Filter-and-fan approaches for scheduling flexible job shops under workforce constraints

David Müller[a] and Dominik Kress[b,*]

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

ABSTRACT
This paper addresses a flexible job shop scheduling problem (FJSP) that takes account of workforce constraints and aims to minimise the makespan. The former constraints ensure that eligible workers that operate the machines and may be heterogeneously qualified, are assigned to the machines during the processing of operations. We refer to this problem as the worker constrained FJSP, and denote it by WFJSP. We develop different variants of filter-and-fan (F&F) based heuristic solution approaches that combine a local search procedure with a tree search procedure. The former procedure is used to obtain local optima, while the latter procedure generates compound transitions in order to explore larger neighbourhoods. In order to be able to adapt neighbourhood structures that have formerly shown to perform well when workforce restrictions are not considered, we decompose the problem into two components for decisions on machine allocation and sequencing and decisions on worker assignment, respectively. Based on this idea, we develop multiple definitions of neighbourhoods that are successively locked and unlocked during runtime of the F&F heuristics. In a computational study, we show that our solution approaches are competitive when compared with the use of a standard constraint programming solver and that they outperform state-of-the-art heuristic approaches on average.

KEYWORDS
Scheduling; Flexible job shop; Workforce constraints; Filter-and-fan; Constraint programming

1. Introduction

The *job shop scheduling problem* (JSP) is a well-known scheduling setting that has attracted a lot of attention in the literature (see, e.g., Blazewicz et al. 2019, for an overview). It arises in traditional manufacturing systems and is composed of a set of jobs and a set of machines. Each job consists of a set of operations that have to be processed in a predefined order to complete the job. Moreover, each operation is

* Corresponding author. Emails: `david.mueller@uni-siegen.de` (D. Müller), `dominik.kress@hsu-hh.de` (D. Kress)

associated to a machine that must be used for its processing as well as a corresponding processing time. A machine can process only one operation at a time and preemption of operations is not permitted. Given these restrictions, the problem is to sequence the operations on the machines so that all jobs are completed and some performance measure is optimised. When considering the minimisation of the makespan, the JSP is known to be strongly NP-hard (Lenstra and Rinnooy Kan 1979).

In the face of large product varieties, short product life cycles, and demand fluctuations, many manufacturing companies implement manufacturing systems that allow for a quick response to market changes. These companies oftentimes make use of multi-purpose machines that are able to process different types of operations (see, e.g., Jain et al. 2013; Beach et al. 2000). This is taken account of in a generalisation of the JSP, which is commonly referred to as the *flexible job shop scheduling problem* (FJSP). It was originally introduced by Brucker and Schlie (1990) and assumes that each operation is associated to a set of *eligible machines*, so that a feasible schedule must specify the machines that are used for processing the operations. FJSP settings are frequently applied in real-world manufacturing systems, e.g., in the production of cardan shaft mounts (Kress, Müller, and Nossack 2019) or in the semiconductor industry (Uzsoy, Lee, and Martin-Vega 1992). They are thus of major practical relevance. In addition, it is oftentimes necessary to take account of the fact that machines usually need to be operated by *workers* (potentially in an alternating manner). We will refer to FJSP settings that explicitly incorporate workers as *worker constrained FJSPs*, and denote them by WFJSPs. Note, however, that these settings are sometimes also referred to as dual-resource constrained systems (see, e.g., Xu, Xu, and Xie 2011; Treleven 1989). Naturally, as a result of the manufacturing flexibility induced by the use of multi-purpose machines, workers are oftentimes not qualified for operating all machines or processing all manufacturing operations, so that one has to take account of a heterogeneous workforce. In general, workforce heterogeneity is a relevant issue that has been addressed across various production systems (Katiraee et al. 2021; De Bruecker et al. 2015). Note that, on a strategic level, it requires companies to recognise the importance of training and accumulation of skills of workers (see, e.g., Altendorfer et al. 2020; Bokhorst and Gaalman 2009; Nembhard and Shafer 2008; Wirojanagud et al. 2007) as well as the potentials of collaboration of human workers and robots (see, e.g., Vieira et al. 2021).

Due to the fact that machine scheduling decisions are oftentimes embedded in rolling horizon based planning approaches or have to be made in online settings, computational time is strongly limited in most practically relevant industry cases. Companies therefore thrive for sophisticated heuristic planning approaches that are likely to quickly compute high quality solutions. Many researchers have therefore proposed diverse variants of heuristic approaches for WFJSPs. As we will show in detail in Section 2, most of the promising approaches in this area are population-based procedures. As pointed out by He, Chen, and Chen (2016), the success of these approaches oftentimes comes at the cost of intricate guiding strategies, complex parameter adjustments, and non-adaptive (with respect to instance size) stopping criteria, as, for example, predefined time limits or a maximum number of iterations. In order to develop competitive local search approaches that tackle these drawbacks, are easy to implement and, thus, attractive for practitioners, one must make use of strategies that allow to very effectively explore the solution space. Potential candidates to achieve this goal are *filter-and-fan* (F&F) methods. On their most general level (details are presented in Section 4), F&F approaches combine two fundamental search strategies that are applied in an alternating manner. A local search procedure is used to obtain local optima, while a tree search procedure generates compound transitions in order to explore larger neighbourhoods to overcome these locally optimal solutions and, thus, aims at effectively guiding the local search. Rego and Glover (2010) summarise the basic functionality and historical development of F&F approaches. They characterise F&F methods as multi-stream neighbourhood search strategies that date back to Glover (1998) and were extended by Rego and Glover (2002) as a specific method of creating efficient and robust combined neighbourhood search strategies. In this sense, they can be seen as a complement to ejection chain procedures (see also Glover 1996; Pesch and Glover 1997; Dorndorf, Jaehn, and Pesch 2008; Kress, Boysen, and Pesch 2017; Kress, Meiswinkel, and Pesch 2019). F&F approaches have shown to be advantageous to or competitive with population-based procedures for many optimisation problems, e.g., the facility location problem (Greistorfer and Rego 2006), the 2D HP model of the protein folding problem (Rego, Li, and Glover 2011), the capacitated minimum spanning tree problem (Rego and Mathew 2011), variants of vehicle-routing problems (Tarantilis, Stavropoulou, and Repoussis 2013; Yang and Tang 2010), and the resource-constrained project scheduling problem (He, Chen, and Chen 2016; Ranjbar 2008). Rego and Duarte (2009) present an extremely successful F&F method for the JSP. The success of their approach suggests adapting this idea to other shop scheduling settings, which is our main driver for developing F&F methods for WFJSPs. In this article, we specifically consider the objective of minimising the makespan and refer to our setting as *the* WFJSP for the sake of simplicity.

The remainder of this paper is structured as follows. In Section 2, we present an overview of the related literature. Next, in Section 3, we provide a formal definition of WFJSP, we introduce the concept of the solution graph, which we use to represent solutions of WFJSP, and we summarise a constraint programming (CP) formulation of Kress and Müller (2019) which will later be used for benchmarking. In Section 4, we describe our F&F approaches in detail. An extensive computational study is subject of Section 5. The paper closes with a summary in Section 6.

2. Literature overview and contribution

The incorporation of workforce related constraints and objectives into the planning process is the backbone of many modern production concepts (see, e.g., the recent overviews by Hashemi-Petroodi et al. 2020; Saadat et al. 2013). Examples include workforce agility (Hopp and Oyen 2004), seru production systems (Liu et al. 2013; Zhang et al. 2017), or the shojinka-principle of the Toyota production philosophy (Sennott, Van Oyen, and Iravani 2006; Monden 2011). Hence, workforce constraints have also become increasingly important in generalisations of classical scheduling settings as the FJSP (see, e.g., the survey by Chaudhry and Khan 2016). Specifically, for WFJSPs in their basic form as described above, many articles are devoted to makespan minimisation (see Table 1 for an overview). With respect to corresponding models and solution approaches, Xianzhou and Zhenhe (2011) and Peng et al. (2018) propose genetic algorithms. Both studies are restricted to the evaluation of the algorithms on a single small size test instance as part of a case study. A variable neighbourhood search is introduced by Lei and Guo (2014). It makes use of four neighbourhood structures. Two of them use classical swap and insertion techniques on operation sequences, the other two are concerned with the reallocation of workers and eligible machines. In a computational study based on literature instances for the FJSP, the authors show that their approach outperforms two modified genetic algorithms that were originally designed for the FJSP. Yazdani et al. (2015) propose a mixed-integer programming (MIP) model

Table 1. Literature overview: WFJSPs aiming at makespan minimisation

Publication	Approach
Xianzhou and Zhenhe (2011)	Genetic algorithm
Lei and Guo (2014)	Variable neighbourhood search
Yazdani et al. (2015)	Simulated annealing and vibration damping optimisation, MIP
Zhang, Wang, and Xu (2015)	Particle swarm optimisation
Zheng and Wang (2016)	Knowledge-guided fruit fly optimisation
Peng et al. (2018)	Genetic algorithm
Kress and Müller (2019)	MIP, CP
This article	F&F approach

and two metaheuristic approaches (simulated annealing and vibration damping optimisation). Using the MIP solver provided by CPLEX, they are able to solve small sized instances to optimality. The authors then show that the vibration damping optimisation approach outperforms the simulated annealing approach on randomly generated instances. Zhang, Wang, and Xu (2015) develop a hybrid discrete swarm optimisation algorithm by incorporating a simulated annealing approach with a variable neighbourhood structure into a particle swarm optimisation approach. In computational tests, they observe that this incorporation pays off when compared to a classical discrete swarm optimisation procedure. Zheng and Wang (2016) propose a knowledge-guided fruit fly optimisation algorithm. The authors extend the standard search techniques used in fruit fly optimisation algorithms, smell-based search and vision-based search, by incorporating a knowledge-guided search stage. In a computational study based on literature instances for the FJSP, they find that this approach outperforms a standard fruit fly optimisation algorithm as well as the variable neighbourhood search heuristic proposed by Lei and Guo (2014). Finally, Kress and Müller (2019) introduce a MIP as well as CP model for WFJSP and compare these models by using the standard solvers provided by IBM ILOG CPLEX. In a computational study, the authors show that the CP solver clearly outperforms the MIP solver for the considered modelling approaches.

Another stream of research on WFJSPs considers generalised problem settings, sometimes motivated from concrete industry applications (e.g., a quality control laboratory scheduling problem in Cunha et al. 2019), or alternative objective functions, either classical scheduling objectives that differ from makespan minimisation or mutliple objectives. As this stream is only indirectly related to the article at hand, we abstain from summarising the articles in detail, but rather present a compact overview of the most relevant articles in Table 2.

Based on Tables 1 and 2, we observe that the majority of heuristic solution approaches proposed in the literature are population-based procedures. In this article, we contribute to the first of the above streams (Table 1) by abstaining from making use of these procedures in order to tackle their aforementioned drawbacks. Consider, for instance, the knowledge-guided fruit fly optimisation approach by Zheng and Wang (2016), who find that their technique is 'more effective than the existing algorithms' summarised in Table 1. In order to tackle the complex parameter adjustments that are typically needed when using population-based procedures, the authors make use of the design-of-experiment method by Taguchi (see, e.g., Montgomery 2012). Only by doing so, they are able to analyse the interdependencies of the key parameters of their approach in order to select a promising setup. Additionally, besides the rather complex guiding strategy applied in the fruit fly approach, another drawback of population-based approaches shows in the stopping criterion applied by the authors, namely a fixed and non-adaptive value for the maximum number of iterations, which is, for ex-

Table 2. Literature overview: WFJSPs with objectives differing from pure makespan minimisation or under additional constraints

Publication	Objective	Approach
Lang and Li (2011)	Delivery satisfaction, process cost, energy consumption, and noise pollution	Genetic algorithm
Liu, Liu, and Tao (2011)	Makespan and production cost	Hybrid genetic algorithm
Zhang et al. (2013)	Makespan and production cost	Hybrid discrete particle swarm optimisation
Lei and Tan (2016)	Makespan and total tardiness	Local search
Paksi and Ma'ruf (2016)	Total tardiness	Genetic algorithm
Gong et al. (2018a)	Makespan, total worker cost and green production factors	Hybrid genetic algorithm
Gong et al. (2018b)	Makespan, maximum workload of machines and total workload of all machines	Memetic algorithm
Vallikavungal Devassia, Salazar-Aguilar, and Boyer (2018)[a]	Makespan	Variable neighbourhood search, MIP
Wu et al. (2018)[b]	Makespan	Hybrid genetic algorithm
Cunha et al. (2019)[c]	Makespan	MIP
Kress, Müller, and Nossack (2019)[d]	Makespan, Total tardiness	Branch-and-cut algorithm, MIP, Decomposition based heuristic approaches
Meng et al. (2019)	Energy consumption	Variable neighbourhood search, MIPs
Yang, Chung, and Lee (2019)[e]	Lateness, makespan and deviation of the workload among the machines	Local search
Yazdani, Zandieh, and Tavakkoli-Moghaddam (2019)	Makespan, critical machine workload and total workload of machines	Genetic algorithms
Andrade-Pineda et al. (2020)	Makespan and mean tardiness	Iterated greedy algorithm, MIP
Wu et al. (2020)[f]	Makespan and total setup time	Genetic algorithm
Zhu et al. (2020)[b]	Makespan, total carbon emission, and total cost of workers	Memetic algorithm

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

ample, also applied by Lei and Guo (2014). Usually, such complex and non-adaptive techniques are not attractive for practitioners. When focussing on F&F settings, we can address these drawbacks by making use of well-known search strategies (local search, tree search) and adaptive stopping criteria (e.g., improvement detection within local search) that result in methods that are easy to implement, to understand, and to adjust due to directly observable interdependencies of their parameters. We are thus able to make use of neighbourhood structures that have proven to be successful for the FJSP (see Mastrolilli and Gambardella 2000) based on a decomposition of WFJSP into two components for decisions on machine allocation and sequencing and decisions on worker assignment, respectively (see Kress, Müller, and Nossack 2019). The resulting methods aim to be competitive when compared with the use of the standard CP solver provided by CPLEX, that itself tends to outperform the state-of-the-art heuristic approaches listed in Table 1 in its ability to quickly determine high quality solutions (see Kress and Müller 2019).

