Flexible Job Shop Scheduling Problems with Arbitrary Precedence Graphs

Gregory A. Kasapidis[1], Dimitris C. Paraskevopoulos[2], Panagiotis P. Repoussis[3], and Christos D. Tarantilis[4]

[1]Department of Management Science and Technology, Athens University of Economics and Business, 76 Patission street, 10434, Athens, Greece, gkasapidis@aueb.gr

[2]The Business School (formerly Cass), City, University of London, 106 Bunhill Row, EC1Y 8TZ, London, UK, dimi@city.ac.uk

[3]Department of Marketing and Communication, School of Business, Athens University of Economics and Business, 76 Patission street, 10434, Athens, Greece, prepousi@aueb.gr

[4]Department of Management Science and Technology, Athens University of Economics and Business, 76 Patission street, 10434, Athens, Greece, tarantil@aueb.gr

Abstract

A common assumption in the shop scheduling literature is that the processing order of the operations of each job is sequential; however, in practice there can be multiple connections and finish-to-start dependencies among the operations of each job. This paper studies flexible job shop scheduling problems with arbitrary precedence graphs. Rigorous mixed integer and constraint programming models are presented, as well as an evolutionary algorithm is proposed to solve large scale problems. The proposed heuristic solution framework is equipped with efficient evolution and local search mechanisms as well as new feasibility detection and makespan estimation methods. To that end, new theorems are derived that extend previous theoretical contributions of the literature. Computational experiments on existing benchmark data sets show that the proposed solution methods outperform the current state-of-the-art. Overall, 59 new best solutions and 61 new lower bounds are produced for a total of 228 benchmark problem instances of the literature. To explore the impact of the arbitrary precedence graphs, lower bounds and heuristic solutions are generated for new large-scale problems. These experiments illustrate that the machine assignment flexibility and density of the precedence graphs, affect not only the makespan, but also the difficulty of producing good upper bounds.

Keywords: Flexible Job Shop Scheduling, Mathematical Programming, Constraint Programming, Evolutionary Algorithms

Received: August 2019; accepted: May 2021 by Kouvelis Panos after two revisions.

1 Introduction

Hard-to-solve production scheduling problems have attracted significant attention in the literature (Jin et al. 2002, Caglar Gencosman et al. 2016). An important class of problems that is often encountered in manufacturing shop floors is the Job Shop Scheduling Problem (JSSP). The JSSP seeks to schedule a set of jobs. Each job consists of a set of operations that are processed by a set of machines. Each operation can be processed by only one machine and each machine can process one operation at a time. The operations of a job must be processed in a predefined chained order and the objective is to minimize the total length of the schedule, i.e., the makespan. The JSSP can be used to model a wide variety of scheduling problems (Zhang et al. 2019), yet real-world settings may involve more complex constraints and operational realities.

A well-known generalization of the JSSP is the Flexible Job Shop Scheduling Problem (FJSSP). In the FJSSP each operation can be processed by a set of parallel unrelated machines and the processing times may vary per machine. The FJSSP provides a more realistic modeling framework and thus, a wide range of variants have been introduced. Sequence dependent setup times (Shen et al. 2018), fuzzy processing times (Gao et al. 2016) and random machine breakdowns (Xiong et al. 2013) are notable examples of features that have been studied in the FJSSP literature. Despite research efforts on exact methods (Ku and Beck 2016, Demir and Kürşat Işleyen 2013, Roshanaei et al. 2013), the FJSSP literature is currently dominated by metaheuristic algorithms that aim to solve large-scale problems. Among early approaches, one may distinguish the Tabu Search (TS) methods presented in Brandimarte (1993), Dell'Amico and Trubian (1993), Dauzère-Pérès and Paulli (1997), and Mastrolilli and Gambardella (2000). In the field of population-based algorithms, Gao et al. (2008) developed a hybrid genetic algorithm combined with variable neighborhood descent, González et al. (2015) proposed a Scatter Search (SS) method coupled with Path Relinking (PR), Yi et al. (2016) presented a memetic algorithm, and Wu and Wu (2017) developed an evolutionary algorithm based on quantum physics principles. It is also worth referring to the discrepancy search algorithm and the harmony search algorithm proposed by Ben Hmida et al. (2010) and Yuan et al. (2013), respectively.

The FJSSP assumes that the processing order of the operations of a job is sequential. Nevertheless, in practice the operational hierarchies are not always linear and multiple dependencies among the operations of each job may occur. For example, in assembly shop floors several components and sub-assemblies are processed in parallel until the assembly of the final product (Komaki et al. 2018). With this topic in mind, Alvarez-Valdes et al. (2005) studied a FJSSP

with precedence relations between the jobs. A similar case is studied by Vilcot and Billaut (2008) where different precedence relations exist among the operations of a job, allowing for an operation to have multiple predecessors but at most one direct successor. Birgin et al. (2013) is, according to our knowledge, the first work that refers to the problem as the extended flexible job-shop scheduling problem (eFJSSP). The authors present a Mixed Integer Programming (MIP) model for the eFJSSP together with a set of benchmark problem instances. More recently, Lunardi and Voos (2018) developed a discrete firefly algorithm for the eFJSSP and Yu et al. (2017) extended the mathematical formulation of Birgin et al. (2013) to include job priorities and sequence flexibility.

Motivated by the practical importance of the eFJSSP as well as the sparse eFJSSP literature, our paper makes methodological contributions and proposes new solution methods, it makes theoretical contributions on how to efficiently explore neighborhood structures, and it studies the impact of important problem properties, i.e., the flexibility of having multiple unrelated parallel machines and the density of the precedence graphs. This paper presents MIP and Constraint Programming (CP) formulations and comparisons are made with the MIP model of Birgin et al. (2013) for the eFJSSP. The proposed CP model in particular, seems to perform very well and produces several new improved lower and upper bounds for both the FJSSP and the eFJSSP. For solving large-scale problems, this paper also presents a hybrid evolutionary algorithm. This heuristic solution framework uses adaptive memory structures to keep track of the search history, as well as local search algorithms that use critical path based neighborhood structures. To that end, efficient neighborhood evaluation schemes are proposed that employ new methods for makespan estimation and early feasibility detection. New theorems are derived that extend and generalize previous theoretical results from the work of Dauzère-Pérès and Paulli (1997) for the case of arbitrary precedence graphs. Computational experiments using various well-known benchmark data sets demonstrate that the proposed solution methods outperform the existing state-of-the-art solution approaches for both the FJSSP and the eFJSSP, while improved results and new optimal solutions are reported. Furthermore, experiments using existing, as well as new data sets for the eFJSSP are performed to explore how the flexibility and the density of the precedence graphs affect the solution process and the associated solution cost. The new data sets contain medium and large-scale problem instances considering precedence graphs with various densities. These experiments assess the efficiency, effectiveness and scalability of the new solution methods. In every case, we report lower and upper bounds and we discuss the effect of

the precedence graphs on the solution quality as well as on the time needed to produce optimal or near optimal solutions. Lastly, various managerial insights are also extracted.

The remainder of this paper is organized as follows. Section 2 describes the MIP and CP formulations, as well as some definitions and propositions that are used throughout the paper. Section 3 presents the proposed evolutionary algorithm along with its components. In Section 4, we present the details of the computational experiments and results, finally the paper concludes in Section 5 by also providing some future research prospects.

2 Preliminaries

This section provides the basic definitions and, in particular, Section 2.1 introduces the notation that is used throughout the paper and paves the way for Section 2.2 and Section 2.3 that present the MIP and CP models, respectively. In the remainder of the paper, we will refer to the problem as the Flexible Job Shop Scheduling with Arbitrary Precedence Graphs, as opposed to the term eFJSSP, since we believe it is more suitable.

2.1 Notation

The FJSSP with arbitrary precedence graphs can be depicted as follows. There exists a set of jobs $J = \{1, \ldots, l\}$ and a set of available machines $M = \{1, \ldots, m\}$. We define two dummy operations i_u° and i_u^* for each job $u \in J$, which correspond to the first and the last operations of the job respectively. Each job u consists of a set of operations O_u, including the dummy operations. There exists a set Ω that includes all the operations of the problem, $\Omega = \bigcup_{u=1}^{l} O_u$. Let $n = |\Omega|$ denote the total number of operations. Each operation $i \in \Omega$ can be executed on a set of available machines $M_i \subseteq M$ and has a processing time $p_{i,k}$, where $k \in M_i$. Each operation is executed once by a single machine, the machines can execute only one operation at a time and no pre-emption is allowed. Note that for the dummy operations i_u° and i_u^*, we assume $M_{i_u^\circ}$=$M_{i_u^*}$=$\emptyset$ and $p_{i_u^\circ,k} = p_{i_u^*,k} = 0$, for all $k \in M$. The flexibility fx of the problem can be defined as a metric of the degrees of freedom regarding the assignment of operations to different machines, and it can be calculated as $\frac{1}{n}\sum_{i=0}^{n} |M_i|$.

For depicting the FJSSP with arbitrary precedence graphs, we adapt the disjunctive directed graph representation as introduced by Mastrolilli and Gambardella (2000) for the FJSSP. Let 0 represent the start node of the schedule that it is connected to the first operation of each job u of the problem i_u°. Similarly, let $*$ represent the end of the schedule that it is connected to the

last operation i_u^* of each job u.

Definition 1. A disjunctive graph $D(\mathcal{V}, \mathcal{A} \cup \mathcal{E})$ consists of a set of nodes (operations) $\mathcal{V} = \Omega \cup \{0, *\}$, a set of conjunctive directed arcs $\mathcal{A}$ and a set of disjunctive arcs $\mathcal{E}$. The conjunctive directed arcs dictate the relations among the operations of the same job. The disjunctive arcs dictate all possible connections between operations of the same machine.

Figure 1 provides an example of a typical disjunctive graph for the FJSSP with arbitrary precedence graphs within the operations of each job that consists of two jobs ($i_1^\circ - i_1^*$ and $i_2^\circ - i_2^*$), three machines, four operations per job and different precedence graphs per job.

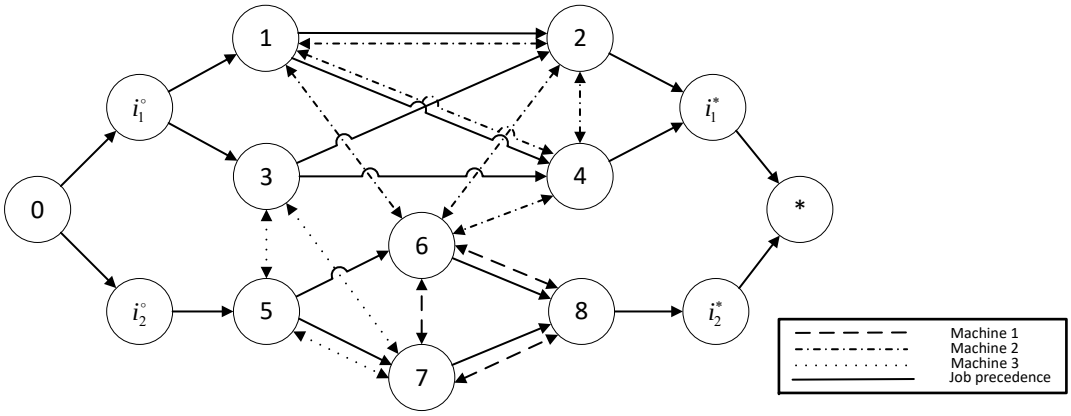


Figure 1: A disjunctive graph of an FJSSP problem with arbitrary precedence constraints with two jobs, three machines and a total of eight operations

For every job $u \in J$, a precedence graph $G_u^P = (O_u, A_u)$ can be defined, where A_u contains the conjunctive arcs that represent the precedence relations between the operations O_u of a job u. Note that $\mathcal{A} = \bigcup_{u=1}^{l} A_u$ and $G_u^P \subset D$. For every operation i, the sets PJ_i and SJ_i denote the immediate job predecessor and successor operations, respectively. Note that $PJ_i \subset \Omega$ and $SJ_i \subset \Omega$. Figure 2 shows an example of a job precedence graph. Sets PJ_i and SJ_i can have an arbitrary cardinality, while in the case that $|PJ_i| \leq 1$ and $|SJ_i| \leq 1$ is true for all $i \in \Omega$, then the problem is equivalent to the simple FJSSP with chain-like precedence graphs. Note that there is no hierarchy on the processing order of the predecessors or successors of an operation, thus all operations of the sets PJ_i and SJ_i are equivalent.

