of available operations:

1. Starting time: Operations are selected in non-decreasing order of their staring times in the given solution graph of the master problem solution. As a tie breaker, we select an operation with smallest job index.
2. Shuffle: The operations are first ordered as in the previous strategy. Next, a shuffling procedure is executed with a probability of 50%. It randomly selects operations and moves them to the first position in the sequence. These operations must be direct successors of the dummy vertex $(n+1)_0$ in the solution graph of the master problem solution with respect to both edge sets E_1 and E_2. The shuffling procedure makes a total of $0.8 \cdot r^1$ selection and moving operations. It results in a sequence of operations that defines the order of selecting available operations within the algorithm.
3. Combined: Execute Algorithm 1 with the starting time strategy. Afterwards, with a probability of 50%, make an additional call of the algorithm with the shuffle strategy that is guaranteed to apply the shuffling procedure. Among the two generated solutions, select the one with the smallest objective function value.

The computational results are presented in Table 11. Due to the non-deterministic nature of the strategies shuffle and combined, the according heuristic variants of H-TS as well as all variants of H-SA were executed three times on each instance when using these strategies. The table presents information on the average objective function values when considering all runs (columns 'WT_{avg}') or only the best of the three runs (column 'WT^*_{avg}') when relevant.

Table 11: Comparison of approaches used for of selecting available operations

	H-TS					H-SA					
	Starting time	Shuffle		Combined		Starting time		Shuffle		Combined	
Sce.	WT_{avg}	WT^*_{avg}	WT_{avg}	WT^*_{avg}	WT_{avg}	WT^*_{avg}	WT_{avg}	WT^*_{avg}	WT_{avg}	WT^*_{avg}	WT_{avg}
C	11280.1	9918.55	11076.22	10439.55	11331.83	8528.15	9302.52	9017.45	9787.95	8664.8	9444.07
D	5138.95	4692.8	4971.63	4945.25	5139.85	4266.85	4501.35	4344.5	4640.48	4277.7	4592.57

For H-TS, we observe that the shuffle strategy outperforms the other approaches with respect to solution quality on average. In case of H-SA, the starting time strategy provides the best average results.

4.3.3. Selecting machine configurations within Algorithm 1

Finally, we implemented various rules for selecting the most promising machine configuration in lines 14–22 of Algorithm 1:

1. Completion time (as presented in Section 3.2): Select a machine configuration that results in the smallest completion time of the considered operation.
2. Shortage value: Select an eligible machine configuration m with smallest shortage value $\frac{|M^2_{m[2]}|}{q^2_{m[2]}} + \frac{|M^3_{m[3]}|}{q^3_{m[3]}}$. This rule prefers machine configurations that use hardware resource classes that tend to be least scarce.
3. Combined: Select machine configuration as in the completion time rule. Use the shortage value as a tie breaker.

The computational results are presented in Table 12. As before, we ran H-SA three times on each instance.

Table 12: Comparison of rules used for selecting machine configurations

	H-TS			H-SA					
	Completion time	Shortage value	Combined	Completion time		Shortage value		Combined	
Sce.	WT_{avg}	WT_{avg}	WT_{avg}	WT^*_{avg}	WT_{avg}	WT^*_{avg}	WT_{avg}	WT^*_{avg}	WT_{avg}
C	11280.1	10780	10357.7	8528.15	9302.52	9085.15	9966.68	8588.4	9399.18
D	5138.95	5283.7	5171.7	4266.85	4501.35	4273.45	4587.03	4332.15	4578.22

We find that the combined rule, on average, tends to result in the best solutions for H-TS. With respect to H-SA, the combined rule and the completion time perform similarly well.

4.3.4. Setup of Algorithm 1

Based on the results of our pre-evaluation, we make use of the following setup in the remainder of this paper: standard mode for computing subproblem solutions, shuffle strategy in case of H-TS or starting time strategy in case of H-SA for selecting available operations, combined rule for selecting machine configurations. In H-PR (and the initialization phase of the heuristics), we make use of the starting time strategy and the completion time rule.

4.4. Small instances: basic evaluation of the heuristic frameworks

The computational results for the heuristic approaches H-PR, H-TS, and H-SA on the small instances are presented in Table 13. The table additionally includes information on the performance of CPLEX,

Table 13: Performance of the heuristic approaches on small problem instances

	CPLEX				H-PR	H-TS		H-SA	
Inst. set	feas.	opt.	WT_{avg}	t_{avg} [s]	WT_{avg}	WT^*_{avg}	WT_{avg}	WT^*_{avg}	WT_{avg}
small1	20	20	225	98.32	250.5 (14)	228.05 (18)	228.05	228.05 (18)	228.15
small2	20	20	527.35	227.49	741.4 (6)	612.7 (12)	620.18	638 (11)	638.72
small3	13	3	693.46	3035.77	899.65 (2)	674.65 (3)	685.77	683.5 (2)	716.88
small4	7	0	1336.57	3600.14	1208.6	694.5	723.18	818.1	825.92
small5	0	0	-	-	1651.15	970.9	989.65	937.9	1061.22

i.e., 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 the average objective function values. As in Section 4.3, we ran H-TS and H-SA three times on each instance, so that the table also includes a column 'WT^*_{avg}' for each of these approaches. Note that all calls of the heuristics resulted in a feasible solution. The numbers in parentheses illustrate the number of instances for which a solution that was known to be optimal from the CPLEX results, was detected by the respective approach on the corresponding instance set.

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, especially for the sets small1 and small2, certainly allow to draw first conclusions on the performance of the heuristic approaches. While H-TS and H-SA are able to detect quite a few optimal solutions, they - at first glance - seem to sometimes also result in solutions with a significantly larger objective function value than the one of a known optimal solution. When looking at the results for the individual instances, however, we find that the respective instances are characterized by relatively large weights of tardy jobs (see Appendix B.1). It is therefore interesting, to additionally analyze the performance of our heuristic approaches when fixing all weights to one. The corresponding results are presented in Table 14 (see also Appendix B.2 for details). We observe that, on these modified instances,

Table 14: Performance of the heuristic approaches on small problem instances (all weights are fixed to one)

	CPLEX				H-PR	H-TS		H-SA	
Inst. set	feas.	opt.	WT_{avg}	t_{avg} [s]	WT_{avg}	WT^*_{avg}	WT_{avg}	WT^*_{avg}	WT_{avg}
small1	20	20	85.15	65.52	95.4 (12)	85.45 (19)	85.45	85.45 (19)	85.92
small2	20	20	198.9	204.95	262.4 (6)	206.15 (15)	208.05	210.65 (16)	216.77
small3	15	2	366	3249.9	341 (1)	246 (2)	251.43	245.95 (1)	260.32
small4	8	1	418.75	3460	449.65 (0)	239.85 (1)	254.23	251.05 (1)	268.78
small5	0	0	-	-	595.75	328.65	334.12	314.45	335.28

our heuristic approaches H-TS and H-SA tend to provide optimal or near-optimal solutions for the sets small1 and small2.

