List scheduling and beam search methods for the flexible job shop scheduling problem with sequencing flexibility

E. G. Birgin* J. E. Ferreira† D. P. Ronconi†

October 3, 2014‡

Abstract

An extended version of the flexible job shop problem is tackled in this work. The considered extension to the classical flexible job shop problem allows the precedences between the operations to be given by an arbitrary directed acyclic graph instead of a linear order. Therefore, the problem consists of allocating the operations to the machines and sequencing them in compliance with the given precedences. The goal in the present work is the minimization of the makespan. A list scheduling algorithm is introduced and its natural extension to a beam search method is proposed. Numerical experiments assess the efficiency of the proposed approaches.

Key words: Scheduling, flexible job shop, makespan, list scheduling, beam search.

1 Introduction

The classical job shop (JS) problem consists of scheduling n jobs on an environment with m machines. Each job is composed by several operations with a linear precedence structure and has a predetermined route through the machines. The flexible job shop scheduling (FJS) problem is a generalization of the JS problem in which there may be several machines, not necessarily identical, capable of processing each operation. The processing time of each operation on each machine is known and no preemption is allowed. The objective is to decide on which machine each operation will be processed, and in what order the operations will be processed on each machine, so that a certain criterion is optimized.

This paper considers the extended version of the FJS problem that allows the precedences between the operations to be given by an arbitrary directed acyclic graph instead of a linear order. Therefore, the problem consists of allocating the operations to the machines and sequencing them in compliance with all given precedences. An example of a job with this general

*Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, SP, Brazil. e-mail: {egbirgin | jeferre}@ime.usp.br

†Department of Production Engineering, Polytechnic School, University of São Paulo, Av. Prof. Almeida Prado, 128, Cidade Universitária, 05508-900, São Paulo SP, Brazil. e-mail: dronconi@usp.br

‡Revisions made on March 12 and June 8, 2015.

type of precedences is presented in Figure 1. This problem appears in practical and industrial environments, such as the printing industry [38], where assembling and disassembling operations are part of the production process. Printing processes can be divided into three major tasks: prepress steps, printing, and postpress steps [40]. Prepress steps include composition and typesetting, graphic arts photography, image assembly, color separation, and image carrier preparation. Printing can be performed by six separate and distinct processes: lithography, letterpress, flexography, gravure, screen printing, and plate-less technologies. Postpress operations consist of four major processes: cutting, folding, assembling, and binding. There are many additional lesser postpress finishing processes such as varnishing, perforating, drilling, etc. In-line finishing may also be considered as a final step of the postpress operations. These three major steps of the printing process have an obvious precedence constraint. However, within each major step, there are operations with no precedence constraint among them. Pages of a book are divided into signatures (bunch of 8, 16, or 32 individual pages) that can be printed, cut, and folded in separate. The book cover is also an element that can be prepared in separate. Then, all printed and non-printed elements need to be gathered in order to continue the process. See Figure 2. It is easy to see that this arbitrary-precedences issue of the printing process may be found in most of the practical industrial applications, making the considered problem a problem with a potential wide range of applications. The scheduling performance measure considered in the present work is the makespan minimization.

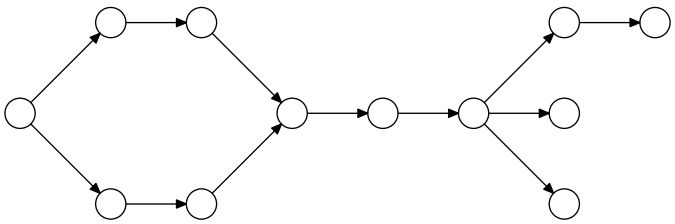


Figure 1: Example of the precedence constraints of a single job of an instance of the extended version of FJS problem, in which precedences between operations are given by an arbitrary directed acyclic graph.

The flexibility of representing the precedences between the operations of a job with an arbitrary directed acyclic graph instead of a linear order is known as sequencing flexibility; while routing flexibility refers to the possibility of an operation to be performed by a subset of machines instead of a single machine (this is the flexibility that transform a JSP into a FJSP). Other types of flexibility exist, like producing the same manufacturing feature with alternative operations or sequences of operations, known as processing flexibility. The effects of sequencing flexibility on the performance of dispatching rules used to schedule operations in manufacturing systems was analysed in [29, 21] (see also the references therein). In [31], a flexible manufacturing system with finite buffer capacities and that considers automated guided vehicles is tackled. Different performance criteria are considered (mean flow time, mean tardiness, and makespan) and an *ad hoc* filtered beam search method is developed. The results of the method are analysed in order to investigate the effects in the performance of the manufacturing system of incorporating

different types of flexibilities. A more recent study can be found in [14].

The extended FJS problem considered in the present work is NP-hard, since it has the JS problem (that is known to be NP-hard [10]) as a particular case. Due to its complexity, the number of publications concerned with the exact solution of the FJS problem is very small. Fattahi, Mehrabad, and Jolai [7] proposed a mixed integer linear programming (MILP) model for the FJS problem and used it to solve small and medium-sized instances with a commercial software. A more concise MILP model, that modifies an earlier one presented in [22] in order to incorporate routing flexibility, was introduced in [26]. More recently, a new MILP model for the extended version of FJS considered in the present work was presented in [3]. This model was analyzed using instances from the literature and instances inspired by the printing industry. According to the numerical experiments, the software CPLEX produced better results with the new model than with the one presented in [26].

Several works from the literature proposed heuristic methods to address the makespan minimization in the classical FJS problem. Brandimarte [4], one of the pioneers of this approach, applied dispatching rules to assign each operation of each job to a machine and, in a second phase, employed a tabu search heuristic to define the sequence of the operations on each machine. This kind of strategy is known as hierarchical approach. Tabu search (TS) based heuristics to