On the basis of the above, the density δ_u of the precedence graph G_u^P of job u is calculated in this paper as follows:

$$\delta_u = \frac{\sum_{j=1}^{|O_u|} |SJ_j|}{|O_u|(|O_u| - 1)} \tag{1}$$

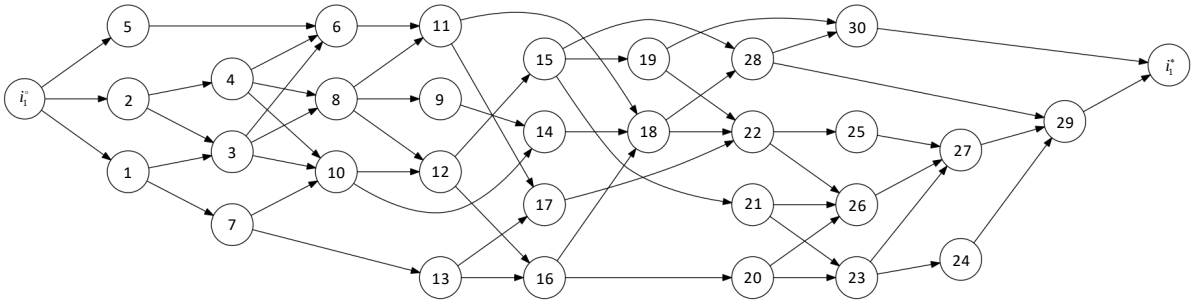


Figure 2: A job precedence graph with 30 operations

The problem's density δ is defined in this paper as the average density of all l precedence graphs as follows:

$$\delta = \frac{1}{l} \sum_{u=1}^{l} \delta_u \tag{2}$$

Given a disjunctive graph $D(\mathcal{V}, \mathcal{A} \cup \mathcal{E})$, an arc $(i, j) \in \mathcal{A} \cup \mathcal{E}$ represents the immediate connection from node i to node j $(i \neq j)$. Set $\mathcal{A}$ can be expressed as follows:

$$\mathcal{A} = \{(i, j), \forall i \in \Omega, \forall j \in SJ_i\} \cup \{(j, i), \forall i \in \Omega, \forall j \in PJ_i\} \tag{3}$$

Set $\mathcal{E}$ can be expressed as the union of all the disjunctive arc sets $\mathcal{E}_k$ for each machine k of the problem.

$$\mathcal{E}_k = \bigcup \{(i, j) : k \in M_i \cap M_j, i \neq j, \forall i, j \in \Omega\}$$
$$\mathcal{E} = \bigcup \mathcal{E}_k, \forall k \in m$$

Note that for the sake of simplicity, we can exclude from sets $\mathcal{A}$ and $\mathcal{E}$ the following sets of arcs: $\{(0, i_u^\circ), \forall u \in J\}$ and $\{(i_u^*, *), \forall u \in J\}$.

Definition 2. A solution s is defined as a pair (α, π), where α is a vector that represents the assignment information of operations to machines and π is a table of vectors that represents the sequence of operations executed at each machine.

More specifically, let $\alpha = \{\alpha(i), \forall i \in \Omega\}$, where $\alpha(i) \in M_i$, and $\pi = \{\pi_k, \forall k \in M\}$, where π_k denotes the permutation of operations processed by machine k. For the sake of completion, every permutation π_k starts and ends with two dummy operations $m_k^\circ, m_k^* \in \Omega$ that denote the start and the end operations of machine k, respectively. Note that $M_{m_k^\circ} = M_{m_k^*} = \{k\}$ and

$p_{m_k^\circ,k} = p_{m_k^*,k} = 0$, for all $k \in M$. Note that given π, one can derive the assignment vector α, but for the sake of simplicity α is also included in the definition of a solution.

Let us consider the problem illustrated by Figure 1. A solution consists of the permutation vectors of the executed operations per machine, for example $\pi_1 = (m_1^\circ, 6, 8, m_1^*)$, $\pi_2 = (m_2^\circ, 1, 4, 2, m_2^*)$, $\pi_3 = (m_3^\circ, 5, 3, 7, m_3^*)$ and $\pi = \{\pi_1, \pi_2, \pi_3\}$. On this basis, one can also determine the assignment vector, in this example $\alpha = (2, 2, 3, 2, 3, 1, 3, 1)$.

We use pm_i (and sm_i) to denote the machine predecessor (and successor) of operation i assigned to machine $\alpha(i)$ in a solution $s(\alpha, \pi)$.

Definition 3. A directed acyclic graph $G = (\mathcal{V}, \mathcal{E}^*) \subset D$ can be used to represent a solution s. The set of directed arcs $\mathcal{E}^* \subset \mathcal{A} \cup \mathcal{E}$ corresponds to the existing machine precedence relations between operations assigned to the same machine as well as the directed arcs that represent precedence relations between operations of the same job.

Definition 4. Given a solution graph $G = (\mathcal{V}, \mathcal{E}^*)$, $\rho(i, j)$ with $i, j \in \Omega$ represents the longest directed path $\{i, o_1, o_2, ..., o_{z-1}, o_z, j\}$ between i and j. The sequence o_1 to o_z denotes all intermediate operations of the corresponding path. The time length of $\rho(i, j)$ is denoted as $L(i, j)$.

Since the graph G is directed, for any path that exists between two operations i, j holds that $L(i, j) > 0$. If there is no path connecting the two operations, then $L(i, j) = 0$. To calculate $L(i, j)$, we need the start and the completion times of the operations in the path. For each operation i its start and completion times can be calculated in $O(n)$ using a labeling algorithm (Adams et al. 1988).

The sets $\mathcal{P}_i$ and $\mathcal{S}_i$ contain all the predecessors and successors of an operation i in the solution graph G, respectively. That is $\mathcal{P}_i = \{v, \forall v \in \Omega : L(v, i) > 0\}$ and $\mathcal{S}_i = \{v, \forall v \in \Omega : L(i, v) > 0\}$. Note that $PJ_i \subset \mathcal{P}_i$ and $SJ_i \subset \mathcal{S}_i$.

Definition 5. The cost of a solution s, namely the makespan of the schedule C_{max}^s, is defined as the length of the longest path from 0 to $*$, i.e., $L(0, *)$. For the sake of brevity, in some cases we may omit the superscript that denotes the corresponding solution.

Definition 6. The head times r_i denote the difference between the start time of the schedule and the start time of an operation i. The head times are also equal to the length of the longest path from node 0 to operation i, i.e., $r_i = L(0, i)$.

Definition 7. The tail times q_i denote the difference between the completion time C_i of the operation i and the makespan C_{max}, i.e., q_i=$C_{max} - C_i$.

The head and tail times can also be determined as follows:

$$r_i = \max\left(\max_{\forall e \in PJ_i}\left(r_e + p_{e,\alpha(e)}\right), r_{pm_i} + p_{pm_i,\alpha(i)}\right) \qquad \forall i \in \Omega \tag{4}$$

$$q_i = \max\left(\max_{\forall e \in SJ_i}\left(q_e + p_{e,\alpha(e)}\right), q_{sm_i} + p_{sm_i,\alpha(i)}\right) \qquad \forall i \in \Omega \tag{5}$$

Definition 8. All operations included in $\rho(0,*)$ are named critical. Critical operations have no flexibility to move back and forth in the scheduling horizon, and thus they define the length of the schedule, i.e., the makespan. An operation i is critical when $C_{max} = r_i + p_{i,\alpha(i)} + q_i$.

Definition 9. A sequence of consecutive operations $B = \{o_1, o_2, ..., o_{e-1}, o_e\} \subseteq \pi_k$ processed on the same machine k is considered as a critical block if all operations $i \in B$ are critical and $|B| \geq 2$.

For the FJSSP with arbitrary precedence graphs, it is also important to provide the definition of the redundant arcs and the maximum density of precedence graphs.

Definition 10. Let i, j be two nodes of a disjunctive graph $D(\mathcal{V}, \mathcal{A} \cup \mathcal{E})$, so that a path $\rho(i,j)$ exists in $\mathcal{A}$. If the conjunctive arc (i,j) also exists in $\mathcal{A}$ and $|\rho(i,j)| \geq 2$, then the arc (i,j) is labeled as redundant.

Given a precedence graph G_u^P of a job u, the maximum density δ_u^{max} represents the density of a fully saturated version of G_u^P that has the maximum possible number of non-redundant arcs. Given an upper bound θ on the number of predecessors and successors per operation $i \in J_u$, the maximum number of non-redundant arcs of G_u^P can be calculated as follows:

$$nRA = \theta + (z_1 - 1)\theta^2 + z_2\theta + \max(z_2, (1 - z_2)\theta) \tag{6}$$

where $z_1 = \lfloor \frac{|O_u|-2}{\theta} \rfloor$ and $z_2 = (|O_u| - 2) \pmod{\theta}$.

Based on nRA, the maximum density of G_u^P as follows:

$$\delta_u^{max} = \frac{nRA}{|O_u|(|O_u| - 1)} \tag{7}$$

2.2 Mixed Integer Programming Formulation

The mathematical formulation presented in this section is inspired by the earlier works of Roshanaei et al. (2013) and Shen et al. (2018). However, one major difference is that we adopt a single index for indexing the operations that leads to a simpler notation.

Let C_i denote the completion time of operation $i \in \Omega$. Let us also define a pair of binary variables $Y_{i,k}$ and $X_{i,j,k}$. The former is equal to one if operation i is assigned to machine k; zero otherwise. The latter is equal to one if operations i and j are assigned to the same machine k and j is processed after i; zero otherwise.

$$\text{minimize } C_{max} \tag{8}$$

subject to:

$$\sum_{k=1}^{m} Y_{i,k} = 1 \qquad \forall i \in \Omega \tag{9}$$

$$C_i \geq C_j + \sum_{k=1}^{m} Y_{i,k} p_{i,k} \qquad \forall i \in \Omega, \forall j \in PJ_i \tag{10}$$

$$C_i \geq C_j + p_{i,k} - \mathcal{M}(2 + X_{i,j,k} - Y_{i,k} - Y_{j,k}) \qquad \forall i, j \in \Omega, \forall k \in M_i \cap M_j \tag{11}$$

$$C_j \geq C_i + p_{j,k} - \mathcal{M}(3 - X_{i,j,k} - Y_{i,k} - Y_{j,k}) \qquad \forall i, j \in \Omega, \forall k \in M_i \cap M_j \tag{12}$$

$$C_i \geq 0 \qquad \forall i \in \Omega \tag{13}$$

$$C_{max} \geq C_{i_u^*} \qquad \forall u \in J \tag{14}$$

$$X_{i,j,k} \in \{0, 1\} \qquad \forall i, j \in \Omega, \forall k \in M \tag{15}$$

$$Y_{i,k} \in \{0, 1\} \qquad \forall i \in \Omega, \forall k \in M \tag{16}$$

The objective (8) is to minimize the makespan C_{max}. Constraints (9) ensure that each operation is assigned to exactly one machine. Constraints (10) ensure that the job precedence relations are satisfied, i.e., an operation is processed after all of its job predecessors are processed. Constraints (11) and (12) enforce that there is no overlap between two operations scheduled on the same machine. Note that the set of constraints (11) and (12) is empty when $M_i \cap M_j = \emptyset$, while $\mathcal{M}$ is a large number. Since the difference between C_j and C_i for any operations i, j cannot exceed the makespan C_{max}, a valid upper bound for $\mathcal{M}$ can be calculated as $\sum_{i \in \Omega} \max_{k \in M_i} p_{i,k}$. Constraints (14) calculate the makespan, while constraints (13) make sure that the completion times are positive. Lastly, constraints (15) and (16) define the variables X and Y. Note that

the dummy machine operations $m_k^\circ, m_k^*, \forall k \in m$ are not considered in the model.

The main difference between the MIP formulation (8) - (16) and the formulation proposed by Birgin et al. (2013) can be found in the representation of the machine precedence constraints. Note that the formulation of Birgin et al. (2013) introduces a smaller number of constraints. A comparison between the two MIP formulations for the FJSSP with arbitrary precedence graphs is provided in Sections 4.4 and 4.6.

2.3 Constraint Programming Formulation

Constraint Programming has been successfully applied for solving various highly constrained and large-scale scheduling problems (Goel et al. 2015, Rasmussen et al. 2017, Unsal and Oguz 2013). The input of a CP model is a set of decision variables, a finite set of alternative values as a domain per decision variable, as well as a set of constraints that have to be satisfied. A CP solver works by enumerating feasible solutions of the problem using branching algorithms. During this process, it also tries to decrease the domain cardinality of each decision variable by propagating through the constraints. Constraint propagation identifies values or combinations of values across multiple decision variables that cannot be part of a feasible solution, and therefore, can be excluded from the domain sets of the corresponding decision variables, which can lead to branch pruning (Laborie et al. 2018).

Specifically, for scheduling applications CP models use *interval* variables. This type of variable is a natural way of describing a task. Interval variables have four attributes: $IsPresent$, $Start$, End and $Size$. $IsPresent$ indicates if the interval variable is included in the solution or not, $Start$ and End denote the start and the end time of the interval variable, i.e., the start and the end time of the task, while $Size$ refers to the size of the interval, i.e., the length of the task.

In our implementation, for each operation i a decision interval variable τ_i is defined. The alternative execution options (modes) of an operation i on a machine $k \in M_i$ are also defined as decision interval variables $\phi_{i,k}$. For these variables a constraint is defined such that the $Size$ attribute of each $\phi_{i,k}$ is equal to the processing time $p_{i,k}$ of i on machine k. We also define a set $\mu_i = \{\phi_{i,k}, \forall k \in M_i\}$ to represent all the available execution modes per operation i, which is also used to denote the domain set of variable τ_i. Lastly, a sequence interval decision variable σ_k is defined per machine k over the set of interval variables $\sigma_k = \{\phi_{i,k}, \forall i \in \Omega\}$.

$$\text{minimize } C_{max} \tag{17}$$

subject to:

$$Alternative(\tau_i, \mu_i) \qquad \forall i \in \Omega \tag{18}$$

$$EndBeforeStart(j, i) \qquad \forall i \in \Omega, \forall j \in PJ_i \tag{19}$$

$$NoOverlap(\sigma_k) \qquad \forall k \in M \tag{20}$$

$$C_{max} \geq EndOf(\tau_i) \qquad \forall i \in \Omega \tag{21}$$

The objective (17) refers to the minimization of the makespan. Constraints (18) are used to enforce a unique selection of the available modes for the interval variable τ_i out of the set μ_i. Constraints (19) are used to cover the precedence relations of the problem, i.e., each operation i can start as soon as all of its job predecessors $j \in PJ_i$ have finished. Constraints (20) ensure that the interval variables included in σ_k do not overlap, since a machine can execute only one operation at a time. They also ensure that each operation starts after its machine predecessor has finished. Lastly, constraint (21) is responsible for the calculation of the makespan. Note that the $EndOf$ function is used to retrieve the End attribute of an interval variable τ_i, i.e. its completion time C_i. Compared to the MIP model described earlier, the number of decision variables of the CP model is smaller. To that end, one may expect smaller memory requirements from the solver side; however, there is no guarantee for the amount of memory that would be actually required during run-time, especially when solving large-scale problem instances. Note that the MIP and CP models are formulated to solve the FJSSP with arbitrary precedence graphs, nevertheless the same formulation can be used to solve the classic FJSSP under the following assumptions:$|PJ_i| \leq 1, |SJ_i| \leq 1, \forall i \in O_u, \forall u \in J$.