3. Problem definition and representation of feasible solutions

In Section 3.1, we formally define the WFJSP. The notation introduced in this section is in line with the notation used in Kress, Müller, and Nossack (2019) and specifically targets a decomposition of WFJSP that we will use below (see Section 4) and that is inspired from the vehicle routing literature (for details, see Kress, Müller, and Nossack 2019). In order to be able to adapt the neighbourhood functions presented by Mastrolilli and Gambardella (2000), we augment the authors' concept of representing solutions by so called solution graphs in Section 3.2. In Section 3.3, we summarise the aforementioned CP formulation of Kress and Müller (2019).

3.1. *Problem description*

The WFJSP is defined as follows. Given is a set $I = \{I_1, \ldots, I_n\}$ of n *jobs*, a set $M = \{M_1, \ldots, M_u\}$ of u *machines*, and a set $W = \{W_1, \ldots, W_v\}$ of v *workers*. Each job $I_i \in I$ is associated with a set of q_i operations $O_i = \{i_1, \ldots, i_{q_i}\}$. The sets O_i are assumed to be linearly ordered for all $i \in \{1, \ldots, n\}$, which relates to the fact that for any pair of operations $i_j, i_{j'} \in O_i$ with $j < j'$, i_j must be completed before the processing of $i_{j'}$ may start. Each operation $i_j \in O_i$, $i \in \{1, \ldots, n\}$, must be processed on exactly one machine out of a non-empty set of *eligible machines* $M_{i_j} \subseteq M$. Moreover, an operation $i_j \in O_i$ of a job $I_i \in I$ can only be processed on a machine, if exactly one worker out of a non-empty set of *eligible workers* $W_{i_j} \subseteq W$ is assigned to the operation for the entire processing time. Processing times are assumed to depend on worker and machine assignments. The processing time of an operation $i_j \in O_i$ of a job $I_i \in I$ assigned to worker $W_w \in W_{i_j}$ on machine $M_m \in M_{i_j}$ is denoted by $p_{i_j}^{m,w} \in \mathbb{N}^+ \cup \{\infty\}$. For each job $I_i \in I$, each operation $i_j \in O_i$, and each eligible machine $M_m \in M_{i_j}$, we assume that there exists at least one eligible worker $W_w \in W_{i_j}$ with a finite processing time $p_{i_j}^{m,w}$. Similarly, for each eligible worker, we assume that there exists at least one eligible machine with a finite processing time. The completion time of an operation $i_j \in O_i$ of job $I_i \in I$ is denoted by C_{i_j}. The completion time of job $I_i \in I$ is denoted by C_i. A job is completed if all of its operations are completed. Hence, $C_i = C_{i_{q_i}}$ for all $i \in \{1, \ldots, n\}$.

We assume that all jobs, machines, and workers are available at time zero. The processing of operations may not be preempted. Furthermore, each machine and each worker can process at most one operation at a time. The problem is to find a schedule, i.e. an allocation of operations to machines and workers as well as corresponding sequences and starting times of the operations on the allocated machines and workers, such that the makespan $C_{max} = \max_{i \in \{1,\ldots,n\}} C_i$ is minimised subject to the above constraints. This problem is strongly NP-hard as it extends the JSP, which - as aforementioned - is strongly NP-hard when aiming to minimise the makespan. We restrict our attention to *left-justified* schedules (see, e.g., Sprecher, Kolisch, and Drexl 1995). That is, whenever considering feasible solutions in the remainder of this paper, we assume that each operation is started to be processed as early as possible when taking the allocation and sequencing decisions as given.

Our notation is summarised in Table 3.

3.2. *Solution graph*

Mastrolilli and Gambardella (2000) represent solutions of the FJSP with the so called

Table 3. Notation used throughout the paper

Notation	Definition	Further details
I	set of jobs	$I = \{I_1, \ldots, I_n\}$
M	set of machines	$M = \{M_1, \ldots, M_u\}$
W	set of workers	$W = \{W_1, \ldots, W_v\}$
O_i	set of operations of job $I_i \in I$	$O_i = \{i_1, \ldots, i_{q_i}\}$, $\lvert O_i \rvert = q_i$
M_{i_j}	set of eligible machines for operation $i_j \in O_i$ of job $I_i \in I$	$M_{i_j} \subseteq M$
W_{i_j}	set of eligible workers for operation $i_j \in O_i$ of job $I_i \in I$	$W_{i_j} \subseteq W$
$p_{i_j}^{m,w}$	processing time of operation $i_j \in O_i$ of job $I_i \in I$ when processed by worker $W_w \in W_{i_j}$ on machine $M_m \in M_{i_j}$	$p_{i_j}^{m,w} \in \mathbb{N}_0^+ \cup \{\infty\}$
C_{i_j}	completion time of operation $i_j \in O_i$ of job $I_i \in I$	
C_i	completion time of job $I_i \in I$	$C_i = C_{i_{q_i}}$
C_{max}	makespan of the schedule	$C_{max} = \max\limits_{i \in \{1,\ldots,n\}} C_i$

solution graph (see also Blazewicz et al. 2019), which – in the presence of workers as an additional resource – can be augmented in a straightforward manner:

- For all jobs $I_i \in I$, each operation $i_j \in O_i$ defines a vertex. We denote the resulting vertex set by V, i.e. $V = \bigcup_{i \in I} O_i$.
- Additional dummy vertices, denoted by 0_1 and $(n+1)_1$, represent the beginning and end of a schedule. We define $D = \{0_1\} \cup \{(n+1)_1\}$.
- Precedence relations among the operations of the jobs are represented by directed edges of the set A_1. For each job $I_i \in I$ and all pairs i_j, i_{j+1} with $j \in \{1, \ldots, q_i - 1\}$, A_1 includes the directed edge (i_j, i_{j+1}). Additionally, A_1 includes dummy edges $(0_1, i_1)$ and $(i_{q_i}, (n+1)_1)$ for all $i \in \{1, \ldots, n\}$. The elements of A_1 are referred to as *precedence edges.*
- Based on the given solution of the considered instance of WFJSP, the set A_2 of directed edges includes an edge (i_j, k_l), if and only if i_j is processed immediately before k_l on some machine $M_m \in M$. For each $M_m \in M$, A_2 additionally includes dummy edges from vertex 0_1 to the vertex that corresponds to the first operation that is processed on M_m and from the vertex that corresponds to the last operation that is processed on M_m to vertex $(n+1)_1$. For each machine M_m that processes no operation, A_2 includes a dummy edge from vertex 0_1 to vertex $(n+1)_1$. The elements of the set A_2 are referred to as *machine edges.*
- Similarly, the set A_3 includes a directed edge (i_j, k_l), if and only if i_j is processed immediately before k_l by some worker $W_w \in W$. Additional dummy edges are defined in line with their definition for machine edges. The elements of the set A_3 are referred to as *worker edges.*

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

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

Figure 1 illustrates the solution graph of a feasible solution for an example instance of WFJSP with three jobs, two machines, and two workers. The solid edges represent

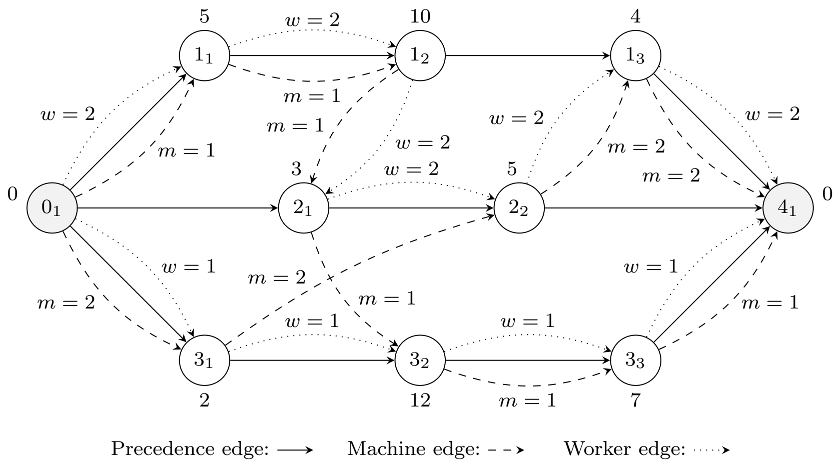


Figure 1. Exemplary illustration of the solution graph

the precedence edges, while the dashed edges and dotted edges represent the machine edges for machines M_m, $m \in \{1,2\}$, and worker edges for workers W_w, $w \in \{1,2\}$, respectively. Edge annotations highlight the corresponding machines or workers. Additionally, vertex weights correspond to the processing times in the specific solution. Hence, in the depicted solution, machine M_1 (M_2) processes operations 1_1, 1_2, 2_1, 3_2, and 3_3 (3_1, 2_2, and 1_3). Similarly, worker W_1 (W_2) processes operations 3_1, 3_2, and 3_3 (1_1, 1_2, 2_1, 2_2, and 1_3).

3.3. *Constraint programming formulation*

As outlined above, Kress and Müller (2019) present a CP formulation for WFJSP. It is based on variables and constraint types provided by the IBM ILOG CPLEX CP Optimizer (see Laborie et al. 2018; IBM 2016, for an introduction). The CP formulation uses *interval variables* to model the start and the end of the processing of the operations. *Sequence variables* represent the sequencing decisions, i.e. orderings of interval variables. An overview is given in Table 4.

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

Table 4. Variables for the CP model as introduced by Kress and Müller (2019)

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

$$\min \max_{i \in \{1,\ldots,n\}} (endOf(I_{OP\{i_{q_i}\}})) \tag{1}$$

s.t.

$$endBeforeStart(I_{OP\{i_j\}}, I_{OP\{i_{j+1}\}}) \quad \forall\, i \in \{1,\ldots,n\}, j \le q_i - 1, \tag{2}$$

$$alternative(I_{OP\{i_j\}}, all\, I_{MO\{i_j,m,w,p\}}) \quad \forall\, i \in \{1,\ldots,n\}, i_j \in O_i, \tag{3}$$

$$noOverlap(S_m) \quad \forall\, m \in \{1,\ldots,u\}, \tag{4}$$

$$noOverlap(S_w) \quad \forall\, w \in \{1,\ldots,v\}. \tag{5}$$

The objective function (1) represents the minimisation of the makespan. Constraints (2) capture the precedence constraints among the operations of the jobs. Constraints (3) guarantee that an eligible processing mode is chosen for each operation. Constraints (4) and (5) ensure that each machine and each worker processes at most one operation at a time.

4. Filter-and-fan approaches

As outlined above, F&F methods combine a local search procedure for obtaining local optima and a tree search procedure that is applied to explore larger neighbourhoods. The tree search procedure essentially corresponds to a beam search approach that generates multiple paths using a breadth-first search strategy. This is illustrated in Figure 2.

