European Journal of Operational Research 265 (2018) 503–516

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Solving the flexible job shop scheduling problem with sequence-dependent setup times

Liji Shen[a,*], Stéphane Dauzère-Pérès[b,c], Janis S. Neufeld[d]

[a] WHU – Otto Beisheim School of Management, Vallendar 56179, Germany
[b] Department of Manufacturing Sciences and Logistics, CMP, Ecole des Mines de Saint-Etienne, CNRS UMR 6158, LIMOS, Gardanne 13541, France
[c] Department of Accounting, Auditing and Business Analytics, BI Norwegian Business School, Oslo 0484, Norway
[d] Faculty of Business and Economics, Technische Universität Dresden, Dresden 01062, Germany

ARTICLE INFO

Article history:
Received 19 May 2016
Accepted 11 August 2017
Available online 25 August 2017

Keywords:
Scheduling
Flexible job shop
Tabu search
Sequence-dependent setup times

ABSTRACT

This paper addresses the flexible job shop scheduling problem with sequence-dependent setup times and where the objective is to minimize the makespan. We first present a mathematical model which can solve small instances to optimality, and also serves as a problem representation. After studying structural properties of the problem using a disjunctive graph model, we develop a tabu search algorithm with specific neighborhood functions and a diversification structure. Extensive experiments are conducted on benchmark instances. Test results first show that our algorithm outperforms most existing approaches for the classical flexible job shop scheduling problem. Additional experiments also confirm the significant improvement achieved by integrating the propositions introduced in this study.

1. Introduction

1.1. Problem description

In this paper, we address the job shop scheduling problem with flexible machines and sequence-dependent setup times. Scheduling refers to the allocation of resources to perform a set of tasks over a period of time. Under economic and commercial market pressures, developing an efficient and accurate schedule becomes increasingly important. Due to the wide applicability for real-world manufacturing systems, the Job shop Scheduling Problem (JSP) is one of the most popular scheduling problems. In the classical JSP, a set of jobs are to be processed on a set of given machines. Each job can be divided into operations, which are assigned to prescribed machines, and are processed according to specific technological orders.

Considering the growing demand on flexibility and reliability, several machines are often capable of performing similar tasks. Especially in Flexible Manufacturing Systems (FMS), a high machine flexibility is characteristic (Rossi, 2014). In this context, the availability of alternative resources can increase performance, and help to manage preventive maintenance or tackle breakdown and other unforeseen events. Taking this into account, the JSP can be extended to the Flexible Job shop Scheduling Problem (FJSP), where an operation can be carried out on a set of compatible machines. The FJSP was originally stated by Brucker and Schlie (1990). The routing flexibility of a job then results in two sub-problems: The assignment of each operation to a specific machine has to be decided and operations have to be sequenced on each machine to optimize a given criterion.

Moreover, setup times are another important characteristic of many practical real-world scheduling settings. The integration of setup times has a significant impact on the applicability and performance of scheduling algorithms for practical manufacturing systems (e.g., Krajewski, King, Ritzman, & Wong, 1987; Wilbrecht & Prescott, 1969). Undeniable savings have also been observed when setup times are explicitly investigated in the scheduling decisions (Allahverdi, Ng, Cheng, & Kovalyov, 2008). Conventional studies of manufacturing lines, however, often focus on the operation time, and setup times are assumed to be zero or constant. In many real applications such as chemical, printing, pharmaceutical, and automobile manufacturing, setups are not only required between jobs but also strongly depend on the preceding process on the same machine. These facts motivate our study.

More precisely, we consider the flexible job shop scheduling problem with sequence-dependent setup times (FJSP-SDST) to minimize the makespan. Sequence dependency indicates that the magnitude of setup times is determined by the current job as well as the direct previous job processed on the same machine. As a generalization of the classical JSP, the FJSP-SDST is known to be strongly NP-hard (Garey, Johnson, & Sethi, 1976). According

* Corresponding author.
E-mail addresses: liji.shen@whu.edu (L. Shen), dauzere-peres@emse.fr (S. Dauzère-Pérès).

http://dx.doi.org/10.1016/j.ejor.2017.08.021
0377-2217/

to Cheng, Gupta, and Wang (2000), the problem to be addressed here has comparatively higher complexity status. In fact, sequence-dependent setup times are generally regarded as one of the most difficult aspects of scheduling problems (Laguna, 1999). Exact solution methods are therefore not adapted to solve problems of large sizes, and our algorithm is developed based on tabu search techniques, which proved to be successful for the FJSP (see Dauzère-Pérès and Paulli (1997) and Mastrolilli and Gambardella (2000)).

1.2. Literature review

Since the 1990s, the FJSP has been extensively investigated by researchers (e.g., Brandimarte (1993); Chen, Wu, Chen, and Chen (2012); Dauzère-Pérès and Paulli (1997); Jia and Hu (2014); Kacem, Hammadi, and Borne (2002); Mastrolilli and Gambardella (2000); Roshanaei, Azab, and ElMaraghy (2013)). A more general FJSP, referred to as general job shop problem, is considered by Dauzère-Pérès, Roux, and Lasserre (1998), Dauzère-Pérès and Pavageau (2003), Imanipour (2006) and Vilcot and Billaut (2008), where routings may be non linear (operations may have more than one predecessor and more than one successor). While Imanipour (2006) presents a non-linear MIP model as well as a hybrid greedy/tabu search algorithm with the makespan as the objective, Vilcot and Billaut (2008) study a bi-criteria optimization problem where both the the makespan and the maximum lateness are minimized. They develop a tabu search algorithm and an efficient genetic algorithm taking into account multiple constraints encountered in the printing and boarding industry. Dauzère-Pérès et al. (1998) extend the results and the approach in Dauzère-Pérès and Paulli (1997) to this problem but also to the case where operations may have to be processed on multiple resources. Dauzère-Pérès and Pavageau (2003) further extend this research to the case where an operation that needs multiple resources might not need all the resources during its entire processing time.

Regarding the FJSP with other extensions, Liu and MacCarthy (1997) discuss the problem arising in an FMS with transportation times and limited buffers. They propose a Mixed Integer Linear Programming model (MILP) and two heuristics to minimize the makespan as well as the mean completion time and the maximum tardiness. Mati, Lahlou, and Dauzère-Pérès (2011) propose a genetic algorithm for a practical FJSP in which blocking constraints are taken into account. Mousakhani (2013) presents an MILP as well as an iterated local search algorithm that minimize total tardiness.

For multi-criteria objectives, Guimaraes and Fernandes (2006) present a genetic algorithm. To simultaneously minimize the makespan and the mean tardiness, Bagheri and Zandieh (2011) develop a variable neighborhood search algorithm.

We next summarize publications addressing the FJSP with various types of setup times. A recent survey on scheduling with setups can be found in Allahverdi (2015). Saidi-Mehrabad and Fattahi (2007) take into account a special case of the FJSP where each operation can be assigned to one of two parallel machines. A tabu search algorithm that first solves the sequencing problem and then the assignment problem is developed. The algorithm is compared to a branch and bound algorithm for several random test instances. Defersha and Chen (2010) study the FJSP with attached and detached setup times, machine release dates, and technological time lag constraints. For this problem, an MILP is formulated, and a parallel genetic algorithm using multiple threads is introduced. Rossi and Dini (2007) solve a problem arising from a practical manufacturing environment with transportation times and overlapping of transportation times with processing times by an ant colony optimization method. Based on their previous study, Rossi (2014) discusses the FJSP with related parallel machines and setup times. An advanced ant colony optimization algorithm is applied to solve the assignment and sequencing problem in a single step. Transportation times, varying release dates for each job as well as an overlapping of transportation and processing times are included as well. For adjusted job shop benchmark instances, the proposed algorithm is compared to existing heuristics such as the genetic algorithm by Chan, Wong, and Chan (2006). Abdelmaguid (2015) considers special cases of the FJS-SDST. More precisely, the authors assume a symmetric definition of setup times and relative small setup times in relation to processing times. Under these conditions, structural properties are examined and formulated. The authors then extend the neighbourhood structure proposed in Mastrolilli and Gambardella (2000) which is proved to be viable.

Another stream of literature adds a third dimension to the FJSP by additionally considering process plan flexibility. Özgüven, Yavuz, and Özbakır (2012) optimize hierarchically the makespan and a balanced workload with an MILP model, which is an extension of the formulation of Özgüven, Özbakır, and Yavuz (2010). Both cases of separable as well as non-separable setup times are taken into account. Adding the no-wait restriction to the problem, Jolai, Rabiee, and Asefi (2012) develop a hybrid meta-heuristic. Assembly job shop systems with setups are considered in Nourali, Imanipour, and Shahriari (2012) and Nourali and Imanipour (2014). In these systems, each job is viewed as a final product and composed of multiple parts. Predecessor and successor relationships are predefined among parts of the same job. While Nourali et al. (2012) propose a mathematical model to integrate process planning and scheduling for assembly job shop systems, Nourali and Imanipour (2014) develop a particle swarm optimization algorithm.

Sharma and Jain (2015) consider the sequence dependent setup times in stochastic dynamic job shop system, where jobs arrive continuously over time and at least one parameter of the job settings is probabilistic. The authors present a simulation model to compare the performance of nine dispatching rules. Based on their previous study, Sharma and Jain (2016) develop four setup-oriented dispatching rules for a stochastic dynamic job shop. Simulation experiments show that the new rules outperform traditional dispatching rules.

González, Vela, Varela, and González-Rodríguez (2015) include non-anticipatory setup times in a flexible job shop environment and present a scatter search algorithm. This settings differ from most conventional setup definitions as they are attached to jobs and cannot be separated from processing time. Ortiz, Neira, Jiménez, and Hernández (2016) investigate a case study in Apparel industry which resembles the flexible job shop system with setups and transfer batches. A dispatching algorithm is presented to minimize average tardiness. Rohaninejad, Kheirkhah, Vahedi Nouri, and Fattahi (2015) considers a particular case of capacitated job shop problem with sequence dependent setup cost. An MIP model and two hybrid approaches are developed to minimize the sum of tardiness cost, overtime cost and setup cost.

Considering its complexity status and broad application, the flexible job shop problem with setup times still attracts significant attention from scheduling community. Intensified research for the FJSP-SDST is thus desirable. The remaining of the paper is organized as follows. To formulate the problem under study, a mathematical model is first presented in the next section. Section 3 introduces a representation based on a disjunctive graph. Section 4 elaborates the key elements of our tabu search algorithm. Computational results and analyzes are summarized in Section 5. Some conclusions and perspectives are provided in Section 6.