Finally, we find that the solutions obtained by H-TS and H-SA are, on average, clear improvements when compared with the solutions determined by H-PR, being the current status quo scheduling approach at our industry partner. This effect is especially pronounced for the larger instances. The average runtimes of the heuristic approaches over all considered instances range from 0.24 to 7.1 milliseconds.

4.5. Large instances: applicability of the heuristic frameworks

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 values returned by H-TS and H-SA with the real-world solution approach at our industry partner (H-PR). Thus, for some given instance and some run of algorithm H-TS and H-SA, 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 considered heuristic framework and H-PR, respectively.

Table 15 summarizes the computational results for the large problem instances grouped by the eight scenarios and the number of jobs. As in the previous sections, H-TS and H-SA were executed three times for each test instance. 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 6 complements Table 15 by illustrating the average quality ratios (Q_{avg}) as

Table 15: Performance of the heuristic approaches on large problem instances

		H-TS		H-SA				H-TS		H-SA	
Sce.	$\|J\|$	Q_{avg}	t_{avg} [s]	Q_{avg}	t_{avg} [s]	Sce.	$\|J\|$	Q_{avg}	t_{avg} [s]	Q_{avg}	t_{avg} [s]
A	20	65.76	5.82	70.47	6.86	B	20	47.02	4.53	54.01	6.51
	30	68.32	19.43	74.45	17.92		30	50.67	14.11	60.46	17.23
	40	69.54	46.9	76.86	37.08		40	55.58	33.57	66.98	34.94
	50	70.99	93.58	79.15	65.31		50	58.1	63.84	71.69	61.08
C	20	59.85	10.1	62.65	9.01	D	20	38.9	8.42	43.15	9.47
	30	64.98	33.46	69.72	24.16		30	46.69	24.48	53.33	25.33
	40	67.45	82.25	73.55	49.24		40	54.34	56.86	61.67	51.5
	50	67.77	175.65	74.48	86.06		50	55.67	111.18	64.03	89.31
E	20	56.12	13.05	58.58	11.11	F	20	33.4	12.52	35.85	11.25
	30	62.27	45.67	66.25	29.59		30	42.51	41.84	46.95	31.11
	40	64.5	114.33	69.47	60.41		40	49.13	96.28	54.51	63.21
	50	65.53	233.94	71.28	105.84		50	52.82	171.74	59.33	109.34
G	20	49.75	16.33	51.7	13.14	H	20	27.3	14.36	29.01	12.39
	30	59.51	53.24	62.64	36		30	37.35	44.92	40.85	33.88
	40	62.51	120.61	66.79	73.03		40	44.08	122.78	48.52	73.52
	50	64.6	255.11	69.79	118.3		50	48.38	221.58	53.51	110.3

well as the average runtimes (t_{avg}) over all resource scenarios. It can be seen that the use of both H-TS

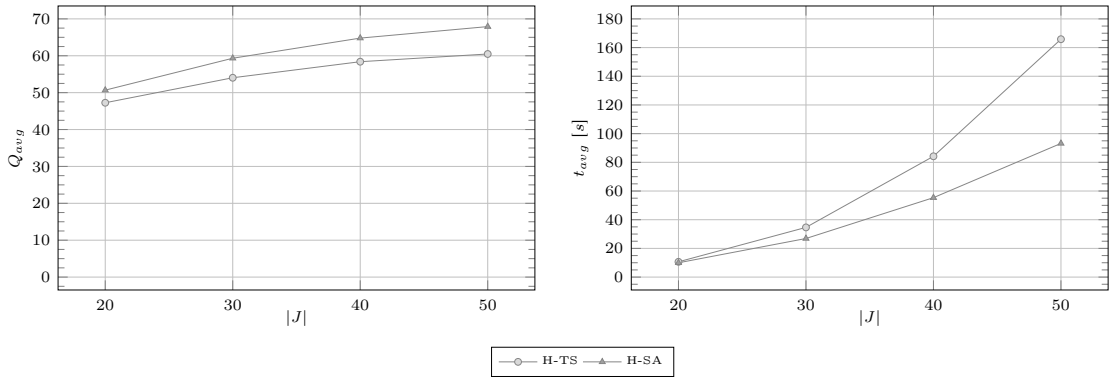

Figure 6: Quality ratios for large problem instances over all resource scenarios

and H-SA significantly improves the solutions that are currently implemented in practice (and used as initial solutions within our framework). Certainly, with respect to the runtimes, both approaches are applicable in practice. However, H-SA tends to outperform H-TS with respect to solution quality. This effect is stronger when the number of jobs is relatively high. As to be expected based on the number of tester and handler entities, we observe that the average quality ratios are smaller in scenarios B, D, F, and H when compared with scenarios A, C, E, and G.