3 Evolutionary Algorithm

This section presents an evolutionary algorithm for generating high quality heuristic upper bounds. The proposed algorithm is inspired from PR and SS frameworks (Martí et al. 2006, Glover et al. 2000). These frameworks have been proven to be very efficient in solving various hard combinatorial optimization problems, including a wide variety of scheduling problems (González et al. 2015, Jia and Hu 2014). Our EA amalgamates the basic principles of these frameworks. It is also equipped with several innovative elements, tailored for addressing the FJSSP with arbitrary precedence graphs, which are unpacked in this section. Among them, the

main highlights are the new frequency map and solution distance metrics used for updating the reference set, the improved feasibility check and makespan estimation methods used during local search that are taken from the FJSSP literature, and the new multi-solution PR mechanism used during subset generation.

Algorithm 1 depicts the proposed solution method. During the *initialization phase*, a reference set R is populated with solutions produced via a greedy randomized construction heuristic method that is described in Section 3.1. The selection of reference solutions is based on the makespan and a score based to an arc frequency map. The arc frequency map can be seen as an adaptive memory structure that records the history of the search process (visited solutions). Therefore, high scores indicate that a solution is comprised of more rarely encountered elements (see Sections 3.2 and 3.3 for details). Next, the *evolutionary phase* is triggered and R is evolved for a number of generations. The size of the reference set R is controlled by parameter ζ. At each generation, a set C of recombined candidate solutions is produced with cardinality equal to $|R|$. A PR mechanism is employed for this purpose that randomly selects and combines β solutions from R. To that end, a TS algorithm is applied to each solution for further improvement (*education phase*). The local search process is repeated for $maxIterations$ iterations until no further improvement is observed, while the number of iterations that a local move is considered "tabu" is controlled by parameter ξ (*tabu tenure*). The improved solutions are used to update the arc frequency map and compete to update R. The algorithm terminates after a number of $maxGenerations$ iterations without observing any improvement on the best solution found.

The arc frequency map, the PR and the TS algorithmic components are described in Sections 3.2, 3.4 and 3.5, respectively. Note that λ, ζ, β, ξ, $maxIterations$ and $maxGenerations$ are hyperparameters and termination conditions that are described in the following sections.

3.1 Reference Set Generation Method

In the initialization phase, the goal is to generate an initial reference set of adequately diverse initial solutions. We adopt a greedy randomized solution construction scheme, similar to that of GRASP (Feo and Resende 1995). The solution construction process works as follows: assume an empty solution s; at each iteration an operation i is added into the partially constructed solution s. Note that all machines $k \in M_i$ and all qualified positions j on the permutation of each machine k are examined. Each feasible combination, i.e., that does not introduce cycles to the solution graph, can be seen as a tuple (i, k, j) that is evaluated and the corresponding

Algorithm 1 Evolutionary Algorithm

Require: λ, ζ, β, ξ, $maxIterations$ and $maxGenerations$

$g \leftarrow 0, R \leftarrow \emptyset$
InitializeArcFrequencyMap()
repeat ▷ 1. Initialization phase
  $s \leftarrow$ GreedyRandomisedConstruction(λ)
  UpdateArcFrequencyMap(s)
  $R \leftarrow$ UpdateReferenceSet(s, R)
until $|R| = \zeta$
$s_* \leftarrow argmin_{s \in R} C^s_{max}$
repeat ▷ 2. Evolutionary phase
  $C \leftarrow \emptyset$
  repeat
    $s \leftarrow$ PathRelinking(β, R)
    $C \leftarrow C \cup \{s\}$
  until $|C| = |R|$
  for all $s_c \in C$ **do**
    $s' \leftarrow$ TabuSearch($s_c, \xi, maxIterations$) ▷ 3. Education phase
    UpdateArcFrequencyMap(s')
    $R \leftarrow$ UpdateReferenceSet(s', R)
  end for
  $\acute{s}_* \leftarrow argmin_{s \in R} C^s_{max}$
  if $C^{\acute{s}_*}_{max} < C^{s_*}_{max}$ **then**
    $g \leftarrow 0$, $s_* \leftarrow \acute{s}_*$
  else
    $g \leftarrow g + 1$
  end if
until $g > maxGenerations$
return s_*

makespan, i.e., C^s_{max} is calculated. The solutions derived from these combinations (i, k, j) are stored in a Candidate List (CL) and are sorted in an ascending order of their makespan. At each iteration, we extract a Restricted Candidate List (RCL) with the top ranked partially constructed solutions from CL, we randomly select an element from RCL, and we continue to the insertion of the next operation. The cardinality of RCL is equal to $\lceil \frac{|CL|}{\lambda} \rceil$. The procedure continues until all operations $i \in \Omega$ are scheduled and a complete solution s is produced.

3.2 Frequency Map and Distance Metric

The proposed algorithm introduces an adaptive memory structure in the form of a frequency map that collects information about the elements of improved solutions visited during the search. This is represented by a three dimensional matrix W and the value of an element $W[i, j, k]$ holds the number of times an arc (i, j) has been observed on machine k. This arc frequency map is used to derive a diversity metric for a solution, referred hereafter as *seldomness*. The intuition

is the following: high seldomness scores show that a solution contains few frequent arcs, and thus it can be seen as a more diversified solution, while low scores of seldomness show that a solution contains many frequent arcs, and thus denoting a less diversified solution. The goal is to identify frequent elements of high quality solutions, and also guide the search towards less frequent elements to encourage diversification.

For a solution $s(\alpha, \pi)$ with a solution graph G, the seldomness $d_r(s)$ is calculated by the inverse sum of the normalized frequencies of each arc (excluding conjunctive arcs) that exists in G. This can be expressed by using the set of permutation vectors π of s:

$$d_r(s) = 1 - \sum_{k=0}^{m} \sum_{j=1}^{|\pi_k|-1} \hat{W}\big[\pi_k(j), \pi_k(j+1), k\big] \tag{22}$$

To ensure that the seldomness is comparable across different solutions, we maintain the matrix $\hat{W}$ that contains normalized frequency values. These values are derived by dividing the frequencies of each element with the number of solutions visited during the search as well as the number of arcs of solution s, i.e., $\sum_{k \in M}(|\pi_k| - 1)$. The sum of the normalized frequency values across all the arcs of G is less than one at all times, and therefore, the seldomness metric takes only values from zero to one.

Let us also define the distance between two solutions. A common metric is the so-called hamming distance that considers the number of different arcs between the two solutions (Mattfeld 2013). For the FJSSP, this definition is extended to capture the assignment information of operations to different machines. In particular, González et al. (2015) proposed two different distance metrics for the assignment of operations to machines and the sequencing of the operations on the machines. In a similar fashion, we combine both decision elements and we introduce a metric for measuring the distance $d_h(s, s')$ between two solutions s and s' as follows:

$$d_h(s, s') = \sum_{k}^{M} \sum_{i}^{\Omega} (Y_{i,k}^{s} \oplus Y_{i,k}^{s'}) + \sum_{k}^{M} \sum_{i}^{\Omega} (1 - Y_{i,k}^{s} \oplus Y_{i,k}^{s'}) \sum_{j}^{\Omega} (X_{i,j,k}^{s} \oplus X_{i,j,k}^{s'}) \tag{23}$$

The symbol $\oplus$ is used to describe a XOR operator between two binary variables, i.e., when comparing $Y_{i,k}^{s} \oplus Y_{i,k}^{s'}$ the result is 0 if and only if operation i is assigned on the same machine k on both solutions s, s' or if it not assigned on machine k on both solutions s, s'. Otherwise, if the assignments are different the operator returns 1. Note that superscripts are used in binary variables X, Y to indicate the solution they refer to.

3.3 Reference Set Update Method

The update of R is important since it partially controls the evolution process. The aim is to maintain reference solutions that are high quality in terms of makespan, but also diverse to each other. For this purpose, we resort to the seldomness metric defined earlier. Assume that s_b and s_w denote the best and the worst solutions in the reference set, respectively. Let s_c be a candidate reference solution. If $C_{max}^{s_c} < C_{max}^{s_w}$, then s_c replaces a solution $s_r \in R$ that satisfies the condition $C_{max}^{s_c} \leq C_{max}^{s_r} \wedge d_r(s_c) \geq d_r(s_r)$. When multiple solutions satisfy the condition, then the one with the worst makespan is selected to be removed. Additionally, if these solutions are tied in terms of makespan, then the solution with the lowest seldomness is selected to be removed. If a tie occurs in terms of both the makespan and the seldomness, then the reference set is not updated. Note that, such ties are rare, since the reference set is continuously evolving and the seldomness metric for the reference solutions are updated based on the history of the search. In the special case that $C_{max}^{s_c} < C_{max}^{s_b}$ and the seldomness criterion is not met, then s_c replaces s_w.

3.4 Subset Generation Method

A critical component for the evolution process is the solution recombination mechanism. The aim is to combine components of parent solutions so as to produce high quality, but also diversified subsets of candidate solutions (offspring). Path relinking offers a powerful and flexible framework (Glover et al. 2000, Tarantilis et al. 2012) and it is used to explore trajectories in the solution space between pairs of reference solutions. Starting from an *initial* solution, local moves are performed in an effort to reach the *guiding* solution by reducing at each local search iteration the distance between them. In our framework, the PR is used to systematically generate paths and intermediate solutions from a subset of solutions β.

The proposed subset generation method works as follows: Firstly, β solutions are selected randomly with equal probability. Based on the selection sequence, an ordered set $Z = (s_1, s_2, \ldots, s_\beta) \subset R$ is derived. The order is also random, but we ensure that no replicates of Z re-appear after the end of the process. Based on this order, the PR is applied initially between s_1 and s_2. The objective is to minimize $d_h(s_1, s_2)$, while the path $LS_1(s_1, s_2)$ is generated that consists of all the solutions visited. Among them, we select the best in terms of makespan $s_1^b = argmin_{s \in LS_1} C_{max}^s$. Next, starting from s_1^b the local search path $LS_2(s_1^b, s_3)$ is generated and the best encountered solution $s_2^b = argmin_{s \in LS_2} C_{max}^s$ is returned. This procedure continues

until all $\beta - 1$ paths are explored and the final best $s^b_{\beta-1} = argmin_{s \in LS_{\beta-1}} C^s_{max}$ is returned. Note that during the selection of the best solution of a path, the initial and the guiding solutions are disregarded. Figure 3 provides a visualization of the above PR for three solutions ($\beta = 3$).

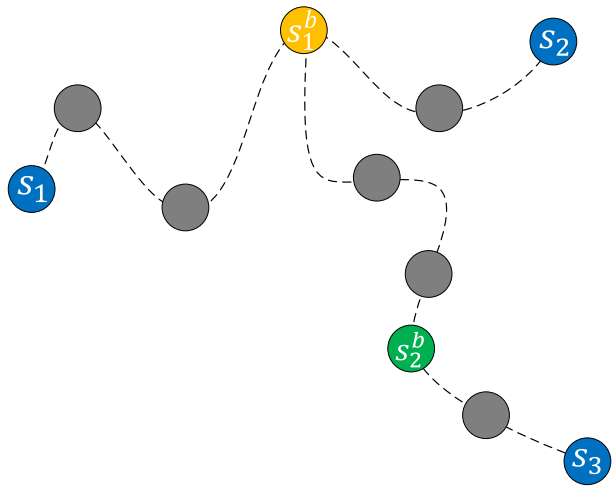


Figure 3: Visualization of PR for a combination of $\beta = 3$ solutions following the ordered set (s_1, s_2, s_3)

Algorithm 2 Path Relinking

Require: : R, β
1: $Z \leftarrow$ SelectAtRandom(R, β)
2: $i \leftarrow 1$
3: $s^b_0 \leftarrow Z(1)$
4: **repeat**
5: $\quad s^b_i \leftarrow$ TabuSearch(s^b_{i-1}, ξ, $maxIterations$) ▷ TS to minimize $d_h(s^b_{i-1}, Z(i+1))$
6: $\quad i \leftarrow i + 1$
7: **until** $i = \beta$
8: **return** $s^b_{\beta-1}$

An overview of the proposed PR mechanism for generating a recombined solution from an ordered solution set Z is presented in Algorithm 2. The TS algorithm that is described later in Section 3.5 is used to generate the search trajectories from an initial solution s^b_{i-1}, which is equal to $Z(1)$ for the first iteration, to a guiding solution $Z(i+1)$. For this purpose, a hierarchical objective is adopted; minimize the distance $d_h(s^b_{i-1}, Z(i+1))$ as primary objective and break ties based on the makespan.

3.5 Local Improvement Method

A TS algorithm is used for improving the recombined solutions derived by the PR. The aim is to explore the solution space by moving at every iteration from a solution s, to the best admissible

solution given a neighborhood structure. In the proposed framework we employ the so-called simple relocation N_1, the critical relocation N_2 and the critical block relocation N_3 (see Section 3.5.1 for details). At each iteration, a neighborhood structure is selected according to an equal probability and the corresponding solution neighborhood is evaluated. To that end, a short term memory records the most recently visited solutions and prevents revisiting them for a number of iterations ξ (tabu tenure). The tabu status can be overridden only if the examined neighboring solution improves the best encountered solution s_{best} (aspiration condition). The overall local search scheme iterates until the termination conditions are met.

Algorithm 3 provides an overview of the proposed local improvement method. Regarding the termination condition, a maximum number of iterations $maxIterations$ without observing any further improvement is imposed. Also note that in Line 5 of Algorithm 3, the best admissible neighboring solution is selected according to a fitness function $\mathcal{F}$ that can be either the makespan, or any weighted or hierarchical combination of solution fitness metrics. The proposed local search is also used for generating paths between an initial and a guiding solution. In that case, the primary objective is to minimize the humming distance of the current neighboring solutions w.r.t. the guiding solution, while the secondary objective is to minimize the makespan.