2. Problem description and MILP modelling

The FJSP-SDST can be stated as follows. A set $\mathcal{J}$ of n jobs and a set $\mathcal{M}$ of m machines are given. Each job i consists of a sequence of n_i consecutive operations, where the jth operation of job i, denoted

by O_{ij}, can be processed on any machine among a subset $\mathcal{M}_{ij} \subseteq \mathcal{M}$ of compatible (also called eligible) machines.

For each operation O_{ij}, let p_{ijk} be its processing time on machine $k \in \mathcal{M}_{ij}$. Motivated by practical applications, we define setup times as both sequence and machine dependent. A setup time $s_{ii'k}$ is incurred when operations of jobs i and i' are processed successively on machine k, where $s_{iik} = 0$ for operations of the same job i. Note that $s_{ii'k}$ is only defined if there exist operations O_{ij} and $O_{i'j'}$ such that $k \in \mathcal{M}_{ij} \cap \mathcal{M}_{i'j'}$. Also, if operations O_{ij} and $O_{i'j'}$ are, in fact, assigned to different machines in the final schedule, then the model ensures that $s_{ii'k}$ is not included. Note that, if the same machine k can be used in more than one operation of each job (e.g., $k \in \mathcal{M}_{ij}$, $k \in \mathcal{M}_{ij'}$ and $j \neq j'$), then we assume that the same setup time $s_{ii'k}$ is incurred between any two consecutive operations of jobs i and i' on k. This is not a limitation of our work since the notation $s_{ii'k}$ could be extended by considering $s_{iji'j'k}$ as the setup time incurred when operations O_{ij} and $O_{i'j'}$ are processed successively on machine k. Moreover, s_{0ik} is the initial setup time of the first job i to be processed on machine k. We assume that the setup times satisfy the following triangular inequality:

$$\begin{aligned} &s_{ii''k} \leq s_{ii'k} + s_{i'i''k} \quad \forall (i, i', i'') \in \mathcal{J} \times \mathcal{J} \times \mathcal{J}, \\ &j = 1, \ldots, n_i,\ j' = 1, \ldots, n_{i'},\ j'' = 1, \ldots, n_{i''}, \\ &k \in \mathcal{M} \text{ such that } k \in \mathcal{M}_{ij} \cap \mathcal{M}_{i'j'} \cap \mathcal{M}_{i''j''} \end{aligned} \tag{1}$$

Preemption is not allowed, i.e., each operation must be completed without interruption once started. Furthermore, machines cannot perform more than one operation at a time. All jobs and machines are available at time 0. The problem is to assign each operation to an eligible machine, and to sequence the operations on the machines in order to minimize the time required to complete all jobs, i.e., the makespan $C_{\max}$.

To model the problem, let us define the following decision variables: t_{ij} as the start time of operation O_{ij}, $C_{\max}$ the resulting makespan, and the two binary variables below:

$$\alpha_{ijk} = \begin{cases} 1 & \text{if } O_{ij} \text{ is assigned to machine } k, \\ 0 & \text{otherwise,} \end{cases}$$

$$\beta_{iji'j'} = \begin{cases} 1 & \text{if } O_{ij} \text{ is scheduled before } O_{i'j'}, \\ 0 & \text{otherwise.} \end{cases}$$

The Mixed Integer Linear Program (MILP) below is used in the numerical experiments of Sections 5.2 and 5.4.

$$\min C_{\max} \tag{2}$$

$$\sum_{k \in \mathcal{M}_{ij}} \alpha_{ijk} = 1 \quad \forall j = 1, \ldots, n_i,\ i \in \mathcal{J} \tag{3}$$

$$t_{ij} \geq t_{i(j-1)} + \sum_{k \in \mathcal{M}_{i(j-1)}} p_{i(j-1)k} \alpha_{i(j-1)k} \quad \forall j = 2, \ldots, n_i, i \in \mathcal{J}, \tag{4}$$

$$t_{ij} \geq t_{i'j'} + p_{i'j'k} + s_{i'ik} - \left(2 - \alpha_{ijk} - \alpha_{i'j'k} + \beta_{iji'j'}\right) H, \tag{5}$$

$$\begin{aligned} &\forall j = 1, \ldots, n_i,\ j' = 1, \ldots, n_{i'}, (i, i') \in \mathcal{J} \times \mathcal{J}, \\ &\text{s.t. } O_{ij} \neq O_{i'j'},\ k \in \mathcal{M}_{ij} \cap \mathcal{M}_{i'j'} \\ &t_{i'j'} \geq t_{ij} + p_{ijk} + s_{ii'k} - \left(3 - \alpha_{ijk} - \alpha_{i'j'k} - \beta_{iji'j'}\right) H, \end{aligned} \tag{6}$$

$$\begin{aligned} &\forall\ j = 1, \ldots, n_i,\ j' = 1, \ldots, n_{i'}, (i, i') \in \mathcal{J} \times \mathcal{J}, \\ &\text{s.t. } O_{ij} \neq O_{i'j'}, k \in \mathcal{M}_{ij} \cap \mathcal{M}_{i'j'} \\ &C_{\max} \geq t_{in_i} + \sum_{k \in \mathcal{M}_{in_i}} p_{in_ik} \alpha_{in_ik} \quad \forall i \in \mathcal{J} \end{aligned} \tag{7}$$

$$\alpha_{ijk} \in \{0, 1\} \quad \forall j = 1, \ldots, n_i, i \in \mathcal{J}, k \in \mathcal{M}_{ij} \tag{8}$$

$$\beta_{iji'j'} \in \{0, 1\} \quad \forall j = 1, \ldots, n_i, j' = 1, \ldots, n_{i'}, (i, i') \in \mathcal{J} \times \mathcal{J} \tag{9}$$

The objective is to minimize the makespan $C_{\max}$. Constraints (3) ensure that each operation is assigned to one and only one of its eligible machine. Constraints (4) ensure the precedence relations between consecutive operations of the same job. Constraints (5) and (6) help to prevent the overlapping of operations on the same machine k. These constraints are only relevant when operations O_{ij} and $O_{i'j'}$ are both assigned to machine k, i.e. $\alpha_{ijk} = \alpha_{i'j'k} = 1$. In this case, Constraint (5) ensures that operation O_{ij} starts after the completion of $O_{i'j'}$ if O_{ij} is scheduled after $O_{i'j'}$, i.e. $\beta_{iji'j'} = 0$. Constraints (6) are bounding for $\beta_{iji'j'} = 1$. In all the other cases where operations O_{ij} and $O_{i'j'}$ are assigned to different machines, these constraints are satisfied automatically, since the right side of the inequalities is negative. Constraints (7) are used to determine the makespan.

3. Structural properties

To study the structural properties of the problem, we utilize the concept of *disjunctive graph* $G = (\mathcal{O}, \mathcal{A}, \mathcal{E})$. Set $\mathcal{O}$ contains two fictitious nodes 0 and $*$, and each remaining node is associated with an operation:

$$\mathcal{O} = \{0, 1, \ldots, o, *\},$$

which is decomposed into subsets containing operations of a single job. Subset $\mathcal{O}_i$ represents n_i operations of job i indexed consecutively by

$$\mathcal{O}_i = \{l_{i-1} + 1, \ldots, l_{i-1} + n_i\},$$

where $l_i = \sum_{h=1}^{i} n_h$ is the total number of operations of the first i jobs, and $\sum_{h=1}^{n} n_h = o$ holds.

For better understanding, and to be consistent with the notation in the MILP, operation O_{ij} is represented by a node v with $v = l_{i-1} + j$. The weight of a node corresponds to the processing time of the operation.

Each job precedence relation is given by a conjunctive arc in set $\mathcal{A}$. The last operation of each job is also connected to node $*$:

$$\mathcal{A} = \bigcup_{i=1}^{n} \{(u, u+1) : u = l_{i-1} + 1, l_i - 1\} \cup \{(v, *) : v = l_i\}.$$

Pairs of operations that can be assigned to the same machine are connected by disjunctive arcs in set $\mathcal{E}$. This set also includes disjunctive arcs with node 0, indicating the potential first operation processed on this machine:

$$\mathcal{E} = \bigcup_{k=1}^{m} \{[0, u], [u, v], m_u = m_v = k\},$$

where m_u and m_v denote the machines on which operations u and v are processed, respectively. Note that $[u, v]$ represents a disjunctive arc whereas (u, v) indicates a precedence relation.

When replaced by a conjunctive arc and to model setup times, the length of the arc is equal to the corresponding sequence-dependent setup time s_{uv} (equivalent to $s_{ii'k}$ for $u = l_{i-1} + j$ and $u = l_{i'-1} + j'$). Furthermore, if an arc originates from node 0, its length then corresponds to the associated initial setup time s_{0u}. A solution consists in replacing each disjunctive arc by a conjunctive arc for the operations assigned to the related machine and in deleting all the other disjunctive arcs (see Dauzère-Pérès & Paulli (1997)). After fully transformed into a conjunctive graph, the

Table 1
Example.

Job i	Number of operations n_i	Operation v	Eligible machines
1	4	1	{1}
		2	{1;2}
		3	{2;3}
		4	{3}
2	2	5	{1;3}
		6	{2}
3	3	7	{3}
		8	{1;3}
		9	{2}

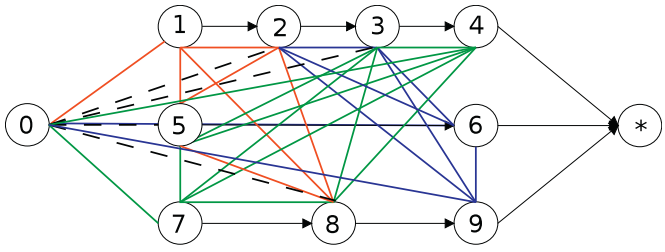

Fig. 1. Disjunctive graph for the example in Table 1.

resulting sequence Π can be expressed as:

$$\Pi = \bigcup_{k=1}^{m} \{(0,u),(u,v), m_u = m_v = k\}.$$

A solution is feasible if and only if a complete selection induces an *acyclic* directed graph.

Table 1 gives data for an example with three jobs, three machines, and nine operations.

In the disjunctive graph of Fig. 1, black arrows are conjunctive arcs connecting operations of the same job and node *. Operations that can be carried out on the same machine are connected with disjunctive arcs of the same color. Note that alternative resources are available for operations 2, 3, 5, and 8. Multiple disjunctive arcs from node 0 are thus available. They are represented with black dashed lines.

On the left side of Fig. 2, a feasible solution is derived from the disjunctive graph in Fig. 1, where each machine processes three operations. However, the solution on the right side is infeasible due to the resulting cycle highlighted with red arcs.