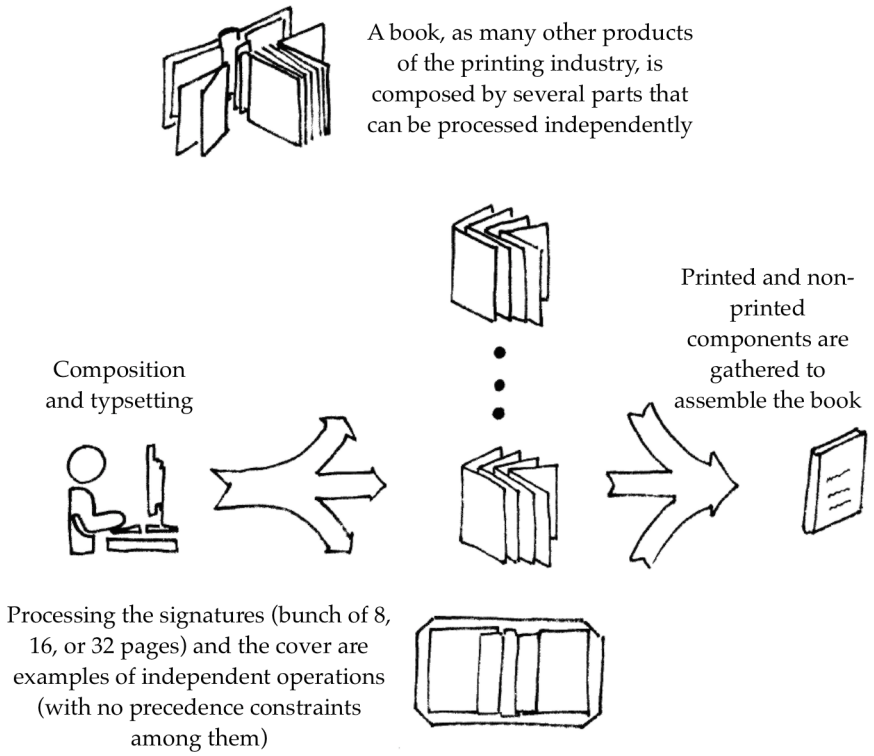


Figure 2: Illustrative scheme including operations with no precedence constraints among them (the different signatures and the cover) in the printing-industry task of producing a book.

solve this problem, but in an integrated way (i.e. considering simultaneously the assignment and schedule of the operations), were also developed in [6, 24]. Recent literature also includes genetic algorithms (GA) to deal with the FJS problem in a integrated approach. The learnable genetic architecture (LEGA), proposed in [12], provides an integration between evolution and learning methodologies within a random search framework. In this context, the learning module is used to influence the diversity and quality of offsprings. On the other hand, a traditional GA with improved components selected from the literature and a new mutation assignment operator named intelligent mutation was introduced in [27]. According to the presented computational tests the proposed GA outperformed other known GAs from the literature and obtained solutions comparable with the ones obtained by the TS described in [24]. A hybrid GA that follows the hierarchical approach can be found in [11]. First the algorithm uses GA to find the assignments and crude schedules (feasibility is not guarantee), and next it applies repairing heuristics. In [11], a quantitative comparison with recent works of the literature, including [24] and considering benchmark problems from [4], was presented to illustrate the performance of the proposed method. Aiming to join the best characteristics of different approaches Yuan and Xu [37] proposed an integrated search heuristic composed by a hybrid harmony search (HHS) and a large neighborhood search (LNS). In first place, the HHS algorithm runs until a solution from which no significant improvement can be done is reached. Then, the operation-machine assignment information from the elite solutions is extracted. Finally, the LNS method is executed on the reduced space to further improve the best solution obtained by the HHS. The authors presented experiments with four different benchmarks from the literature and concluded that the HHS/LNS integrated search shows a competitive performance with respect to the state-of-the-art methods. Additionally, several works have considered the FSJ with multiple objectives that include the makespan criterion. See, for example, [15, 39, 35, 20, 32], and the references therein.

A few works, most of them inspired in practical applications, deal with the extension of the FJS in which precedences are given by an arbitrary directed acyclic graph. An environment coming from a glass factory, that requires an even more general variant of the FJS problem including, for example, no-wait constraints, is described in [1]. The authors proposed heuristic methods based on priority rules and a local search to minimize a criterion based on earliness and tardiness penalties. A class of instances that includes arbitrary precedence relations among operations (with the constraint of having an ending assembling operation in each job) of a problem from the printing and boarding industry is also tackled in [36]. Considering the makespan and the maximum lateness criteria, TS and GA are applied with the aim of building an approximation of the Pareto frontier. A symbiotic evolutionary algorithm that considers routing, sequencing, and processing flexibility in a job shop scheduling problem was introduced in [17]. Another extended version of the FJSP, inspired in real manufacturing environments, in which precedence constraints are arbitrary and can be of AND/OR type, is studied in [19]. A real scheduling problem in a mould manufacturing shop problem with several kind of flexibilities is described in [9]. The considered problem possesses process planning flexibility, that includes sequencing flexibility. In the proposed method, process planing and scheduling are tackled in an integrated way.

Due to the limited amount of works that approach the extended version of the FJS problem and its practical applicability, the purpose of this paper is to contribute to the development of

heuristic techniques able to produce reasonable results in acceptable time. First, a list scheduling algorithm is proposed, motivated by its simplicity and applicability to job scheduling in production environments (see, for example, [16, 18, 23]). The natural extension of the list scheduling algorithm to a filtered beam search method is also investigated. The filtered beam search is a technique for searching decision trees that involves systematically developing a small number of solutions in parallel so as to attempt to maximize the probability of finding a good solution with minimal search effort [25]. The evaluation process that determines which partial solutions are the promising ones is a crucial component of this method [28]. The main idea of the presented filtered beam search method is to apply a customized version of the list scheduling algorithm to locally and globally evaluate partial solutions in an effective way. After the pioneer work of Ow and Morton [25], other authors addressed scheduling environments with this approach. Sabuncuoglu and Bayiz [30] tackled the JS problem minimizing the makespan and conducted several numerical experiments to analyze the influence of some of the filtered beam search parameters on its performance. Moreover, a comparison considering metaheuristic algorithms and dispatching rules was also conducted to expose the competitive performance of the proposed method. More recently, a filtered beam search approach for the flexible job shop minimizing the weighted sum of three objectives (the makespan, the total workload of machines, and the workload of the most loaded machine) was introduced in [33].

Considering the component-based view of metaheuristics suggested in [34], it can be said the beam search method is a matheuristic that combines the search-tree strategy of branch-and-bound methods with a constructive heuristic used for potentially pruning apparently fruitless nodes of the search tree. Some of the highlights of the introduced methods are that they both have finite termination and that they depend on a very reduced set of parameters whose meaning is very simple to interpret. In fact, the list scheduling algorithm has no parameters while the beam search method has only three parameters. Other than the promising numerical results, additional main features of the heuristic methods presented in this work are (a) their precise description (all possible ties are decided with explicitly given rules) and (b) availability of their C/C++ implementation. Moreover, together with a complete description of the obtained solutions, these facts allow full reproducibility of the presented results and open the possibility of using them as a benchmark for future developments. The remaining of this work is organized as follows. Section 2 gives the description of the tackled problem by presenting its mixed-integer linear programming formulation. Section 3 presents a list scheduling algorithm while Section 4 introduces the proposed beam search method. Section 5 is devoted to numerical experiments. Section 6 presents some final remarks and lines for future research.

2 Extended flexible job shop scheduling problem

A precise description of the considered problem can be given by the MILP formulation introduced in [3], that we reproduce in this section for completeness and to introduce the notation that will be used in the present work.