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

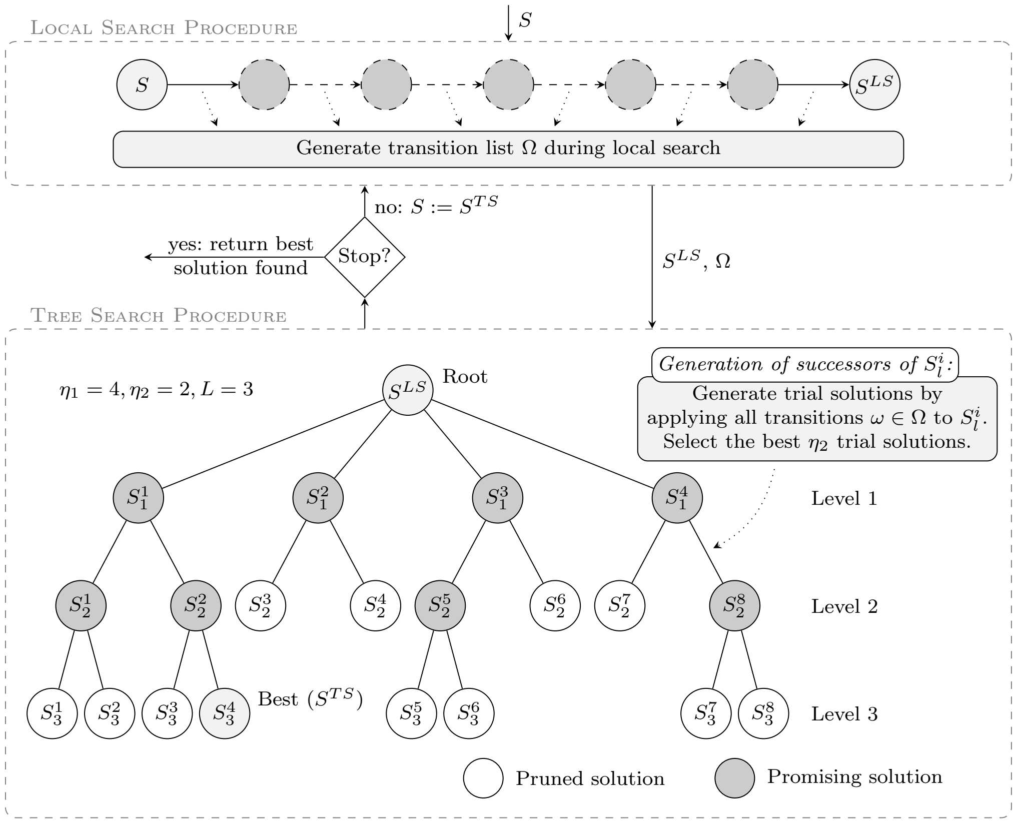


Figure 2. Alternating structure of F&F approaches

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

4.1. *Neighbourhood structure*

Mastrolilli and Gambardella (2000) construct a neighbour of a given solution of FJSP by *moving* an operation, i.e. deleting it from its current machine sequence and inserting it in some other feasible position on a corresponding (not necessarily different) eligible machine. By making use of the concept of solution graphs (see Section 3.2), they show that the resulting set of potential neighbours of a solution can be reduced to a specific subset that is guaranteed to include a neighbour with the lowest makespan. Unfortunately, this result does not immediately carry over to the case of the WFJSP because the processing times (and therefore the vertex weights of the solution graph) in the latter problem depend on both the machine and worker allocation. Nevertheless, we make use of this method as it guarantees the construction of feasible solutions, i.e. solutions with acyclic solution graphs, when adapted appropriately.

In order to handle the interdependencies between machine and worker allocations, we follow the main ideas of the hierarchical (decomposition based) approach introduced by Kress, Müller, and Nossack (2019) for a WFJSP with sequence-dependent setup times. In a first step, their approach solely takes account of the allocation of operations to eligible machines and the sequencing of these operations on the machines. The second step then determines a corresponding assignment of operations to eligible workers as well as the sequences of these operations for each worker. Given the solution graph G of some feasible solution of WFJSP and an operation a_b that we want to move, we adapt the underlying hierarchical idea as shown in Algorithm 1, where we first delete either all machine edges or all worker edges of the graph (construction of G' in line 2). We are left with a graph that solely considers one of the two resources, so

Algorithm 1. Generate a set of neighbouring solution graphs

```
   Input: Solution graph $G = (V \cup D, A_1 \cup A_2 \cup A_3, \mu)$ with starting and tail times, operation $a_b \in V$
   Output: Set $N$ of neighbouring solution graphs, including starting and tail times
1  Initialise $N := \emptyset$;
2  Determine $\gamma \in \{2, 3\}$ (Algorithm 2). Set $G' := (V \cup D, A_1 \cup A_\gamma, \mu)$. The starting times and tail times
     of the vertices of $G'$ are set to the ones of the vertices of $G$;
3  forall $M_m \in M_{a_b}$ (in case of $\gamma = 2$) or $W_w \in W_{a_b}$ (in case of $\gamma = 3$) do
4      Apply an adapted version of the procedure presented by Mastrolilli and Gambardella (2000) on
         $G'$ to determine a set of neighbouring solution graphs resulting from moving $a_b$ to machine $M_m$
         (worker $W_w$) and select a most promising candidate $\hat{G} = (V \cup D, A_1 \cup \hat{A}_\gamma, \mu)$ (Algorithm 3);
5      Recompute (feasible) set of machine edges (in case of $\gamma = 3$) or worker edges (in case of $\gamma = 2$)
         and add it to $\hat{G}$ (Algorithm 4). While doing so, redefine vertex weights $\mu$ and compute starting
         times and tail times of the vertices of $\hat{G}$ based on the corresponding worker and machine
         assignment;
6      Set $N := N \cup \{\hat{G}\}$;
7  end
```

that we can directly apply the ideas of Mastrolilli and Gambardella (2000) in order to construct feasible (with respect to the resource that has not been deleted as well as the precedence constraints) neighbours and select a promising candidate (line 4). Finally, we recompute a feasible allocation and sequencing decision for the remaining resource and update the solution graph accordingly (line 5). This routine is executed for all eligible machines or workers of operation a_b (line 3). It is important to note that line 4 of Algorithm 1 is based on the vertex weights of the input graph, even though one resource is neglected. Nevertheless, the feasibility of the solutions that correspond to the graphs constructed in line 5 is implied by the feasibility results presented by Mastrolilli and Gambardella (2000) because the assignment of workers to operations given an allocation and sequencing decision for the machines (and similarly an assignment of machines to operations given the worker decisions) will only cause temporal shifts of the operations on the machines (or in the worker sequences). Details of Algorithm 1 are given in the following sections.

4.1.1. Determine set of edges to be deleted

In order to determine the set of edges that is deleted from G in line 2 of Algorithm 1, we analyse the completion times of the predecessor operations of the input operation a_b with respect to the edge sets A_2 and A_3, i.e. operations $P_2(a_b)$ and $P_3(a_b)$, as stated in Algorithm 2. The basic idea is to determine the resource that has the strongest effect

Algorithm 2. Determine γ

```
   Input: Solution graph $G = (V \cup D, A_1 \cup A_2 \cup A_3, \mu)$ with starting and tail times, operation $a_b \in V$
   Output: Integer $\gamma$
1  if $s_{P_2(a_b)} + \mu(P_2(a_b)) > s_{P_3(a_b)} + \mu(P_3(a_b))$ then $\gamma = 2$;
2  else if $s_{P_2(a_b)} + \mu(P_2(a_b)) < s_{P_3(a_b)} + \mu(P_3(a_b))$ then $\gamma = 3$;
3  else randomly select $\gamma \in \{2, 3\}$;
```

on the 'delayed' start of a_b due to the sequencing decisions and later delete the edge set that corresponds to the other resource.

Consider an exemplary feasible solution of WFJSP as illustrated in Figure 3 as a Gantt chart ($m \in \{1, 2, 3\}$ refers to the index of machine M_m). In case of $a_b = 2_2$

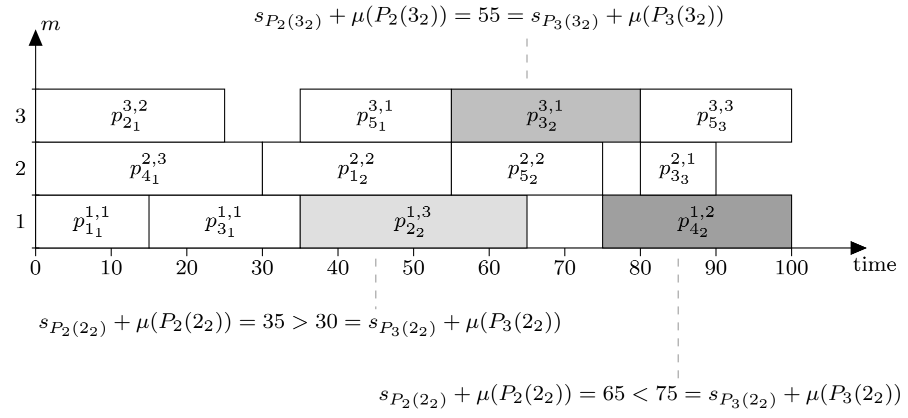


Figure 3. Exemplary solution of an instance of WFJSP

(processed on machine M_1 by worker W_3) we have $P_2(a_b) = 3_1$ and $P_3(a_b) = 4_1$, so that Algorithm 2 will return $\gamma = 2$. Similarly, for $a_b = 4_2$, the algorithm will return $\gamma = 3$. In case of $a_b = 3_2$, the algorithm will randomly determine $\gamma \in \{2, 3\}$.

4.1.2. Determine neighbouring solution graph

After having deleted either all machine (if $\gamma = 3$) or all worker (if $\gamma = 2$) edges from G in line 2 of Algorithm 1, we are left with a graph $G' = (V \cup D, A_1 \cup A_\gamma, \mu)$. Given operation a_b and some machine $M_m \in M_{a_b}$ (if $\gamma = 2$) or worker $W_w \in W_{a_b}$ (if $\gamma = 3$) as selected in line 3 of Algorithm 1, our adaption of the procedure presented by Mastrolilli and Gambardella (2000) is presented in Algorithm 3. For the sake of notational convenience, we denote the index of the resource under evaluation in the algorithm by θ, i.e. we set $\theta = m$ if $\gamma = 2$ and $\theta = w$ if $\gamma = 3$. The algorithm starts by deleting a_b from its current machine (worker) sequence in lines 1–2. The starting time and tail time of a_b is updated accordingly (line 3). Next, in lines 4–12, the algorithm checks potential trial moves of operation a_b to the operation sequence of the resource indexed by θ. Denote the set of operations that is processed by the resource with index θ by Q_θ (note that this will not include a_b, as this operation has been removed from its sequence) and assume that the elements of this set are ordered in non-decreasing order of their starting times. The algorithm computes the sets $L_\theta = \{i_j \in Q_\theta \mid \mu(i_j) + q_{i_j} > q_{a_b}\}$ and $R_\theta = \{i_j \in Q_\theta \mid s_{i_j} + \mu(i_j) > s_{a_b}\}$ (line 4). As shown by Mastrolilli and Gambardella (2000) for the FJSP, all insertions of a_b after the operations of the set $L_\theta \setminus R_\theta$ and before the operations of the set $R_\theta \setminus L_\theta$ result in feasible solutions. As indicated above, this result immediately carries over to the case of the WFJSP, so that Algorithm 3 restricts the construction of potential neighbours to these insertions. Let $\xi_\theta = Q_\theta \setminus (L_\theta \setminus R_\theta) \setminus (R_\theta \setminus L_\theta)$ and assume that the elements of this set are ordered in non-decreasing order of their starting times (see Figure 4). Furthermore, denote the i-th element of this set by $\xi_\theta[i]$ and define $\xi_\theta[|\xi_\theta| + 1]$ to be an operation with smallest starting time in the set $R_\theta \setminus L_\theta$ or, if this set is empty, the dummy operation $(n + 1)_1$. The algorithm approximates the length of the longest paths from 0_1 to $(n+1)_1$ that include operation a_b and that result from inserting a_b at