Next, we introduce some important definitions. A *path* from nodes u to v contains a finite sequence of nodes and conjunctive arcs between u and v. The *length* of a path $L(u,v)$ is equal to the sum of the weights of nodes and the length of the arcs on the path (Dauzère-Pérès & Paulli, 1997). A *critical path* refers to a longest path between the two fictitious nodes, and its length $L(0,{}^*)$ is the *makespan*. Operations and arcs on such paths are called *critical operations* and *critical arcs*, respectively. A *block* consists of a maximum sequence of adjacent critical operations that are processed on the same machine (Nowicki & Smutnicki, 1996). An operation is *internal* if it is neither the first nor the last operation of a particular block (Nowicki & Smutnicki, 1996).

For each operation u, we specify its *head* r_u and *tail* q_u. Graphically, r_u can be viewed as a longest path from 0 to u without this node weight, whereas q_u indicates the time required to deliver all successors of operation u, i.e., the longest path from u to *:

$$r_u = \max\left\{r_u^J, r_u^M\right\}, \tag{10}$$

with

$$r_u^J = r_{u-1} + p_{u-1} \quad (u-1,u) \in \mathcal{A}, \tag{11}$$

$$r_u^M = r_w + p_w + s_{wu} \quad (w,u) \in \Pi, \tag{12}$$

where r_u^J is the head of operation u in its job routing, and r_u^M the head of u on the machine to which it is assigned. Operation w denotes the immediate predecessor of u on the same machine. Based on the definitions, we also have

$$q_u = \max\left\{q_u^J, q_u^M\right\}, \tag{13}$$

with

$$q_u^J = q_{u+1} + p_{u+1} \quad (u,u+1) \in \mathcal{A}, \tag{14}$$

$$q_u^M = s_{uv} + p_v + q_v \quad (u,v) \in \Pi, \tag{15}$$

where q_u^J and q_u^M are the tails of operation u on its job routing and the machine it is assigned. Operation v is the direct successor of u on the same machine.

In addition, it is assumed that the predecessors of the first operations in a job routing and on a machine are the fictitious node 0 with $r_0 = p_0 = 0$. Similarly, the successors of the last operations in a job routing and on a machine are the fictitious node * with $q_* = p_* = 0$. The makespan is then determined by

$$C_{\max} = \max_{u \in \mathcal{O}} \{r_u + p_u + q_u\}. \tag{16}$$

In turn, a critical operation u satisfies $r_u + p_u + q_u = C_{\max}$.

The graphical representation also builds the basis for further analyses, which will be presented with the design of our neighborhood.

4. Tabu search algorithm

In order to thoroughly test the performance of our algorithm, we use a randomly generated initial solution. Key components of the algorithm include a neighborhood structure with feasibility checks and preliminary neighbor evaluations, a tabu list and a specific diversification.

4.1. Neighborhood structure

4.1.1. Feasibility checks

Recall that a directed graph may contain cycles which then lead to infeasible solutions. Before constructing a neighborhood function, our first concern is to exclude infeasibilities and to focus on promising moves. In this context, Dauzère-Pérès and Paulli (1997) developed the following proposition.

Proposition 4.1. *(Dauzère-Pérès & Paulli, 1997) For FJSPs, sequencing operation u between v and w does not lead to infeasible solutions if*

1. $r_v < r_{u+1} + p_{u+1}$ *with* $(u,u+1) \in \mathcal{A}$, *and*
2. $r_w + p_w > r_{u-1}$ *with* $(u-1,u) \in \mathcal{A}$ *are valid.*

Proposition 4.1 gives sufficient but not necessary conditions, which may then discard promising moves. Furthermore, these conditions identify intervals for feasible moves. However, in the presence of sequence-dependent setup times, the movable interval becomes smaller, which again limits the number of possible moves. Alternatively, we present the next proposition.

Proposition 4.2. *For FJSPs, sequencing operation u between v and w does not lead to infeasible solutions if*

1. $\max\{r_x; r_{v-1}\} < r_{u+1} + p_{u+1}$ *with* $(v-1,v),(u,u+1) \in \mathcal{A}, (x,v) \in \Pi$, *and*
2. $\min\{r_y + p_y; r_{w+1} + p_{w+1}\} > r_{u-1}$ *with* $(u-1,u),(w,w+1) \in \mathcal{A}, (w,y) \in \Pi$ *are valid.*

Proof. As depicted in Fig. 3, moving operation u between operations v and w corresponds to removing arcs (a,u), (u,b), and (v,w), and adding arcs (a,b), (v,u), and (u,w). If the new graph contains cycles, they are induced by the newly added arcs. A path

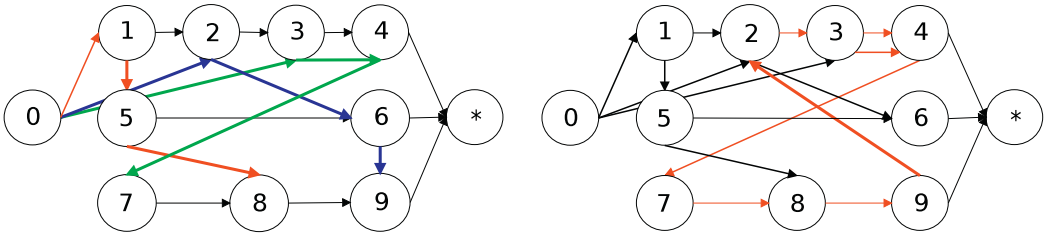

Fig. 2. Examples of a feasible solution and an infeasible solution.

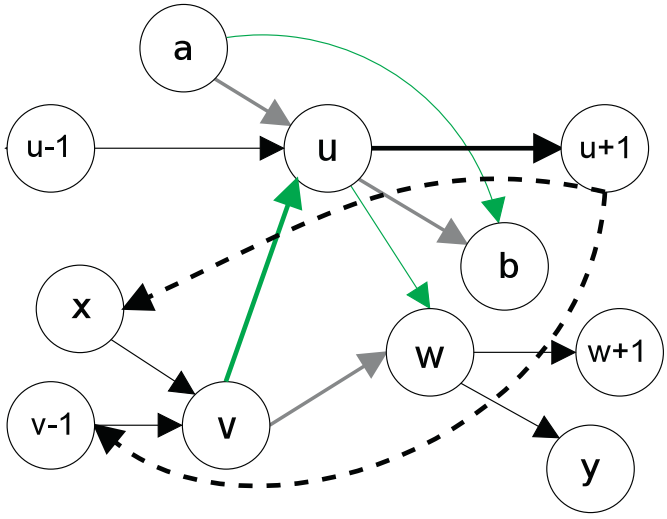

Fig. 3. Illustration of Proposition 4.2.

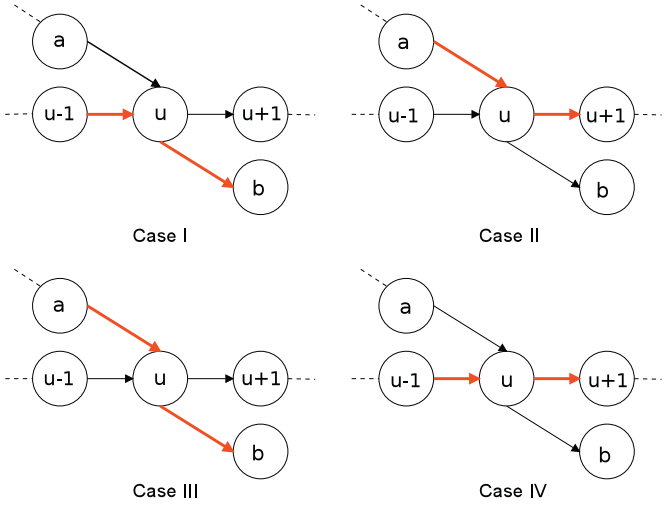

Fig. 4. Four cases for performing moves.

from b to a cannot exist since it would incur cycles in the current graph. Therefore, a cycle can only be associated to either (1) a path from u to v or (2) a path from w to u.

For the first case, a path from u to v cannot pass through w, otherwise the current graph would contain cycles, for example $(w, \ldots, v, w)$. The path from u to v must then go through $u+1$, since the arc connecting u and b is being removed. Note that the incoming arcs to v are (x, v) and $(v-1, v)$. The first condition given in the proposition requires that operation $u+1$ is completed after x and $v-1$ are released. This prevents a path connecting $u+1$ and v.

In a similar manner, it can be proved that the second condition prevents paths from w to u, which pass through operations $w+1, y$ and $u-1$. Hence, the proposition ensures feasibility after moving operation u between v and w. □

Note that this proposition is generally true if related operations are processed on different machines. Next, we show that Proposition 4.2 imposes weaker restrictions compared to Proposition 4.1.

Lemma 4.1. *For FJSPs, if Proposition 4.1 is valid, then Proposition 4.2 is also satisfied.*

Proof. By definition, the head of predecessors is not greater than the head of the corresponding operation, i.e., $\max\{r_x, r_{v-1}\} \leq r_v$. In the case where v is the first operation in a job routine, the equality holds. Combining with the first condition given in Proposition 4.1 follows

$$\max\{r_x; r_{v-1}\} \leq r_v < r_{u+1} + p_{u+1}.$$

The first condition of Proposition 4.2 is thus also satisfied. Since y and $w+1$ are successors of w, we have

$$\min\{r_y + p_y; r_{w+1} + p_{w+1}\} \geq \min\{r_y; r_{w+1}\} \geq r_w + p_w > r_{u-1},$$

where the equalities are valid if w is the last operation of a job routing and on a machine. Subject to Proposition 4.1, the second condition of Proposition 4.2 is automatically satisfied as well. □

4.1.2. Move definition

Assume that $(a, u), (u, b)$, and $(v, w) \in \Pi$ in the current solution. Based on the feasibility propositions, we define move types as follows:

Move type 1. Inserting a candidate operation u directly after v (between v and w) on the same machine k with $r_u < r_v$. This insertion must satisfy Proposition 4.2.

Move type 2. Inserting a candidate operation u directly before w (between v and w) on the same machine k with $r_u > r_w$. This insertion must satisfy Proposition 4.2.

Move type 3. Removing a candidate operation u from its current machine k, and reinserting u directly after v (between v and w) on another eligible machine k' with $r_u \leq r_v$. The reinsertion satisfies Proposition 4.2.

Move type 4. Removing a candidate operation u from its current machine k, and reinserting u directly before w (between v and w) on another eligible machine k' with $r_u \geq r_w$. The reinsertion satisfies Proposition 4.2.

Move types 1 and 2 focus on resequencing operations on the same machine. Reassignments are involved in move types 3 and 4. Here, candidates for moves refer to critical operations as they have the potential of immediately improving the makespan. Furthermore, moves relating to an operation u are differentiated by the concrete values of r_u^J, q_u^J, r_u^M and q_u^M. As shown in Fig. 4, red arcs indicate critical paths.