Let n, o, and m be the number of jobs, operations, and machines, respectively. The number of jobs will not play any explicit role in the problem formulation. For each operation i ($i = 1, \ldots, o$), let $F_i \subseteq \{1, 2, \ldots, m\}$ ($F_i \neq \emptyset$) be the subset of machines that can process operation i and let p_{ik}

($i = 1, \ldots, o,\ k \in F_i$) be the corresponding processing times. Moreover, let A be a set of pairs (i, j) with $i, j \in \{1, \ldots, o\}$ such that, if (i, j) belongs to A, this means that operation i precedes operation j, i.e. operation j can not start to be processed until operation i ends to be processed. This set of precedences A is the place where jobs are implicitly defined, since it is expected to exist precedences between operations of the same job but not to exist precedences between operations of different jobs. Moreover, it is assumed that if the elements of A represent the arcs of a directed graph with vertices labeled from 1 to o then this graph is a directed acyclic graph, i.e. there are no cyclic precedences and, in consequence, the problem is well defined. The problem consists in assigning each operation i to a machine $k \in F_i$ and to determine a starting processing time s_i such that precedences are satisfied. Of course, a machine can not process more than an operation at a time and preemption is not allowed. The objective is to minimize the makespan, i.e. the completion time of the last operation (or, equivalently, last job).

The model uses binary variables x_{ik} ($i = 1, \ldots, o,\ k \in F_i$) to indicate whether operation i is allocated to be processed by machine k (in this case $x_{ik} = 1$) or not (in this case $x_{ik} = 0$). It also considers binary variables y_{ij} ($i, j = 1, \ldots, o,\ F_i \cap F_j \neq \emptyset$) to indicate, whenever two operations are allocated to the same machine, which one is processed first. Assume, for example, that there are two operations i and j such that $k \in F_i \cap F_j$ and that both operations are allocated to machine k, i.e. $x_{ik} = x_{jk} = 1$. In this case, we must have exactly one between y_{ij} and y_{ji} equal to 1. If $y_{ij} = 1$ and $y_{ji} = 0$ then operation i is processed on machine k before operation j. On the other hand, if $y_{ij} = 0$ and $y_{ji} = 1$ then operation i is processed on machine k after operation j. Finally, the model uses variables s_i ($i = 1, \ldots, o$), to represent the starting time of operation i (on the machine to which it was allocated) and a variable $C_{\max}$ to represent the makespan. With these variables, the MILP model of the extended flexible job shop problem introduced in [3] can be written as:

$$\text{Minimize} \quad C_{\max} \tag{1}$$

$$\text{subject to} \quad \sum_{k \in F_i} x_{ik} = 1, \quad i = 1, \ldots, o, \tag{2}$$

$$p'_i = \sum_{k \in F_i} x_{ik} p_{ik}, \quad i = 1, \ldots, o, \tag{3}$$

$$C_{\max} \geq s_i + p'_i, \quad i = 1, \ldots, o, \tag{4}$$

$$s_i + p'_i \leq s_j, \quad i, j = 1, \ldots, o \text{ such that } (i, j) \in A, \tag{5}$$

$$y_{ij} + y_{ji} \geq x_{ik} + x_{jk} - 1, \quad i, j = 1, \ldots, o, i \neq j, \text{ and } k \in F_i \cap F_j, \tag{6}$$

$$s_i + p'_i - (1 - y_{ij})L \leq s_j, \quad i, j = 1, \ldots, o \text{ and } i \neq j \text{ such that } F_i \cap F_j \neq \emptyset, \tag{7}$$

$$s_i \geq 0, \quad i = 1, \ldots, o. \tag{8}$$

Constraint (2) says that each operation i must be assigned to exactly one machine $k \in F_i$. Constraint (3) defines the actual processing time p'_i of each operation i (that depends on the machine to which it was assigned). In fact, there is no need to consider p'_i in (3) as a variable of the model. It can be seen as an auxiliary value that simplifies the model presentation, while it can be avoided by removing constraint (3) and replacing each appearance of p'_i in the other constraints with the expression in the right hand side of (3). Constraint (4), together with the minimization of the objective function in (1) defines $C_{\max}$ as the makespan. Constraint (5) represents the precedence constraints. For every pair of operations assigned to the same machine,

constraints (6,7) say that both operations can not be processed at the same time and determine which one is processed first. If two operations i and j that could have been both assigned to a machine k (i.e. $k \in F_i \cap F_j \neq \emptyset$) are not then we have that at most one between x_{ik} and x_{jk} is equal to 1. In this case, constraints (6,7) are trivially satisfied with $y_{ij} = y_{ji} = 0$. In (7), L represents a sufficiently large positive constant (see [3] for a suggested value that may be used in practice). Finally, constraint (8) says that the starting times of the operations must be not smaller than the beginning of the considered planning horizon that, without loss of generality, was set to 0.

Model (1–8) was introduced in [3], where a comparison with a simple extension of the model presented in [26] was given. Up to the authors acknowledge these two models are the only published ones that include the possibility of the precedences between the operations to be given by an arbitrary directed acyclic graph.

3 List scheduling algorithm

In this section we describe a non-hierarchical list scheduling algorithm. It is non-hierarchical in the sense that, at each iteration, it selects an operation, assigns it to a machine, and determines a starting time. This is in contrast with hierarchical methods that in a first phase assign the operations to the machines and in a second phase determine the starting times. A similar but simpler heuristic (named EST) was very briefly described in [3, p.1426], where it was used, in the context of evaluating a MILP model, to provide an upper bound on the solution (i.e. an initial feasible solution) to the exact solver CPLEX.

The proposed list scheduling algorithm iterates o times and, at each iteration, selects an operation i, assigns it to a machine $k \in F_i$, and determines the starting time st of operation i on machine k. This process is guided by customized rules for the extended version of the FJS problem. The decision is recorded by setting $s[i] \leftarrow st$ (starting time of operation i) and $w[i] \leftarrow k$ (machine to which operation i was assigned). In the context of the algorithm, if the values of $s[i]$ and $w[i]$ were defined, we say that operation i was already *handled* (by the algorithm) or scheduled. Otherwise, if $s[i]$ and $w[i]$ are undefined, we say that operation i is still *unhandled*. At a given iteration, operations that are candidates to be handled are those that are still unhandled and such that do not have unhandled predecessors.

Parameters n, m, o, F_i $(i = 1, \ldots, o)$, p_{ik} $(i = 1, \ldots, o,\ k \in F_i)$, and A, that determine an instance of the FJS problem, are the input data of the list scheduling algorithm. From them, some auxiliary instance data that aid to apply the algorithm can be computed. Those quantities, that are defined for each operation i $(i = 1, \ldots, o)$, are:

(a) The predecessors and successors sets $\mathcal{P}_i$ and $\mathcal{S}_i$ given by

$$\mathcal{P}_i = \{j \in \{1, \ldots, o\} \mid (j, i) \in A\}$$

and

$$\mathcal{S}_i = \{j \in \{1, \ldots, o\} \mid (i, j) \in A\}.$$

(b) The average processing times $\bar{p}_i$ given by

$$\bar{p}_i = \frac{1}{|F_i|} \sum_{k \in F_i} p_{ik}.$$

(c) The remaining (or blocked) work RW_i given by the longest (directed) path from i to any other operation j in the digraph with set of arcs A, set of nodes $\{1, \ldots, o\}$, and nodes' weights $\bar{p}_1, \ldots, \bar{p}_o$.