4.6. Managerial implications

Based on the above results we conclude that, in general, our heuristic frameworks are adequate methods for deriving managerial insights. It furthermore seems reasonable to focus on the use of the most promising framework H-SA. The following results therefore rely on this approach. It was executed three times for each test instance. With regard to the test instances, we restrict our attention to the representative resource scenarios C and D, as already motivated in Section 4.3. 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 7 illustrates the corresponding computational results over increasing values of q_i^2 and q_i^3. Naturally, we find that

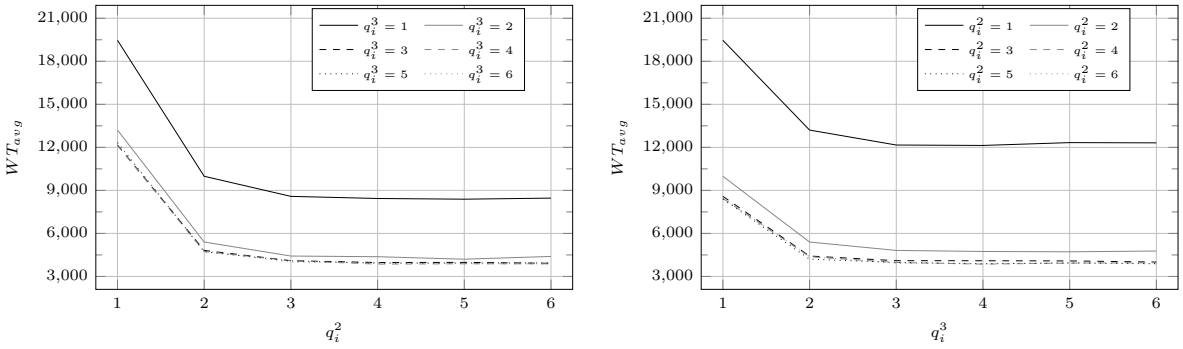

Figure 7: Impact of increasing the number of copies of handler and adapter classes

increasing the number of copies of handler and adapter classes has a 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 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 8 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 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

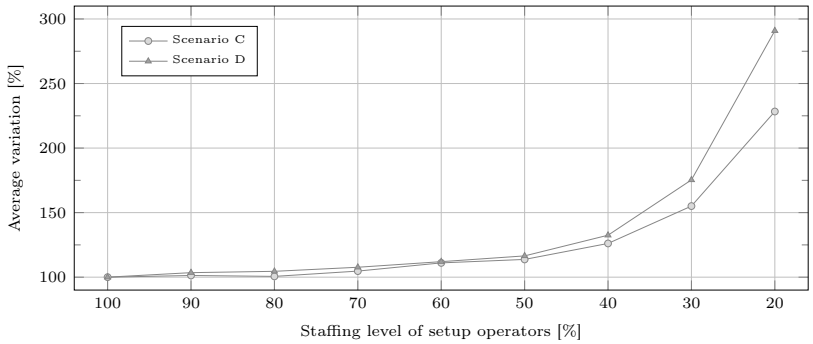

Figure 8: Analysis of staffing level (Setup operators)

(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 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 H-SA. 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 H-SA 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 H-SA, 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 16. We observe that rescheduling jobs with our solution approach has a 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

Table 16: 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	0.82	1.03	0.59
	6	0.8	1.02	0.58
	8	0.79	1.02	0.59
	10	0.79	1.02	0.6
	12	0.79	1.02	0.62
	14	0.79	1.01	0.64
	16	0.79	1	0.65
	18	0.8	1	0.66
	20	0.8	1	0.68
	25	0.8	0.99	0.7
D	4	0.77	1.02	0.54
	6	0.75	1.02	0.56
	8	0.73	1.01	0.57
	10	0.73	1	0.58
	12	0.72	1	0.59
	14	0.72	0.99	0.61
	16	0.72	0.99	0.62
	18	0.72	0.98	0.63
	20	0.72	0.97	0.65
	25	0.73	0.95	0.67

above, we first compute an initial solution S_{Init} by means of H-SA, determine a breakpoint at which the handler or adapter failures occur, and then apply H-SA 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 9. It depicts the average quality

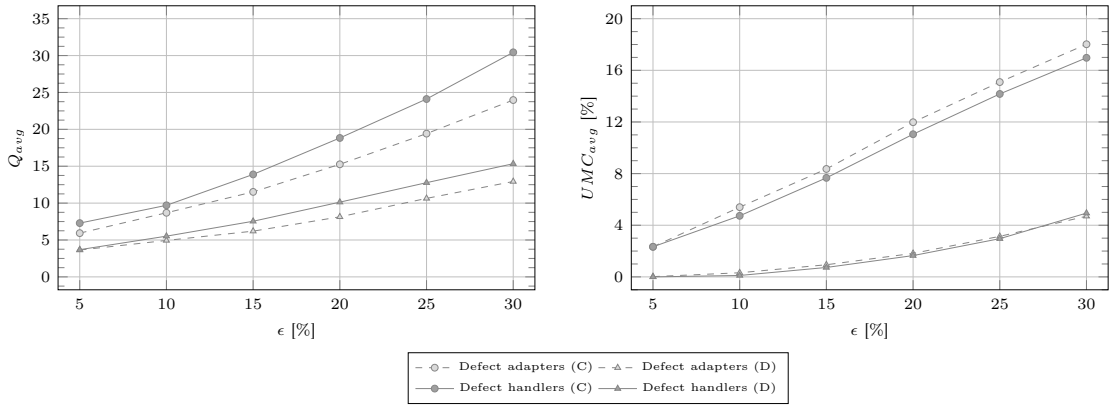

Figure 9: Impact of defect hardware resources

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 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. 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 an MIP and proposed two heuristic frameworks. 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 frameworks, 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.

References

Adams, J., Balas, E., Zawack, D., 1988. The shifting bottleneck procedure for job shop scheduling. Management Science 34, 391–401.

Ahmadi, E., Zandieh, M., Farrokh, M., Emami, S.M., 2016. A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms. Computers & Operations Research 73, 56–66.

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

Allahverdi, A., Gupta, J.N.D., Aldowaisan, T., 1999. A review of scheduling research involving setup considerations. Omega 27, 219–239.

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

Aschauer, A., Roetzer, F., Steinboeck, A., Kugi, A., 2017. An efficient algorithm for scheduling a flexible job shop with blocking and no-wait constraints. IFAC-PapersOnLine 50, 12490–12495.

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

Benkalai, I., Rebaine, D., Baptiste, P., 2019. Scheduling flow shops with operators. International Journal of Production Research 57, 338–356.

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

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

Burke, E.K., Kendall, G., 2014. Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. 2nd ed., Springer, New York.

Cao, Z., Lin, C., Zhou, M., Huang, R., 2018. Scheduling semiconductor testing facility by using cuckoo search algorithm with reinforcement learning and surrogate modeling. IEEE Transactions on Automation Science and Engineering 16, 825–837.

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

Chen, R., Yang, B., Li, S., Wang, S., 2020. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. Computers & Industrial Engineering 149, 106778.

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

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

Defersha, F.M., Rooyani, D., 2020. An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time. Computers & Industrial Engineering 147, 106605.