Case I. Move candidate u satisfying $C_{\max} = r_u^J + p_u + q_u^M$. For this case, perform move types 1, 3, and 4.

Case II. Move candidate u satisfying $C_{\max} = r_u^M + p_u + q_u^J$. For this case, perform move types 2, 3 and 4.

Case III. Move candidate u satisfying $C_{\max} = r_u^M + p_u + q_u^M$. For this case, perform move types 1, 2, 3 and 4.

Case IV. Move candidate u satisfying $C_{\max} = r_u^J + p_u + q_u^J$. No movement is involved.

We can see that move types are intentionally selected for each case. The next lemma justifies the exclusion of the remaining move types for these cases. Note that this lemma is based on the theorem of Nowicki and Smutnicki (1996). We also generalize the proof by extending the classical job-shop case to include sequence-dependent setup times.

Lemma 4.2. *No immediate improvement in the makespan can be found by performing*

- *move type 1 for case II,*
- *move type 2 for case I, and*
- *any move type for case IV.*

Proof. Let $\tilde{C}_{\max}$, $\tilde{r}_u^M$ and $\tilde{q}_u^J$ be the makespan, head, and tail of u after performing a move, respectively. First, we consider case II where $C_{\max} = r_u^M + p_u + q_u^J$ holds. By definition, operation u is critical, and is the last operation of its associated block. Operation u cannot be on a path from $u+1$ to *, since this would have created a cycle in the current graph. Thus, moving u cannot change the value of q_u^J, i.e., $\tilde{q}_u^J = q_u^J$.

Performing move type 1 indicates sequencing u after one of its successors v on the same machine k. Assume that the current partial sequence on k is $a, u, b, c, \ldots, x, v$. Heads r_u^M and $\tilde{r}_u^M$ can be respectively expressed as

$$r_u^M = r_a + p_a + s_{au}, \tag{17}$$

$$\tilde{r}_u^M \geq r_a + p_a + s_{ab} + p_b + s_{bc} + p_c + \cdots + p_x + s_{xv} + p_v + s_{vu}. \tag{18}$$

Note that equality of (18) is satisfied only when all operations $a, b, \ldots, v$ are processed contiguously, i.e., all operations are on a longest path. The variation of setup times Δs by moving u after v is thus equal to:

$$\Delta s = s_{ab} + s_{bc} + \cdots + s_{xv} + s_{vu} - s_{au}.$$

Applying triangular inequalities repeatedly follows:

$$\Delta s \geq s_{ac} + \cdots + s_{xv} + s_{vu} - s_{au} \geq s_{av} + s_{vu} - s_{au} \geq s_{au} - s_{au} = 0.$$

By including the processing times of operations currently processed between u and v, we have $\tilde{r}_u^M > r_u^M$. Therefore,

$$\tilde{C}_{\max} \geq \tilde{r}_u^M + p_u + \tilde{q}_u^J > r_u^M + p_u + q_u^J = C_{\max}.$$

The remaining cases can be proved true in a similar manner. □

4.1.3. Preliminary neighbor evaluation

According to the neighborhood functions, the algorithm performs four move types. In particular, reassignment-related moves are applied to three cases. Depending on the number of critical operations, a search process may require large computational times. Therefore, we conduct preliminary neighbor evaluations to identify promising neighbors. These evaluations are primarily based on the calculation of lower bounds on the new makespan. Assume that sets P_v and S_w contain all predecessors of v and successors of w, respectively.

Proposition 4.3. *Assume that* $(a,u), (u,b)$, and $(v,w) \in \Pi$ *in the current solution. By performing move types 1 and 3, the new makespan is not smaller than*

$$LB_1 = \max\left\{r_u^J; \hat{r}_v + p_v + s_{vu}\right\} + p_u + \max\left\{q_u^J; s_{uw} + p_w + q_w\right\}, \tag{19}$$

where

$$\hat{r}_v = \begin{cases} r_v & u \notin P_v \\ \max\left\{r_b^J; r_u^M - s_{au} + s_{ab}\right\} + r_v - r_b & u \in P_v \end{cases} \tag{20}$$

Proof. Let $\tilde{C}_{\max}, \tilde{r}$ and $\tilde{q}$ be the makespan, head and tail after a move. According to the definition of move types 1 and 3, no infeasibility occurs, and $r_u \leq r_v < r_w$ holds. Moving an operation u cannot change its head and tail in its job routing, since otherwise it would induce cycles, i.e., $\tilde{r}_u^J = r_u^J$ and $\tilde{q}_u^J = q_u^J$. Since w is always processed after u, it follows that $\tilde{q}_w = q_w$. A longest path passing through u is not larger than the new makespan, and can be expressed as

$$\begin{aligned}\tilde{C}_{\max} &\geq \max\left\{\tilde{r}_u^J; \tilde{r}_v + p_v + s_{vu}\right\} + p_u + \max\left\{\tilde{q}_u^J; s_{uw} + p_w + \tilde{q}_w\right\} \\ &= \max\left\{r_u^J; \tilde{r}_v + p_v + s_{vu}\right\} + p_u + \max\left\{q_u^J; s_{uw} + p_w + q_w\right\}.\end{aligned}$$

We next show that $\hat{r}_v$ is a lower bound of $\tilde{r}_v$.

If u is not a predecessor of v, then moving u does not change the head of v, i.e., $\tilde{r}_v = r_v$ for $u \notin P_v$.

For the case with $u \in P_v$, it follows that $b \in P_v \cup v$, otherwise $u+1 \in P_v$ which violates the first condition in Proposition 4.2. Therefore, $r_b \leq \max\{r_x; r_{v-1}\}$ is valid. The length $L(0, b)$ depends on $\tilde{r}_b^J$ and $\tilde{r}_b^M$. If u is on a longest path from 0 to $b-1$ in the current graph, this path must go through $u+1$ to $b-1$, thus

$$r_{u+1} + p_{u+1} \leq r_{b-1} < r_b \leq \max\{r_x; r_{v-1}\},$$

which contradicts the first condition of Proposition 4.2. Operation u cannot be on a longest path from 0 to $b-1$ after the insertion either, since it is then sequenced after $b-1$. Hence, $\tilde{r}_b^J = r_b^J = r_{b-1} + p_{b-1}$.

Using similar arguments, we can derive that

$$\tilde{r}_b^M = \tilde{r}_a + p_a + s_{ab} = r_a + p_a + s_{ab}.$$

By definition, $r_u^M = r_a + p_a + s_{au}$, and the length $L(0, b)$ can thus be written as

$$\begin{aligned}L(0,b) &= \max\left\{\tilde{r}_b^J; \tilde{r}_b^M\right\} \\ &= \max\left\{r_b^J; r_u^M - s_{au} + s_{ab}\right\}.\end{aligned}$$

Moreover, the length $L(b, v)$ remains unchanged, since u is processed before b in the current graph and after v in the new graph. Otherwise, this would create cycles. Hence, $L(b, v) = r_v - r_b$. The length of a path from 0 to v is then equal to

$$\begin{aligned}L(0,v) &= L(0,b) + L(b,v) \\ &= \max\left\{r_b^J; r_u^M - s_{au} + s_{ab}\right\} + r_v - r_b.\end{aligned}$$

The length $\tilde{r}_v$ of a longest path from 0 to v in the new graph can thus be expressed as

$$\tilde{r}_v \geq \max\left\{r_b^J; r_u^M - s_{au} + s_{ab}\right\} + r_v - r_b = \hat{r}_v. \tag{21}$$

Hence, LB_1 is a valid lower bound on the new makespan after performing move types 1 and 3. □

Similarly, the following proposition can be derived.

Proposition 4.4. *Assume that* $(a,u), (u,b)$, and $(v,w) \in \Pi$ *in the current solution. By performing move types 2 and 4, the new makespan is not smaller than*

$$LB_2 = \max\left\{r_u^J; r_v + p_v + s_{vu}\right\} + p_u + \max\left\{q_u^J; s_{uw} + p_w + \hat{q}_w\right\}, \tag{22}$$

where

$$\hat{q}_w = \begin{cases} q_w & u \notin S_w \\ \max\left\{q_a^J; q_u^M - s_{ub} + s_{ab}\right\} + q_w - q_a & u \in S_w. \end{cases} \tag{23}$$

Table 2
Problem sizes and parameter settings.

Instance	m	n	Tabu tenure (default)	MaxItr
mk1-2	6	10	6	22,308
mk3-4	8	15	9	42,065
mk5	4	15	6	14,872
mk6	15	10	10	88,182
mk7	5	20	8	24,000
mk8-9	10	20	11	67,882
mk10	15	20	14	124,708
m06	6	6	5	21,600
m10	10	10	8	60,000
m20	5	20	8	60,000
la01-05	5	10	6	30,000
la06-10	5	15	7	45,000
la11-15	5	20	8	60,000
la16-20	10	10	8	60,000
la21-25	10	15	10	90,000
la26-30	10	20	11	120,000
la31-35	10	30	14	180,000
la36-40	15	15	12	135,000
test1-6a	5	10	6	30,000
test7-12a	8	15	9	72,000
test13-18a	10	20	11	120,000

Particular attention should be paid to case III, where all move types apply. Case III actually refers to internal operations. Considering their large number, a refined evaluation is also used according to the following lemma.

Lemma 4.3. *If sequencing an internal operation u between two critical operations v and w of the same block does not create cycles, then the new makespan does not improve if*

$$s_{ab} + s_{vu} + s_{uw} - s_{au} - s_{ub} - s_{vw} > 0. \tag{24}$$

Proof. Operations u, v and w belong to the same block, and a longest path through this block defines the current makespan. Sequencing internal operation u between two critical operations v and w of the same block does not change the head or tail of the corresponding block since no cycles exist. Inequality (24) indicates that the total setup time increases after the move, which then leads to a longer block. The new makespan is not smaller than the length of a longest path passing through this block. Therefore, the new makespan cannot be improved after the move. □

4.2. Tabu list

Note that by employing different move types, a single tabu list is not sufficient to support various neighborhoods. For each move type, we specify tabu definition as follows:

- For move type 1, operations (v, u) and machine k are stored in the tabu list;
- For move type 2, operations (u, w) and machine k are stored in the tabu list;
- For move type 3, operations (v, u) and machine k' are stored in the tabu list;
- For move type 4, operations (u, w) and machine k' are stored in the tabu list.

The reversals of stored tabu moves are then characterized as tabu. The tabu tenure is determined by using reactive tabu search (Battiti & Giampietro, 1994; Buscher & Shen, 2009).