3.1 Characterization of partial solutions

To support, at each iteration, the selection of an operation (plus the machine that will process it and its corresponding starting time), the algorithm keeps track of the following information:

- $\boldsymbol{s[i]}$ that saves, for each handled operation i, its processing starting time. The value is undetermined if operation i is still unhandled.
- $\boldsymbol{w[i]}$ that saves, for each handled operation i, the machine to which it was assigned. The value is undetermined if operation i is still unhandled.
- $\boldsymbol{C_{\max}}$ the maximum among the completion times of the handled operations, i.e. the makespan of the *partial solution* (where by "partial solution" we mean the solution, not yet complete, composed by the already handled operations).
- $\boldsymbol{u[i]}$ that saves, for each operation i, the maximum among the completion times of its handled predecessors.
- $\boldsymbol{v[k]}$ that saves, for each machine k, the maximum among the completion times of the handled operations that were assigned to it.
- $\boldsymbol{L[k]}$ that saves, for each machine k, the sum of the processing times of the unhandled operations that can potentially be assigned to it (this is an upper bound on the future load of the machine).
- $\boldsymbol{\eta[i]}$ that says, for each operation i, how many unhandled predecessors it has.
- $\boldsymbol{\Omega}$ the set of unhandled operations that have no unhandled predecessors (i.e. operations that are natural candidates to be handled).

A partial solution with ℓ handled operations is characterized by the 9-tuple $(\ell, s, w, C_{\max}, u, v, L, \eta, \Omega)$, that carries all the information needed (by the list scheduling algorithm that will be presented below) to generate a new partial solution with $\ell + 1$ handled operations. For the particular case of the partial solution with $\ell = 0$ scheduled operations, we have that:

$$
\begin{cases}
s[i] & \text{undetermined for } i = 1, \ldots, o, \\
w[i] & \text{undetermined for } i = 1, \ldots, o, \\
C_{\max} &= 0, \\
u[i] &= 0 \text{ for } i = 1, \ldots, o, \\
v[k] &= 0 \text{ for } k = 1, \ldots, m, \\
L[k] &= \sum_{\{i \mid k \in F_i\}} p_{ik} \text{ for } k = 1, \ldots, m, \\
\eta[i] &= |\mathcal{P}_i| \text{ for } i = 1, \ldots, o, \\
\Omega &= \{i \in \{1, \ldots, o\} \mid \mathcal{P}_i = \emptyset\}.
\end{cases}
$$

Let $(\ell, s, w, C_{\max}, u, v, L, \eta, \Omega)$ be a partial solution with $0 \leq \ell < o$. If an operation $i \in \Omega$ is assigned to a machine $k \in F_i$ and scheduled to start its processing at time st, the 9-uple that characterizes the new partial solution with $\ell + 1$ handled operation is given by $(\ell+1, s', w', C'_{\max}, u', v', L', \eta', \Omega')$, where s', w', $C'_{\max}$, u', v', L', η', and Ω' receive, respectively, the values of s, w, $C_{\max}$, u, v, L, η, and Ω and then are updated as follows:

$$\begin{cases} s'[i] & \leftarrow & st, \\ w'[i] & \leftarrow & k, \\ C'_{\max} & \leftarrow & \max\{C'_{\max}, st + p_{ik}\}, \\ u'[j] & \leftarrow & \max\{u'[j], st + p_{ik}\} \text{ for all } j \in S_i, \\ v'[k] & \leftarrow & \max\{v'[k], st + p_{ik}\}, \\ L'[r] & \leftarrow & L'[r] - p_{ir} \text{ for all } r \in F_i, \\ \eta'[j] & \leftarrow & \eta'[j] - 1 \text{ for all } j \in S_i, \\ \Omega' & \leftarrow & \Omega' \setminus \{i\} \cup \{j \in S_i \mid \eta'[j] = 0\}. \end{cases}$$

3.2 Selecting rules

The list scheduling algorithm introduced in this section starts with the partial solution with $\ell = 0$ handled operations, handles a single operation per iteration, and ends after o iterations with a feasible solution to the given instance of the FJS problem. We now describe the rules to choose, at each iteration, the operation to be scheduled, the machine to which the operation is assigned, and the operation's starting time.

At each iteration, the set of candidate pairs operation/machine Ψ_0 is given by

$$\Psi_0 = \{(i, k) \mid i \in \Omega \text{ and } k \in F_i\}. \tag{9}$$

The three rules below are sequentially applied until a single pair operation/machine $(i, k) \in \Psi_0$ is selected and a processing starting time st is determined for the schedule of operation i on machine k.

Rule 1: *Pairs operation/machine with the earliest starting time.*

In accordance with the minimization of the makespan, the first attribute used to select a pair (i, k) from Ψ_0 is based on the earliest starting time st_{ik} for operation i on machine k, given by the maximum between two quantities: (a) the maximum between the completion times of the (already handled) predecessors of operation i, given by $u[i]$, and (b) the time at which machine k ends to process all the already handled operations assigned to it, given by $v[k]$. Thus, we have that

$$st_{ik} = \max\{u[i], v[k]\} \tag{10}$$

for all $(i, k) \in \Psi_0$. Let

$$\widehat{st} = \min\{st_{ik} \mid (i, k) \in \Psi_0\} \tag{11}$$

be the smallest possible starting time among the candidate pairs $(i, k) \in \Psi_0$. Only pairs (i, k) such that $st_{ik} = \widehat{st}$ remain as a possibility for the iteration assignment (with the natural choice

$s[i] = \widehat{st}$ and $w[i] = k$), while the other pairs are discarded. Thus, let

$$\Psi_1 = \{(i,k) \mid (i,k) \in \Psi_0 \text{ and } st_{ik} = \widehat{st}\}. \tag{12}$$

Rule 2: *Fastest machine for each operation.*

Let

$$\Psi_1^i = \{(j,k) \mid (j,k) \in \Psi_1 \text{ and } j = i\} \text{ for } i = 1, \ldots, o,$$

i.e. Ψ_1^i is the subset of pairs in Ψ_1 that correspond to the same operation i (associated with different machines in F_i). The focus of this rule is on the operations i such that $|\Psi_1^i| > 1$ and the objective is to select, for each one of those operations, a single machine among the several possibilities.