Delahaye, D., Chaimatanan, S., Mongeau, M., 2019. Simulated annealing: from basics to applications, in: Gendreau, M., Potvin, J.Y. (Eds.), Handbook of Metaheuristics. Springer, Cham, pp. 1–35.

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

Freed, T., Doerr, K.H., 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, 5075–5093.

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

Freed, T., Leachman, R.C., 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, IEEE. pp. 252–259.

Giffler, B., Thompson, G.L., 1960. Algorithms for solving production-scheduling problems. Operations Research 8, 487–503.

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

Graham, R.L., Lawler, E.L., Lenstra, J.K., 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., Sivakumar, A.I., 2006. Job shop scheduling techniques in semiconductor manufacturing. International Journal of Advanced Manufacturing Technology 27, 1163–1169.

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

He, J., Chen, X., 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., Hinchman, J., 1995. Global job shop scheduling with a genetic algorithm. Production and Operations Management 4, 30–45.

Kato, E.R.R., de Aguiar Aranha, G.D., Tsunaki, R.H., 2018. A new approach to solve the flexible job shop problem based on a hybrid particle swarm optimization and random-restart hill climbing. Computers & Industrial Engineering 125, 178–189.

Kim, Y.D., Kang, J.H., Lee, G.E., 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, 14–26.

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

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

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

Li, X., Gao, L., 2016. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. International Journal of Production Economics 174, 93–110.

Li, X., Peng, Z., Du, B., Guo, J., Xu, W., Zhuang, K., 2017. Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems. Computers & Industrial Engineering 113, 10–26.

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

Lunardi, W.T., Birgin, E.G., Ronconi, D.P., Voos, H., 2021. Metaheuristics for the online printing shop scheduling problem. European Journal of Operational Research 293, 419–441.

Luo, Q., Deng, Q., Gong, G., Zhang, L., Han, W., Li, K., 2020. An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers. Expert Systems with Applications 160, 113721.

Mastrolilli, M., Gambardella, L.M., 2000. Effective neighbourhood functions for the flexible job shop problem. Journal of Scheduling 3, 3–20.

Mathirajan, M., 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, 990–1001.

Mihoubi, B., Bouzouia, B., Gaham, M., 2020. Reactive scheduling approach for solving a realistic flexible job shop scheduling problem. International Journal of Production Research , 1–19.

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

Müller, D., Kress, D., 2021. Filter-and-fan approaches for scheduling flexible job shops under workforce constraints. International Journal of Production Research doi:10.1080/00207543.2021.1937745. (in press).

Nouri, H.E., Driss, O.B., Ghédira, K., 2018. Solving the flexible job shop problem by hybrid metaheuristics-based multiagent model. Journal of Industrial Engineering International 14, 1–14.

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

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

Ovacik, I.M., Uzsoy, R., 1995. Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. International Journal of Production Research 33, 3173–3192.

Ovacik, I.M., Uzsoy, R., 1996. Decomposition methods for scheduling semiconductor testing facilities. International Journal of Flexible Manufacturing Systems 8, 357–387.

Pearn, W.L., Chung, S.H., Chen, A.Y., Yang, M.H., 2004. A case study on the multistage IC final testing scheduling problem with reentry. International Journal of Production Economics 88, 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., Li, J.Q., 2018. An effective invasive weed optimization algorithm for scheduling semiconductor final testing problem. Swarm and Evolutionary Computation 38, 42–53.

Shen, L., Dauzère-Pérès, S., Neufeld, J.S., 2018. Solving the flexible job shop scheduling problem with sequence-dependent setup times. European Journal of Operational Research 265, 503–516.

Uzsoy, R., Lee, C.Y., 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, 47–60.

Uzsoy, R., Lee, C.Y., 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, 369–388.

Uzsoy, R., Lee, C.Y., 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, 44–55.

Uzsoy, R., Martin-Vega, L.A., Lee, C.Y., Leonard, P.A., 1991. Production scheduling algorithms for a semiconductor test facility. IEEE Transactions on Semiconductor Manufacturing 4, 270–280.

Wang, S., 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., Xu, Y., 2015. A hybrid estimation of distribution algorithm for the semiconductor final testing scheduling problem. Journal of Intelligent Manufacturing 26, 861–871.

Wu, J.Z., Chien, C.F., 2008. Modeling semiconductor testing job scheduling and dynamic testing machine configuration. Expert Systems with Applications 35, 485–496.

Wu, J.Z., Hao, X.C., Chien, C.F., Gen, M., 2012. A novel bi-vector encoding genetic algorithm for the simultaneous multiple resources scheduling problem. Journal of Intelligent Manufacturing 23, 2255–2270.

Xiong, H.H., Zhou, M., 1998. Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search. IEEE Transactions on Semiconductor Manufacturing 11, 384–393.

Zhang, Z., Zhang, M.T., Niu, S., Zheng, L., 2006. Capacity planning with reconfigurable kits in semiconductor test manufacturing. International Journal of Production Research 44, 2625–2644.

Zhang, Z., Zheng, L., Hou, F., Li, N., 2011. Semiconductor final test scheduling with Sarsa(λ, k) algorithm. European Journal of Operational Research 215, 446–458.

Zheng, X.L., Wang, L., 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., Wilhelm, W.E., 2006. Scheduling and lot sizing with sequence-dependent setup: A literature review. IIE Transactions 38, 987–1007.

Appendix A. MIP formulation

In this appendix, we present an MIP for SFTPS. 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.