Algorithm 3. Determine neighbouring solution graph

```
Input: Solution graph $G' := (V \cup D, A_1 \cup A_\gamma, \mu)$ with starting and tail times (as determined in line 2
       of Algorithm 1), operation $a_b \in V$, eligible machine $M_m \in M_{a_b}$ or worker $W_w \in W_{a_b}$ (index
       denoted by $\theta$)
Output: Solution graph $\hat{G} = (V \cup D, A_1 \cup \hat{A}_\gamma, \mu)$
1  Initialise $\hat{A}_\gamma := A_\gamma$ and $\hat{G} := (V \cup D, A_1 \cup \hat{A}_\gamma, \mu)$ with starting and tail times identical to the ones of
     $G'$ (all following deletion, adding, and updating operations are performed on $\hat{G}$);
2  Delete $(P_\gamma(a_b), a_b)$ and $(a_b, S_\gamma(a_b))$ from $\hat{A}_\gamma$ and add $(P_\gamma(a_b), S_\gamma(a_b))$ to $\hat{A}_\gamma$;
3  Set $s_{a_b} := s_{P_1}(a_b) + \mu(P_1(a_b))$ and $q_{a_b} := \mu(S_1(a_b)) + q_{S_1}(a_b)$;
4  Compute $L_\theta$ and $R_\theta$;
5  Initialise $LP^* := \infty$ and $x^* := 0$;
6  forall potential positions $x \in \{0, \ldots, |\xi_\theta|\}$ after $L_\theta \setminus R_\theta$ and before $R_\theta \setminus L_\theta$ do
7      Compute $LP(a_b, \theta, x)$;
8      if $LP(a_b, \theta, x) < LP^*$ then
9          $LP^* = LP(a_b, \theta, x)$;
10         $x^* = x$;
11     end
12 end
13 if $x^* = 0$ then
14     Delete $(P_\gamma(\xi_\theta[1]), \xi_\theta[1])$ from $\hat{A}_\gamma$;
15     Add $(P_\gamma(\xi_\theta[1]), a_b)$ and $(a_b, \xi_\theta[1])$ to $\hat{A}_\gamma$;
16 end
17 else if $x^* = |\xi_\theta|$ then
18     Delete $(\xi_\theta[x^*], S_\gamma(\xi_\theta[x^*]))$ from $\hat{A}_\gamma$;
19     Add $(\xi_\theta[x^*], a_b)$ and $(a_b, S_\gamma(\xi_\theta[x^*]))$ to $\hat{A}_\gamma$;
20 end
21 else
22     Delete $(\xi_\theta[x^*], \xi_\theta[x^* + 1])$ from $\hat{A}_\gamma$;
23     Add $(\xi_\theta[x^*], a_b)$ and $(a_b, \xi_\theta[x^* + 1])$ to $\hat{A}_\gamma$;
24 end
```

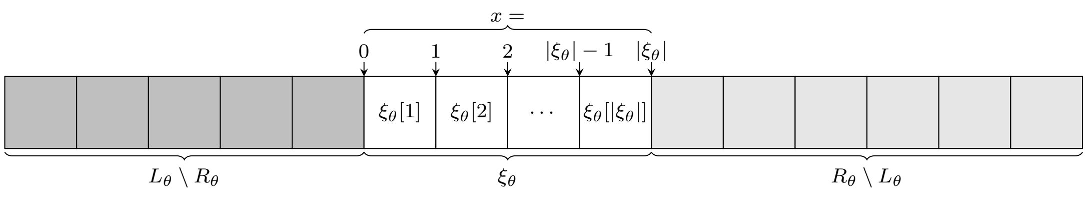


Figure 4. Illustration of Q_θ

positions $x = 0$ (immediately before the first operation of ξ_θ) to $x = |\xi_\theta|$ (immediately after the last operation of ξ_θ) as follows:

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

Here, $p_{i_j}^{m,\min} = \min_{W_w \in W_{i_j}} p_{i_j}^{m,w}$ for all $i_j \in V$ and $M_m \in M_{i_j}$. Similarly $p_{i_j}^{w,\min} = \min_{M_m \in M_{i_j}} p_{i_j}^{m,w}$ for all $i_j \in V$ and $W_w \in W_{i_j}$. Algorithm 3 selects a position x^* that results in the shortest approximate length (lines 5–12) and terminates after having updated the corresponding solution graph $\hat{G}$ (lines 13–24).

4.1.3. Recompute missing edge set

Once a neighbouring solution graph $\hat{G}$ has been selected in line 4 of Algorithm 1, we are left with having to recompute the edge set that has previously been deleted. To do so, we follow a simple greedy approach that is illustrated in Algorithm 4. Given the

Algorithm 4. Recompute missing edge set

```
Input: Solution graph $\hat{G} = (V \cup D, A_1 \cup \hat{A}_\gamma, \mu)$
Output: Modified solution graph $\hat{G}$ with starting times and tail times, values $C_{max}$ and $\bar{C}_{max}$
1 Initialise $\hat{A}_{\bar{\gamma}} := \emptyset$, where $\bar{\gamma} \in \{2,3\} \setminus \gamma$;
2 Add edge set $\hat{A}_{\bar{\gamma}}$ to $\hat{G}$ (all following adding and updating operations are performed on $\hat{G}$);
3 Set $\mu(i_j) := p_{i_j}^{\theta(i_j),\min}$, where $\theta(i_j)$ refers to the index of the resource (specific machine index, if
    $\gamma = 2$, or worker index, if $\gamma = 3$) that operation $i_j$ is currently allocated to, for all $i_j \in V$ in solution
    graph $\hat{G}$;
4 Recompute starting times $s_{i_j}$ of vertices $i_j \in V \cup D$ in $\hat{G}$ (see Section 3.2) and set $\bar{C}_{max} := s_{(n+1)_1}$;
5 Assign an eligible worker $W_w \in W_{i_j}$ (if $\gamma = 2$) or machine $M_m \in M_{i_j}$ (if $\gamma = 3$) to each operation
    $i_j \in V$ in non-decreasing order of the starting times determined in line 4 in a greedy manner. While
    doing so, update the corresponding vertex weight $\mu(i_j)$ and starting time $s_{i_j}$. Additionally, define a
    corresponding worker or machine edge and add it to $\hat{A}_{\bar{\gamma}}$;
6 Recompute tail times $q_{i_j}$ of vertices $i_j \in V \cup D$ in $\hat{G}$ (see Section 3.2) and set $C_{max} := s_{(n+1)_1}$;
```

machine (in case of $\gamma = 2$) or worker allocation (in case of $\gamma = 3$) of the input solution graph $\hat{G}$, it first updates the weights of the vertices $i_j \in V$ to the values $p_{i_j}^{\theta(i_j),\min}$ (line 3), where $\theta(i_j)$ refers to the index of the specific resource (a machine index, if $\gamma = 2$, or a worker index, if $\gamma = 3$) that operation i_j is currently allocated to. The starting times of the vertices are recomputed accordingly (line 4). The vertices $i_j \in V$ are then traversed in non-decreasing order of these starting times and are allocated to an eligible resource (a worker, if $\gamma = 2$, or a machine, if $\gamma = 3$) in a greedy manner (line 5). That is, the resource is chosen such that the resulting completion time of the corresponding operation is as small as possible, given all previous resource allocation decisions. Based on the resource allocation, the vertex weight and starting time of the corresponding operation may change, so that these values are updated to their correct values. The worker or machine edge that corresponds to the allocation decision is added to the edge set of the solution graph. Finally, we are left with having to compute the tail times of the vertices of the solution graph (line 6).

4.1.4. Neighbourhood definitions

Based on the above deliberations, we define three different neighbourhoods of some solution graph G. The first neighbourhood, denoted by $N_1(G)$, corresponds to the union of the sets returned by Algorithm 1 when being called for all critical operations of G. Similarly, $N_2(G)$ is constructed by calling the algorithm for all non-critical operations. Finally, we define a neighbourhood $N_3(G)$ that is somewhat similar to a neighbourhood generated by traditional swap moves. It is constructed by calling a modification of Algorithm 1 on all critical operations $a_b \in V$ of G. This modification executes the loop of lines 3–7 for all eligible machines or workers that are not identical to the one (labelled with resource index θ) that processes the input operation a_b in the input solution graph. After having executed line 5 and thus (potentially) having moved a_b to some other resource (labelled with resource index θ'), the modified algorithm once more deletes all machine or worker edges as in line 2, i.e. without recomputing γ, and then constructs all neighbouring solution graphs that result from moving some operation $k_l \neq a_b$ on the resource with index θ' to the resource with index θ (if eligible) in analogy to lines 4 and 5. All these graphs are then added to the set N as in line 6 of Algorithm 1.

4.2. *Details of the filter-and-fan algorithms*

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

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

4.2.1. Constructive procedure

In line with our deliberations in Section 4.1, we make use of the hierarchical approach of Kress, Müller, and Nossack (2019) in order to construct a first feasible solution S (and its solution graph) of a given instance of WFJSP. Hence, we first allocate all operations to eligible machines and make the corresponding sequencing decisions without considering the workers. Here, we apply a priority-rule based heuristic proposed by Kress, Müller, and Nossack (2019) (see therein for details; setup times can easily be neglected) that follows an algorithmic idea of Giffler and Thompson (1960) for the classical JSP. Basically, this heuristic iteratively allocates operations that can start being processed at the respective point of time with respect to all corresponding precedence constraints. Among all operations that compete for the same machine in some iteration, exactly one operation is chosen based on the *most work remaining* (MWKR) priority rule. Given the corresponding solution graph that solely includes precedence and machine edges, our constructive procedure then proceeds in a greedy manner as in Algorithm 4 in order to generate the missing worker edges.

4.2.2. Transition list

The transition list Ω generated within the local search procedure is a crucial component of any F&F approach. As defined above, it contains information regarding the ‘best’

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

We define a transition ω to include information on the operation $a_b \in V$ that serves as an input of Algorithm 1 when generating a neighbouring solution, the corresponding value of γ determined in line 2 and the index θ of the resource that the operation a_b is moved to in line 4, a boolean value *swap* that indicates whether or not the transition refers to a solution generated when using neighbourhood definition N_3 (one for yes, zero for no), as well as the objective function values C_{max} and $\bar{C}_{max}$ returned by the final call of Algorithm 4 in the course of generating the solution. Thus, we denote a transition ω by a tuple $(a_b, \gamma, \theta, swap, C_{max}, \bar{C}_{max})$. The value C_{max} defines the quality of the transition. $\bar{C}_{max}$ is used as a tie-breaker when comparing transitions with identical C_{max}.

In order to apply a transition ω to a solution within the tree search procedure, we call a modified version of Algorithm 1 with input operation a_b, where γ is fixed to the given value and where the loop 3–7 is executed solely for the resource with index θ. In case of $swap = 1$, we additionally incorporate the modifications highlighted in Section 4.1.4, where k_l is additionally fixed to the operation with the smallest starting time that is processed on the resource with index θ' and is eligible to be processed on the resource with index θ. If no such operation exists, the transition is considered non-applicable. As in case of transitions, the corresponding values C_{max} and $\bar{C}_{max}$ are used as a quality measure of the solution.

4.2.3. Local search

Our local search approach uses a best-fit strategy with a predefined neighbourhood operator $\phi \in \{1, 2, 3\}$ (corresponding to neighbourhood N_ϕ) on an input solution S. The transition list Ω of length η_0 (or less, if less neighbours have been evaluated) is generated during runtime of the procedure as illustrated in Algorithm 5.

Algorithm 5. Local search procedure

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

4.2.4. Tree search procedure

Our tree search procedure is outlined in Algorithm 6. It follows the main principles of tree search procedures within F&F approaches described above and uses a neigh-

Algorithm 6. Tree search procedure

```
Input: Solution $S^{LS}$, transition list $\Omega$, parameters $\eta_0$, $\eta_1$, $\eta_2$, $L$, and $\phi$
Output: Solution $S^{TS}$
▷ Create first level of the tree ($l = 1$)
1  Initialise $l := 1$ and an empty tabu list;
2  Construct solutions of the first level of the tree by applying (if applicable) the $\eta_1$ best (or all, if there
     exists less than $\eta_1$) transitions included in $\Omega$ to $S^{LS}$. Mark all of these solutions;
3  If no solution was generated in line 1, call Algorithm 7 to generate at most $\eta_1$ alternative solutions
     (and update $\Omega$ accordingly). Mark all of these solutions. If no solution is generated, terminate the
     procedure and return $S^{TS} := S^{LS}$;
4  Initialise $S^{TS}$ with the best solution generated in lines 2 and 3;
5  if $\ddot{S}^{TS} < \ddot{S}^{LS}$ then terminate the procedure;
▷ Create further levels of the tree ($1 < l \leq L$)
6  forall marked solutions on the current level $l$ do
7      Apply all transitions in $\Omega$ (if applicable) to obtain potential trial solutions for the next level $l + 1$.
         Among these solutions, discard all but the best $\eta_2$ candidates, the transitions of which are
         non-tabu or the makespan of which is smaller than $\ddot{S}^{LS}$ (aspiration criterion). If there are less
         than $\eta_2$ corresponding candidates, call Algorithm 7 to generate additional solutions (and update
         $\Omega$ accordingly), until a total of at most $\eta_2$ solutions has been constructed;
8  end
9  Initialise $S^*_{l+1}$ with the best solution generated in loop 6–8. If no solution has been generated,
     terminate the procedure;
10 if $\ddot{S}^*_{l+1} < \ddot{S}^{TS}$ then set $S^{TS} := S^*_{l+1}$;
11 if $\ddot{S}^{TS} < \ddot{S}^{LS}$ then terminate the procedure;
12 Mark the best $\eta_1$ (or all, if there exists less than $\eta_1$) trial solutions on level $l + 1$;
13 Update the tabu list;
14 Delete the worst $\max\{0, |\Omega| - \eta_0\}$ transitions from $\Omega$;
15 if $l < L$ then set $l := l + 1$ and go to line 6;
16 else exit the procedure;
```

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

Algorithm 6 includes details on how to handle situations where less than η_1 or η_2 transitions or solutions are available in the corresponding steps of a F&F approach. The algorithm, for instance, calls Algorithm 7 in lines 3 and 7 to potentially generate additional solutions as well as the corresponding transitions based on alternative neighbourhood operators. Note that Algorithm 7 potentially alters the transition list Ω, so that Algorithm 6 updates this list in line 14 in order to balance the computational effort and to only keep the most promising transitions.