Let i be such that $|\Psi_1^i| > 1$ and let $(i,k_1),(i,k_2),\ldots$ be the elements of Ψ_1^i. Each machine $k_1, k_2, \ldots$ is associated with the processing time $p_{ik_1}, p_{ik_2}, \ldots$ and the upper bound on the machine load $L[k_1], L[k_2], \ldots$ Consider the triplets

$$(p_{ik_1}, L[k_1], k_1),\ (p_{ik_2}, L[k_2], k_2),\ \ldots$$

and let $\Psi_{1.5}^i = \{(i,k_\nu)\}$ be the singleton such that $(p_{ik_\nu}, L[k_\nu], k_\nu)$ is the smallest triplet in the lexicographical order. It means that, among all possible machines that can process operation i, we select the one with the smallest processing time. In the case of a tie, the smallest upper bound on the machine load is used as a tie-break and, in the case of a second tie, the machine with the smallest index is chosen with no purpose other than defining a deterministic rule. Define

$$\Psi_2 = \{\cup\Psi_1^i \mid |\Psi_1^i| \leq 1\} \cup \{\cup\Psi_{1.5}^i \mid |\Psi_1^i| > 1\}.$$

We have that $\Psi_2 \subseteq \Psi_1$ contains no more than one pair operation/machine for each operation and, for the operations that had more than one possible machine, it preserves only the most promising one (fastest and, in case of tie, less loaded).

Rule 3: *Operation with the largest remaining or blocked work.*

The next attribute used to reduce the number of candidate pairs $(i,k) \in \Psi_2$ is based on an estimate of the remaining (or blocked) work RW_i that is associated with an operation i. Recall that RW_i is defined as the longest path from i to any other operation j in the digraph with set of arcs A, set of nodes $\{1,\ldots,o\}$, and nodes' weights $\bar{p}_1, \ldots, \bar{p}_o$. Note that if the longest path starting at operation i ends at an operation j then $\mathcal{S}_j = \emptyset$ and i and j belong to the same job t. Moreover, $st_{ik} + RW_i$ may be seen as an estimate of the completion time of job t, computed from the perspective of the candidate pair (i,k). However, since $st_{ik} = \widehat{st}$ for all $(i,k) \in \Psi_2$, we simple consider the value of RW_i as an estimate of the remaining or blocked work associated with operation i.

For the pairs $(i_1,k_1),(i_2,k_2),\ldots \in \Psi_2$ consider the triplets

$$(-RW_{i_1}, -L[k_1], i_1),\ (-RW_{i_2}, -L[k_2], i_2),\ \ldots$$

and let $(-RW_{i_\nu}, -L[k_\nu], i_\nu)$ be the smallest triplet in the lexicographical order, associated with the pair (i_ν, k_ν). This means that a final selection was made and that it consists in assigning

operation i_ν to machine k_ν with starting time $\widehat{st}$. In this way, we selected the operation with the largest remaining or blocked work as the one to be scheduled. The idea behind this choice is to rapidly handling operations that impair the processing of a large amount of work. In the case of a tie, the upper bound on the future load of the machines is used as a tie-break, in order to assign as soon as possible operations to a machine that has a large expected future load (trying to minimize its idle time). Finally, in the case of a second tie, the operation with the smallest index is chosen with no purpose other than defining a deterministic rule.

3.3 Pseudo-code and complexity

The computation of the auxiliary instance data (items (a–c) at the beginning of Section 3), the initialization of the empty partial solution (Section 3.1), and the iterative application of rules 1–3 (Section 3.2) plus the update of the partial solution characterization (Section 3.1) define the method introduced in this section, that is fully described in Algorithm 1.

Sets $\mathcal{P}_i$ and $\mathcal{S}_i$ are computed in lines 2–3, the average processing times $\bar{p}_i$ are computed in lines 4–7, the remaining work estimates RW_i are computed in lines 8–16 by Dijkstra's shortest path method [5, pp.655–658] (adapted to compute the longest path and to the case in which there may be several sources and several targets). The values of $C_{\max}$, u, v, L, η, and the set Ω are initialized in lines 17–21. The main loop, from line 22 to line 41, executes o times rules 1–3. The selected pair at each iteration is named (λ, θ) and the determined starting time is named $\widehat{st}$. The starting time $s[\lambda]$ for operation λ is set at line 36, as well as the the machine θ to which operation λ is assigned is recorded, in the same line, in $w[\lambda]$. The quantities $C_{\max}$, u, v, L, η, and the set Ω are updated in lines 36–41.

A few words regarding the worst-case time complexity of Algorithm 1 are in order. Initializations from line 2 to 7 and from line 17 to 21 are $O(|A|+m+\sum_{i=1}^{o}|F_i|)$. The Dijkstra's algorithm implemented in lines 8–16 has complexity $O(|A|+o)$. It remains to analyze the main loop that goes from line 22 to line 41 and is executed o times. If we consider that at each iteration we have that $|\Omega| \leq o$ then we have that the loop is $O(o\sum_{i=1}^{o}|F_i|+|A|)$. Thus, summing up, we have that Algorithm 1 is

$$O\left(|A|+m+o\sum_{i=1}^{o}|F_i|\right).$$

On the other hand, a better bound for $|\Omega|$ can be given. Consider the directed acyclic graph $D=(\{1,\ldots,o\},A)$. An *antichain* in D is a set of nodes, no two of which are included in any path of D. Noting that operations in Ω are operations that have no precedence constraints among each other, it is not hard to see that the number of elements in Ω is limited by the size of a maximum antichain in D (that can be computed in polynomial time [8]). Let w be the size of a maximum antichain in D and let

$$q = |F_{i_1}| + |F_{i_2}| + \ldots + |F_{i_w}|, \tag{13}$$

where $F_{i_1}, F_{i_2}, \ldots, F_{i_w}$ are the w largest sets among the sets $F_i, i=1,\ldots,o$. We can say that the main loop (from line 22 to line 41) is $O(oq+|A|)$. Then, we have that Algorithm 1 is

$$O(|A|+m+oq),$$

that provides a tighter bound on the worst-case time complexity of the algorithm.

Algorithm 1: List Scheduling.