Appendix A.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 A.10. 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 associated to a completion time variable $C_{j_i} \in \mathbb{R}_0^+$ as defined in Section 2. 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

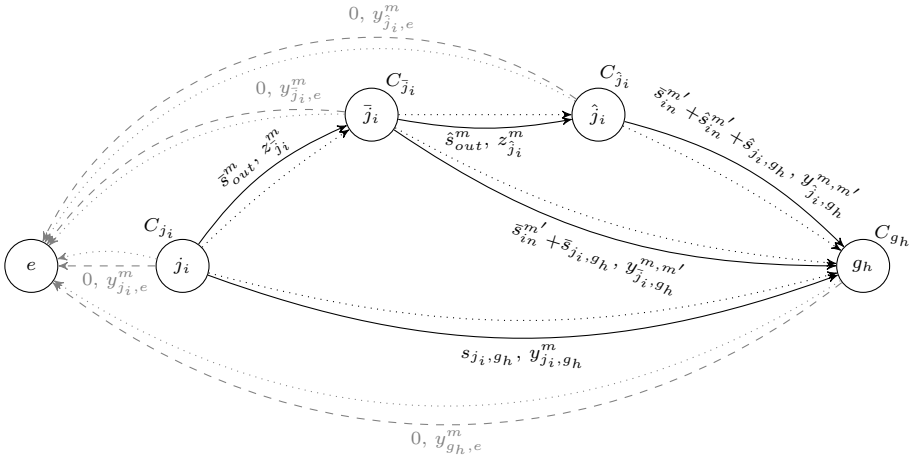

Figure A.10: Illustration of the ASAS graph, $j_i, g_h \in \hat{O}$, $j_i \neq g_h$

allocation, sequencing and setup decisions are represented by directed edges that are weighted with the setup times defined in Section 2 and that are associated with the following variables:

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

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

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

$$z^m_{\bar{j}_i} := \begin{cases} 1 & \text{if adapter } m[3] \text{ is disassembled after processing } j_i \text{ on } m, \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{A.4}$$

$$z^m_{\hat{j}_i} := \begin{cases} 1 & \text{if handler } m[2] \text{ is disassembled after processing } j_i \text{ on } m, \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}. \tag{A.5}$$

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 (dotted edges in Figure A.10). 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 (A.1)–(A.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 2, the tardiness of each job $j \in J$ is represented by the variable $T_j \in \mathbb{R}_0^+$.

Appendix A.2. Objective, processing time related constraints, and precedence constraints

The objective

$$\min \sum_{j \in J} w_j T_j \tag{A.6}$$

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

$$T_j \geq C_{j_{q_j}} - d_j \qquad \forall\, j \in J. \tag{A.7}$$

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:

$$C_{j_i} + s_{j_i,g_h} + p^m_{g_h} - C_{g_h} \leq (1 - y^m_{j_i,g_h})B \qquad \forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, m \in M_{j_i} \cap M_{g_h}, \tag{A.8}$$

$$C_{\bar{j}_i} + \bar{s}^{m'}_{in} + \bar{s}_{j_i,g_h} + p^{m'}_{g_h} - C_{g_h} \leq (1 - y^{m,m'}_{\bar{j}_i,g_h})B \qquad \forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, \; m \in M_{j_i}, m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}, \tag{A.9}$$

$$C_{\hat{j}_i} + \bar{s}^{m'}_{in} + \hat{s}^{m'}_{in} + \hat{s}_{j_i,g_h} + p^{m'}_{g_h} - C_{g_h} \leq (1 - y^{m,m'}_{\hat{j}_i,g_h})B \qquad \forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, \; m \in M_{j_i}, m' \in M_{g_h} \cap M^1_{m[1]}, \tag{A.10}$$

$$C_{j_i} + \bar{s}^m_{out} z^m_{\bar{j}_i} \leq C_{\bar{j}_i} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{A.11}$$

$$C_{\bar{j}_i} + \hat{s}^m_{out} z^m_{\hat{j}_i} \leq C_{\hat{j}_i} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}. \tag{A.12}$$

The restrictions

$$\begin{aligned} C_{j_i} \leq C_{j_{i+1}} &- \sum_{g_h \in \hat{O} \setminus \{j_{i+1}\}} \sum_{m \in M_{g_h}} \sum_{m' \in M_{j_{i+1}} \cap M^1_{m[1]}} y^{m,m'}_{\bar{g}_h,j_{i+1}} p^{m'}_{j_{i+1}} \\ &- \sum_{g_h \in \hat{O} \setminus \{j_{i+1}\}} \sum_{m \in M_{g_h}} \sum_{m' \in M_{j_{i+1}} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m,m'}_{\bar{g}_h,j_{i+1}} p^{m'}_{j_{i+1}} \\ &- \sum_{g_h \in \hat{O} \setminus \{j_{i+1}\}} \sum_{m \in M_{g_h} \cap M_{j_{i+1}}} y^m_{g_h,j_{i+1}} p^m_{j_{i+1}} \qquad \forall\, j_i \in O \text{ with } i \leq q_j - 1 \end{aligned} \tag{A.13}$$

enforce the precedence relations among the operations of each job.

Appendix A.3. Allocation and sequencing constraints

Constraints (A.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:

$$\sum_{j_i \in \hat{O} \setminus \{g_h\}} \sum_{m \in M_{j_i} \cap M_{g_h}} y^{m}_{j_i,g_h} + \sum_{j_i \in \hat{O} \setminus \{g_h\}} \sum_{m \in M_{j_i}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m,m'}_{\bar{j}_i,g_h} + \sum_{j_i \in \hat{O} \setminus \{g_h\}} \sum_{m \in M_{j_i}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m,m'}_{\hat{j}_i,g_h} = 1 \qquad \forall\, g_h \in O. \tag{A.14}$$

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 $\hat{j}_i$, $j_i \in \hat{O}$:

$$\sum_{m \in M_{j_i}} z^{m}_{\bar{j}_i} \leq 1 \qquad \forall\, j_i \in \hat{O}, \tag{A.15}$$

$$\sum_{m \in M_{j_i}} z^{m}_{\hat{j}_i} \leq 1 \qquad \forall\, j_i \in \hat{O}. \tag{A.16}$$

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:

$$\sum_{g_h \in O \setminus \{j_i\}} \sum_{m \in M_{j_i} \cap M_{g_h}} y^{m}_{j_i,g_h} + \sum_{m \in M_{j_i}} z^{m}_{\bar{j}_i} + \sum_{m \in M_{j_i}} y^{m}_{j_i,e} = 1 \qquad \forall\, j_i \in \hat{O}. \tag{A.17}$$

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

$$\sum_{g_h \in O \setminus \{j_i\}} \sum_{m \in M_{j_i}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m,m'}_{\bar{j}_i,g_h} + \sum_{m \in M_{j_i}} z^{m}_{\bar{j}_i} + \sum_{m \in M_{j_i}} y^{m}_{\bar{j}_i,e} \leq 1 \qquad \forall\, j_i \in \hat{O}, \tag{A.18}$$

$$\sum_{g_h \in O \setminus \{j_i\}} \sum_{m \in M_{j_i}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m,m'}_{\hat{j}_i,g_h} + \sum_{m \in M_{j_i}} y^{m}_{\hat{j}_i,e} \leq 1 \qquad \forall\, j_i \in \hat{O}. \tag{A.19}$$