Algorithm 7. Generate alternative neighbours

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

4.2.5. Filter-and-fan framework

Our overall F&F framework including neighbourhood switching is presented in Algorithm 8. It consists of two phases, the *initialisation phase* and the *filter-and-fan procedure*. The former phase makes use of the constructive procedure described in Section 4.2.1 (line 1) to determine a feasible solution. It furthermore unlocks all neighbourhood operators (line 2) in a specific order. It is important to note that the corresponding get-procedure (line 3) will always consider this ordering. An operator is locked, if a call of the tree search procedure does not improve its input solution (line 9). Similarly, all operators are unlocked, whenever the overall best solution S^* is improved (lines 5 and 8). The filter-and-fan procedure calls the local search procedure (line 4) and the tree search procedure (line 7) in an alternating manner. This process is repeated until all neighbourhood operators are locked (line 10). If the transition list Ω includes less than η_0 elements before executing a tree search, the framework calls Algorithm 7 to potentially determine more transitions.

Algorithm 8. Filter-and-fan framework

```
Input: Instance $\mathcal{I}$ of WFJSP, parameters $\eta_0$, $\eta_1$, $\eta_2$, and $L$
Output: Solution $S^*$
   ▷ Initialisation phase
1  Determine a feasible solution $S$ of $\mathcal{I}$ with the constructive procedure described in Section 4.2.1 and
     initialise $S^* := S$;
2  Unlock all neighbourhood operators in the specific order $\{1, 3, 2\}$;
   ▷ Filter-and-fan procedure
3  Get the first unlocked neighbourhood operator $\phi$;
4  Determine $S^{LS}$ and $\Omega$ by calling Algorithm 5 on solution $S$ with operator $\phi$. ;    ▷ Local search
5  if $\ddot{S}^{LS} < \ddot{S}^*$ then set $S^* := S^{LS}$ and unlock all neighbourhood operators in the specific order
     $\{1, 3, 2\}$;
6  if $|\Omega| < \eta_0$ then modify $\Omega$ by calling Algorithm 7 on $S^{LS}$ with $\lambda := \eta_0 - |\Omega|$;
7  Determine $S^{TS}$ by calling Algorithm 6 on $S^{LS}$ with transition list $\Omega$ and operator $\phi$. ; ▷ Tree search
8  if $\ddot{S}^{TS} < \ddot{S}^*$ then set $S^* := S^{TS}$ and unlock all neighbourhood operators in the specific order
     $\{1, 3, 2\}$;
9  else if $\ddot{S}^{TS} \geq \ddot{S}^{LS}$ then lock current neighbourhood operator $\phi$;
10 if there is at least one unlocked neighbourhood operator then set $S := S^{TS}$ and go to line 3;
11 else exit the procedure;
```

4.2.6. A modified filter-and-fan procedure

As described above, F&F procedures rely on transition lists that are generated during calls of a local search procedure and that are later globally applied in a tree search procedure. We will additionally consider a variant of our F&F framework that makes local decisions within the tree search procedure. Our corresponding modifications are as follows. Algorithm 5 (local search) does not compute a transition list. It solely returns a local optimum. Therefore, in Algorithm 6 (tree search), whenever generating the successors of some solution S with solution graph G in the tree, we randomly select η_1 (adaption of line 2) or η_2 (adaption of line 7) solutions from the neighbourhood $N_1(G)$ (the other neighbourhood operators are not used in the tree search) instead of using a transition list. Hence, we make use of the concrete characteristics, i.e. the critical operations, of S, rather than relying on information generated during local search. We additionally do not make use of Algorithm 7 in our modified F&F framework.

5. Computational study

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

In total, we implemented four approaches as illustrated in Table 5.

Table 5. Algorithms

Abbreviation	Algorithm	Reference
FaF	F&F framework	Section 4.2.5
FaFM	Modified F&F framework	Section 4.2.6
TS	Tabu search procedure	Section 5
CPA	IBM's CP solver on the CP model of WFJSP	Section 3.3

Our approaches include a tabu search (TS) benchmark heuristic that is presented in Algorithm 9. All elements of Algorithm 9 are in line with the setup of our F&F framework. It uses neighbourhood switching and the structure of the tabu list (line 11) is identical to the one used in Algorithm 6. The tabu length is dynamically updated. It is set to the average number of critical (in case of $\phi \in \{1, 3\}$) or non-critical (in case of $\phi = 2$) operations of the current solution S. The input parameter τ specifies the termination criterion of the subroutine of lines 4–13.

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

The remainder of section is split into two parts. In the first part, we analyse the performance of our F&F approaches (FaF and FaFM) on randomly generated test instances when compared with TS (as a benchmark heuristic using the same

Algorithm 9. Tabu search heuristic

```
Input: Instance $\mathcal{I}$ of WFJSP, parameter $\tau$
Output: Solution $S^*$
▷ Initialisation phase
1  Determine a feasible solution $S$ of $\mathcal{I}$ with the constructive procedure described in Section 4.2.1 and
     initialise $S^* := S$ and $S^{TBS} := S$;
2  Unlock all neighbourhood operators in the specific order $\{1, 3, 2\}$ and initialise an empty tabu list;
▷ Tabu search phase
3  Get the first unlocked neighbourhood operator $\phi$, clear tabu list, and set $NoImprCounter := 0$;
4  forall critical (if $\phi \in \{1, 3\}$) or non-critical (if $\phi = 2$) operations $a_b \in V$ of the solution graph of $S$ do
5      Call (modified, if $\phi = 3$) Algorithm 1 to determine the set $N$ of neighbouring solution graphs;
6      Discard all elements of $N$, where the corresponding transition from $S$ is tabu if their makespan is
         not smaller than $\ddot{S}^*$ (aspiration criterion);
7      Select the best solution $S^{NBS}$ among the remaining solutions in $N$. If no solution remains, i.e. if
         $N = \emptyset$, lock current neighbourhood operator $\phi$ and go to line 16;
8  end
9  if $\ddot{S}^{NBS} < \ddot{S}^{TBS}$ then set $S^{TBS} := S^{NBS}$ and $NoImprCounter := 0$;
10 else set $NoImprCounter := NoImprCounter + 1$;
11 Update the tabu list;
12 Set $S := S^{NBS}$;
13 if $NoImprCounter < \tau$ then go to line 4;
14 else if $\ddot{S}^{TBS} < \ddot{S}^*$ then set $S^* := S^{TBS}$ and unlock all neighbourhood operators in the specific
     order $\{1, 3, 2\}$;
15 else lock current neighbourhood operator $\phi$;
16 if there is at least one unlocked neighbourhood operator then set $S := S^{TBS}$ and go to line 3;
17 else exit the procedure;
```

neighbourhood structure) and CPA (as a standard solver benchmark). In the second part, we make use of benchmark instances from the literature in oder to compare the performance of our heuristics with existing metaheuristics from the literature. All test instances are available in supplementary files that accompany this paper (`https://doi.org/10.6084/m9.figshare.8082059`).

5.1. *Random testbed*

Our random testbed is composed of 24 instances sets, denoted by wfjsp1–wfjsp24. Each set represents a different scenario and features 10 randomly generated instances with the parameter ranges illustrated in Table 6. The numbers of jobs, machines, and workers are fixed for the instances of each set. The number of operations q_i was drawn from uniform distributions over the intervals given in the table for all jobs $I_i \in I$. Similarly, the eligible machines M_{i_j} for operations $i_j \in O_i$ of jobs $I_i \in I$ were randomly determined based on a random generation of their number $|M_{i_j}|$. The sets of eligible workers for each operation were generated indirectly by iterating over all machines $M_m \in M$ and randomly generating a subset of workers (denoted by $\hat{W}^m$ in Table 6) of a given size that is assumed to be able to process the respective machines. The processing times were then generated as follows (cf. Kress, Müller, and Nossack 2019). First, auxiliary integer parameters p_{i_j} for all jobs $I_i \in I$ and operations $i_j \in O_i$ were drawn from uniform distributions over the interval $[10, 100]$. Based on these parameters, we constructed varying processing times over the corresponding eligible machines $M_m \in M_{i_j}$ by drawing integer values $p^m_{i_j}$ from uniform distributions over $[\lfloor 0.9 \cdot p_{i_j} \rfloor, \lfloor 1.1 \cdot p_{i_j} \rfloor]$. In the last step, we incorporated dependencies on the workers $W_w \in W$ by drawing integer values $p^{m,w}_{i_j}$ from uniform distributions over $[\lfloor 0.9 \cdot p^m_{i_j} \rfloor, \lfloor 1.1 \cdot p^m_{i_j} \rfloor]$ for all $M_m \in \hat{M}^w \cap M_{i_j}$, where $\hat{M}^w$ denotes the set of machines that worker $W_w \in W$ may process.

Table 6. Parameters of random testbed

Instance set	n	u	v	q_i	$\lvert M_{i_j}\rvert$	$\lvert \hat{W}^m\rvert$	Instance set	n	u	v	q_i	$\lvert M_{i_j}\rvert$	$\lvert \hat{W}^m\rvert$
wfjsp1	5	3	3	$[2,4]$	$[1,2]$	2	wfjsp13	25	15	15	$[8,12]$	$[2,5]$	8
wfjsp2	7	3	3	$[2,4]$	$[1,2]$	2	wfjsp14	30	15	15	$[5,10]$	$[2,5]$	8
wfjsp3	7	4	4	$[2,6]$	$[1,2]$	2	wfjsp15	30	15	15	$[10,15]$	$[3,5]$	8
wfjsp4	10	5	5	$[2,4]$	$[1,3]$	3	wfjsp16	40	15	15	$[5,10]$	$[2,5]$	8
wfjsp5	10	5	5	$[2,10]$	$[1,3]$	3	wfjsp17	15	5	4	$[8,10]$	$[2,3]$	3
wfjsp6	12	5	5	$[2,10]$	$[1,3]$	3	wfjsp18	20	10	8	$[5,10]$	$[1,3]$	5
wfjsp7	12	5	5	$[5,10]$	$[2,3]$	3	wfjsp19	20	10	8	$[5,15]$	$[2,3]$	5
wfjsp8	15	5	5	$[5,10]$	$[2,3]$	3	wfjsp20	25	10	8	$[5,10]$	$[2,4]$	6
wfjsp9	15	5	5	$[8,10]$	$[2,3]$	3	wfjsp21	25	15	12	$[8,12]$	$[2,5]$	8
wfjsp10	20	10	10	$[5,10]$	$[1,3]$	5	wfjsp22	30	15	12	$[5,10]$	$[2,5]$	8
wfjsp11	20	10	10	$[5,15]$	$[2,3]$	5	wfjsp23	30	15	12	$[10,15]$	$[3,5]$	8
wfjsp12	25	10	10	$[5,10]$	$[2,4]$	6	wfjsp24	40	15	12	$[5,10]$	$[2,5]$	8

It follows directly from the sets $\hat{W}^m$ for all $M_m \in M$. All remaining processing times were set to infinity. Note that the basic parameters of instance sets wfjsp9–wfjsp16 and wfjsp17–wfjsp24 differ solely in the staffing levels, i.e. in the ratio of the number of workers and the number of machines. The staffing level is 1 for sets wfjsp9–wfjsp16 and 0.8 for sets wfjsp17–wfjsp24.

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

The computational results are presented in Table 7. For each instance set, the table presents information about the percentage of test instances that were solved to optimality with CPA within the time limit (column 'opt.'), the average quality ratios of the heuristic approaches over all runs and all instances of the set (columns 'Q_{avg}'), the average values of the best quality ratios among the five runs of the heuristics for each instance of the set (columns 'Q_{best}'), as well as the average runtimes (columns 't_{avg}') of the algorithms. Entries 'tl' indicate that the time limit was reached by CPA for all instances of the set. Bold entries highlight the best heuristic approaches with respect to the quality ratios.

We find that CPA is able to determine optimal solutions only for the small instances of the sets wfjsp1–wfjsp5 within the time limit. For instance sets wfjsp1–wfjsp4, CPA solves the majority of instances to optimality. Here, all heuristics provide high quality solutions. In general, however, both F&F approaches tend to outperform TS. Moreover, it can be concluded that FaFM provides the overall best performance among the heuristics with respect to the solution quality. This effect is particularly pronounced for instances of larger size. The average computational times of the F&F approaches are quite similar, except for the largest instance sets (wfjsp13–wfjsp16 and wfjsp21–wfjsp24), where FaFM terminates faster than FaF. Based on these results, it can be concluded that it certainly pays off to locally decide on the transitions applied within the tree search procedure. Finally, when comparing the results for instances wfjsp17–wfjsp24 and wfjsp9–wfjsp16, we observe that smaller staffing levels tend to result in smaller quality ratios.