Input: m, o, F_i $(i = 1, \ldots, o)$, p_{ik} $(i = 1, \ldots, o,\ k \in F_i)$, A
Output: $s[i]$, $w[i]$ $(i = 1, \ldots, o)$, $C_{\max}$
ListScheduling(m, o, F, p, A)

begin
  for $i \leftarrow 1$ **to** o **do** $\mathcal{P}_i \leftarrow \emptyset,\ \mathcal{S}_i \leftarrow \emptyset$
  foreach $(i, j) \in A$ **do** $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup \{i\},\ \mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{j\}$
  for $i \leftarrow 1$ **to** o **do**
    $\bar{p}_i \leftarrow 0$
    foreach $k \in F_i$ **do** $\bar{p}_i \leftarrow \bar{p}_i + p_{ik}$
    $\bar{p}_i \leftarrow \bar{p}_i / |F_i|$
  $\mathcal{Q} \leftarrow \{i \mid |\mathcal{S}_i| = 0\}$
  for $i \leftarrow 1$ **to** o **do**
    if $i \in \mathcal{Q}$ **then** $RW_i \leftarrow \bar{p}_i$ **else** $RW_i \leftarrow 0$
  while $\mathcal{Q} \neq \emptyset$ **do**
    Let $z \in \mathcal{Q}$, $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{z\}$
    foreach $i \in \mathcal{P}_z$ **do**
      if $RW_z + \bar{p}_i > RW_i$ **then**
        $RW_i \leftarrow RW_z + \bar{p}_i$
        if $i \notin \mathcal{Q}$ **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{i\}$
  $C_{\max} \leftarrow 0,\ u[1 \ldots o] \leftarrow 0,\ v[1 \ldots m] \leftarrow 0,\ L[1 \ldots m] \leftarrow 0$
  for $i \leftarrow 1$ **to** o **do**
    foreach $k \in F_i$ **do** $L[k] \leftarrow L[k] + p_{ik}$
  for $i \leftarrow 1$ **to** o **do** $\eta[i] \leftarrow |\mathcal{P}_i|$
  $\Omega \leftarrow \{i \mid \eta[i] = 0\}$
  do o **times**
    $\widehat{st} \leftarrow \infty,\ \widehat{RW} \leftarrow 0$
    foreach $i \in \Omega$ **do**
      $\widetilde{st} \leftarrow \infty,\ \widetilde{pt} \leftarrow \infty$
      foreach $k \in F_i$ **do**
        $st \leftarrow \max(u[i], v[k])$
        if $st < \widetilde{st}$ or ($st = \widetilde{st}$ and $p_{ik} < \widetilde{pt}$)
          or ($st = \widetilde{st}$ and $p_{ik} = \widetilde{pt}$ and $L[k] < L[\widetilde{k}]$)
          or ($st = \widetilde{st}$ and $p_{ik} = \widetilde{pt}$ and $L[k] = L[\widetilde{k}]$ and $k < \widetilde{k}$) **then**
          $\widetilde{st} \leftarrow st,\ \widetilde{pt} \leftarrow p_{ik},\ \widetilde{k} \leftarrow k$
      if $\widetilde{st} < \widehat{st}$ or ($\widetilde{st} = \widehat{st}$ and $RW_i > \widehat{RW}$)
        or ($\widetilde{st} = \widehat{st}$ and $RW_i = \widehat{RW}$ and $L[\widetilde{k}] > L[\theta]$)
        or ($\widetilde{st} = \widehat{st}$ and $RW_i = \widehat{RW}$ and $L[\widetilde{k}] = L[\theta]$ and $i < \lambda$) **then**
        $\widehat{st} \leftarrow \widetilde{st},\ \widehat{RW} \leftarrow RW_i,\ \theta \leftarrow \widetilde{k},\ \lambda \leftarrow i$
    $s[\lambda] \leftarrow \max(u[\lambda], v[\theta]),\ w[\lambda] \leftarrow \theta,\ C_{\max} \leftarrow \max(C_{\max}, s[\lambda] + p_{\lambda\theta})$
    $v[\theta] \leftarrow s[\lambda] + p_{\lambda\theta},\ \Omega \leftarrow \Omega \setminus \{\lambda\}$
    foreach $k \in F_\lambda$ **do** $L[k] \leftarrow L[k] - p_{\lambda k}$
    foreach $i \in \mathcal{S}_\lambda$ **do**
      $\eta[i] \leftarrow \eta[i] - 1,\ u[i] \leftarrow \max(u[i], s[\lambda] + p_{\lambda\theta})$
      if $\eta[i] = 0$ **then** $\Omega \leftarrow \Omega \cup \{i\}$
  return $s[i], w[i]$ $(i = 1, \ldots, o), C_{\max}$
end

4 Beam search method

At each iteration of the list scheduling algorithm described in the previous section, an operation is selected, assigned to a machine, and its starting time is determined. On the one hand, all decision are made based on heuristic rules. On the other hand, even if we were able to make a decision based on a local optimal strategy, it is well-known that greedy algorithms not necessarily provide good quality solutions. Therefore, paying the price of increasing the complexity of the method, it makes sense to select, at each iteration, not a single operation to be handled, but a small set of operations to be handled. In this way, a partial solution can be split into several partial solutions with one more handled operation. This strategy gives rise to a search tree and fits into the framework of a beam search method. In this sense, we say that the presented beam search method is a "natural extension" of the list scheduling method described in the previous section.

The introduced beam search method is a filtered beam search method with approximately $f(\beta)$ nodes at each level of the search tree, where $\beta \in (0, 1]$ is a given real parameter and $f : I\!R \to I\!N$ is a function to be defined later. Each node at a given level ℓ of the search tree represents a partial solution with ℓ handled operations. From the point of view of the beam search method, we can say that the (simple, thus non-expensive) rules 1–3 described in the previous section place the role of the *local* evaluation used to choose the $g(\alpha)$ more promising operations that would be considered to add one more handled operation to a given partial solution (where $\alpha \in (0, 1]$ is a given real parameter and $g : I\!R \to I\!N$ is a function to be defined later). Therefore, a partial solution may be split into $g(\alpha)$ partial solutions with one more handled operation. Then, a *global* (and more time consuming) evaluation is considered in order to select one partial solutions among the set of $g(\alpha)$ partial solutions. As expected, the global evaluation consists in completing each partial solution by running to the end the list scheduling algorithm described in the previous section. The one with the smallest makespan is chosen as the child of the parent being considered and, in this way, a new level of the search tree is built. In this work, we adopted the strategy of keeping a single child for each node as suggested in [30].

4.1 Local and global strategies

We now describe how rules 1–3 from the list scheduling algorithm are used to generate the children of a given node or partial solution. Consider a partial solution with $\ell < o$ scheduled operations, or, equivalently, a node at level ℓ of the search tree, given by $(\ell, s, w, u, v, L, C_{\max}, \eta, \Omega)$. Rules 1–3 can be applied to select a pair $(i_1, k_1) \in \Psi_0$ and a starting time $st_{i_1k_1}$ for operation i_1 on machine k_1; and the pair (i_1, k_1) with the starting time $st_{i_1k_1}$ can be used to generate a partial solution with an additional handled operation. We will say that the generated child is the *most promising* child from the point of view of the local strategy. Assume now that the pair (i_1, k_1) is forbidden and that rules 1–3 are applied again (to the node at level ℓ). Then, another pair $(i_2, k_2) \in \Psi_0$ is selected and a starting time $st_{i_2k_2}$ for operation i_2 on machine k_2 determined. This pair can also be used to generate a second child. We will say that this second child is the *second more promising* child from the point of view of the local strategy.