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

$$\begin{aligned} \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in \{m\} \cap M_{g_h}} y^{m'}_{g_h,j_i} + \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m',m}_{\bar{g}_h,j_i} \\ + \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m',m}_{\hat{g}_h,j_i} \\ = \sum_{g_h \in O \setminus \{j_i\}} \sum_{m' \in \{m\} \cap M_{g_h}} y^{m'}_{j_i,g_h} + z^{m}_{\bar{j}_i} + y^{m}_{j_i,e} \qquad \forall\, j_i \in O, m \in M_{j_i}, \end{aligned} \tag{A.20}$$

$$z^{m}_{\bar{j}_i} = \sum_{g_h \in O \setminus \{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m,m'}_{\bar{j}_i,g_h} + z^{m}_{\hat{j}_i} + y^{m}_{\bar{j}_i,e} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{A.21}$$

$$z^{m}_{\hat{j}_i} = \sum_{g_h \in O \setminus \{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m,m'}_{\hat{j}_i,g_h} + y^{m}_{\hat{j}_i,e} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}. \tag{A.22}$$

Appendix A.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:

$$u^m_{j_i,t} := \begin{cases} 1 & \text{if the setup for processing } j_i \text{ on } m \text{ starts at time instant } t-1, \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T\}, \tag{A.23}$$

$$\bar{v}^{m,k}_{j_i,t} := \begin{cases} 1 & \text{if the use of resource class } m[k] \text{ of type } k \text{ in machine configuration } m \text{ for operation } j_i \text{ finishes at time instant } t, \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{0,\dots,T\}, k \in \{2,3\}, \tag{A.24}$$

$$x^{m,k}_{j_i,t} := \begin{cases} 1 & \text{if resource class } m[k] \text{ of type } k \text{ is being used in machine configuration } m \text{ for operation } j_i \text{ in time slot } [t-1,t], \\ 0 & \text{else,} \end{cases} \quad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\dots,T\}, k \in \{2,3\}. \tag{A.25}$$

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 (A.23), i.e., the start of the setups that precede the processing of operations, are handled by the following constraints:

$$\sum_{m \in M_{j_i}} \sum_{t \in \{1,\dots,T\}} u^m_{j_i,t} = 1 \qquad \forall\, j_i \in \hat{O}, \tag{A.26}$$

$$\sum_{m \in M_{0_i}} u^m_{0_i,1} = 1 \qquad \forall\, i \in \{1,\dots,r^1\}, \tag{A.27}$$

$$\begin{aligned} \sum_{m \in M_{j_i}} \sum_{t \in \{1,\dots,T\}} (t-1) u^m_{j_i,t} = C_{j_i} &- \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m \in M_{j_i} \cap M_{g_h}} y^m_{g_h,j_i} (s_{g_h,j_i} + p^m_{j_i}) \\ &- \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h}} \sum_{m \in M_{j_i} \cap M^1_{m'[1]} \cap M^2_{m'[2]}} y^{m',m}_{\bar{g}_h,j_i} (\bar{s}^m_{in} + \bar{s}_{g_h,j_i} + p^m_{j_i}) \\ &- \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h}} \sum_{m \in M_{j_i} \cap M^1_{m'[1]}} y^{m',m}_{\hat{g}_h,j_i} (\bar{s}^m_{in} + \hat{s}^m_{in} + \hat{s}_{g_h,j_i} + p^m_{j_i}) \qquad \forall\, j_i \in O, \end{aligned} \tag{A.28}$$

$$\begin{aligned} \sum_{t \in \{1,\dots,T\}} u^m_{j_i,t} \leq &\sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in \{m\} \cap M_{g_h}} y^{m'}_{g_h,j_i} + \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}} y^{m',m}_{\bar{g}_h,j_i} \\ &+ \sum_{g_h \in \hat{O} \setminus \{j_i\}} \sum_{m' \in M_{g_h} \cap M^1_{m[1]}} y^{m',m}_{\hat{g}_h,j_i} \qquad \forall\, j_i \in O, m \in M_{j_i}. \end{aligned} \tag{A.29}$$

Here, conditions (A.26) and (A.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 3 are taken account of in restrictions (A.28). Conditions (A.29) ensure that the machine configurations are consistent.

In line with the system of restrictions (A.26)–(A.29), the following seven sets of conditions handle the end of the usage of resources used for processing the operations:

$$\sum_{t \in \{0,\dots,T\}} \bar{v}^{m,k}_{j_i,t} = \sum_{t \in \{1,\dots,T\}} u^m_{j_i,t} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, k \in \{2,3\}, \tag{A.30}$$

$$\sum_{t\in\{0,\ldots,T\}} t\bar{v}^{m,k}_{j_i,t} - C_{g_h} + p^m_{g_h} + s_{j_i,g_h} \geq (y^m_{j_i,g_h} - 1)B \qquad \forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, \; m \in M_{j_i} \cap M_{g_h}, k \in \{2,3\}, \tag{A.31}$$

$$\bar{v}^{m,k}_{j_i,T} \geq y^m_{j_i,e} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, k \in \{2,3\}, \tag{A.32}$$

$$\bar{v}^{m,2}_{j_i,T} \geq y^m_{\bar{j}_i,e} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{A.33}$$

$$\sum_{t\in\{0,\ldots,T\}} t\bar{v}^{m,2}_{j_i,t} - C_{g_h} + p^{m'}_{g_h} + \bar{s}^{m'}_{in} + \bar{s}_{j_i,g_h} \geq (y^{m,m'}_{\bar{j}_i,g_h} - 1)B \quad \forall\, j_i \in \hat{O}, g_h \in O, j_i \neq g_h, m \in M_{j_i}, m' \in M_{g_h} \cap M^1_{m[1]} \cap M^2_{m[2]}, \tag{A.34}$$

$$\sum_{m\in M_{j_i}} \sum_{t\in\{0,\ldots,T\}} t\bar{v}^{m,3}_{j_i,t} \geq C_{\bar{j}_i} \qquad \forall\, j_i \in \hat{O}, \tag{A.35}$$

$$\sum_{m\in M_{j_i}} \sum_{t\in\{0,\ldots,T\}} t\bar{v}^{m,2}_{j_i,t} \geq C_{\hat{j}_i} \qquad \forall\, j_i \in \hat{O}. \tag{A.36}$$

Conditions (A.30) ensure that the use of some resource class must end if it has previously been setup. Depending on the allocation and sequencing decisions (Appendix A.1), constraints (A.31)–(A.36) then set the variables (A.24) to their correct values.