4.3. Diversification

Recall that our neighborhood function concentrates on critical operations. In addition, no movement is activated for case IV. Considering these facts, we integrate a specific diversification structure.

If no improvement of the makespan is found after a prescribed number of iterations (MaxItr), the operation u identified in case IV is removed from its current position and is reassigned to another eligible machine. Moreover, non-critical operations of the current solution are arbitrarily swapped. The search then resumes from the new solution. This process is repeated for mn iterations.

At the highest level of the algorithm, we implement a so-called *multi-start-diversification*. After executing the entire search procedure, the algorithm restarts from another randomly generated initial solution. The algorithm terminates after several restarts (MaxDiv).

5. Computational results

5.1. Experimental design

Computational experimentation consists of four major parts:

1. Comparing our MILP with the model presented in Saidi-Mehrabad and Fattahi (2007);
2. Solving the classical FJSP without setup times, where test results are compared with those of eight existing approaches;
3. Solving the FJSP with SDST, where
   - comparison of our MILP with our tabu search algorithm is conducted; and
   - comparison of our tabu search algorithm with two metaheuristics in the literature is conducted.
4. Examining functionalities of our tabu search algorithm.

The tabu search algorithm is implemented in C++ on an AMD Opteron 6282SE, 2.6 GHz computer with 144 GB RAM. The MILP is implemented using IBM ILOG CPLEX 12.5.0 and run on the same personal computer. Benchmark problem instances available from the literature are considered for testing.

- *The data set of* Brandimarte (1993): They consist of 10 problem instances where the number of jobs ranges from 10 to 20, the number of machines from 6 to 15, the number of operations for each job from 5 to 15, and the maximum number of equivalent machines per operation from 3 to 6;
- *The data set of* Dauzère-Pérès and Paulli (1997): The data consist of 18 test problems (test1-18a) where the number of jobs ranges from 10 to 20, the number of machines from 5 to 10, and the number of operations for each job from 5 to 25. The set of machines capable of processing an operation was constructed by letting a machine be in that set with a probability ranging from 0.1 to 0.5;
- *The data set of* Hurink, Jurisch, and Thole (1994): The problem instances were generated by modifying benchmark problems for the classical job shop scheduling problem. More specifically, they were generated from three problems by Fisher and Thompson (1963) (mt06, mt10, mt20), and 40 problems from Lawrence (1984) (la01-la40). Each set $\mathcal{M}_{ij}$ is equal to the machine to which an operation is assigned in the original problem, plus any of the other machines with a given probability. Depending on this probability, the authors generated three sets of test problem instances: edata, rdata and vdata. The first set contains the problems with the least amount of flexibility, whereas the average size of $\mathcal{M}_{ij}$ is equal to 2 in rdata and $m/2$ in vdata.

Processing times are randomly generated using a uniform distribution between given limits [1, 99]. Furthermore, we extend these benchmark instances to include sequence-dependent setup times. The values of setup times, similar to processing times, are

Table 3
Comparing MILP with Saidi-Mehrabad and Fattahi (2007).

n	m	MILP (Saidi-Mehrabad & Fattahi, 2007)				MILP 2016			
		Variables	Constraints	Nonzeros	Density	Variables	Constraints	Nonzeros	Density
3	3	289	615	2103	0,011832	136	222	912	0,030207
4	4	1121	2340	8552	0,003260	385	616	2736	0,011537
10	5	12,851	26,110	101,370	0,000302	3001	5320	25,540	0,001600
10	10	101,201	204,210	805,220	0,000039	12001	20,620	101,040	0,000408
20	5	50,701	102,220	402,740	0,000078	11,001	20,640	101,080	0,000445
30	10	903,601	1,812,630	7,215,660	0,000004	96,001	181,860	903,120	0,000052
30	30	24,328,801	48,709,830	194,536,860	0,000000	864,001	1,625,460	8,109,120	0,000006

Table 4
Comparing Mean Relative Errors (MRE %) on lower bounds.

Reference algorithm	Data sets				
	Brandimarte (1993)	Dauzère-Pérès and Paulli (1997)	Hurink et al. (1994)		
			edata	rdata	vdata
CIL 1999	19.55	7.91	5.59	4.41	2.59
MG 2000	15.41	2.24	2.27	1.40	0.14
JMFZ 2003	19.11	10.62	9.01	8.34	3.24
HTL 2007	24.61	–	–	–	–
GSG 2008	14.91	2.94	2.40	1.16	0.09
PMC 2008	17.53	7.63	6.00	4.42	2.04
YAZ 2010	16.39	5.11	3.86	1.88	0.42
JH 2014	19.36	14.12	–	–	–
TS	15.10	2.19	2.21	1.27	0.10

randomly chosen from [1, 99], where triangular inequalities are valid.

Table 2 gives our parameter settings according to the problem sizes. The settings were decided after several pilot runs to balance solution quality and computational time. Due to the non-deterministic nature of the algorithm, each problem instance is run four times and the average values are taken for evaluation. The algorithm terminates after completing four multi-start diversification steps (MaxDiv= 4).

5.2. Comparing MILPs

We first compare our MILP with the model proposed in Saidi-Mehrabad and Fattahi (2007) where decision variables, especially binary variables are defined differently. Note that both models can solve small instances to optimality. On the other hand, neither can reach optima for medium-sized instances. In order to provide a precise evaluation, Table 3 reports the resulting model size in terms of total number of variables, constraints, and nonzeros with varying problem classes. We can see that our MILP leads to considerably smaller models. It becomes more obvious as the problem size increases.

5.3. Solving classical flexible job shop scheduling problems

As mentioned earlier, limited studies dealing with the FJSP-SDST are currently available. To validate the performance of our algorithm, we next solve the classical FJSP without setup times. Table 4 summarizes the Mean Relative Error (MRE) over lower bound in comparison to that of other approaches. The following notations are used:

- *CIL 1999* denotes the genetic algorithm of Chen, Ihlow, and Lehmann (1999);
- *MG 2000* denotes the local search techniques proposed by Mastrolilli and Gambardella (2000);
- *JMFZ 2003* denotes the genetic algorithm developed by Jia, Nee, Fuh, and Zhang (2003);
- *HTL 2007* denotes the specific procedure - LEGA (GENACE) of Ho, Tay, and Lai (2007);
- *GSG 2008* denotes the hybrid metaheuristic of Gao, Sun, and Gen (2008);
- *PMC 2008* denotes the genetic algorithm proposed by Pezzella, Morganti, and Ciaschetti (2008);
- *YAZ 2010* denotes the variable neighborhood search presented in Yazdani, Amiri, and Zandieh (2010);
- *JH 2014* denotes the path-relinking tabu search algorithm developed by Jia and Hu (2014);
- *TS* denotes the tabu search algorithm proposed in this study.

The MRE is calculated as follows:

$$\text{MRE} = 100\frac{C_{\max} - \text{LB}}{\text{LB}}. \tag{25}$$

Cells filled with "–" indicate that the results are not reported in original studies. Overall, the proposed algorithm outperforms most approaches developed so far. Our solutions generally achieve small deviation from lower bounds. Significant improvement can be observed especially for the data sets of Dauzère-Pérès and Paulli (1997) and Hurink et al. (1994). This proves that our tabu search algorithm mainly focusing on setup times, is still able to obtain competitive solutions for the classical FJSP without setups.

5.4. Solving flexible job shop scheduling problems with sequence-dependent setups

Considering FJSP with SDST, very limited studies addressed exactly the same problem settings, as summarized in Section 1.2. In order to examine the performance of our algorithm, we use different references for comparison. This includes our MILP as well as two adapted metaheuristics which deal with similar FJSPs:

- *MILP* presented in Section 2;
- *SF2007* denotes the tabu search algorithm of Saidi-Mehrabad and Fattahi (2007) generalized for multiple machines instead of two machines. In this context, moves do not focus on critical operations, and an operation is allowed to be assigned on every eligible machine; and
- *AB2015* denotes the neighborbood search of Abdelmaguid (2015) generalized for common SDSTs instead of being subject to $s_{uv} = s_{vu}$ and $s_{vu} \ll p_u$. In this context, the properties given in Abdelmaguid (2015) are adjusted to accommodate standard SDSTs.

SF2007 and AB2015 are adapted, extended and implemented in C++ and run on the same computer. Note that the original termination criteria do not apply after the generalization. To ensure a fair comparison, the same amount of computational time is given to SF2007, AB2015 and TS.

The MILP can solve problem instances up to 10 jobs and 10 machines optimally. Most problem instances, however, remain unsolvable. For these cases, the IBM ILOG CPLEX is terminated after 6

Table 5
Comparing MILP and Tabu Search algorithm.

Instance	PI	Instance	PI	Instance	PI	Instance	PI
mt06 (edata)	0.00	mt06 (rdata)	0.00	mt06 (vdata)	0.00	test1a	9.71
mt10 (edata)	0.12	mt10 (rdata)	0.07	mt10 (vdata)	0.13	test2a	20.04
mt20 (edata)	1.27	mt20 (rdata)	5.77	mt20 (vdata)	8.69	test3a	29.85
el1	0.00	rl1	1.26	vl1	8.33	test4a	11.62
el2	0.00	rl2	0.00	vl2	5.30	test5a	21.86
el3	0.00	rl3	2.17	vl3	3.03	test6a	35.71
el4	0.00	rl4	2.35	vl4	3.67	test7a	18.66
el5	0.00	rl5	3.55	vl5	5.64	test8a	42.68
el6	0.00	rl6	8.00	vl6	10.62	test9a	57.04
el7	1.64	rl7	6.65	vl7	13.53	test10a	20.61
el8	2.10	rl8	5.50	vl8	11.25	test11a	43.16
el9	1.51	rl9	4.41	vl9	10.51	test12a	45.62
el10	5.22	rl10	4.81	vl10	14.23	test13a	21.01
el11	1.18	rl11	16.17	vl11	17.18	test14a	50.45
el12	4.08	rl12	10.77	vl12	17.12	test15a	–
el13	4.71	rl13	14.97	vl13	11.97	test16a	23.38
el14	4.26	rl14	9.51	vl14	15.67	test17a	46.17
el15	3.79	rl15	13.85	vl15	15.59	test18a	–
el16	0.00	rl16	3.41	vl16	16.18		
el17	0.00	rl17	1.77	vl17	20.35		
el18	0.00	rl18	2.04	vl18	21.84		
el19	0.00	rl19	5.09	vl19	16.34		
el20	0.00	rl20	1.59	vl20	16.56		
el21	2.99	rl21	6.02	vl21	28.36		
el22	3.15	rl22	9.74	vl22	27.09		
el23	2.50	rl23	7.79	vl23	24.36		
el24	1.99	rl24	6.57	vl24	28.44		
el25	0.13	rl25	6.99	vl25	26.64		
el26	5.44	rl26	12.35	vl26	27.98		
el27	5.17	rl27	10.35	vl27	46.30		
el28	6.05	rl28	10.66	vl28	30.05		
el29	9.11	rl29	14.82	vl29	42.24		
el30	6.03	rl30	12.25	vl30	25.59		
el31	8.89	rl31	14.05	vl31	57.81		
el32	11.56	rl32	25.97	vl32	44.89		
el33	13.99	rl33	26.43	vl33	58.33		
el34	6.64	rl34	22.79	vl34	–		
el35	13.95	rl35	31.08	vl35	29.66		
el36	1.47	rl36	6.49	vl36	53.26		
el37	1.26	rl37	8.39	vl37	41.11		
el38	0.00	rl38	7.59	vl38	37.70		
el39	3.25	rl39	7.39	vl39	51.67		
el40	7.58	rl40	7.04	vl40	37.86		
Mean	3.28		8.80		26.86		31.10