In the way described in the previous paragraph, several children may be generated. On the one hand, since a relatively expensive global strategy must be applied to all children in order

to keep a single child, a limit on the number of generated children is imposed. This limit is a function of the input parameter $\alpha \in (0, 1]$ and, ideally, should be independent of the size of the instance. However, to avoid a degradation on the method's performance for instances of increasing sizes, we considered a limit given by $\hat{\alpha}_1 \equiv \lceil \alpha |\Psi_0| \rceil$. Note that Ψ_0 (as defined in (9)) depends on the partial solution being considered and, therefore, $\hat{\alpha}_1$ may vary from node to node. On the other hand, a second limit that depends on the input parameter $\xi > 0$ and aims to preserve the children's quality is also imposed. Note that rule 1 constructs the set Ψ_1 (see 12) picking up from Ψ_0 (see (9)) the pairs (i, k) such that $st_{ik} = \widehat{st}$, where st_{ik} and $\widehat{st}$ are given by (10) and (11), respectively. This means that (in the list scheduling algorithm) only pairs operation/machine with the earliest possible starting time are candidates for producing a new partial solution with an additional handled operation. However, this may not be the case when rules 1–3 are applied iteratively to pick several pairs up from Ψ_0. Therefore, it makes sense to impose an upper bound on the starting time of the selected pairs. Consider the set

$$\Psi_{0.5} = \{(i, k) \mid (i, k) \in \Psi_0 \text{ and } st_{ik} \leq \widehat{st} + \xi\ \widehat{p}\} \subseteq \Psi_0, \tag{14}$$

where

$$\widehat{st} = \min\{st_{ik} \mid (i, k) \in \Psi_0\} \text{ and } \widehat{p} = \max\{p_{ik} \mid (i, k) \in \Psi_0\}. \tag{15}$$

By imposing that no more than $\hat{\alpha}_2 \equiv |\Psi_{0.5}|$ children can be generated, we guarantee that the children's starting time will not be larger than $\widehat{st} + \xi\ \widehat{p}$. This means that the bound on the number of children is given by $\hat{\alpha} \equiv \min\{\hat{\alpha}_1, \hat{\alpha}_2\}$. This defines the function $g(\alpha)$ that depends on the node and determines how many children will be chosen by the local strategy.

We aim at generating a single child for each node (or, in other words, keeping a single child open and closing or bounding all the others). Given the node $(\ell, s, w, u, v, L, C_{\max}, \eta, \Omega)$, a natural way to do that would be to complete the partial solution it represents in different ways by selecting every possible pair in $(i, k) \in \Psi_0$ (associated with the earliest starting time of operation i on machine k) and then completing all those partial solutions, that have $\ell + 1$ scheduled operations, using the list scheduling algorithm. Since applying the list scheduling algorithm to all the $|\Psi_0|$ children may be too expensive, the strategy described in the previous paragraph *filters* the children preserving only the $\hat{\alpha}$ more promising children.

After having selected a small set of promising children, the so called global strategy is used to pick a single child from the set. As already suggested, the global strategy consists of computing the makespan of a solution that can be obtained by completing each child using the list scheduling algorithm. Note that we are not interested in the completed solution, but only in its makespan. The completed solution has an associated makespan that we will name $C^{\text{est}}_{\max}$. Among all children, the one with the smallest $C^{\text{est}}_{\max}$ is the chosen one, while all the others are discarded, closed, or bounded. A few technical details related to tie breaks between children and some other minor details will be given below, when describing the pseudo-code of the introduced beam search method.

4.2 Pseudo-code and complexity

The whole beam search method is described in Algorithms 2–5. Algorithm 2 corresponds to the initializations and to the construction of the first level of the search tree, that is slightly

different from the construction of the other levels. Algorithm 5 is the core of the beam search method that iterates $o-1$ times generating the successive levels of the search tree. Algorithm 4 consists of subroutine SELECT that receives a partial solution and implements rules 1–3 to select a pair operation/machine that can be used to expand the given partial solution. Algorithm 3 consists of subroutine CMAXEST that receives a partial solution and, by completing it using the list scheduling algorithm, computes an estimate of the completion time that could be obtained by completing the given partial solution. Algorithm 3 is a re-implementation of Algorithm 1 in the form of a subroutine that does not perform the initializations and receives a partial solution. Subroutine SELECT plays the role of the local evaluation while subroutine CMAXEST plays the role of the global evaluation.

The input parameters of Algorithm 2 are the data that describes the instance (n, m, o, F_i $(i = 1, \ldots, o)$, p_{ik} $(i = 1, \ldots, o, k \in F_i)$, A) plus the scalar real values $\xi > 0$ and $\alpha, \beta \in (0, 1]$. Parameter ξ is related to the tolerance for the earliest starting time st_{ik} of a candidate operation/machine pair (i, k) with respect to the minimum earliest starting time $\widehat{st}$ among all pairs in Ψ_0, as described in (14,15). Parameter α is related to the number of children of a given node that are selected by the local strategy and to which the global strategy is applied in order to keep a single child per node. Parameter β determines the number of nodes that remain open at each level of the search tree.

Lines 2 to 16 of Algorithm 2 initializes (in the same way it is done in Algorithm 1) $\mathcal{P}_i$, $\mathcal{S}_i$, $\bar{p}_i$, and RW_i $(i = 1, \ldots, o)$. Quantities $C'_{\max}$, u', v', L', η', and the set Ω' associated with the partial solution with none scheduled operation (i.e. the root node of the search tree) are initialized in lines 17–21. Since the "quality" of the partial solutions at the first level of the search tree has a strong influence in the overall performance of the method, no filter is considered to construct the first level. It means that all possible pairs operation/machine will be evaluated by the global strategy and the most promising ones will constitute the first level of the search tree (partial solutions with a single scheduled or handled operation). Lines 22 to 34 are devoted to this task. All possible pairs operation/machine (i.e. the ones in Ψ_0 as defined in (9)) are considered, the partial solutions are constructed, its estimated completion times are computed (see the call to subroutine CMAXEST at line 33), and the partial solutions are saved in the set $\mathcal{N}'$. In lines 35 and 36, the fraction β of the most promising partial solutions in $\mathcal{N}'$ is saved into the set $\mathcal{N}$ that turns out to be the first level of the search tree. Note that due to possible ties, the cardinality of $\mathcal{N}$ may be larger than $\hat{\beta} \equiv \lceil \beta \ |\mathcal{N}'| \rceil$. In line 37 subroutine BEAMSEARCH (implemented in Algorithm 5) is called. It receives the first level of the search tree, constructs the remaining of the tree, and returns the best leave.