Now, based on the choice of the variables (A.23) and (A.24), conditions (A.37) fix the variables (A.25) that model the actual resource usage over all time slots of the planning horizon:

$$x^{m,k}_{j_i,t} - x^{m,k}_{j_i,t-1} = u^m_{j_i,t} - \bar{v}^{m,k}_{j_i,t-1} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\ldots,T+1\}, k \in \{2,3\}. \tag{A.37}$$

Finally, constraints (A.38) restrict the usage of the handler and adapter classes based on their availability:

$$\sum_{j_i\in\hat{O}} \sum_{m\in M_{j_i}\cap M^k_r} x^{m,k}_{j_i,t} \leq q^k_r \qquad \forall\, t \in \{1,\ldots,T\}, k \in \{2,3\}, r \in R^k. \tag{A.38}$$

Appendix A.5. Setup operator variables and constraints

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

$$\bar{u}^m_{j_i,t} := \begin{cases} 1 & \text{if the setup for processing } j_i \text{ on } m \text{ finishes at time instant } t, \\ 0 & \text{else,} \end{cases} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{0,\ldots,T\}, \tag{A.39}$$

$$v^{m,k}_{j_i,t} := \begin{cases} 1 & \text{if the procedure of disassembling resource class } m[k] \text{ of type } k \text{ from machine configuration } m \text{ after processing operation } j_i \text{ starts at time instant } t-1, \\ 0 & \text{else,} \end{cases} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\ldots,T\}, k \in \{2,3\}, \tag{A.40}$$

$$w^m_{j_i,t} := \begin{cases} 1 & \text{if a setup operator is required for the setup of machine configuration } m \text{ for operation } j_i \text{ in time slot } [t-1,t], \\ 0 & \text{else,} \end{cases} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\ldots,T\}, \tag{A.41}$$

$$\tilde{w}^{m,k}_{j_i,t} := \begin{cases} 1 & \text{if a setup operator is required for disassembling resource class } m[k] \text{ of type } k \text{ from machine configuration } m \text{ for operation } j_i \text{ in time slot } [t-1,t], \\ 0 & \text{else,} \end{cases} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\ldots,T\}, k \in \{2,3\}. \tag{A.42}$$

While the variables (A.39) model the end of the setup operations, variables (A.40) are used to identify time instants at which handlers and adapters are started to be disassembled. The variables (A.41) and (A.42) are used to indicate the need for setup operators for setup or disassembly operations. As in Appendix A.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 Appendix A.4. Conditions (A.43)–(A.45) relate to variables (A.39), while constraints (A.46)–(A.48) relate to variables (A.40):

$$\sum_{t\in\{0,\ldots,T\}} \bar{u}^m_{j_i,t} = \sum_{t\in\{1,\ldots,T\}} u^m_{j_i,t} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, \tag{A.43}$$

$$\sum_{m\in M_{0_i}} \bar{u}^m_{0_i,0} = 1 \qquad \forall\, i \in \{1,\ldots,r^1\}, \tag{A.44}$$

$$\begin{aligned}\sum_{m\in M_{j_i}} \sum_{t\in\{0,\ldots,T\}} t\bar{u}^m_{j_i,t} = C_{j_i} &- \sum_{g_h\in\hat{O}\setminus\{j_i\}} \sum_{m\in M_{j_i}\cap M_{g_h}} y^m_{g_h,j_i} p^m_{j_i} \\ &- \sum_{g_h\in\hat{O}\setminus\{j_i\}} \sum_{m'\in M_{g_h}} \sum_{m\in M_{j_i}\cap M^1_{m'[1]}\cap M^2_{m'[2]}} y^{m',m}_{\bar{g}_h,j_i} p^m_{j_i} \\ &- \sum_{g_h\in\hat{O}\setminus\{j_i\}} \sum_{m'\in M_{g_h}} \sum_{m\in M_{j_i}\cap M^1_{m'[1]}} y^{m',m}_{\hat{g}_h,j_i} p^m_{j_i} \qquad \forall\, j_i \in O,\end{aligned} \tag{A.45}$$

$$\sum_{t\in\{1,\ldots,T\}} v^{m,k}_{j_i,t} = \sum_{t\in\{1,\ldots,T\}} u^m_{j_i,t} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, k \in \{2,3\}, \tag{A.46}$$

$$\sum_{m\in M_{j_i}} \sum_{t\in\{1,\ldots,T\}} (t-1) v^{m,3}_{j_i,t} = C_{\bar{j}_i} - \sum_{m\in M_{j_i}} z^m_{\bar{j}_i} \bar{s}^m_{out} \qquad \forall\, j_i \in \hat{O}, \tag{A.47}$$

$$\sum_{m\in M_{j_i}} \sum_{t\in\{1,\ldots,T\}} (t-1) v^{m,2}_{j_i,t} = C_{\hat{j}_i} - \sum_{m\in M_{j_i}} z^m_{\hat{j}_i} \hat{s}^m_{out} \qquad \forall\, j_i \in \hat{O}. \tag{A.48}$$

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 Appendix A.4):

$$w^m_{j_i,t} - w^m_{j_i,t-1} = u^m_{j_i,t} - \bar{u}^m_{j_i,t-1} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\ldots,T+1\}, \tag{A.49}$$

$$\tilde{w}^{m,k}_{j_i,t} - \tilde{w}^{m,k}_{j_i,t-1} = v^{m,k}_{j_i,t} - \bar{v}^{m,k}_{j_i,t-1} \qquad \forall\, j_i \in \hat{O}, m \in M_{j_i}, t \in \{1,\ldots,T+1\}, k \in \{2,3\} \tag{A.50}$$

$$\sum_{j_i\in\hat{O}} \sum_{m\in M_{j_i}} (w^m_{j_i,t} + \tilde{w}^{m,2}_{j_i,t} + \tilde{w}^{m,3}_{j_i,t}) \le h \qquad \forall\, t \in \{1,\ldots,T\}. \tag{A.51}$$

Appendix B. Details on the performance of the heuristic approaches on small instances

Appendix B.1. Original sets small1 and small2

The computational results of CPLEX as well as the heuristics H-PR, H-TS, and H-SA on the instances of the set small1 and small2 are presented in Table B.17. The table presents the objective function values of the solutions returned by each run of the algorithms (columns 'WT', potentially indexed with the number of the run). For CPLEX, the table additionally includes the runtime (column 't').