Algorithm 3 Tabu Search

Require: $s, maxIterations, \xi$
1: $s_{best} \leftarrow s, i \leftarrow 0$
2: **while** $i \leq maxIterations$ **do**
3: $\quad y \leftarrow RandomSelection()$ ▷ $y \in \{1, 2, 3\}$
4: $\quad N_y \leftarrow NeighborhoodEvaluation(y, s)$ ▷ Solution neighborhood
5: $\quad s \leftarrow \arg\min_{s_c \in N_c} \mathcal{F}(s_c)$ ▷ $\mathcal{F}(s)$ represents the fitness function
6: $\quad$ **if** $C^s_{max} < C^{s_{best}}_{max}$ **then**
7: $\quad\quad s_{best} \leftarrow s$
8: $\quad\quad i \leftarrow 0$
9: $\quad$ **else**
10: $\quad\quad i \leftarrow i + 1$
11: $\quad$ **end if**
12: $\quad UpdateTabuList(s, \xi)$
13: **end while**
14: **return** s_{best}

3.5.1 Neighborhood structures

Given a solution $s(\alpha, \pi)$ and its associated graph $G = (\mathcal{V}, \mathcal{E}^*)$, the neighborhood operator y modifies a machine permutation π_k of a mechine k and produces a set of neighboring solutions N_y. The solutions that are derived by these amendments are considered as neighbors of the

solution s and form a neighborhood $N_y(s)$.

All neighborhood structures involve the relocation of a single operation i from its permutation $\pi_{\alpha(i)}$ to the same or a different permutation of operations π_k. Let v, w be two consecutive operations with $v, w \in \pi_k$, $(v, w) \in \mathcal{E}^*$ and $i \neq v, w$. The relocation of an operation i between operations v and w results in a new solution $\acute{s}$ and its corresponding solution graph $\acute{G} = (\mathcal{V}, \acute{\mathcal{E}}^*)$, where $\acute{\mathcal{E}}^* = \mathcal{E}^* \cup \{(pm_i, sm_i), (v, i), (i, w)\} \setminus \{(pm_i, i), (i, sm_i), (v, w)\}$. The feasibility conditions of a relocation move are discussed in Section 3.5.2.

The neighborhood structure N_1 involves the relocation of every operation $i \in \Omega$ to every feasible position of a machine permutation π_k. The size of N_1 is $O(n^2)$. To the contrary, N_2 and N_3 focus only on critical operations and seek to reduce the length of the critical paths (see Definition 8). Similar neighborhood structures also appear in the works of Mastrolilli and Gambardella (2000), Bozejko et al. (2010), González et al. (2015). In particular, N_2 considers the relocation of every critical operation on every feasible position of a machine permutation π_k. Since the number of critical operations is at most n, the size of N_2 is also $O(n^2)$. Nevertheless, in practice only a small subset of operations are critical, and therefore, we expect that N_2 is only a small subset of N_1. Lastly, the neighborhood structure N_3 focuses on the critical blocks of the solution and it is inspired from the earlier work of Zhang et al. (2008) for the JSSP. Let $B = \{o_1, o_2, \ldots, o_{e-1}, o_e\}$ be a critical block. At first, we examine all insertion positions inside the block to relocate o_1 and o_e. For all other "internal" operations starting from o_2 to o_{e-1} we examine their relocation to the first and the last position of the block. The worst case size for N_3 is also $O(n^2)$; however, one may expect that N_3 is significantly smaller than N_2. Similar neighborhood structures that appear in the literature include the reversal of all arcs within critical blocks (van Laarhoven et al. 1992), the relocation of all internal operations of a block to the first and the last positions of the block (Dell'Amico and Trubian 1993), and the reversal of the first and the last arcs of a block (Nowicki and Smutnicki 1996). Note that N_3 works only on the permutations of operations and not on the assignment of operations to machines.

3.5.2 Feasibility checks and objective function evaluation

During the evaluation of a solution neighborhood, a number of relocation moves are a priori infeasible since they introduce cycles in the solution graph. For this reason it is important to detect infeasible relocation moves early so as to avoid the expensive process of recalculating all the start and completion times of all operations. However, the cycle detection for the FJSSP

with arbitrary precedence graphs can be quite challenging. Figure 4 shows six typical examples of infeasible relocation moves of an operation i between two consecutive operations v and w. Cycles are introduced in the graph (denoted with dashed lines), whenever the solution graph contains paths from any of the immediate job successors of i to any predecessor of v, or from the immediate successors of w to any job predecessor of i.

We extend the theorem proposed by Dauzère-Pérès and Paulli (1997), regarding the feasibility of relocation moves for the FJSSP, to capture also the case of arbitrary precedence graphs.

Theorem 1. The relocation of an operation $i \in \pi_{\alpha(i)}$, between two consecutive operations $v, w \in \pi_k$, where $v \notin SJ_i$ and $w \notin PJ_i$ does not create a cycle in the solution graph when all following conditions are true:

i. $r_v < r_e + p_{e,\alpha(e)} \quad \forall e \in SJ_i, v \neq e$

ii. $r_w + p_{w,k} > r_e \quad \forall e \in PJ_i, w \neq e$

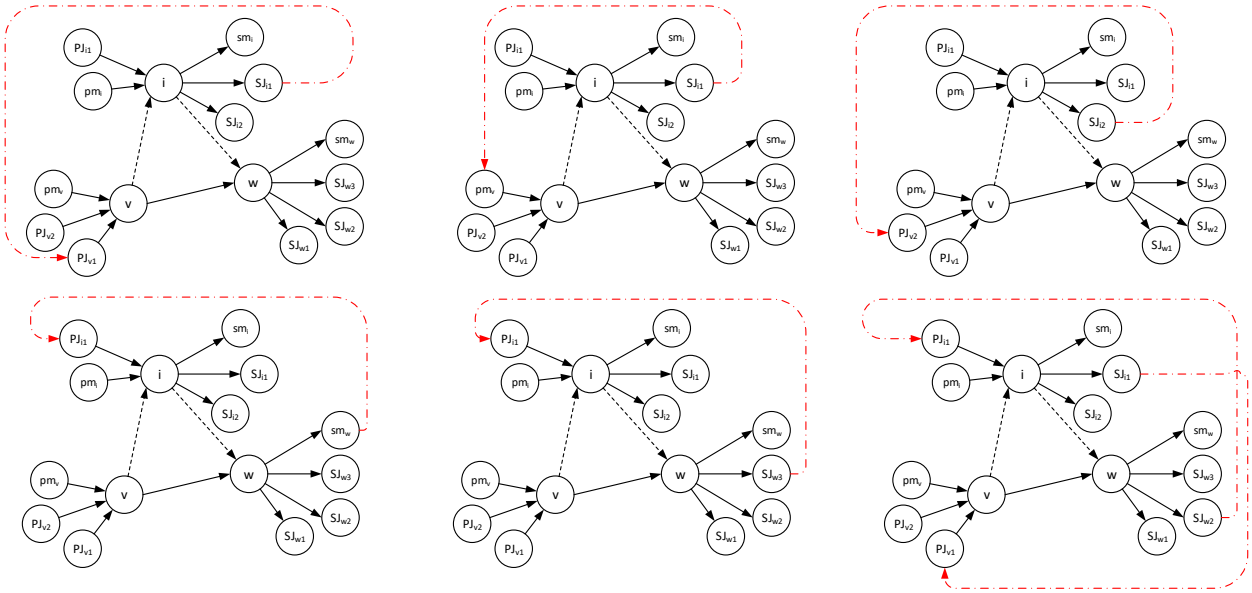


Figure 4: Examples of infeasible moves and occurrence of cycles in the solution graph when relocating operation i between operations v and w

.

Theorem 1 allows a speedier neighborhood evaluation process, since the feasibility of a relocation move is predicted before its application. It is important to highlight that feasible moves can be classified as infeasible due to Theorem 1; however, it is guaranteed that no infeasible move will be classified as feasible. The proof of Theorem 1 is provided in Appendix A, while

computational experiments that assess the classification accuracy of Theorem 1 are presented in Section 4.2.

Apart from the assessment of the feasibility of a solution, another computationally expensive procedure is the calculation of the cost of the move, i.e., the makespan. For this purpose, various methods have been proposed in the JSSP and FJSSP literature for estimating the cost of a relocation move, without the need of recalculating the start, and the completion times of all operations. Dell'Amico and Trubian (1993) introduced the *lpath* method for the JSSP, which estimates the cost of the relocation of an operation i within the permutation $\pi_{\alpha(i)}$. Regarding the relocation of i to a different permutation $\pi_k, k \neq \alpha(i)$, González et al. (2015) proposed a makespan estimate method for the FJSSP inspired by the *lpath* method. We extended the *lpath* method for the FJSSP with arbitrary precedence graphs by considering equations (4) and (5) provided earlier for the calculation of head and tail times.

Dauzère-Pérès and Paulli (1997) also proposed a method to calculate lower bounds for the makespan of relocation moves regarding the FJSSP. In this paper, we introduce an extension of these lower bounds for the case of arbitrary precedence graphs below:

Theorem 2. Assume a relocation of an operation $i \in \pi_{\alpha(i)}$ to the permutation π_k of machine $k \in M_i$, between operations $v, w \in \pi_k$, so that Theorem 1 holds. Then, the makespan of the new solution $\acute{s}$, $C^{\acute{s}}_{\max}$, is always larger or equal than:

$$LB(i,v,w) = \max\left(\hat{r}_v + p_{v,k}, \max_{\forall e \in PJ_i}\left(r_e + p_{e,\alpha(e)}\right)\right) + p_{i,k} + \max\left(\hat{q}_w + p_{w,k}, \max_{\forall e \in SJ_i}\left(q_e + p_{e,\alpha(e)}\right)\right) \tag{24}$$

where

$$\hat{r}_v = \begin{cases} r_v - r_{sm_i} + \max\left(\max_{\forall e \in PJ_{sm_i}}\left(r_e + p_{e,\alpha(e)}\right), r_{pm_i} + p_{pm_i,\alpha(pm_i)}\right) & \text{if } i \in \mathcal{P}_v \\ r_v & \text{if } i \notin \mathcal{P}_v \end{cases} \tag{25}$$

$$\hat{q}_w = \begin{cases} q_w - q_{pm_i} + \max\left(\max_{\forall e \in SJ_{pm_i}}\left(q_e + p_{e,\alpha(e)}\right), q_{sm_i} + p_{sm_i,\alpha(sm_i)}\right) & \text{if } i \in \mathcal{S}_w \\ q_w & \text{if } i \in \mathcal{S}_w \end{cases} \tag{26}$$

The proof of Theorem 2 is provided in Appendix A. For assessing the accuracy of the adaptation of the *lpath* method for the FJSSP with arbitrary precedence graphs and the lower bounds

calculated by Theorem 2, various computational experiments have been conducted that are presented in Section 4.2.

4 Computational Experiments

This section presents various computational experiments for exploring the properties of arbitrary precedence graphs, as well as for evaluating the efficiency and effectiveness of the proposed exact and heuristic solution methods. At first, Section 4.1 discusses the parameter settings and the tuning process. Section 4.2 assesses the accuracy of the theorems discussed in Section 3.5.2. Section 4.3 provides computational experiments as well as a discussion on the performance of the proposed heuristic and exact approaches on existing FJSSP benchmark data sets. Detailed results for the FJSSP, a comprehensive comparative analysis w.r.t. the current state-of-the art approaches of the FJSSP literature, and a performance assessment discussion for the proposed EA can be found in Appendices B, C and D, respectively. Section 4.4 provides a comparative performance analysis on the existing benchmark sets for the FJSSP with arbitrary precedence graphs. Lastly, Sections 4.5 and 4.6 present exact and heuristic solutions obtained for a new benchmark data set with various graph densities. To that end, detailed results for the FJSSP with arbitrary precedence graphs as well as the instance generation process can be found in Appendices E and F, respectively.

4.1 Parameter Settings and Termination Conditions

The proposed evolutionary algorithm uses six user-defined parameters: ζ the size of the reference set, β the number of recombined solutions, $maxIterations$ the number of local search iterations without observing any improvement, ξ the tabu tenure, λ the size of RCL, and $maxGenerations$ the maximum number of generations without improving the overall best found solution.

Based on our computational experience, the proposed framework is quite reliable and one can determine well-performing parameter settings within reasonable value ranges. All computational experiments reported in subsequent sections consider fixed parameters with the following settings: $\zeta = 16$, $\beta = 4$, $maxIterations = 1000$, $\xi = 20$, $\lambda = 4$ and $maxGenerations = 250$. Ten runs of the EA are performed (unless otherwise stated) for each problem instance, using an Intel Xeon E52650 v2 CPU clocked at 2.6GHz, and the overall best solution found is reported. The proposed EA is implemented in C++ using the gcc compiler with the following flags: "-O3 -ffast-math". Note that a limit on the CPU time to 10800 seconds is imposed (unless other-

wise stated) for every run, if the maximum generation limit is not reached. In case the same best solution is produced in different runs, the reported time refers to the quickest run (unless otherwise stated). Below, we provide suitable value ranges for each parameter.

Parameters ζ and β can be used to control the evolution process. Large values of ζ may enhance the diversity of the reference set; however, it may also reduce the convergence velocity. To the contrary, small values may not be adequate to capture the information included in the encountered reference solutions. The reference set size of scatter search schemes of the literature rarely exceeds 30. In our case, a value range between 8 to 20 was found to provide a good compromise for most problem instances. On the other hand, large values for β may allow a more thorough exploration of the search space, nevertheless the computational time invested for the solution recombination will also increase. Values of β up to 4 are appropriate for reference sets with up to 20 solutions.