hours. A detailed comparison of the MILP and our tabu search algorithm is shown in Table 5, where the Percentage Improvement (PI) is calculated as:

$$\mathrm{PI} = 100\frac{C_{\max}^{MILP} - C_{\max}^{TS}}{C_{\max}^{MILP}} \tag{26}$$

where $C_{\max}^{MILP}$ and $C_{\max}^{TS}$ are the makespans obtained using the MILP and the TS, respectively. Cells filled with "–" indicate that no feasible solution is found with CPLEX after 6 hours. For small problem instances, both the MILP and the TS are able to reach optimal solutions. However, the performance of the MILP quickly drops. Note that the percentage improvement strongly increases with the problem sizes but also with the flexibility of the instances, for example for rdata, vdata, and test-data.

Table 6 reports test results of the metaheuristics proposed in Saidi-Mehrabad and Fattahi (2007), Abdelmaguid (2015), and our tabu search, where the MRE is determined according to

$$\mathrm{MRE} = 100\frac{C_{\max}^{M} - C_{\max}^{*}}{C_{\max}^{*}}. \tag{27}$$

Note that $C_{\max}^{M}$ and $C_{\max}^{*}$ are respectively the makespan of the corresponding metaheuristic and the best makespan of SF2007, AB2015, and TS. We can see that our TS generally outperforms SF2007 and AB2015. The improvement becomes especially apparent as the flexibility and problem size increase. For eight out of 137 instances (marked with asterisk), our TS was not able to reach the best solution. These results again suggest that specifically designed metaheuristics for FJSP-SDSTs are desirable.

5.4.1. Computational times

Table 7 classifies the required CPU time according to problem sizes. The times required for solving both the FJSP and the FJSP-SDST are given. When considering the FJSP, moves of internal operations on the same machine are not included since they cannot provide immediate improvement. As a result, a smaller computational time is necessary for the same class of problem instances.

Generally, the CPU time increases with the problem size. The flexibility of the instances also plays a significant role, for example when considering the same problem size for Hurink-data and Test-data. It is reasonable that the latter requires more CPU time as reassignments are more frequent for instances of higher flexibility. The largest instances can be solved within 15 minutes. It is also worth mentioning that the search can be notably accelerated with similar solution quality, if we reduce MaxItr and keep the same MaxDiv.

Table 6
Comparing MRE (%) of metaheuristics in Saidi-Mehrabad and Fattahi (2007), Abdelmaguid (2015) and TS.

Instance	TS	SF2007	AB2015	Instance	TS	SF2007	AB2015	Instance	TS	SF2007	AB2015	Instance	TS	SF2007	AB2015
mt06(edata)	0.00	0.00	0.00	mt06(rdata)	0.00	0.00	0.00	mt06(vdata)	0.00	0.00	0.00	test1a	0.00	5.71	5.08
mt10(edata)	0.00	0.00	0.00	mt10(rdata)	0.00	0.00	0.00	mt10(vdata)	0.00	0.00	0.00	test2a	0.00	3.32	3.17
mt20(edata)	0.00	0.00	0.00	mt20(rdata)	0.00	0.24	0.23	mt20(vdata)	0.00	0.51	0.57	test3a	0.00	5.01	4.82
el01	0.00	0.70	0.90	rl01	0.00	0.70	1.97	vl01	0.00	3.46	2.56	test4a	0.00	6.25	4.81
el02	0.00	1.42	2.27	rl02	0.00	0.75	1.25	vl02	0.00	2.59	2.05	test5a	0.00	4.26	4.96
el03	0.00	0.21	0.00	rl03	0.00	0.00	0.26	vl03	0.00	2.55	2.70	test6a	0.00	7.98	3.60
el04	0.00	0.32	1.27	rl04	0.00	0.51	1.14	vl04	0.00	0.00	1.18	test7a	0.00	4.92	3.93
el05	0.00	0.24	1.29	rl05	* 1.66	3.19	0.00	vl05	0.00	1.99	2.84	test8a	0.00	5.35	4.62
el06	0.00	1.03	1.54	rl06	0.00	1.57	2.15	vl06	0.00	0.53	1.85	test9a	0.00	7.49	4.45
el07	0.00	1.36	0.98	rl07	0.00	1.67	0.79	vl07	0.00	1.83	0.91	test10a	0.00	8.11	3.17
el08	0.00	1.04	1.92	rl08	0.00	2.35	2.17	vl08	0.00	1.42	1.61	test11a	* 0.85	5.92	0.00
el09	0.00	1.25	1.95	rl09	0.00	1.41	1.41	vl09	0.00	1.44	1.77	test12a	0.00	5.58	4.90
el10	0.00	1.45	1.52	rl10	0.00	1.46	1.20	vl10	0.00	2.22	2.04	test13a	0.00	8.78	4.52
el11	0.00	1.25	1.03	rl11	0.00	1.31	0.69	vl11	0.00	2.41	2.55	test14a	0.00	4.53	4.01
el12	0.00	0.30	1.08	rl12	0.00	1.43	0.95	vl12	0.00	0.00	1.33	test15a	0.00	6.62	3.47
el13	0.00	0.45	0.68	rl13	0.00	1.19	1.92	vl13	0.00	0.07	1.03	test16a	0.00	4.25	4.84
el14	0.00	1.92	2.20	rl14	0.00	1.03	1.15	vl14	0.00	1.54	1.74	test17a	0.00	8.46	3.91
el15	0.00	0.05	1.19	rl15	0.00	0.49	1.73	vl15	0.00	0.53	0.53	test18a	0.00	4.68	3.37
el16	0.00	1.38	1.53	rl16	0.00	1.57	1.37	vl16	0.00	0.62	0.62	Mean	0.05	5.96	3.98
el17	0.00	2.61	2.07	rl17	0.00	1.49	1.49	vl17	0.00	1.25	1.25				
el18	0.00	2.96	0.56	rl18	0.00	0.83	0.62	vl18	0.00	0.65	0.65				
el19	0.00	1.96	2.53	rl19	0.00	2.20	1.63	vl19	0.00	1.24	1.12				
el20	0.00	2.95	1.63	rl20	0.00	0.85	0.85	vl20	0.00	0.62	0.62				
el21	0.00	1.67	1.91	rl21	0.00	0.65	0.43	vl21	0.00	2.32	2.32				
el22	* 0.13	1.06	0.00	rl22	0.00	1.23	1.23	vl22	* 0.28	1.69	0.00				
el23	0.00	0.81	0.69	rl23	0.00	2.35	1.54	vl23	0.00	0.61	0.61				
el24	0.00	0.79	0.79	rl24	0.00	0.74	0.30	vl24	0.00	1.21	1.21				
el25	0.00	1.43	1.82	rl25	* 0.07	0.97	0.00	vl25	0.00	0.95	0.10				
el26	0.00	0.41	0.20	rl26	0.00	0.85	0.56	vl26	0.00	2.05	1.30				
el27	0.00	1.09	1.54	rl27	0.00	2.13	1.51	vl27	0.00	0.66	0.66				
el28	0.00	0.56	0.35	rl28	0.00	1.08	0.63	vl28	0.00	1.71	1.37				
el29	0.00	0.86	0.81	rl29	0.00	1.09	0.42	vl29	0.00	1.07	1.07				
el30	* 0.10	0.29	0.00	rl30	0.00	1.12	0.78	vl30	0.00	0.74	0.74				
el31	0.00	0.73	0.88	rl31	* 0.60	0.00	0.48	vl31	0.00	2.14	2.14				
el32	0.00	0.21	0.42	rl32	0.00	0.42	0.65	vl32	*1.38	2.18	0.00				
el33	0.00	0.29	0.11	rl33	0.00	0.16	0.24	vl33	0.00	0.05	0.05				
el34	0.00	0.39	0.00	rl34	0.00	0.68	0.68	vl34	0.00	3.11	3.55				
el35	0.00	1.32	0.89	rl35	0.00	0.67	0.71	vl35	0.00	0.33	0.33				
el36	0.00	1.60	1.54	rl36	0.00	0.51	0.39	vl36	0.00	1.39	1.39				
el37	0.00	1.52	2.16	rl37	0.00	0.12	0.61	vl37	0.00	0.65	0.65				
el38	0.00	1.41	0.54	rl38	0.00	1.35	1.21	vl38	0.00	0.88	0.70				
el39	0.00	2.46	1.82	rl39	0.00	0.00	0.44	vl39	0.00	1.22	0.78				
el40	0.00	2.18	1.12	rl40	0.00	1.26	1.33	vl40	0.00	0.53	0.44				
Mean	0.01	1.07	1.06	Mean	0.05	1.01	0.91	Mean	0.04	1.23	1.18				

Table 7
Computational time.

m	n		FJSP (sec.)	FJSP-SDST (second)	m	n	FJSP (second)	FJSP-SDST (second)
5	10		5.59	14.15	10	15	89.22	125.81
5	15		15.52	51.47	10	20 (Hurink-data)	167.52	408.29
5	20		66.19	123.46	10	30	345.82	788.57
8	15	(Test-data)	133.49	172.42	15	15	98.67	158.86
10	10		22.23	39.00	10	20 (Test-data)	256.33	676.59

5.5. Analysis of feasibility checks

In order to analyze the feasibility checks, the Hurink-data are solved three times: (1) Without feasibility propositions, (2) With Proposition 4.1, and (3) With Proposition 4.2.

The Percentage Improvement (PI), based on the makespan obtained without feasibility checks, is shown in Table 8. By integrating Proposition 4.1, some (about 0.2%) improvement is achieved. Using Proposition 4.2 instead of Proposition 4.1 leads to further improvement (over 1%).