It is interesting to additionally compare the performance of CPA when the time limit for some instance is set to the average time needed by the five corresponding

Table 7. Performance of the heuristic approaches on random testbed

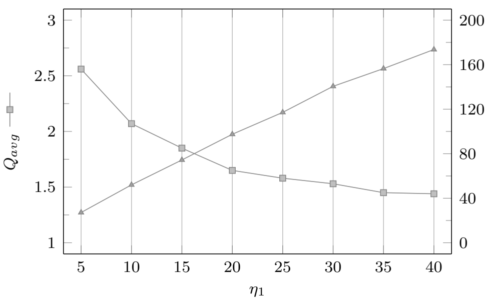
	CPA		FaF			FaFM			TS		
Instance set	opt. [%]	t_{avg} [s]	Q_{best}	Q_{avg}	t_{avg} [s]	Q_{best}	Q_{avg}	t_{avg} [s]	Q_{best}	Q_{avg}	t_{avg} [s]
wfjsp1	100.00	0.83	0.22	0.70	0.16	**0.11**	**0.64**	0.26	0.35	1.32	0.02
wfjsp2	90.00	372.80	0.78	1.38	0.29	**0.35**	**0.84**	0.35	0.59	1.42	0.03
wfjsp3	100.00	59.68	**0.89**	2.85	0.40	1.02	**2.00**	0.77	2.23	3.80	0.04
wfjsp4	90.00	1031.33	2.21	3.93	0.56	**1.52**	**2.38**	0.68	2.57	4.13	0.06
wfjsp5	10.00	3272.70	3.88	5.61	2.84	**2.33**	**3.37**	3.59	3.74	5.49	0.66
wfjsp6	-	tl	1.92	3.53	4.80	**0.80**	**1.89**	5.69	2.63	4.57	1.02
wfjsp7	-	tl	2.03	3.22	12.53	**2.02**	**2.80**	9.74	2.43	3.57	3.16
wfjsp8	-	tl	**1.51**	**2.91**	24.57	2.51	3.27	17.56	2.54	3.44	7.43
wfjsp9	-	tl	**1.46**	**2.38**	41.04	1.97	2.57	22.69	1.85	2.76	10.51
wfjsp10	-	tl	6.60	8.77	19.02	**1.50**	**3.02**	18.14	4.90	7.45	7.34
wfjsp11	-	tl	6.65	8.20	45.31	**2.97**	**4.19**	34.14	6.17	7.73	21.83
wfjsp12	-	tl	2.72	3.68	52.45	**0.77**	**1.69**	33.60	2.63	3.62	20.27
wfjsp13	-	tl	5.09	6.37	127.92	**1.39**	**2.51**	48.59	4.24	5.45	50.59
wfjsp14	-	tl	3.61	4.67	70.17	**1.45**	**2.22**	33.47	3.53	4.48	32.15
wfjsp15	-	tl	3.51	4.03	265.15	**1.63**	**2.33**	98.45	3.19	3.77	136.70
wfjsp16	-	tl	1.83	2.45	141.58	**0.28**	**0.88**	68.23	1.80	2.56	63.86
wfjsp17	-	tl	**0.61**	1.55	56.48	0.83	**1.34**	30.90	1.12	2.38	17.91
wfjsp18	-	tl	2.49	3.98	22.38	**1.25**	**1.93**	22.22	3.50	4.28	10.34
wfjsp19	-	tl	2.95	3.81	49.62	**1.46**	**2.11**	46.64	3.20	3.81	31.82
wfjsp20	-	tl	1.97	2.64	55.28	**1.47**	**1.95**	37.03	2.87	3.32	37.72
wfjsp21	-	tl	2.11	2.91	119.50	**1.29**	**1.81**	61.17	2.50	3.05	71.50
wfjsp22	-	tl	2.28	2.89	82.84	**1.82**	**2.33**	44.28	2.96	3.44	47.39
wfjsp23	-	tl	2.05	2.64	347.88	**1.37**	**2.05**	115.74	2.40	2.80	295.22
wfjsp24	-	tl	1.30	1.86	185.73	**0.82**	**1.28**	84.99	2.12	2.55	132.10

calls of FaFM and when the parallel processing mode in the CPLEX CP Optimizer is deactivated for the sake of comparability. This is subject of Table 8, where quality ratios are determined in relation to the previous CPA calls. Again, bold entries highlight the best solution approaches with respect to the quality ratios. We observe that in this setting, i.e. when time becomes a limiting factor as it is, for example, often the case in rolling horizon based planning approaches, the standard CP solver provides feasible solutions for all considered instances. However, FaFM clearly outperforms the standard solver. This provides first evidence for the fact that FaFM may also be competitive with state-of-the-art metaheuristics from the literature.

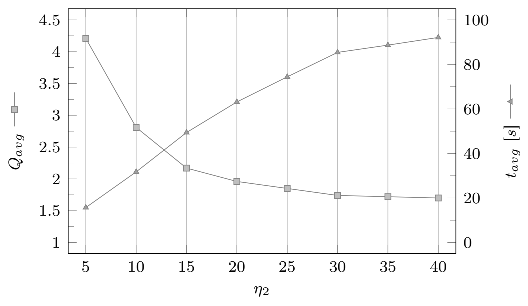
Table 8. Comparison of FaFM and CPA with FaFM-based time limit

	CPA	FaFM				CPA	FaFM		
Instance set	Q_{avg}	Q_{best}	Q_{avg}	t_{avg} [s]	Instance set	Q_{avg}	Q_{best}	Q_{avg}	t_{avg} [s]
wfjsp1	**0.03**	0.11	0.64	0.26	wfjsp13	4.22	**1.39**	**2.51**	48.59
wfjsp2	0.89	**0.35**	**0.84**	0.35	wfjsp14	3.79	**1.45**	**2.22**	33.47
wfjsp3	1.34	**1.02**	**2.00**	0.77	wfjsp15	**0.94**	1.63	2.33	98.45
wfjsp4	4.94	**1.52**	**2.38**	0.68	wfjsp16	3.07	**0.28**	**0.88**	68.23
wfjsp5	5.95	**2.33**	**3.37**	3.59	wfjsp17	3.58	**0.83**	**1.34**	30.90
wfjsp6	4.53	**0.80**	**1.89**	5.69	wfjsp18	4.37	**1.25**	**1.93**	22.22
wfjsp7	4.39	**2.02**	**2.80**	9.74	wfjsp19	3.42	**1.46**	**2.11**	46.64
wfjsp8	4.32	**2.51**	**3.27**	17.56	wfjsp20	4.75	**1.47**	**1.95**	37.03
wfjsp9	3.48	**1.97**	**2.57**	22.69	wfjsp21	4.04	**1.29**	**1.81**	61.17
wfjsp10	7.46	**1.50**	**3.02**	18.14	wfjsp22	5.18	**1.82**	**2.33**	44.28
wfjsp11	3.62	**2.97**	4.19	34.14	wfjsp23	2.13	**1.37**	**2.05**	115.74
wfjsp12	3.45	**0.77**	**1.69**	33.60	wfjsp24	2.91	**0.82**	**1.28**	84.99

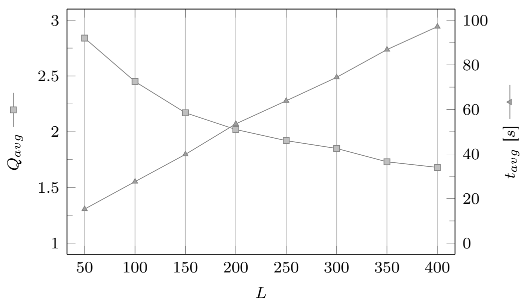
We close this section with an illustration of the effect of the choice of different values for the parameters η_1, η_2, and L within FaFM in Figure 5. The plots are based on fixing all but one parameter to their original values as stated above and setting the other



(a) Analysis of parameter η_1



(b) Analysis of parameter η_2



(c) Analysis of parameter L

Figure 5. Parameter analysis for FaFM

parameter to distinct values, i.e., $\eta_1 \in \{5, 10, 15, 20, 25, 30, 35, 40\}$ (Figure 5a), $\eta_2 \in \{5, 10, 15, 20, 25, 30, 35, 40\}$ (Figure 5b), and $L \in \{50, 100, 150, 200, 250, 300, 350, 400\}$ (Figure 5c). For each resulting set of parameters, we ran FaFM five times on each instance of the largest instance sets wfjsp14–wfjsp16 and wfjsp22–wfjsp24. The plots illustrate the corresponding average quality ratios and runtimes. We observe that increasing values of the parameters result in increasing runtimes. Furthermore, they first tend to improve the solution quality. At some point, however, this latter effect stagnates. Hence, the parameters must be chosen carefully in order to balance the trade-off between solution quality and runtime, especially when applying FaFM in practice.

5.2. *Literature instances*

In our additional computational tests, we make use of benchmark instances that are based on the ones provided by Lei and Guo (2014), who extend two FJSP benchmark sets with worker related information. This includes ten instances (MK1–MK10) of the set by Brandimarte (1993) with 10 to 20 jobs, 4 to 15 machines, 3 to 15 operations per job, 2 to 6 eligible machines for each operation, and processing times that range from 1 to 20 time units. Moreover, this includes twelve instances (DP1–DP12) of the set by Dauzère-Pérès and Paulli (1997) with 10 to 15 jobs, 5 to 8 machines, and 15 to 25 operations per job. The set of eligible machines of the operations of the instances within this latter set has been randomly constructed, by assuming that each machine is eligible with a 0.1 to 0.5 probability. The processing times range from 10 to 100 time units. The worker related information of these instances provided by Lei and Guo (2014) is summarised in Table 9. In order to keep the notation simple, we will not change the labels of the augmented instance sets, and refer to them by MK1–MK10 and DP1–DP12. Table 9 includes information on the number of workers and the sets

Table 9. Worker related information of the literature instances (Lei and Guo 2014)

Literature instance	v	$\hat{M}^w$
MK1–MK2	4	$\hat{M}^1 = \{1,3,5\}$, $\hat{M}^2 = \{2,4,5\}$, $\hat{M}^3 = \{1,4,6\}$, $\hat{M}^4 = \{2,3,6\}$
MK3–MK4; DP7–DP12	6	$\hat{M}^1 = \{1,5\}$, $\hat{M}^2 = \{2,4\}$, $\hat{M}^3 = \{1,4,6\}$, $\hat{M}^4 = \{2,3,6,7\}$, $\hat{M}^5 = \{6,7,8\}$, $\hat{M}^6 = \{5,8\}$
MK5	3	$\hat{M}^1 = \{1,3,4\}$, $\hat{M}^2 = \{2,4\}$, $\hat{M}^3 = \{1,2,3\}$
MK6,MK10	8	$\hat{M}^1 = \{1,8,10\}$, $\hat{M}^2 = \{2,7,11\}$, $\hat{M}^3 = \{3,4,6,11\}$, $\hat{M}^4 = \{2,9,12,13\}$, $\hat{M}^5 = \{6,7,8,15\}$, $\hat{M}^6 = \{5,8,10\}$, $\hat{M}^7 = \{4,9,14,15\}$, $\hat{M}^8 = \{1,3,10,14\}$
MK7; DP1–DP6	4	$\hat{M}^1 = \{1,3,5\}$, $\hat{M}^2 = \{2,4\}$, $\hat{M}^3 = \{3,4\}$, $\hat{M}^4 = \{1,2,5\}$
MK8–MK9	6	$\hat{M}^1 = \{1,3,5\}$, $\hat{M}^2 = \{2,4,9\}$, $\hat{M}^3 = \{3,4,8,10\}$, $\hat{M}^4 = \{1,7,9\}$, $\hat{M}^5 = \{5,6,7\}$, $\hat{M}^6 = \{2,4,8,10\}$

of machines that can be operated by the workers. Unfortunately, the generation of the processing times is only very briefly summarised by Lei and Guo (2014). Moreover, the instances are not available publicly. Hence, based on the related information given by Lei and Guo (2014), we made use of the following procedure. For all workers $W_w \in W$, all operations $i_j \in O_i$ of jobs $I_i \in I$, and all $M_m \in \hat{M}^w \cap M_{i_j}$, we drew the processing times from a uniform distributions over the intervals $[\bar{p}^m_{i_j}, \bar{p}^m_{i_j} + \delta_{i_j}]$, where $\bar{p}^m_{i_j}$ are the original processing time stated by Brandimarte (1993) or Dauzère-Pérès and Paulli (1997), and the values δ_{i_j} were drawn from a uniform distributions over the interval $[2, 8]$. All remaining processing times were set to infinity.