The termination condition of the local improvement method is critical to the overall performance of the EA. Large values of $maxIterations$ may increase the effectiveness; however, a balance is needed between exploration and exploitation since large values may result in excessive computational times invested for local search, without providing sufficient time to the evolution process. Regarding ξ, a value range from 10 to 30 seems to work well, while for λ values between 3 to 6 provided a good sampling of the initial reference set for most of the problem instances.

Overall, we did not observe any significant differences or patterns during our experimentation when we tried different parameter settings for the FJSSP, with or without arbitrary precedence graphs. To that end, for selecting the final set of parameters the following process was followed. We chose a subset of hard-to-solve large-scale FJSSP problem instances given that tighter lower bounds are available to compare the heuristic upper bounds of the proposed EA. We used the CP % optimality gap as the measure of difficulty to solve. In total, 10 problems were selected, namely the $mk6$ and the $mk13$ from the $BRDATA$ set, the $10a$ and the $16a$ from the $DPDATA$ set, the $abz7$ and the $abz8$ from the $HUEData$ set, the $abz7$ and the $car1$ from the $HURData$ set and the $abz7$ and the $abz8$ from the $HUVData$ set. Each of these problems was solved considering 36 combinations of parameter settings: $\zeta \in \{8, 12, 16, 20\}$, $\beta \in \{2, 3, 4\}$ and $maxIterations \in \{200, 500, 1000\}$. Note that ten runs were performed for each combination considering time limits of 600 and 10800 seconds, while the remaining parameters were fixed (i.e., $\xi = 20$, $\lambda = 4$ and $maxGenerations = 250$). Overall, the combination ($\zeta = 16$, $\beta = 4$, and $maxIterations = 1000$) seemed to provide a good trade-off for both time limits.

In terms of implementation, the MIP and CP models were coded with IBM's Optimization Programming Language (OPL) and solved using the MIP and CP solvers of IBM ILOG CPLEX version 12.8.0. The default parameter settings were considered for both solvers, the number of threads was set equal to one, and no starting solution was used. Lastly, a time limit of 10800 seconds was imposed (unless otherwise stated) and the optimality tolerance was set to 0.01%.

4.2 Accuracy of feasibility prediction and makespan estimation methods

This section discusses the accuracy and impact of the feasibility detection and the makespan estimation methods presented in Section 3.5.2 for the FJSSP with arbitrary precedence graphs.

Initially, we examine the classification accuracy and the predictability of infeasible relocation moves based on Theorem 1. For different problem instances and starting from randomly generated solutions (five per instance) as described in Section 3.1, we evaluated the generic relocation structure (N_1) and we examined the feasibility of all corresponding neighboring solutions. Overall, during this experiment we examined ten million neighboring solutions and the classification results are summarized in the form of a confusion matrix. Table 1 provides the test outcome as well as the True Positive Rate (TPR), the True Negative Rate (TNR), the Positive Predictive Value (PPV), the Negative Predictive Value (NPV), and the Accuracy (ACC). Theorem 1 can predict if a move is feasible (positive) or not (negative). However, it is worth highlighting that the negative outcome of Theorem 1 does not provide any guarantee that the move is actually infeasible. There is a guarantee only for the positive outcome.

Table 1: Confusion matrix for the classification assessment of Theorem 1

	Actual Feasible	Actual Infeasible	
Predicted Feasible	2557936	0	100% (PPV)
Not Predicted Feasible	884462	6510841	88.04%(NPV)
	74.31% (TPR)	100.00% (TNR)	91.11%(ACC)

The results show that the overall accuracy, i.e., the sum of true positives and true negatives, is 91.11% w.r.t. the total population. More importantly, there are no false positives, i.e., cases where Theorem 1 predicted incorrectly an actual infeasible neighboring solution as feasible. On the other hand, there is a 8.89% of the total population that are false negatives w.r.t. the total population. To that end, one needs to examine whether this prediction inaccuracy is counterbalanced from the benefit of early feasibility detection during the neighborhood evaluation process (see Table 3).

Subsequently, we assess the accuracy of the lower bound method based on Theorem 2 and

the extended *lpath* method for estimating the makespan prior to the actual application of a relocation move. For this purpose, we repeated the previous experiment, but this time we examined the accuracy of the proposed makespan estimation methods.

Table 2 summarizes the obtained results. The first column refers to the makespan estimation methods. Each of the next three columns represents a possible outcome of the comparison between the value of the estimate and the actual move cost, that is $C_{est} < C_{max}$, $C_{est} = C_{max}$ and $C_{est} > C_{max}$, respectively. The reported values are expressed in terms of percentages w.r.t. the total number of makespan evaluations. Lastly, the rightmost column contains the computational time in microseconds μs that each method requires to evaluate a move.

Results suggest that the extended *lpath* method is more accurate compared to the lower bound estimation method. Specifically, the extended *lpath* method accurately calculates the cost of relocation moves at a percentage of 88.25%, compared to 60.40% that is observed for the lower bound method. A large portion of moves for both methods consists of predictions that are lower than the actual cost of the move. Paying particular attention to the extended *lpath* method, there is also a small percent of moves, that is 2.75%, where the estimated makespan is larger than the actual cost. On the other hand, the lower bound estimation method based on Theorem 2 is five times faster.

Table 2: Accuracy results of makespan estimation methods

Estimation Methods	$< C_{max}(\%)$	$= C_{max}(\%)$	$> C_{max}(\%)$	Time(μs)
Lower bound (Theorem 2)	39.60	60.40	0.00	0.01
Extended *lpath*	9.00	88.25	2.75	0.05

Table 3: Impact of feasibility prediction and makespan estimation methods on neighborhood evaluation

Early feasibility detection	Makespan estimation method	Speed-up
None	None	1.0
Theorem 1	None	2.72
Theorem 1	Extended *lpath*	26.76
Theorem 1	Lower Bound (Theorem 2)	136.48

The above described feasibility prediction and the makespan estimation methods can significantly accelerate the neighborhood evaluation process. Overall, one may identify four cases as depicted in Table 3. For each case, an experiment is conducted using the above specifications. In particular, we measure the average elapsed time for the evaluation of a single neighboring solution. On this basis, we measure the relative speed up observed, compared to the baseline case of not involving any method. Overall, one can observe that the effect of makespan estima-

tion methods is significant. Note that when the lower bound estimation method is employed the neighborhood evaluation is approximately 136 times faster compared to the regular procedure of calculating the start and the completion times of all operations (baseline).

It is worth noting that both combinations of Theorem 1 with the extended *lpath*, as well as with the Lower Bound, prove to be effective and may significantly help the performance of local search based improvement methods. In our computational experiments, we found that it is better to trade efficiency (computational time) for accuracy (solution quality). More specifically, we found that overall using the slower but more accurate extended *lpath* method for estimating the makespan of local moves, is a better choice compared to the much quicker, but less accurate Lower Bound makespan estimation method. However, if efficiency and speed are more important, then the Lower Bound method is superior. All computational experiments reported in subsequent sections adopt the Theorem 1 coupled with the extended *lpath* method.

4.3 Computational results for the FJSSP

This section provides exact and heuristic results for the FJSSP produces by the MIP and CP models as well as the proposed evolutionary algorithm, that is abbreviated hereafter as EA. For this purpose, we use the well-known benchmark data sets of Brandimarte (1993) (BRData), Dauzère-Pérès and Paulli (1997) (DPData), Barnes and Chambers (1996) (BCData) and Hurink et al. (1994) (HUData). Note that the HUData set contains three groups of problems, namely *edata*, *vdata* and *rdata*.

Summarized results, grouped by benchmark set, are provided in Table 4. The first set of columns includes the benchmark set information and more specifically, the name, the number of instances, the average instance flexibility (fx) and the average known lower bound (LB) respectively. The next sets of columns include results for the CP, MIP and EA approaches, respectively. C_{max} refers to the average makespan of the best solution found for every problem instance of each data set. The Gap(%) refers to the average optimality gap recorded from the tightest available lower bounds. A single run is performed for the CP and MIP approaches, and we record the best solution found either before the time limit is reached, or when optimality conditions are met. On the other hand, regarding the EA, 10 runs are performed for each problem instance and we record the best found solution. The three bottom rows of the table provide the average optimality gap(%), the number of proven optimal solutions, and the number of new best solutions obtained by each solution method respectively.

Table 4: Summary of computational experiments on FJSSP benchmark data sets

Benchmark Data Sets					CP		MILP		**EA**	
Name	Instances	Size	fx	LB	C_{max}	Gap (%)	C_{max}	Gap (%)	C_{max}	Gap (%)
BRData	15	60-300	2.45	277.3	284.6	5.87	335.9	24.87	283.4	5.30
HURData	66	15-75	1.97	1420.2	1428.3	0.96	1499.5	5.43	1425.8	0.63
HUVData	66	15-75	4.13	1365.3	1366.0	0.07	1487.8	12.88	1366.0	0.10
HUEData	66	15-75	1.15	1694.1	1697.4	0.47	1735.5	3.39	1696.5	0.33
CBData	21	100-225	1.18	995.2	995.2	0.00	997.5	0.20	995.5	0.03
DPData	18	200-387	2.49	2172.3	2212.1	1.87	-	-	2196.3	1.13
Average Gap (%)					1.54		-		1.25	
# Optimal Solutions					180		104		179	
# NB					14		0		32	

Clearly, the CP seems to be more effective compared to the MIP considering both upper and lower bounds generated across all data sets. This is evident from the difference in the Gap(%) recorded between the two methods. Furthermore, in many cases the MIP failed even to produce feasible upper bounds within the time limit. The flexibility fx appears to be an important indicator of the difficulty of the problem, especially when solving the MIP model. For example, the CP manages to close the gap in all problem instances of the BCData set, while problem instances with high flexibility are much harder to solve. On the other hand, the EA was able to find high quality solutions within reasonable computational times. For the vast majority of problem instances, the EA successfully matched the optimal solutions, or provided heuristic solutions with small gaps compared to the CP and MIP lower bounds. In total, the EA was able to find 32 new best solutions, while the CP managed to update 49 lower bounds for a total of 178 FJSSP instances. In terms of the number of optimal solutions, CP performed equally well. Furthermore, the EA seemed to scale well as it was able to produce high quality solutions relatively early in the search process, and well before the time limit is reached, even for large scale problem instances. It is also important to highlight that the flexibility of the problem seems to have little or no effect in the performance of the EA. In particular, an average gap of 1.25% was observed for the EA considering the best run per instance. Also, when all ten runs per instance are considered, the average gap was 1.36%, while when only the worst run per instance is considered the average gap is 1.46%. This is a strong indication of a low variability of the solutions obtained by the EA. Lastly, the worst performance of the EA among all data sets and considering all ten runs per instance was recorded for the BRData set, with an average gap of 5.49%. Detailed results for all FJSSP benchmark data sets as well as comparative performance analysis with state of the art approaches are provided in Appendix B.

4.4 Comparisons to the state of the art for the FJSSP with arbitrary precedence graphs

This section presents the results obtained by the proposed exact and heuristic methods on the benchmark data set of Birgin et al. (2013). This set includes two groups of problem instances, namely the DAFJS and the YFJS. More specifically, the authors have generated these problems using six types of directed acyclic graphs (DAG) to generate the precedence graphs for DAFJS and YFJS instances. For the YFJS instances, only one type is adopted, while combinations of these DAG types are used for the DAFJS problems. The YFJS problems are mostly linear, but there is one operation on every job that actually splits execution into two parallel linear streams. The DAFJS are more generalised, but again due to the predefined nature of the DAG types used, they are also limited in terms of the density of the precedence graphs.

Tables 5 and 6 present the results of the CP, MIP and EA for the DAFJS and YFJS data sets. The first four columns contain the name, the size ($l \times m$), the flexibility (fx) and the best available lower bounds, respectively. The lower bounds are calculated using the maximum values between the lower bounds from the proposed MIP and CP models as well as the reported lower bounds in the work of Birgin et al. (2013). The remaining three columns provide details regarding the solutions produced by the CP, the MIP and the EA, respectively. In particular, C_{max} is the upper bound and the Gap(%) is the optimality gap that is calculated as $\frac{C_{max}-LB}{LB}\%$ for every problem instance. The column $Time$ refers to the CPU time elapsed in seconds either to fully close the gap, regarding the CP and MIP, or to obtain the best feasible heuristic solution. The last two rows report the Average Gap (%) w.r.t the lower bound and the number of new best solutions obtained by the each method. As previously stated, the CP and MIP approaches use a single run, and the best solution found either before the time limit is reached, or when optimality conditions are met is reported. Also, regarding the EA, 10 runs are performed for each problem instance and the best solution found is reported.

One can observe in Table 5 that the CP is more effective than the MIP. The EA is capable of producing high quality solutions in reasonable computational times, especially on the largest instances of the DAFJS set. In most cases, the EA is capable of finding better solutions compared to the CP. An average gap of 29.14% was recorded, considering the best run per instance. Additionally, when all ten runs are considered the average gap was 29.51%, while when only the worst run is considered the average gap was 29.97%. The results are different for the YFJS set (see Table 6) that includes problems with quite high numbers of machines. It is worth mentioning

that the CP performs very well, as it managed to close the gap for all instances in very short computational times. Similar results are observed for the EA and it is worth mentioning that all ten runs per instance converged to the optimal solution.