Note that the Propositions 4.1 and 4.2 provide sufficient but not necessary conditions. This implies that a search with feasibility checks has additional restrictions. Nevertheless, the algorithm shows better results in these cases. Compared to Proposition 4.2, Proposition 4.1 imposes stronger conditions, which can overly restrict the search by excluding promising feasible moves. This may then explain the best results obtained with Proposition 4.2. When the feasibility propositions are not effective (see the negative values in Table 8), we conjecture that this is because the feasibility checks lead to a more focused search.

Table 8
Comparing feasibility propositions on improvement of the makespan.

	Percentage improvement (PI)					
	Edata		Rdata		Vdata	
	Prop. 4.1	Prop. 4.2	Prop. 4.1	Prop. 4.2	Prop. 4.1	Prop. 4.2
Mean	0.03	0.92	0.18	1.11	0.34	1.78
Max	2.11	2.95	1.71	2.89	3.08	3.78
Min	−1.40	−0.53	−0.90	0.00	−1.05	−0.48

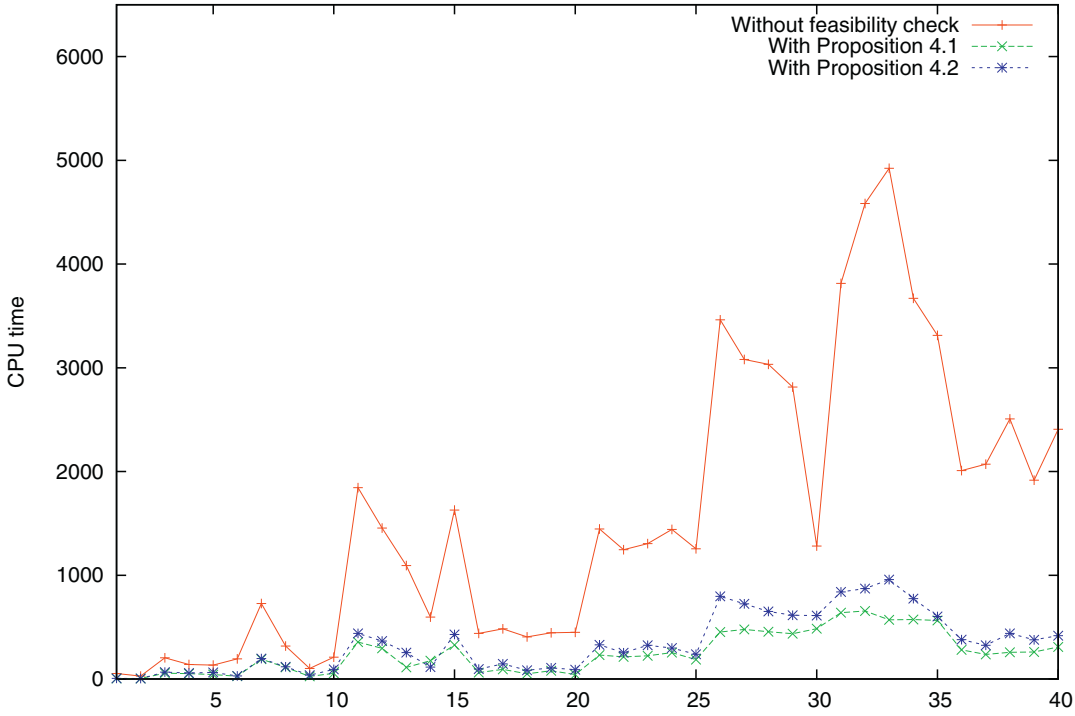

Fig. 5. Comparing CPU times with and without feasibility propositions on Hurink instances.

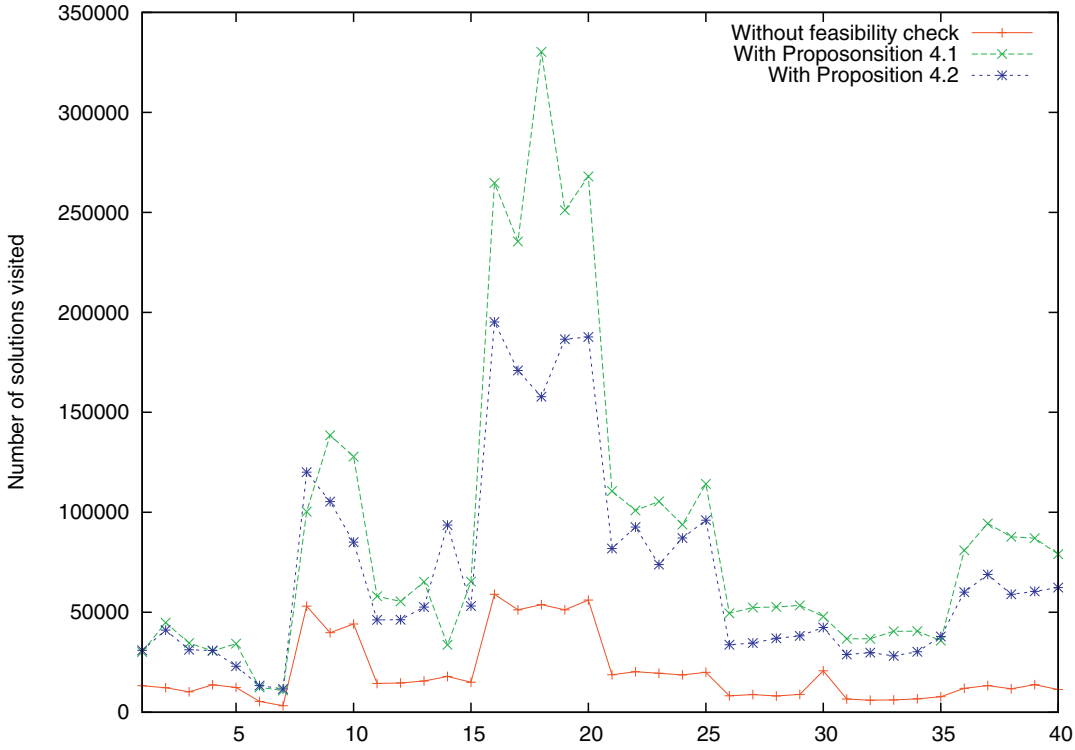

Fig. 6. Comparing numbers of visited solutions in 100 seconds with and without feasibility propositions on Hurink instances.

One of the major advantages of the feasibility checks is that they help to reduce computational times. As shown in Fig. 5, integrating Propositions 4.1 or 4.2 leads to a significant reduction of the CPU times. An unrestricted search requires more CPU time because moves are performed regardless of their feasibility status. In this context, the feasibility of solutions is only verified after actually performing the moves. When comparing Propositions 4.1 and 4.2, note that the latter requires slightly more computational time, as it allows more moves in the search process. More specifically, a search with feasibility checks explores more solutions in the same computational time. This is confirmed in Fig. 6, where the same amount of CPU time (100 seconds) is given to three cases. A search without feasibility checks visits the smallest number of solutions. This is probably because a considerable portion of CPU time is consumed on checking infeasible moves. The search with Proposition 4.1 explores the largest number of solutions.

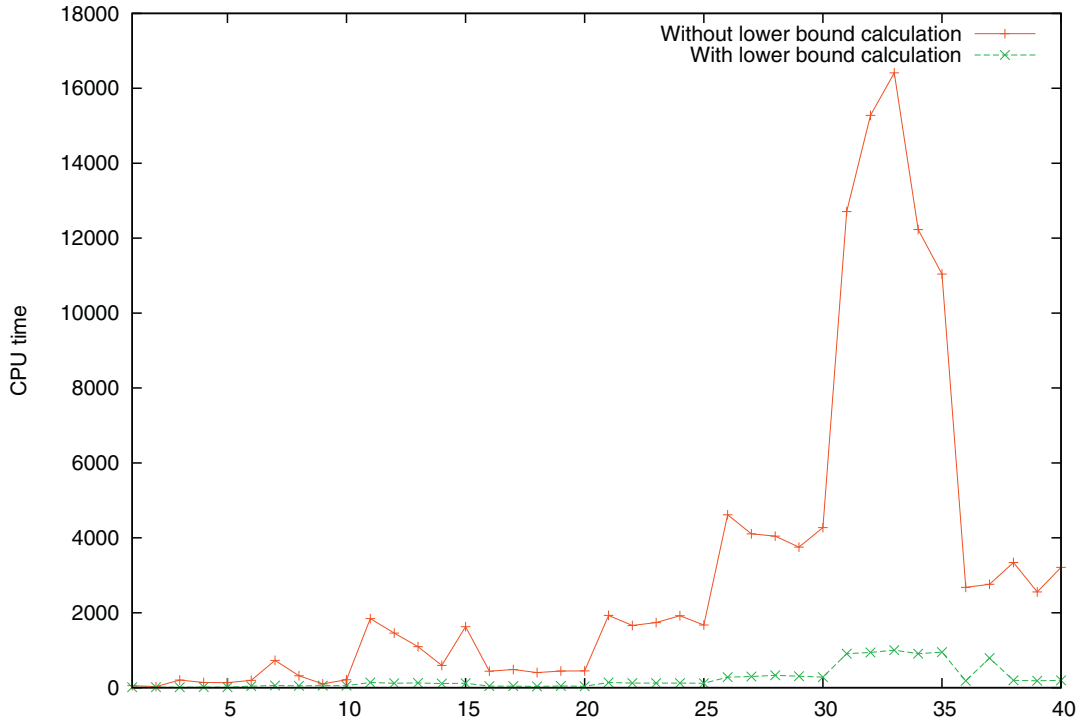

Fig. 7. Comparing CPU times with and without lower bound calculations on Hurink instances.