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

$$LB = \max\left(\max_{i\in\{1,\dots,n\}}\left(\sum_{i_j\in O_i} p^{\min}_{i_j}\right), \left\lceil\frac{P}{|M|}\right\rceil, \left\lceil\frac{P}{|W|}\right\rceil\right).$$

Here, $p^{\min}_{i_j} = \min_{M_m\in M_{i_j}} p^{m,\min}_{i_j}$ (see Section 4.1.2) for all $I_i \in I$, $i_j \in O_i$, and $P = \sum_{i=1}^{n}\sum_{i_j\in O_i} p^{\min}_{i_j}$. Note that, for the sake of brevity, we do not explicitly state the concrete instance in the definition of the bound.

As in the previous section, we initiated five runs of the heuristics FaF, FaFM, and TS on each instance. Moreover, we ran CPA with the FaFM-based time limit as introduced in the context of Table 8. All calls of the algorithms returned feasible solutions. We measure the quality of a solution with makespan $C_{\max}$ with the corresponding quality ratio $100 \cdot (C_{\max} - LB)/LB$. The computational results are presented in Table 10.

The table includes information on the best (columns 'Q_{best}') and the average (columns 'Q_{avg}') quality ratios over all calls of some algorithm on the respective instance, as well as the average runtime of the heuristic approaches (columns 't_{avg}'). Bold entries highlight the best approaches with respect to the quality ratios among FaF, FaFM, and TS. Figure 6 additionally plots the values given in Table 10.

As in case of the random testbed, FaFM tends to outperform FaF and TS. FaF, FaFM and TS also prove to be competitive with the standard solver (with FaFM-based time limit) for the MK instances, which is in line with our results for the random testbed. In case of the DP instances, CPA sometimes also performs slightly better than our F&F approaches. We believe that this is a result of the small average numbers of eligible machines for the operations, that allow the CP solver to quickly determine

Table 10. Performance of heuristic approaches on literature instances

		CPA	FaF			FaFM			TS		
Instance	LB	Q_{best}	Q_{best}	Q_{avg}	t_{avg} $[s]$	Q_{best}	Q_{avg}	t_{avg} $[s]$	Q_{best}	Q_{avg}	t_{avg} $[s]$
MK1	66	18.18	**10.61**	12.73	2.85	10.61	**11.82**	4.16	12.12	15.45	0.48
MK2	65	10.77	**7.69**	**11.08**	6.36	10.77	12.62	5.41	10.77	12.62	3.01
MK3	182	37.91	34.62	37.58	24.99	**34.07**	**34.73**	21.99	34.62	36.59	11.08
MK4	80	32.50	26.25	31.00	11.29	**22.50**	**25.75**	11.54	30.00	32.50	1.06
MK5	295	5.76	**3.39**	5.69	31.79	**3.39**	**4.20**	37.54	4.41	5.63	13.10
MK6	78	41.03	64.10	71.79	31.14	**44.87**	**51.28**	29.05	55.13	65.38	13.01
MK7	213	11.74	**8.92**	12.39	26.15	9.86	**12.30**	18.54	11.27	13.05	8.28
MK8	488	28.89	29.10	30.98	72.59	**28.07**	**28.52**	51.14	29.10	30.61	11.76
MK9	443	20.32	12.87	14.09	291.95	**11.06**	**12.78**	74.79	12.64	14.13	133.00
MK10	289	19.72	17.99	23.60	339.27	**13.84**	**16.12**	73.38	20.07	25.88	72.65
DP1	2881	5.07	8.05	9.68	31.35	**4.58**	**5.23**	64.10	7.71	8.97	8.60
DP2	2881	1.80	2.26	3.41	60.92	**1.91**	**2.43**	32.54	2.29	2.95	19.91
DP3	2881	1.49	**1.67**	3.05	74.05	2.12	**2.43**	41.97	1.80	3.44	41.44
DP4	2862	3.67	7.69	9.93	38.63	**3.74**	**4.70**	64.87	7.44	8.41	8.48
DP5	2832	2.68	**2.97**	3.64	65.31	3.04	**3.62**	41.10	3.07	4.37	17.57
DP6	2799	3.61	3.29	4.91	106.79	3.75	**4.12**	53.70	**3.18**	4.14	73.58
DP7	2843	3.80	7.70	11.09	78.38	**4.43**	**5.41**	88.30	6.54	9.69	17.68
DP8	2835	1.83	**2.01**	3.28	316.57	2.57	**3.08**	62.01	2.22	4.04	64.08
DP9	2824	2.48	**1.84**	3.80	484.24	2.41	**3.67**	78.27	2.20	4.96	104.79
DP10	2840	5.21	9.40	11.09	89.07	**6.51**	**7.74**	67.61	7.50	10.15	15.98
DP11	2786	4.06	**3.41**	**4.03**	379.10	4.06	4.31	76.49	3.70	5.18	64.48
DP12	2723	5.77	**3.82**	5.77	771.17	5.18	**5.51**	82.36	4.59	6.71	171.03

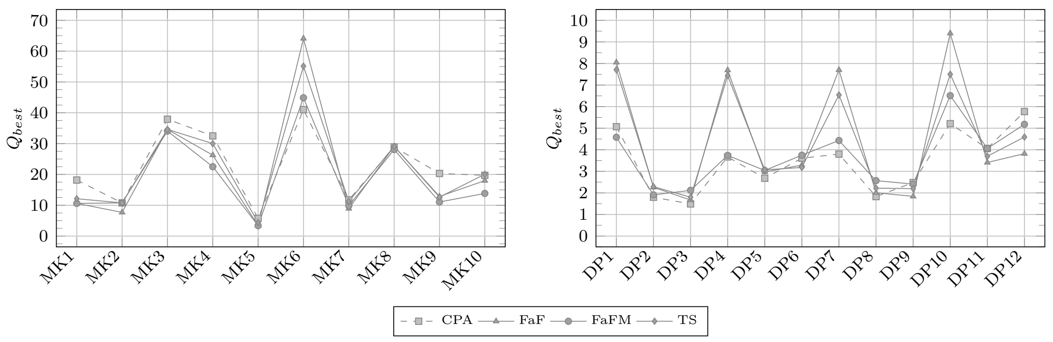


Figure 6. Performance of heuristic approaches on literature instances

promising allocation decisions of machines to operations.

With respect to the relevant metaheuristic algorithms that have been presented in the literature (see Section 2, Table 1), Peng et al. (2018) and Xianzhou and Zhenhe (2011) provide only basic case studies or examples in order to evaluate their approaches. Zhang, Wang, and Xu (2015) and Yazdani et al. (2015) do not provide detailed information on their test instances. Thus, we can only compare our approaches with the variable neighbourhood search approach (denoted by VNS) by Lei and Guo (2014) and the knowledge-guided fruit fly optimisation algorithm (denoted by KF) by Zheng and Wang (2016). As Zheng and Wang (2016) find that KF outperforms 'existing algorithms' (including VNS), this is a comparison with the state-of-the-art. Both, Lei and Guo (2014) and Zheng and Wang (2016), make use of instances with the parameters illustrated in Table 9. Additionally, their computational tests were performed on similar hardware when compared with our setup. Lei and Guo (2014) report to have used a PC with a 2.2 GHz CPU. They used Microsoft Visual C^{++} in version 6.0. Zheng and Wang (2016) coded in C^{++} and ran their tests an a PC with a 2.3 GHz CPU. However, as mentioned above, the processing times used by these authors are not publicly known, so that comparisons with the above results have to be made carefully. Nevertheless, as our process of generating processing times mimics the procedures of the respective authors as close as possible, the lower bounds stated by these authors only slightly differ from the ones listed in Table 10. This is highlighted in Tables 11 and 12, where we compare the lower bounds as well as the results listed by all relevant studies (note that Zheng and Wang (2016) reimplemented VNS, so that this approach is listed twice). As can be seen, the differences in the lower bounds are sufficiently small to allow our results to be utilised for detecting tendencies on the relative performance of the approaches. Besides the lower bounds for the instances (labelled as described above), the tables include the average (Table 11) and best (Table 12) quality ratios (computed by using the bounds presented in the respective studies) and processing times as reported by the authors. Additionally, the tables include average ($C^{avg}_{\max}$) as well as best ($C^{best}_{\max}$) makespan values achieved for each instance by the respective algorithm. These values have been calculated and rounded based on the quality ratios reported in the relevant studies.

Based on the results presented in Tables 11 and 12, we observe that, while especially KF tends to use sightly less computational time than FaFM on average, FaFM tends to outperform KF and VNS for most instances with respect to solution quality. When comparing the average (best) quality ratios for each instance, FaFM performs best for 6 (5) out of 10 MK instances and on 12 (12) out of 12 DP instances. It can thus be observed that FaFM performs significantly better on instances with a larger average number of operations per job, as encountered in instances MK3, MK9–MK10 and DP1–DP12.

Overall, we therefore conclude that FaFM is suitable for application in real-world scenarios. It is competitive with the use of the standard CP solver provided by CPLEX when runtime becomes a limiting factor. Additionally, it outperforms state-of-the-art heuristics on average.

6. Conclusion

In this paper, we have addressed a flexible job shop scheduling problem that aims at makespan minimisation and takes account of a heterogeneous workforce by making use of processing times that do not only depend on the machine, but also on the specific

Table 11. Comparison with state-of-the-art approaches (average values)

	This article				Lei and Guo (2014)				Zheng and Wang (2016)						
	FaFM				VNS					VNS			KF		
Instance	LB	Q_{avg}	$C_{\max}^{avg}$	t_{avg} $[s]$	LB	Q_{avg}	$C_{\max}^{avg}$	t_{avg} $[s]$	LB	Q_{avg}	$C_{\max}^{avg}$	t_{avg} $[s]$	Q_{avg}	$C_{\max}^{avg}$	t_{avg} $[s]$
MK1	66	11.8	73.8	4.2	63	17.3	73.9	4.3	61	17.1	71.4	3.0	8.2	66.0	3.1
MK2	65	12.6	73.2	5.4	51	22.6	62.5	4.4	52	23.1	64.0	3.1	7.7	56.0	3.2
MK3	182	34.7	245.2	22.0	190	48.8	282.7	31.0	201	49.2	299.9	12.8	41.0	283.4	13.4
MK4	80	25.8	100.6	11.5	69	32.5	91.4	5.4	67	31.8	88.3	4.9	26.3	84.6	5.3
MK5	295	4.2	307.4	37.5	337	12.7	379.7	34.8	287	13.6	325.9	14.4	9.2	313.4	15.1
MK6	78	51.3	118.0	29.1	89	49.7	133.2	22.2	86	44.7	124.4	11.1	36.9	117.7	11.2
MK7	213	12.3	239.2	18.5	184	24.7	229.4	21.9	164	21.6	199.5	10.0	13.4	185.9	10.5
MK8	488	28.5	627.2	51.1	536	22.9	658.8	114.0	551	23.2	678.9	58.3	19.0	655.9	59.2
MK9	443	12.8	499.6	74.8	437	35.1	590.3	117.0	407	35.7	552.4	51.5	28.2	521.6	52.9
MK10	289	16.1	335.6	73.4	328	42.2	466.3	89.5	315	44.1	454.0	47.8	37.1	431.9	49.4
DP1	2881	5.2	3031.6	64.1	2885	13.5	3273.6	61.4	2887	13.5	3277.9	33.6	11.2	3209.8	35.2
DP2	2881	2.4	2951.0	32.5	2775	12.9	3133.5	62.1	2779	13.0	3139.7	33.9	9.9	3055.2	35.3
DP3	2881	2.4	2951.0	42.0	2983	10.6	3300.4	61.6	2990	11.0	3319.2	34.5	10.7	3308.4	35.6
DP4	2862	4.7	2996.6	64.9	2761	15.3	3184.0	60.8	2768	15.4	3195.1	34.4	13.1	3130.9	35.8
DP5	2832	3.6	2934.4	41.1	2984	13.2	3377.9	60.2	2988	13.3	3386.0	34.9	11.4	3329.5	35.1
DP6	2799	4.1	2914.2	53.7	2650	13.8	3015.7	60.8	2647	13.6	3006.7	33.8	10.2	2916.7	34.4
DP7	2843	5.4	2996.8	88.3	2747	29.3	3550.5	169.0	2757	29.5	3571.4	51.3	27.0	3501.7	52.3
DP8	2835	3.1	2922.2	62.0	2437	29.5	3155.2	164.0	2451	29.3	3169.9	51.1	25.2	3069.1	52.1
DP9	2824	3.7	2927.6	78.3	2338	29.2	3020.2	165.0	2321	29.5	3004.8	51.2	27.3	2955.1	52.4
DP10	2840	7.7	3059.8	67.6	2742	31.1	3595.6	164.0	2747	31.2	3605.2	51.1	26.9	3485.1	52.6
DP11	2786	4.3	2906.2	76.5	2450	34.3	3289.1	166.0	2460	33.9	3293.2	51.3	31.2	3227.8	53.0
DP12	2723	5.5	2873.0	82.4	2213	32.6	2935.1	168.0	2221	32.8	2949.3	51.3	30.1	2889.3	53.2