Table 5: Computational results for the DAFJS data set

Problem instances				CP			MIP			EA(10800)		
Name	$l \times m$	fx	LB	C_{max}	Gap(%)	Time	C_{max}	Gap(%)	Time	C_{max}	Gap(%)	Time
DAFJS01	4x5	3.15	257	**257***	0.00	1	**257***	0.00	6	**257***	0.00	1
DAFJS02	4x5	3.16	289	**289***	0.00	2	**289***	0.00	121	**289***	0.00	1
DAFJS03	4x10	5.07	576	**576***	0.00	0	**576***	0.00	3	**576***	0.00	1
DAFJS04	4x10	5.12	606	**606***	0.00	0	**606***	0.00	2	**606***	0.00	1
DAFJS05	6x5	2.67	384	**384***	0.00	4	**384***	0.00	3493	**384***	0.00	1
DAFJS06	6x5	3.09	404	**404***	0.00	454	410	1.49	10800	**404***	0.00	3
DAFJS07	6x10	5.07	505	**505***	0.00	12	530	4.95	10800	**505***	0.00	56
DAFJS08	6x10	4.74	628	**628***	0.00	0	**628***	0.00	30	**628***	0.00	1
DAFJS09	8x5	3.00	324	461	42.28	10800	468	44.44	10800	**460**	41.98	68
DAFJS10	8x5	2.90	337	519	54.01	10800	546	62.02	10800	**516**	53.12	213
DAFJS11	8x10	4.73	658	**658***	0.00	21	666	1.22	10800	**658***	0.00	2
DAFJS12	8x10	5.15	530	602	13.58	10800	658	24.15	10800	**588**	10.94	1109
DAFJS13	10x5	3.11	306	635	107.52	10800	699	128.43	10800	**634**	107.19	195
DAFJS14	10x5	2.99	367	715	94.82	10800	736	100.54	10800	**708**	92.92	263
DAFJS15	10x10	4.96	512	640	25.00	10800	691	34.96	10800	**626**	22.27	512
DAFJS16	10x10	5.02	641	646	0.78	10800	711	10.92	10800	**642**	0.16	254
DAFJS17	12x5	3.00	309	776	151.13	10800	820	165.37	10800	**771**	149.51	405
DAFJS18	12x5	3.12	328	771	135.06	10800	828	152.44	10800	**766**	133.54	309
DAFJS19	8x7	4.04	512	**512***	0.00	95	538	5.08	10800	**512***	0.00	5
DAFJS20	10x7	3.92	434	675	55.53	10800	726	67.28	10800	**660**	52.07	636
DAFJS21	12x7	3.97	504	761	50.99	10800	880	74.60	10800	**755**	49.80	829
DAFJS22	12x7	3.88	464	672	44.83	10800	840	81.03	10800	**659**	42.03	2638
DAFJS23	8x9	4.83	450	468	4.00	10800	495	10.00	10800	**461**	2.44	227
DAFJS24	8x9	5.03	476	540	13.45	10800	596	25.21	10800	**533**	11.97	656
DAFJS25	10x9	5.03	584	704	20.55	10800	832	42.47	10800	**689**	17.98	228
DAFJS26	10x9	5.09	565	705	24.78	10800	746	32.04	10800	**681**	20.53	1220
DAFJS27	12x9	4.92	503	774	53.88	10800	839	66.80	10800	**768**	52.68	1281
DAFJS28	8x10	5.02	535	**535***	0.00	671	542	1.31	10800	**535***	0.00	22
DAFJS29	8x10	4.93	609	**618**	1.48	10800	650	6.73	10800	620	1.81	197
DAFJS30	10x10	5.19	467	522	11.78	10800	588	25.91	10800	**519**	11.13	1795
Average Gap (%)				30.18			39.23			29.14		
# NB				25			4			25		

Tables 7 and 8 compare the results obtained from the EA considering different time limits (i.e., 50, 200 and 3600 seconds), the proposed MIP model considering a time limit of 3600 seconds, the exact MIP approach of Birgin et al. (2013) and the metaheuristic Firefly Algorithm (FA) of Lunardi and Voos (2018) for the FJSSP with arbitrary precedence graphs. The proven optimal solutions are marked with the $(*)$ symbol, while bold face is used to indicate the best upper bounds. The last rows provide information for the Average Gap (%) w.r.t the lower bound, the number of new best solutions obtained by the EA, as well as the sum of the average CI-CPU (computer independent CPU) times per problem instance for every data set.

Regarding the calculation of CI-CPU times, it is common in the literature to use normalization coefficients from Dongarra (1992). However, our CPU is not included in Dongarra (1992), and for this reason we used the single thread CPU rating as reported at *cpubenchmark.net* to derive the normalization coefficient for our machine. The computational times of FA and MIP are used as the baseline, and in particular we have used a normalization coefficient of

Table 6: Computational results for the YFJS data set

Problem instances				CP			MIP			EA(10800)		
Name	$l \times m$	fx	LB	C_{max}	Gap(%)	Time	C_{max}	Gap(%)	Time	C_{max}	Gap(%)	Time
YFJS01	4x7	2.60	773	**773***	0.00	0	**773***	0.00	6	**773***	0.00	1
YFJS02	4x7	2.60	825	**825***	0.00	0	**825***	0.00	6	**825***	0.00	1
YFJS03	6x7	2.63	347	**347***	0.00	0	**347***	0.00	3	**347***	0.00	1
YFJS04	7x7	2.54	390	**390***	0.00	0	**390***	0.00	3	**390***	0.00	1
YFJS05	8x7	2.53	445	**445***	0.00	1	**445***	0.00	102	**445***	0.00	1
YFJS06	9x7	2.64	446	**446***	0.00	3	**446***	0.00	2895	**446***	0.00	1
YFJS07	9x7	2.58	444	**444***	0.00	1	**444***	0.00	75	**444***	0.00	1
YFJS08	9x12	2.78	353	**353***	0.00	0	**353***	0.00	2	**353***	0.00	1
YFJS09	9x12	6.08	242	**242***	0.00	0	**242***	0.00	17	**242***	0.00	1
YFJS10	10x12	2.83	399	**399***	0.00	1	**399***	0.00	4	**399***	0.00	1
YFJS11	10x10	2.68	526	**526***	0.00	1	**526***	0.00	11	**526***	0.00	3
YFJS12	10x10	2.66	512	**512***	0.00	2	**512***	0.00	65	**512***	0.00	1
YFJS13	10x10	2.74	405	**405***	0.00	1	**405***	0.00	58	**405***	0.00	1
YFJS14	13x26	2.90	1317	**1317***	0.00	2	**1317***	0.00	679	**1317***	0.00	5
YFJS15	13x26	2.93	1239	**1239***	0.00	1	**1239***	0.00	9741	**1239***	0.00	16
YFJS16	13x26	2.86	1222	**1222***	0.00	6	1225	0.25	10800	**1222***	0.00	5
YFJS17	17x26	4.60	1133	**1133***	0.00	2	**1133***	0.00	10800	**1133***	0.00	8
YFJS18	17x26	4.71	1220	**1220***	0.00	3	1261	3.36	10800	**1220***	0.00	8
YFJS19	17x26	4.66	926	**926***	0.00	37	1008	8.85	10800	**926***	0.00	265
YFJS20	17x26	4.65	968	**968***	0.00	82	993	2.58	10800	**968***	0.00	92
Average Gap (%)				0.00			-			0.00		
# NB				2			0			2		

1.008=(1248/1238) based on the single thread rating of the corresponding machines. As indicated by both Yuan et al. (2013) and González et al. (2015), the comparison between CI-CPU times is meant to be indicative, because we do not have access to other information that influences the computation time, such as the operating systems, the programming language, the compiler selection, and the overall code quality. Note that the average CI-CPU time for the EA refers to the average running time across 10 runs per problem instance.

The computational results demonstrate the efficiency and effectiveness of the proposed EA. Our method successfully found the optimal solutions for 11 out of the 30 instances of the DAFJS set, and was able to improve upon the existing upper bounds on the remaining problem instances of the set. The results are similar for the YJFS set. The EA finds the optimal solution for the two largest problem instances. Regarding the two exact approaches, one will notice that the average optimality gap recorded for the proposed MIP approach is lower compared to the MIP of Birgin et al. (2013) for the DAFJS and the YFJS data sets. In terms of computational times, the results demonstrate that the EA has the best average optimality gap with very competitive CI-CPU times.

4.5 Effect of flexibility and density for the FJSSP with arbitrary precedence graphs

This section examines the effect of the flexibility of the machines and the density of the precedence graphs on the solution cost, as well as the time needed to solve the problem to optimality.

Table 7: Comparative analysis for the DAFJS data set

Name	LB	EA(50)	EA(200)	EA(3600)	MIP(3600)	MIP(Birgin et al.)	FA
DAFJS01	257	**257***	**257***	**257***	**257***	**257***	**257***
DAFJS02	289	**289***	**289***	**289***	**289***	**289***	**289***
DAFJS03	576	**576***	**576***	**576***	**576***	**576***	**576***
DAFJS04	606	**606***	**606***	**606***	**606***	**606***	**606***
DAFJS05	384	**384***	**384***	**384***	**384***	403	389
DAFJS06	404	**404***	**404***	**404***	415	435	412
DAFJS07	505	506	**505***	**505***	536	562	512
DAFJS08	628	**628***	**628***	**628***	**628***	631	**628***
DAFJS09	324	461	**460**	**460**	482	475	464
DAFJS10	337	518	517	**516**	562	575	533
DAFJS11	658	**658***	**658***	**658***	713	708	659
DAFJS12	530	598	592	**588**	727	720	645
DAFJS13	306	635	**634**	**634**	723	708	653
DAFJS14	367	710	710	**708**	786	860	726
DAFJS15	512	637	628	**626**	781	818	671
DAFJS16	641	648	643	**642**	728	819	679
DAFJS17	309	775	774	**771**	864	909	787
DAFJS18	328	769	767	**766**	886	951	789
DAFJS19	512	**512***	**512***	**512***	541	592	524
DAFJS20	434	665	662	**660**	759	815	696
DAFJS21	504	765	758	**755**	962	965	803
DAFJS22	464	669	661	**659**	817	902	697
DAFJS23	450	464	462	**461**	502	538	476
DAFJS24	476	540	535	**533**	650	666	564
DAFJS25	584	705	693	**689**	868	897	752
DAFJS26	565	688	683	**681**	844	903	745
DAFJS27	503	781	773	**768**	958	981	831
DAFJS28	535	**535***	**535***	**535***	601	671	543
DAFJS29	609	632	620	620	754	726	654
DAFJS30	467	527	521	**519**	591	656	555
Average Gap (%)		30.03	29.40	29.14	46.10	52.16	34.05
# NB		25	25	25	-	-	-
CI-CPU		1428	4786	11897	90240	94968	1372

Table 8: Comparative analysis for the YFJS group

Name	LB	EA(50)	EA(200)	EA(3600)	MIP(3600)	MIP(Birgin et al.)	FA
YFJS01	773	**773***	**773***	**773***	**773***	**773***	**773***
YFJS02	825	**825***	**825***	**825***	**825***	**825***	**825***
YFJS03	347	**347***	**347***	**347***	**347***	**347***	**347***
YFJS04	390	**390***	**390***	**390***	**390***	**390***	**390***
YFJS05	445	**445***	**445***	**445***	**445***	**445***	**445***
YFJS06	446	**446***	**446***	**446***	**446***	449	**446***
YFJS07	444	**444***	**444***	**444***	**444***	**444***	**444***
YFJS08	353	**353***	**353***	**353***	**353***	**353***	**353***
YFJS09	242	**242***	**242***	**242***	**242***	**242***	**242***
YFJS10	399	**399***	**399***	**399***	**399***	**399***	**399***
YFJS11	526	**526***	**526***	**526***	**526***	**526***	**526***
YFJS12	512	**512***	**512***	**512***	**512***	**512***	**512***
YFJS13	405	**405***	**405***	**405***	**405***	**405***	**405***
YFJS14	1317	**1317***	**1317***	**1317***	**1317***	**1317***	**1317***
YFJS15	1239	**1239***	**1239***	**1239***	**1239***	1244	**1239***
YFJS16	1222	**1222***	**1222***	**1222***	1237	1245	**1222***
YFJS17	1133	**1133***	**1133***	**1133***	1145	2379	**1133***
YFJS18	1220	**1220***	**1220***	**1220***	1265	2082	**1220***
YFJS19	926	938	928	**926***	1034	1581	941
YFJS20	968	970	**968***	**968***	1066	2312	973
Average Gap (%)		0.08	0.01	0.00	1.39	19.66	0.11
# NB		2	2	2	-	-	-
CI-CPU		710	1065	6847	19953	35316	886

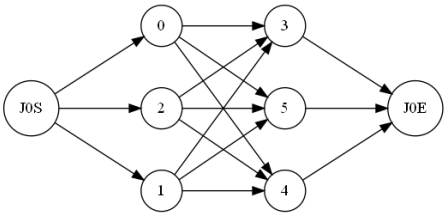
For this purpose, a new set of problem instances was generated with various precedence graph structures and different flexibility values. The following procedure is used to generate the new set of problem instances. Initially, the number of operations was chosen to be from 20 to 30, while the number of jobs was set equal to two. These settings are selected so that each job is well-populated with operations, and on return, this will allow the generation of rich precedence graphs without significantly increasing the difficulty of solving the problem. On this basis, three graph structures are defined using different values of the maximum number of successors and predecessors of each operation, i.e., $\theta = 2, 3, 4$, respectively. Given these θ values, three different base problems are generated with the maximum possible density δ_u^{max}. All jobs share the same precedence graphs, and therefore, $\delta^{max} = \delta_u^{max} \quad \forall u \in J$. Next, we iteratively remove arcs in order to generate precedence graphs of lower densities. This procedure is repeated, until the job precedence graph is reduced to linear streams of operations (see for example Figure 5). By reducing the density of the precedence graphs, three groups of problems emerge, namely d1–d29, t1–t36 and q1–q50. In addition, three versions of each problem instance are produced with different flexibility values allowed per operation, i.e., $fx = 1, 2, 3$. Note that each instance version with $fx > 1$ includes the machine assignments of the corresponding instance with less flexibility, and therefore, the results between the same instance with different flexibility are comparable.