Appendix B.2. Set small1 and small2 with all weights fixed to one

Table B.18 focusses on the results after having fixed all weights to one. It is in line with Table B.17

Table B.17: Performance of the heuristic approaches on the sets small1 and small2

	CPLEX		H-PR	H-TS			H-SA		
Instance	WT	t [s]	WT	WT_1	WT_2	WT_3	WT_1	WT_2	WT_3
small1-1	295	13.64	295	295	295	295	295	295	295
small1-2	674	3.45	674	674	674	674	674	674	674
small1-3	0	254.13	0	0	0	0	0	0	0
small1-4	71	10.1	71	71	71	71	71	71	71
small1-5	187	0.39	187	187	187	187	187	187	187
small1-6	175	11.09	260	175	175	175	175	175	175
small1-7	140	14.32	140	140	140	140	140	140	140
small1-8	260	0.75	316	260	260	260	260	260	260
small1-9	416	16.66	429	429	429	429	429	429	429
small1-10	360	276.29	425	408	408	408	408	408	408
small1-11	0	6.66	0	0	0	0	0	0	0
small1-12	358	12.51	643	358	358	358	358	358	358
small1-13	242	1215.91	248	242	242	242	248	242	242
small1-14	312	82.59	312	312	312	312	312	312	312
small1-15	72	33.84	72	72	72	72	72	72	72
small1-16	650	6.32	650	650	650	650	650	650	650
small1-17	89	6.87	89	89	89	89	89	89	89
small1-18	185	0.23	185	185	185	185	185	185	185
small1-19	4	0.53	4	4	4	4	4	4	4
small1-20	10	0.2	10	10	10	10	10	10	10
small2-1	1206	279.71	1206	1206	1206	1206	1206	1206	1206
small2-2	810	44.74	1356	1259	810	810	1259	1259	1259
small2-3	774	98.93	1522	1306	1306	1306	1306	1306	1306
small2-4	515	103.07	749	735	735	735	735	735	749
small2-5	495	542.91	495	495	495	495	495	495	495
small2-6	484	105.32	679	679	679	679	679	679	679
small2-7	250	182.83	364	250	250	250	250	250	250
small2-8	248	174.09	336	248	248	248	248	248	248
small2-9	136	164.84	404	188	188	188	136	136	136
small2-10	653	14.16	653	653	653	653	653	653	653
small2-11	448	61.61	448	448	448	448	448	448	448
small2-12	172	198.66	211	211	211	211	211	211	211
small2-13	54	212.28	54	54	54	54	54	54	54
small2-14	184	356.65	371	184	184	184	293	293	293
small2-15	358	965.1	815	358	358	358	358	358	368
small2-16	1832	352.56	1832	1832	1832	1832	1832	1832	1832
small2-17	724	55.11	1352	1352	1352	1352	1352	1352	1352
small2-18	786	448.49	1487	795	795	795	795	795	795
small2-19	293	111.72	294	293	293	293	293	293	293
small2-20	125	77.12	200	157	157	157	157	157	176

Table B.18: Performance of the heuristic approaches on sets small1 and small2 (all weights are fixed to one)

	CPLEX		H-PR	H-TS			H-SA		
Instance	WT	t [s]	WT	WT_1	WT_2	WT_3	WT_1	WT_2	WT_3
small1-1	79	5.04	83	79	79	79	79	79	79
small1-2	168	7.09	168	168	168	168	168	168	168
small1-3	0	259.39	0	0	0	0	0	0	0
small1-4	64	12.65	81	64	64	64	64	64	64
small1-5	187	0.38	187	187	187	187	187	187	187
small1-6	35	7.36	52	35	35	35	35	35	35
small1-7	70	13.55	70	70	70	70	70	70	70
small1-8	65	0.58	106	65	65	65	65	65	65
small1-9	189	5.02	259	189	189	189	189	189	189
small1-10	96	308.19	120	102	102	102	102	102	102
small1-11	0	3.65	0	0	0	0	0	0	0
small1-12	179	10.33	197	179	179	179	179	179	179
small1-13	121	571.15	135	121	121	121	135	121	135
small1-14	133	41.4	133	133	133	133	133	133	133
small1-15	36	40.32	36	36	36	36	36	36	36
small1-16	130	16.44	130	130	130	130	130	130	130
small1-17	89	6.9	89	89	89	89	89	89	89
small1-18	55	0.21	55	55	55	55	55	55	55
small1-19	2	0.52	2	2	2	2	2	2	2
small1-20	5	0.2	5	5	5	5	5	5	5
small2-1	523	268.06	523	523	523	523	523	523	523
small2-2	243	63.66	354	348	243	243	348	348	348
small2-3	226	280.36	388	266	266	266	266	266	266
small2-4	147	59.39	233	147	147	147	147	147	233
small2-5	99	282.95	99	99	99	99	99	99	99
small2-6	121	36.02	121	121	121	121	121	121	121
small2-7	250	182.68	364	250	250	250	250	250	250
small2-8	202	227.61	212	202	202	202	202	202	202
small2-9	34	405.95	143	47	47	47	47	34	34
small2-10	238	23.68	290	238	238	238	290	290	238
small2-11	112	27.51	112	112	112	112	112	112	112
small2-12	172	47.4	243	211	211	211	211	211	211
small2-13	45	331.65	45	45	45	45	45	45	45
small2-14	92	493.66	151	92	92	92	92	92	141
small2-15	177	300.79	244	179	179	179	179	179	177
small2-16	458	292.87	458	458	458	458	458	458	458
small2-17	293	253.82	475	293	293	293	404	293	293
small2-18	259	245.96	421	259	259	259	259	259	259
small2-19	221	191.24	239	221	221	221	221	221	221
small2-20	66	83.81	133	117	126	117	117	117	117