Table 12. Comparison with state-of-the-art approaches (best values)

	This article			Lei and Guo (2014)			Zheng and Wang (2016)				
	FaFM			VNS				VNS		KF	
Instance	LB	Q_{best}	$C_{\max}^{best}$	LB	Q_{best}	$C_{\max}^{best}$	LB	Q_{best}	$C_{\max}^{best}$	Q_{best}	$C_{\max}^{best}$
MK1	66	10.6	73	63	7.9	68	61	8.2	66	8.2	66
MK2	65	10.8	72	51	7.8	55	52	7.7	56	7.7	56
MK3	182	34.1	244	190	43.2	272	201	44.8	291	38.8	279
MK4	80	22.5	98	69	24.6	86	67	25.4	84	20.9	81
MK5	295	3.4	305	337	11.0	374	287	12.5	323	8.0	310
MK6	78	44.9	113	89	44.9	129	86	41.9	122	33.7	115
MK7	213	9.9	234	184	17.4	216	164	17.1	192	11.0	182
MK8	488	28.1	625	536	21.5	651	551	21.1	667	17.8	649
MK9	443	11.1	492	437	30.7	571	407	31.9	537	26.5	515
MK10	289	13.8	329	328	38.4	454	315	40.3	442	35.6	427
DP1	2881	4.6	3013	2885	10.5	3187	2887	11.2	3211	10.1	3178
DP2	2881	1.9	2936	2775	10.1	3055	2779	11.0	3084	8.5	3014
DP3	2881	2.1	2942	2983	9.6	3269	2990	9.3	3269	7.8	3223
DP4	2862	3.7	2969	2761	12.0	3093	2768	12.1	3104	11.1	3074
DP5	2832	3.0	2918	2984	11.5	3328	2988	11.0	3318	9.3	3265
DP6	2799	3.8	2904	2650	10.4	2925	2647	10.2	2916	9.5	2898
DP7	2843	4.4	2969	2747	28.0	3517	2757	27.2	3507	25.0	3447
DP8	2835	2.6	2908	2437	27.3	3101	2451	27.3	3119	23.8	3033
DP9	2824	2.4	2892	2338	28.3	2999	2321	29.0	2995	25.4	2911
DP10	2840	6.5	3025	2742	29.6	3555	2747	27.6	3505	24.8	3427
DP11	2786	4.1	2899	2450	31.5	3221	2460	32.7	3264	29.8	3194
DP12	2723	5.2	2864	2213	31.4	2907	2221	31.3	2916	28.2	2848

worker that operates the machine while it processes some operation. We have developed two filter-and-fan based heuristic solution approaches. These methods combine a local search procedure with a tree search procedure that generates compound transitions in order to explore larger neighbourhoods to overcome locally optimal solutions. They make use of a decomposition of the problem that allows to make use of a neighbourhood structure that has formerly shown to perform well when worker restrictions are not considered. In a computational study, we have shown that our heuristic approaches outperform existing heuristic approaches from the literature on average. They have also proven competitive when compared with a standard constraint programming solver. With respect to the setup of the filter-and-fan framework, we found that it pays off to make local decisions when generating solutions within the tree search procedure instead of relying on transitions lists that are generated during local search.

There remain several interesting questions to be answered in future research. They are, for example, concerned with the performance of filter-and-fan approaches for objectives other than makespan minimisation that are relevant in practice. Similarly, the problem under consideration can be generalised to include more restrictions found in real-world scheduling environments, e.g., sequence-dependent setup times and the need for setup operators. Moreover, it may be interesting to design methods that adjust the parameters of our filter-and-fan approaches depending on the given instance. Due to the convincing computational results, we believe that related future research activities should furthermore focus on developing filter-and-fan based heuristic solution approaches for other problem settings. As highlighted by Rego and Glover (2010), it is especially promising to consider settings, where simple neighbourhood structures have shown to be relatively effective in rather simple approaches. Here, the additional computational burden needed to take advantage of compound neighbourhood structures is likely to pay off.

Funding

David Müller has been supported by the European Union and the state North Rhine-Westphalia through the European Fund for Regional Development (EFRD), as this work was conducted as part of the project 'EKPLO: Echtzeitnahes kollaboratives Planen und Optimieren' (EFRE-0800463). This work has furthermore partially been supported by the German Research Foundation (DFG) through the grant 'Sustainable Personnel Planning in Highly Customized Assembly Lines with Work Sharing' (KR 4926/3-1, OT 500/4-1).

References

Altendorfer, K., A. Schober, J. Karder, and A. Beham. 2020. "Service level improvement due to worker cross training with stochastic worker absence." *International Journal of Production Research* (in press).

Andrade-Pineda, J. L., D. Canca, P. L. Gonzalez-R, and M. Calle. 2020. "Scheduling a dual-resource flexible job shop with makespan and due date-related criteria." *Annals of Operations Research* 291 (1): 5–35.

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

Blazewicz, J., K. H. Ecker, E. Pesch, G. Schmidt, M. Sterna, and J. Weglarz. 2019. *Handbook on scheduling: from theory to practice.* 2nd ed. Berlin: Springer.

Bokhorst, J. A. C., and G. J. C. Gaalman. 2009. “Cross-training workers in Dual Resource Constrained systems with heterogeneous processing times.” *International Journal of Production Research* 47 (22): 6333–6356.

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

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

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

Cunha, M. M., J. L. Viegas, M. S. E. Martins, T. Coito, A. Costigliola, J. Figueiredo, J. M. C. Sousa, and S. M. Vieira. 2019. “Dual resource constrained scheduling for quality control laboratories.” *IFAC-PapersOnLine* 52 (13): 1421–1426.

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

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

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

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

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

Glover, F. 1998. “A template for scatter search and path relinking.” In *Artificial Evolution*, edited by J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, Vol. 1363 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 1–51. Springer.

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

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

Greistorfer, P., and C. Rego. 2006. “A simple filter-and-fan approach to the facility location problem.” *Computers & Operations Research* 33 (9): 2590–2601.

Hashemi-Petroodi, S. E., A. Dolgui, S. Kovalev, M. Y. Kovalyov, and S. Thevenin. 2020. “Workforce reconfiguration strategies in manufacturing systems: a state of the art.” *International Journal of Production Research* (in press).

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

Hopp, W. J., and M. P. Oyen. 2004. “Agile workforce evaluation: a framework for cross-training and coordination.” *IIE Transactions* 36 (10): 919–940.

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

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

Katiraee, N., M. Calzavara, S. Finco, D. Battini, and O. Battaïa. 2021. “Consideration of workers’ differences in production systems modelling and design: State of the art and directions for future research.” *International Journal of Production Research* (in press).

Kress, D., N. Boysen, and E. Pesch. 2017. “Which items should be stored together? A basic partition problem to assign storage space in group-based storage systems.” *IISE Transactions*

49 (1): 13–30.

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

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

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

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

Lang, M., and H. Li. 2011. "Research on dual-resource multi-objective flexible job shop scheduling under uncertainty." In *Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce*, AIMSEC '11, 1375–1378. IEEE.

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

Lei, D., and X. Tan. 2016. "Local search with controlled deterioration for multi-objective scheduling in dual-resource constrained flexible job shop." In *Proceedings of the 28th Chinese Control and Decision Conference*, CCDC '16, 4921–4926. IEEE.

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

Liu, C. G., N. Yang, W. J. Li, J. Lian, S. Evans, and Y. Yin. 2013. "Training and assignment of multi-skilled workers for implementing seru production systems." *The International Journal of Advanced Manufacturing Technology* 69 (5-8): 937–959.

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

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

Meng, L., C. Zhang, B. Zhang, and Y. Ren. 2019. "Mathematical modeling and optimization of energy-conscious flexible job shop scheduling problem with worker flexibility." *IEEE Access* 7: 68043–68059.

Monden, Y. 2011. *Toyota production system: an integrated approach to just-in-time.* 3rd ed. Boca Raton: CRC Press.

Montgomery, D. C. 2012. *Design and Analysis of Experiments.* 8th ed. Hoboken, New Jersey: Wiley.

Nembhard, D. A., and S. M. Shafer. 2008. "The effects of workforce heterogeneity on productivity in an experiential learning environment." *International Journal of Production Research* 46 (14): 3909–3929.

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

Peng, C., Y. Fang, P. Lou, and J. Yan. 2018. "Analysis of double-resource flexible job shop scheduling problem based on genetic algorithm." In *Proceedings of the 15th International Conference on Networking, Sensing and Control*, ICNSC '18, 1–6. IEEE.

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

Ranjbar, M. 2008. "Solving the resource-constrained project scheduling problem using filter-and-fan approach." *Applied Mathematics and Computation* 201 (1-2): 313–318.

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

Rego, C., and F. Glover. 2002. "Local search and metaheuristics for the traveling salesman problem." In *The traveling salesman problem and its variations*, edited by G. Gutin and A. Punnen, Vol. 12 of *Combinatorial Optimization Series*, Boston, 309–368. Kluwer.

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

Rego, C., H. Li, and F. Glover. 2011. “A filter-and-fan approach to the 2D HP model of the protein folding problem.” *Annals of Operations Research* 188 (1): 389–414.

Rego, C., and F. Mathew. 2011. “A filter-and-fan algorithm for the capacitated minimum spanning tree problem.” *Computers & Industrial Engineering* 60 (2): 187–194.

Saadat, M., M. C. L. Tan, M. Owliya, and G. Jules. 2013. “Challenges and trends in the allocation of the workforce in manufacturing shop floors.” *International Journal of Production Research* 51 (4): 1024–1036.

Sennott, L. I., M. P. Van Oyen, and S. M. R. Iravani. 2006. “Optimal dynamic assignment of a flexible worker on an open production line with specialists.” *European Journal of Operational Research* 170 (2): 541–566.

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

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

Tarantilis, C. D., F. Stavropoulou, and P. P. Repoussis. 2013. “The capacitated team orienteering problem: a bi-level filter-and-fan method.” *European Journal of Operational Research* 224 (1): 65–78.

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

Uzsoy, R., C.-Y. Lee, and L. A. Martin-Vega. 1992. “A review of production planning and scheduling models in the semiconductor industry part I: system characteristics, performance evaluation and production planning.” *IIE Transactions* 24 (4): 47–60.

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

Vieira, M., S. Moniz, B. S. Gonçalves, T. Pinto-Varela, A. P. Barbosa-Póvoa, and P. Neto. 2021. “A two-level optimisation-simulation method for production planning and scheduling: the industrial case of a human–robot collaborative assembly line.” *International Journal of Production Research* (in press).

Wirojanagud, P., E. S. Gel, J. W. Fowler, and R. Cardy. 2007. “Modelling inherent worker differences for workforce planning.” *International Journal of Production Research* 45 (3): 525–553.

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

Wu, X., J. Peng, X. Xiao, and S. Wu. 2020. “An effective approach for the dual-resource flexible job shop scheduling problem considering loading and unloading.” *Journal of Intelligent Manufacturing* 1–22.

Xianzhou, C., and Y. Zhenhe. 2011. “An improved genetic algorithm for dual-resource constrained flexible job shop scheduling.” In *Proceedings of the 4th International Conference on Intelligent Computation Technology and Automation*, ICICTA ’11, 42–45. IEEE.

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

Yang, G., B. D. Chung, and S. J. Lee. 2019. “Limited search space-based algorithm for dual resource constrained scheduling problem with multilevel product structure.” *Applied Sciences* 9 (19): 4005.

Yang, Y., and L. Tang. 2010. “A filter-and-fan approach to the multi-trip vehicle routing problem.” In *Proceeding of the International Conference on Logistics Systems and Intelligent Management*, ICLSIM ’10, 1713–1717. IEEE.

Yazdani, M., M. Zandieh, and R. Tavakkoli-Moghaddam. 2019. “Evolutionary algorithms for multi-objective dual-resource constrained flexible job-shop scheduling problem.” *OPSEARCH* 56 (3): 983–1006.

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

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

Zhang, J., W. Wang, X. Xu, and J. Jie. 2013. “A multi-objective particle swarm optimization for dual-resource constrained shop scheduling with resource flexibility.” In *Proceedings of the IEEE Symposium on Computational Intelligence for Engineering Solutions*, CIES ’13, 29–34. IEEE.

Zhang, X. L., C. G. Liu, W. J. Li, S. Evans, and Y. Yin. 2017. “Effects of key enabling technologies for seru production on sustainable performance.” *Omega* 66 (Part B): 290–307.

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

Zhu, H., Q. Deng, L. Zhang, X. Hu, and W. Lin. 2020. “Low carbon flexible job shop scheduling problem considering worker learning using a memetic algorithm.” *Optimization and Engineering* 21 (4): 1691–1716.