A visualization of the results obtained from the above experiment is presented in Figure 6. Note that the computational time in the subfigures d and f are scaled with a factor of 0.1 to better fit in the plot, while all detailed results can be found in Tables E.1 – E.3 of Appendix E. To that end, Table 9 provides the corresponding summary statistics. Particularly, the first set of columns provide the range of problem instances as well as the flexibility fx and θ parameters, the second set of columns provides the slope and the intercept of the linear regression line from the density (x-values) and makespan (y-values), while the last sets of columns provide the minimum, average and maximum values of the density, the makespan and the elapsed run time (in seconds), respectively.

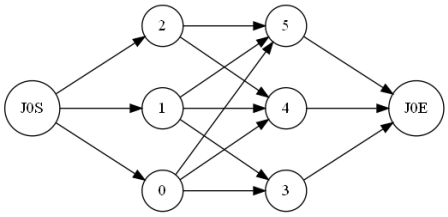
In the first instance, one can observe that as the density of the precedence graphs increases, the C_{max} also increases. This is plausible since the problem becomes more constrained and the completion times of the operations, as well as the C_{max}, are typically prolonged. Furthermore, this effect becomes stronger as the fx decreases (see Figures 6a, 6c and 6e). The average rate of change of the makespan w.r.t. the density is depicted by the Slope values in Table 9. In most cases as the machine flexibility increases, the slope decreases. This shows that larger flexibility

Table 9: Summary statistics for the computational experiments on the machine flexibility and the density of the precedence graphs

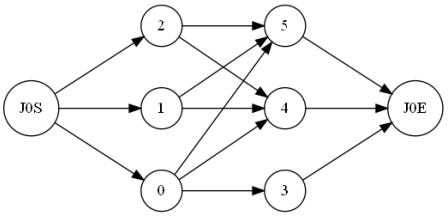
Group			Linear Regression		δ			C_{max}			Time		
Range	fx	θ	Slope	Intercept	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
d1–d29	1	2	3515.49	523.09	0.03	0.05	0.06	615	686.10	721	1	2	6
d1–d29	2	2	1166.46	362.53	0.03	0.05	0.06	399	416.62	428	1	4	11
d1–d29	3	2	1193.33	327.28	0.03	0.05	0.06	367	382.62	397	1	7	35
t1–t36	1	3	816.51	377.98	0.05	0.08	0.12	416	446.58	475	0	1	2
t1–t36	2	3	555.65	227.76	0.05	0.08	0.12	253	274.44	287	3	12	63
t1–t36	3	3	413.41	212.32	0.05	0.08	0.12	235	247.06	260	20	493	5071
q1–q50	1	4	656.73	341.92	0.05	0.10	0.16	373	411.72	442	0	1	2
q1–q50	2	4	167.02	223.22	0.05	0.10	0.16	231	241.02	248	7	30	139
q1–q50	3	4	124.44	205.45	0.05	0.10	0.16	214	218.53	226	138	1279	18875



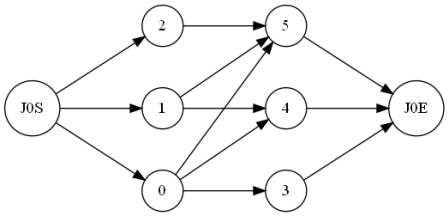
(a) $\delta = \delta_{max} = 0.27$



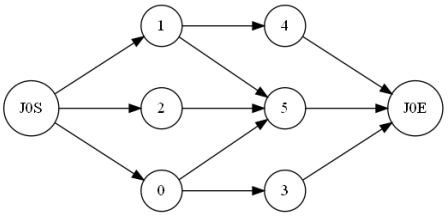
(b) $\delta = 0.25$



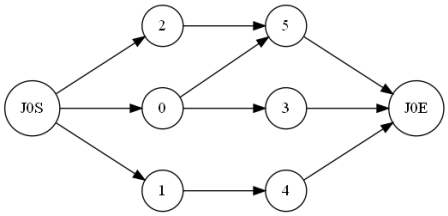
(c) $\delta = 0.23$



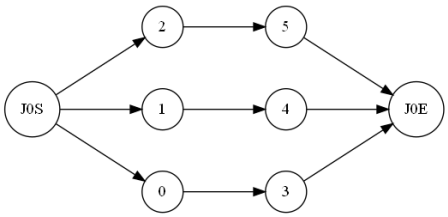
(d) $\delta = 0.21$



(e) $\delta = 0.20$

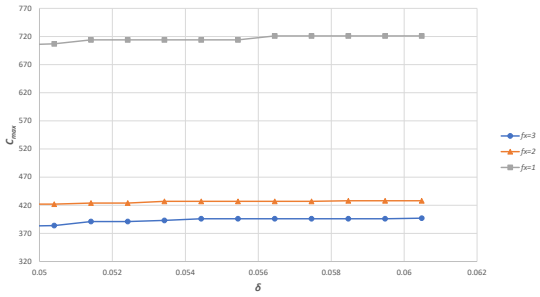


(f) $\delta = 0.18$

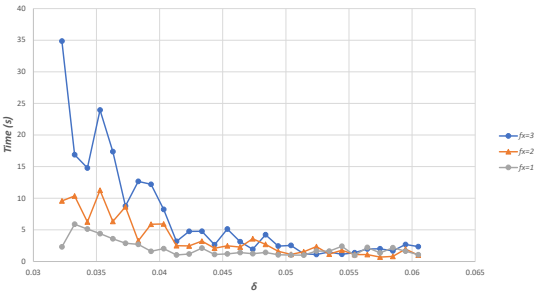


(g) $\delta = 0.16$

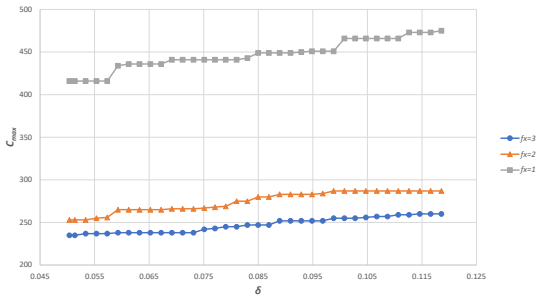
Figure 5: Example of a gradual reduction of the density of an arbitrary precedence graph on an FJSSP problem with $\theta = 3$



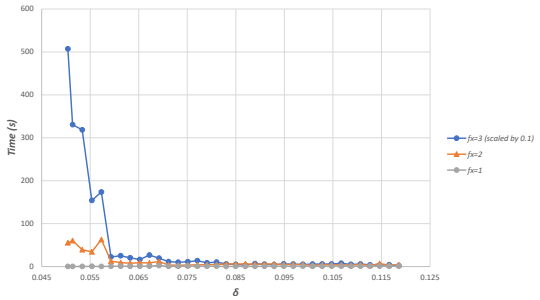
(a) d1–d29 problem instances: Makespan vs Density for different flexibility values.



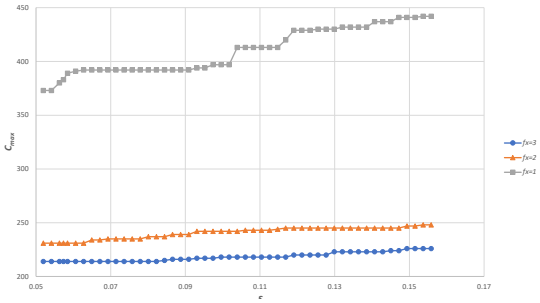
(b) d1–d29 problem instances: Computational time vs Density for different flexibility values.



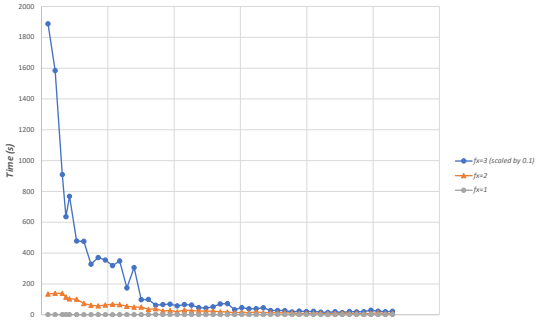
(c) t1–t36 problem instances: Makespan vs Density for different flexibility values.



(d) t1–t36 problem instances: Computational time vs Density for different flexibility values.



(e) q1–q50 problem instances: Makespan vs Density for different flexibility values.



(f) q1–q50 problem instances: Computational time vs Density for different flexibility values.

Figure 6: Effect of the flexibility fx and density of the precedence graphs δ on the makespan and the time needed to solve the problems to optimality

values tend to weaken the effect of the density on the makespan. For example, when the fx is set to one for the problem instances q1-q50, the makespan decreases with a rate of 656.73 as density increases. To the contrary, when the fx is set to three, the rate drops to 124.44. This behavior is consistent for all problem instances apart from the problem instances d1–d29. In particular, there is a marginal 2% increase of the slope as the fx values increase from 2 to 3. This is probably due to the small values and the narrow range of δ. On the other hand, as the complexity of the precedence graphs increases, i.e. by increasing θ, the slope values also decrease. Lastly, it seems that for the more complex precedence graphs the overall effect of the

density is weaker.

Another way to quantify the effect of the machine flexibility is to calculate the difference of the average makespan between different flexibility values for the same set of problem instances, i.e, pairs of the fx values (1,2) and (2,3). In particular, when the fx increases from 1 to 2, the average makespan decreases by 39.28%, 38.55% and 41.46% for the benchmark sets d1–d29, t1–t36 and q1–q50, respectively. Similarly, when the fx increases from 2 to 3, the average makespan decreases by 8.16%, 9.98% and 9.33% for the problem sets d1–d29, t1–t36 and q1–q50, respectively.

Regarding the time needed to close the optimality gap, it is clear that it becomes smaller as the density of the precedence graph increases. This can be attributed to the fact that as the problem becomes more constrained, the solution space is also reduced and thus, less computational time is potentially required to fully solve the problem. Finally, as the flexibility increases, the effect of the density of the precedence graphs on the time needed to solve the problem seems to be reduced (see Figures 6b, 6d, 6f and Table 9). This means that the manufacturing shop-floors with more available parallel machines, are more capable of absorbing increments in the C_{max} caused by dense job precedence graphs.

On the basis of the above observations, the following managerial insights can be drawn; it is important to reduce precedence constraints and linearize relationships. This will decrease scheduling sensitivity, reduce critical paths and as a result reduce the makespan. However, if the manufacturing recipe of the product requires complex precedence constraints to be respected, then one way to alleviate the effect on the makespan is to invest in flexibility.

4.6 Computational results for new large scale data sets of the FJSSP with arbitrary precedence graphs

In this section, we present computational results produced by the CP, MIP and EA on a new large-scale benchmark data set generated for the FJSSP with arbitrary precedence graphs. Additionally, we evaluate the performance of the MIP model proposed by Birgin et al. (2013) on this new data set. Compared to the problem instances used in the previous section, precedence graphs are now randomly generated and no specific structure is imposed. A full description of the process that was followed to generate these new benchmark sets, is provided in Appendix F. Overall, the new data set is divided into two groups of problems, namely SCPC and BCPC. The former group features problems with up to 120 operations, at most eight machines and

$1 \leq fx \leq 2$, while the latter group features problems with up to 1200 operations, 15 machines and $3 \leq fx \leq 4$. The flexibility ranges per problem group were selected so that the CP or the MIP could provide reasonably good lower and upper bounds within a time limit of 10800 seconds. A slope parameter γ is introduced to control the density of the generated precedence graphs. We refer the reader to Appendix F for more details. The examined values for γ are 0.2, 0.5 and 0.8. Therefore, for every problem instance, three versions are generated, each corresponding to a different value of γ. Note also that no starting solution is provided for the CP and MIP solution methods.

Tables 10 and 11 summarize the results obtained for the problem instances SCPC and BCPC. The first set of columns contain information about the average flexibility and the average slope γ. The remaining sets of columns provide the computational results, i.e., the average upper bounds, the average optimality gap %, and the number of proven optimal solutions for the CP, the MIP, the MIP of Birgin et al. (2013) and the EA, respectively. The last row reports the average gap out of all instances, when considering the best result per instance for each solution method. Given that BCPC instances are large scale, instead of reporting optimality gaps from lower bounds, we report % deviations from the best known solutions. The symbol "-" is used to indicate cases where no integer solution was found within the time limit, and therefore, no average result can be extracted for the particular sub-group. Similarly, the symbol "n/a" is used to indicate that these instances do not include arbitrary precedence constraints and therefore γ does not apply. Detailed results for both problem groups can be found in Tables E.4 – E.8 of Appendix E.

Overall, CP seems to perform better compared to both MIP approaches, which is consistent with the observations made earlier in Section 4.3. Regarding the low flexibility instances (SCPC01–SCPC12), nine out of the 12 instances can be solved optimally within the time limit. We also notice that the makespan as well as the lower bounds increase when the density of the precedence graphs increases (see values for γ). On the other hand, regarding high flexibility instances (SCPC13–SCPC24), the difficulty of the problem significantly increases. In this case, the gap is closed for only four out of 12 problem instances. For the sake of completeness, regarding the EA, an average gap of 30.68% was calculated considering the best run per instance. In this case, the average gap when considering all ten runs per instance as well as the worst run per instance were also equal to 30.68%. The worst case performance in this benchmark set when considering all ten runs per instance was 70.84% for the SCPC13–SCPC24 problem sub-group.