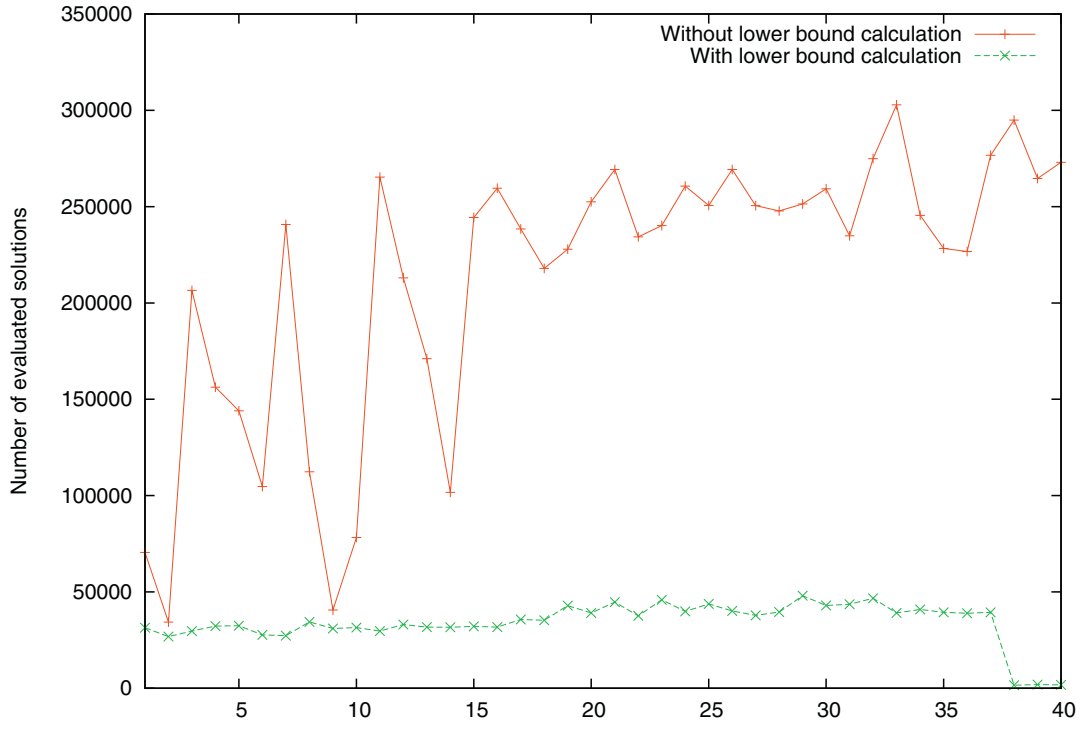

Fig. 8. Comparing numbers of evaluated solutions with and without lower bound calculations on Hurink instances.

In comparison, the search with Proposition 4.2 visits a slightly smaller number of solutions. Recall that Proposition 4.1 uses a stronger condition that may not consider a move which is actually feasible. The search with Proposition 4.2 preserves and evaluates such moves, at the cost of additional time. There are also some exceptions, as each search does not follow the same course for an individual instance.

5.6. Impact of the preliminary neighbor evaluation

To evaluate the impact of the preliminary neighbor evaluation (Propositions 4.3 and 4.4 and Lemma 4.2), the algorithm is also repeated twice on the Hurink data, respectively with and without preliminary evaluation. Similar effects that with feasibility checks are observed. In comparison, the values are even more impressive.

First, solutions of very similar qualities are observed for the search with and without preliminary neighbor evaluation. More importantly, the lower bound calculation saves a large amount of computational time as depicted in Fig. 7. This is due to a significantly reduced number of evaluated solutions. Fig. 8 indicates that a relatively constant number of solutions are considered by using the preliminary evaluation. Of particular interest is that the number of evaluated solutions suddenly drops for instances la38-40. It probably depends on the individual instances, where a reduction of the makespan is quickly achieved. Additional moves hardly provide any improvement and are therefore discarded by the preliminary evaluation.

6. Conclusions

In this paper, we investigate the flexible job shop scheduling problem with sequence-dependent setup times. A mathematical model is first presented which is used in our numerical experiments. After studying structural properties based on a disjunctive graph model, we propose a tabu search algorithm with novel neighborhood functions and a specific diversification structure. Computational experiments are conducted on well-known benchmark instances. They show that our algorithm outperforms most existing approaches used to solve the classical flexible job shop scheduling problem. For the case with sequence-dependent setup times, we compare the performances of the proposed algorithm with our mathematical model and two adapted metaheuristics in the literature. Improvement is achieved by using our tabu search algorithm. Additional experiments also confirm the benefits of the propositions in this paper, showing that studying and integrating structural properties of problems in algorithms is of particular importance for future studies.

In future work, we would like to consider other objectives than the makespan as well as more complex manufacturing systems. Following the work of Dauzère-Pérès et al. (1998) and Dauzère-Pérès and Pavageau (2003), considering multiple resources to perform an operation could significantly impact the sequence-dependent setup times, which could be associated to the combination of selected resources. Moreover, it could be relevant to model explicitly the resource (or resources) that are necessary to perform setups, and that cannot be used simultaneously.

References

Abdelmaguid, T. F. (2015). A neighborhood search function for flexible job shop scheduling with separable sequence-dependent setup times. *Applied Mathematics and Computation, 260*, 188–203.

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

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

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

Battiti, R., & Giampietro, T. (1994). The reactive tabu search. *ORSA Journal on Computing, 6*(2), 126–140.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research, 41*, 57–83.

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

Buscher, U., & Shen, L. (2009). An integrated tabu search algorithm for the lot streaming problem in job shops. *European Journal of Operational Research, 199*, 385–399.

Chan, F., Wong, T., & Chan, L. (2006). Flexible job shop scheduling problem under resource constraints. *International Journal of Production Research, 11*, 2071–2089.

Chen, H., Ihlow, J., & Lehmann, C. (1999). A genetic algorithm for flexible job shop scheduling. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 1120–1125).

Chen, J. C., Wu, C., Chen, C., & Chen, K. (2012). Flexible job shop scheduling with parallel machines using genetic algorithm and grouping genetic algorithm. *Expert Systems with Applications, 39*(11), 10016–10021.

Cheng, T., Gupta, J., & Wang, G. (2000). A review of flowshop scheduling research with setup times. *Production and Operations Management, 9*, 262–282.

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

Dauzère-Pérès, S., & Pavageau, C. (2003). Extensions of an integrated approach for multi-resource shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 33*(2), 207–213.

Dauzère-Pérès, S., Roux, W., & Lasserre, J. (1998). Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research, 107*, 289–305.

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

Fisher, H., & Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In J. Muth, & G. Thompson (Eds.), *Industrial scheduling*. Englewood Cliffs, NJ: Prentice Hall.

Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers and Operations Research, 35*, 2892–2907.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematical Methods of Operations Research, 1*, 117–129.

González, M. A., Vela, C. R., Varela, R., & González-Rodríguez, I. (2015). An advanced scatter search algorithm for solving job shops with sequence dependent and non-anticipatory setups. *AI Communications, 28*(2), 179–193.

Guimaraes, K. F., & Fernandes, M. A. (2006). An approach for flexible job-shop scheduling with separable sequence-dependent setup time. In *Proceedings of the IEEE international conference on systems, man and cybernetics: 5* (pp. 3727–3731). IEEE.

Ho, N., Tay, J., & Lai, E. (2007). An effective architecture for learning and evolving flexible job shop schedules. *European Journal of Operational Research, 179*, 316–333.

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

Imanipour, N. (2006). Modeling & solving flexible job shop problem with sequence dependent setup times. In *Proceedings of the international conference on service systems and service management, 2006: 2* (pp. 1205–1210). IEEE.

Jia, H., Nee, A., Fuh, J., & Zhang, Y. (2003). A modified genetic algorithm for distributed scheduling problems. *International Journal of Intelligent Manufacturing, 14*, 351–362.

Jia, S., & Hu, Z. (2014). Path-relinking tabu search for the multi-objective flexible job shop scheduling problem. *Comuters and Operations Research, 47*, 11–26.

Jolai, F., Rabiee, M., & Asefi, H. (2012). A novel hybrid meta-heuristic algorithm for a no-wait flexible flow shop scheduling problem with sequence dependent setup times. *International Journal of Production Research, 50*(24), 7447–7466.

Kacem, I., Hammadi, S., & Borne, P. (2002). Approach by localization and multiobjective evolutionary optimization for flexible job shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics, 32*, 1–13.

Krajewski, L. J., King, B. E., Ritzman, L. P., & Wong, D. S. (1987). Kanban, MRP, and shaping the manufacturing environment. *Management Science, 33*(1), 39–57.

Laguna, M. (1999). A heuristic for production scheduling and inventory control in the presence of sequence-dependent setup times. *IIE Transactions, 31*, 125–134.

Lawrence, S. (1984). Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement). *Technical Report*. Pittsburgh, PA: Graduate School of Industrial Administration, Carnegie-Mellon University.

Liu, J., & MacCarthy, B. L. (1997). A global MILP model for FMS scheduling. *European Journal of Operational Research, 100*(3), 441–453.

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

Mati, Y., Lahlou, C., & Dauzère-Pérès, S. (2011). Modelling and solving a practical flexible job-shop scheduling problem with blocking constraints. *International Journal of Production Research, 49*(8), 2169–2182.

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

Nourali, S., & Imanipour, N. (2014). A particle swarm optimization-based algorithm for flexible assembly job shop scheduling problem with sequence dependent setup times. *Scientia Iranica. Transaction E, Industrial Engineering, 21*(3), 1021–1033.

Nourali, S., Imanipour, N., & Shahriari, M. R. (2012). A mathematical model for integrated process planning and scheduling in flexible assembly job shop environment with sequence dependent setup times. *Review of International Journal of Mathematical Analysis, 6*(43), 2117–2132.

Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science, 42*(6), 797–813.

Ortiz, M., Neira, D., Jiménez, G., & Hernández, H. (2016). Solving flexible job-shop scheduling problem with transfer batches, setup times and multiple resources in apparel industry. In *Proceedings of the international conference in swarm intelligence* (pp. 47–58). Springer.

Özgüven, C., Özbakır, L., & Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling, 34*(6), 1539–1548.

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

Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible job shop scheduling problem. *Computers and Operations Research, 35*, 3202–3212.

Rohaninejad, M., Kheirkhah, A. S., Vahedi Nouri, B., & Fattahi, P. (2015). Two hybrid tabu search–firefly algorithms for the capacitated job shop scheduling problem with sequence-dependent setup cost. *International Journal of Computer Integrated Manufacturing, 28*(5), 470–487.

Roshanaei, V., Azab, A., & ElMaraghy, H. (2013). Mathematical modelling and a meta-heuristic for flexible job shop scheduling. *International Journal of Production Research, 51*(20), 6247–6274.

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

Rossi, A., & Dini, G. (2007). Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing, 23*(5), 503–516.

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

Sharma, P., & Jain, A. (2015). Performance analysis of dispatching rules in a stochastic dynamic job shop manufacturing system with sequence-dependent setup times: Simulation approach. *CIRP Journal of Manufacturing Science and Technology, 10*, 110–119.

Sharma, P., & Jain, A. (2016). New setup-oriented dispatching rules for a stochastic dynamic job shop manufacturing system with sequence-dependent setup times. *Concurrent Engineering, 24*(1), 58–68.

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

Wilbrecht, J. K., & Prescott, W. B. (1969). The influence of setup time on job shop performance. *Management Science, 16*(4), B–274.

Yazdani, M., Amiri, M., & Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications, 37*(1), 678–687.