Table 10: Summary of computational experiments on the SCPC benchmark data sets

Group SCPC			CP			MIP			MIP (Birgin et.al.)			EA		
Range	fx	γ	C_{max}	Gap (%)	#	C_{max}	Gap (%)	#	C_{max}	Gap (%)	#	C_{max}	Gap (%)	#
SCPCN01–SCPCN08	1.90	n/a	67.6	8.72	6	70.0	12.59	5	71.1	14.32	5	67.6	8.20	6
SCPC01–SCPC12	1.61	0.5	61.0	5.26	9	64.4	10.13	3	66.3	12.37	3	60.9	5.16	9
SCPC13–SCPC24	2.19	0.5	47.7	70.22	4	53.3	89.91	3	58.4	107.94	3	47.8	70.84	3
Average Gap (%)			30.48			40.66			48.70			30.68		

Table 11: Summary of computational experiments on the BCPC benchmark data sets

Group BCPC			CP			MIP			MIP (Birgin et.al.)			EA		
Range	fx	γ	C_{max}	Dev (%)	#	C_{max}	Dev (%)	#	C_{max}	Dev (%)	#	C_{max}	Dev (%)	#
BCPCN01–BCPCN27	3.51	n/a	235.7	1.82	3	-	-	1	-	-	1	231.7	0.17	2
BCPC01–BCPC27	3.51	0.2	219.5	1.45	0	-	-	0	-	-	0	218.3	0.38	0
BCPC28–BCPC54	3.51	0.5	219.9	1.26	0	-	-	0	-	-	0	218.7	0.2	0
BCPC55–BCPC81	3.51	0.8	220.4	1.08	0	-	-	0	-	-	0	219.3	0.24	0
Average Dev (%)			1.40			-			-			0.28		

Regarding the results for the instances BCPCN01–BCPCN27, one can observe that the CP managed to solve optimally three out of the 27 instances, while the average deviation to the best known solutions is 1.82%. The EA was able to find highly competitive upper bounds, recording a deviation of 0.17% from the best solutions, which indicates that the EA performed much better compared to the CP within the same time limit. For this set of problems, only three integer solutions out of 27 problem instances were derived by the MIP approaches. It is also worth noting that the % deviation from the best solutions are in favor of the proposed MIP model.

The addition of precedence constraints among the operations seems to make the baseline problems much harder to solve, especially for low densities. Regarding the problem instances BCPC01–BCPC27 the CP did not manage to find any optimal solutions. The same observations can be made for the more dense versions of the problem instances, i.e. BCPC28–BCPC81. For these problems, the CP recorded an average deviation from the best known solutions that ranges from 1.08% to 1.26%. This deviation seems to decrease as the problem's density increases. Similar trends are observed for the EA as well, but the range is much smaller i.e., $0.24\% - 0.48\%$, which shows the superiority of the EA, regardless of the density of the precedence graphs. Overall, for the EA an average deviation of 0.28% was calculated considering the best run per instance. Also, when considering all ten runs per instance the average deviation was 0.46%, while when considering the worst run per instance the average deviation was 0.64%. Lastly, the worst performance of the EA among all BCPC problem sub-groups when considering all ten runs per instance was recorded for the BCPC55–BCPC81 sub-group, with an average deviation of 0.53%.

Regarding the two MIP approaches, the proposed MIP appears superior, with smaller deviations and a higher number of integer solutions, compared to the MIP model of Birgin et al. (2013).

On the other hand, when we compare the CP and the EA, one can observe that the difference on their performance is more significant on the baseline problems (BCPCN01–BCPCN27), rather than the problems with arbitrary precedence graphs (BCPC01–BCPC81). Another interesting remark is that the strengths of CP and EA seem to be complementary. However, it is difficult to explain this behaviour, since it may be the combined effect of many different elements. For example, the constraint propagation during the solution process of specific problem instances for the CP, anomalies of the solution space and the existence of strong local optima for the EA. We also observe that the CP becomes more efficient, when more constraints are added (e.g. more dense precedence graphs).

Based on the above observations, the following managerial insights can be drawn; the arbitrary precedence graphs allow more operations to be executed in parallel that compete for the same machines, whereas for the baseline FJSSP instances the operations of the same job have to be processed one after the other in a linear fashion. In particular, when an operation finishes, it may allow for more than one successor operations to be processed. These operations possibly compete over the same machines that may, or may not, be available yet. Therefore, even though waiting times on the operations might be introduced, the makespan typically reduces with the presence of arbitrary precedence graphs, as more operations can be scheduled in parallel at different machines. Nevertheless, as the precedence graphs become more dense, the makespan increases since the succeeding operations have to wait for more predecessor operations to finish.

5 Conclusions

The JSSP and the FJSSP are widely-studied combinatorial optimization problems due to their inherent complexity and the range of real-world applications. A common assumption for both problems is that each operation of a job has up to one successor and one predecessor operation. In practice, the operational execution hierarchies are often not linear and different work flows can be processed in parallel. As a result, multiple dependencies may occur among the job operations. In this paper, we examined the lesser studied FJSSP with arbitrary precedence graphs and we explored the effect of the machine flexibility, and the density of the precedence graphs on the makespan, as well as on the difficulty, to solve the problem.

MIP and CP models were proposed for exactly solving the FJSSP with arbitrary precedence graphs. Furthermore, an EA was presented for producing heuristic solutions, and for solving large-scale problem instances in reasonable amounts of time . The proposed EA is an amalgam of

SS and PR, and uses an efficient local search mechanism for educating the recombined reference solutions, as well as adaptive long-term memory structures for exploiting information gathered throughout the search process. Previous theoretical work for the FJSSP was extended and new methods for estimating the makespan, and for detecting cycles in the solution graphs before the application of relocation moves were provided. Both mechanisms resulted in significant savings regarding the computational effort of evaluating critical path-based neighborhood structures.

Initially, the efficiency and effectiveness of the proposed exact and heuristic solution methods were assessed on well-established FJSSP benchmark data sets. Notably, the CP managed to close the gap and improve the existing lower bounds for many problem instances, especially for those with low flexibility, while the performance of the MIP was inferior. On the other hand, the EA outperformed the current state-of-the-art solution methods. It was able to match the best known solutions and lower bounds for the vast majority of problems, and also produced 32 new best solutions for large-scale hard-to-solve FJSSP problem instances.

The proposed solution methods were also evaluated on the benchmark data set of Birgin et al. (2013). This set of experiments also demonstrated the competitiveness of the proposed solution methods. Regarding the YFJS set, the CP and the EA successfully closed the gaps in short computational times. Furthermore, six problems of the DAFJS set were solved to optimality, while the upper bounds for the majority of the problems were updated. We also observed that within the same time limit, the EA outperformed the CP and produced better upper bounds.

Regarding the FJSSP with arbitrary precedence graphs, we examined the effect of flexibility and density on new benchmark data sets. Overall, the computational results suggested that the presence of multiple dependencies among the operations made the problem harder to solve compared to the baseline FJSSP with linear dependencies. In all cases, the optimality gaps increased dramatically for both the CP and MIP solution methods, especially when the flexibility of the problem instances was high. To the contrary, the proposed EA seemed to scale relatively well within reasonable computational times. Problem instances with high average precedence graph densities were apparently easier to solve compared to those with lower densities. One could speculate that this is due to the increased scheduling flexibility in terms of processing more operations in parallel.

Overall, our results confirmed that as density increases, the makespan tends also to increase, since the succeeding operations have to wait for more predecessor operations to finish . Therefore, one major takeaway is that during the product design phase, it is important to reduce precedence

constraints and also to try to linearize relationships, this will decrease scheduling sensitivity and reduce critical paths. If precedence constraints cannot be avoided, our results indicated that adding flexibility can be a good alternative option to reduce the makespan. For example, our computational experiments indicated that if the flexibility is increased from 1 to 2 (i.e. allowing up to two machines per operation), the average makespan across problems with precedence graphs of various densities was decreased from 38.55% to 41.46%.

In terms of future research, there are major avenues that the literature could pursue. It may prove valuable to examine the effect of arbitrary precedence graphs when considering other objective functions and operational realities. The presence of multiple dependencies makes scheduling problems harder to solve, and thus it is worth exploring hybrid frameworks that will combine CP with metaheuristics to solve them.

Acknowledgments

The authors would like to gratefully acknowledge support from the European Commission and the Athens University of Economics and Business Research Center [DISRUPT Project, award number 723541, FACTLOG Project, award number 869951]. We also thank the two anonymous reviewers and the associated editor for their constructive comments and for helping us improve our work.

References

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

Alvarez-Valdes, R., Fuertes, A., Tamarit, J., Giménez, G., and Ramos, R. (2005). A heuristic to schedule flexible job-shop in a glass factory. *European Journal of Operational Research*, 165(2):525–534.

Barnes, J. W. and Chambers, J. B. (1996). Tabu Search for the Flexible-Routing Job Shop Problem. *The University of Texas, Austin, TX, Technical Report Series ORP96-10, Graduate Program in Operations Research and Industrial Engineering*, pages 1–11.

Ben Hmida, A., Haouari, M., Huguet, M.-J., and Lopez, P. (2010). Discrepancy search for the flexible job shop scheduling problem. *Computers and Operations Research*, 37(12):2192–2201.

Birgin, E. G., Feofiloff, P., Fernandes, C. G., de Melo, E. L., Oshiro, M. T., and Ronconi, D. P. (2013). A MILP model for an extended version of the Flexible Job Shop Problem. *Optimization Letters*, 8(4):1417–1431.

Bozejko, W., Uchroński, M., and Wodecki, M. (2010). Parallel hybrid metaheuristics for the flexible job shop problem. *Computers and Industrial Engineering*, 59(2):323–333.

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

Caglar Gencosman, B., Begen, M. A., Ozmutlu, H. C., and Ozturk Yilmaz, I. (2016). Scheduling Methods for Efficient Stamping Operations at an Automotive Company. *Production and Operations Management*, 25(11):1902–1918.

Dauzère-Pérès, S. and Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70(0):281–306.

Dell'Amico, M. and Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252.

Demir, Y. and Kürşat Işleyen, S. (2013). Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling*, 37(3):977–988.

Dongarra, J. J. (1992). Performance of various computers using standard linear equations software. *ACM SIGARCH Computer Architecture News*.

Feo, T. A. and Resende, M. G. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6(2):109–133.

Gao, J., Sun, L., and Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers and Operations Research*, 35(9):2892–2907.

Gao, K. Z., Suganthan, P. N., Pan, Q. K., Chua, T. J., Chong, C. S., and Cai, T. X. (2016). An improved artificial bee colony algorithm for flexible job-shop scheduling problem with fuzzy processing time. *Expert Systems and Applications*, 65:52–67.

Glover, F., Laguna, M., and Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684.

Goel, V., Slusky, M., Van Hoeve, W. J., Furman, K. C., and Shao, Y. (2015). Constraint programming for LNG ship scheduling and inventory management. *European Journal of Operational Research*, 241(3):662–673.

González, M. A., Vela, C. R., and Varela, R. (2015). Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research*, 245(1):35–45.

Hurink, J., Jurisch, B., and Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, 225(1981):223–225.

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

Jin, Z., Ohno, K., Ito, T., and Elmaghraby, S. (2002). Scheduling hybrid flowshops in printed circuit board assembly lines. *Production and Operations Management*, 11(2):216–230.

Komaki, G. M., Sheikh, S., and Malakooti, B. (2018). Flow shop scheduling problems with assembly operations: a review and new trends. *International Journal of Production Research*, 0(0):1–30.

Ku, W.-y. and Beck, J. C. (2016). Mixed Integer Programming models for job shop scheduling : A computational analysis. *Computers and Operations Research*, 73:165–173.

Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. (2018). IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 23(2):210–250.

Lunardi, W. T. and Voos, H. (2018). An extended flexible job shop scheduling problem with parallel operations. *SIGAPP Applied Computing Review*, 18(2):46–56.

Martí, R., Laguna, M., and Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, 169(2):359–372.

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

Mattfeld, D. C. (2013). *Evolutionary search and the job shop: investigations on genetic algorithms for production scheduling*. Springer Science & Business Media, Berlin/Heidelberg, Germany.

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

Rasmussen, K. M., Ejlertsen, L. S., M. Pour, S., Burke, E. K., and Drake, J. H. (2017). A hybrid Constraint Programming/Mixed Integer Programming framework for the preventive signaling maintenance crew scheduling problem. *European Journal of Operational Research*, 269(1):341–352.

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

Shen, L., Dauzère-Pérès, S., and Neufeld, J. S. (2018). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265(2):503–516.

Tarantilis, C. D., Anagnostopoulou, A. K., and Repoussis, P. P. (2012). Adaptive Path Relinking for Vehicle Routing and Scheduling Problems with Product Returns. *Transportation Science*, 47(3):356–379.

Unsal, O. and Oguz, C. (2013). Constraint programming approach to quay crane scheduling problem. *Transportation Research Part E: Logistics and Transportation Review*, 59:108–122.

van Laarhoven, P. J. M., Aarts, E. H. L., and Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125.

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

Wu, X. and Wu, S. (2017). An elitist quantum-inspired evolutionary algorithm for the flexible job-shop scheduling problem. *Journal of Intelligent Manufacturing*, 28(6):1441–1457.

Xiong, J., Xing, L. N., and Chen, Y.-w. (2013). Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. *International Journal of Production Economics*, 141(1):112–126.

Yi, W., Li, X., and Pan, B. (2016). Solving flexible job shop scheduling using an effective memetic algorithm. *International Journal of Computer Applications in Technology*, 53(2):157.

Yu, L., Zhu, C., Shi, J., and Zhang, W. (2017). An extended flexible job shop scheduling model for flight deck scheduling with priority, parallel operations, and sequence flexibility. *Science Programming*, 2017:1–15.

Yuan, Y., Xu, H., and Yang, J. (2013). A hybrid harmony search algorithm for the flexible job shop scheduling problem. *Applied Soft Computing*, 13(7):3259–3272.

Zhang, C., Shao, X., Rao, Y., and Qiu, H. (2008). Some new results on tabu search algorithm applied to the job-shop scheduling problem. In *Tabu Search*, chapter 8. IntechOpen, London, UK.

Zhang, J., Ding, G., Zou, Y., Qin, S., and Fu, J. (2019). Review of job shop scheduling research and its new perspectives under Industry 4.0. *Journal of Intelligent Manufacturing*, 30(4):1809–1830.