Algorithm 5 implements subroutine BEAMSEARCH. It iterates $o-1$ times (see the main loop in line 2). Each iteration starts with the set $\mathcal{N}$ composed by the nodes of the current level of the search tree. Nodes in $\mathcal{N}$ are temporary labeled (numbered from 1 to $|\mathcal{N}|$) with the identifier *nid* (that stands for "node identifier") (see lines 3 and 5) and a set of children $\mathcal{D}_{\text{nid}}$ for each node is constructed. The limit on the number of children of each node was already described in the previous subsection and is implemented in lines 6–16. Then, a single child for each node is selected and saved into the set $\mathcal{N}'$. The iteration finishes replacing $\mathcal{N}$ by $\mathcal{N}'$. It means that no tree structure is in fact build, since only the nodes of the current level are needed as each node carries full information of the partial solution it represents.

The set of children $\mathcal{D}_{\text{nid}}$ of a given node identified by *nid* is build in lines 17–28. The set $\mathcal{D}_{\text{nid}}$ is

initialized as the empty set in line 17. Exactly $\min\{\hat{\alpha}_1, \hat{\alpha}_2\}$ children are generated, where $\hat{\alpha}_1$ is the one computed in line 6 and $\hat{\alpha}_2$ is computed in lines 12–16. A copy $(s', w', C'_{\max}, u', v', L', \eta', \Omega')$ of the current solution $(s, w, C_{\max}, u, v, L, \eta, \Omega)$ is made at line 19. A pair operation/machine (λ, θ) is selected (at line 20) by calling to subroutine SELECT. The new partial solution with an extra handled operation is build (lines 21–26) and its estimated completion time $C^{\text{est}}_{\max}$ is computed by calling to subroutine CMAXEST at line 27. The new partial solution, together with λ, θ, and $C^{\text{est}}_{\max}$, is saved in $\mathcal{D}_{\text{nid}}$. To guarantee that a different pair operation/machine will be selected in the forthcoming calls to subroutine SELECT (to generate the siblings of the just generated partial solution), the selected pair (λ, θ) is added to the set $\mathcal{F}$, that stands for "forbidden pairs". Subroutine SELECT (see Algorithm 4) implements rules 1–3 with the only difference that the *most promising pair* is chosen from $\Psi_0 \setminus \mathcal{F}$.

It remains to explain how a single child for node *nid* is chosen from the set $\mathcal{D}_{\text{nid}}$ for $nid = 1, \ldots, |\mathcal{N}|$. This selection is based on the value of $C^{\text{est}}_{\max}$ and it uses (λ, θ) to solve tie-breaks. Other than that, there is an additional task that requires to inspect all $\mathcal{D}_{\text{nid}}$ simultaneously: to avoid identical children of different partial solutions in $\mathcal{N}$. Note that all nodes in the first level of the search tree are different (since they consist of partial solutions with a single handled operation coming from different pairs operation/machine). However, when expanded, by considering an additional handled operation, two different partial solutions may become identical. Moreover, if some level of the search tree has identical solutions, their expansions will continue being identical to the end, affecting the diversity of the leaves (complete solutions). To avoid this situation, before choosing a single child per node, identical partial solutions are removed from $\cup\mathcal{D}_{\text{nid}}$. If identical partial solutions exist, the one with smallest λ is preserved and, in case of ties, the one with smallest θ. (This strategy solves all possible ties since identical partial solutions can not have identical λ and θ. Otherwise, there must be identical parents, which is not the case.) This elimination of identical partial solutions is done in the loop that goes form line 30 to line 36, marking the solutions to be removed by setting $C^{\text{est}}_{\max}$ equal to ∞. Solutions are in fact removed in line 37. Finally, the best child of each parent is chosen in lines 38 and 39 and saved in the set $\mathcal{N}'$ that, at the end of the iteration, substitutes the current set of nodes $\mathcal{N}$ (see line 40).

A few words regarding the complexity of the beam search method described by Algorithms 2–5 are in order. Algorithm 3 is $O(oq + |A|)$ while Algorithm 4 is $O(q)$ with q given by (13), as already analysed in Section 3.3. Algorithms 2 (disregarding line 37 where subroutine BEAM-SEARCH described by Algorithm 5 is called, i.e. considering initializations and construction of the first level of the search tree only) has a worst-case time complexity

$$O\left(\sum_{i=1}^{o} |F_i| + r + oq^2 + q|A|\right),$$

where

$$r = |F_{i_1}||S_{i_1}| + |F_{i_2}||S_{i_2}| + \ldots + |F_{i_w}||S_{i_w}|,$$

w is the size of a maximum antichain in $D = (\{1, \ldots, o\}, A)$, and the indices $i_1, i_2, \ldots, i_o$ are such that $|F_{i_1}||S_{i_1}| \geq |F_{i_2}||S_{i_2}| \geq \ldots \geq |F_{i_o}||S_{i_o}|$. To analyze Algorithm 5, we will first assume that $\hat{\alpha}$ and $\hat{\beta}$ are constants that do not depend on the instance size. It implies that each level of the search tree has $|\mathcal{N}| \leq \hat{\beta}$ and that $|\mathcal{D}_{\text{nid}}| \leq \hat{\alpha}$ for $nid = 1, \ldots, |\mathcal{N}|$. Thus, we have that the

worst-case time complexity of Algorithm 5 is given by

$$O\left(\hat{\alpha}\hat{\beta}(o|A| + o^2q) + \hat{\alpha}^2o\right). \tag{16}$$

However, in practice, we consider $\hat{\alpha} = O(q)$ and $\hat{\beta} = O(q)$. Therefore, (16) becomes

$$O\left(oq^2|A| + o^2q^3\right).$$

Summing up, the overal worst-case time complexity of the introduced beam search method is

$$O\left(r + q|A| + \hat{\alpha}\hat{\beta}(o|A| + o^2q) + \hat{\alpha}^2o\right),$$

if $\hat{\alpha}$ and $\hat{\beta}$ are given constants or $O\left(r + oq^2|A| + o^2q^3\right)$, if $\hat{\alpha}$ and $\hat{\beta}$ are $O(q)$.

Algorithm 2: Beam Search initialization and construction of the search tree's first level.

Input: m, o, F_i $(i = 1, \ldots, o)$, p_{ik} $(i = 1, \ldots, o, k \in F_i)$, A, α, β, ξ
Output: $s^*[i]$, $w^*[i]$ $(i = 1, \ldots, o)$, $C^*_{\max}$
BeamSearchIni$(m, o, F, p, A, \alpha, \beta, \xi)$

begin
  for $i \leftarrow 1$ **to** o **do** $\mathcal{P}_i \leftarrow \emptyset$, $\mathcal{S}_i \leftarrow \emptyset$
  foreach $(i, j) \in A$ **do** $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup \{i\}$, $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{j\}$
  for $i \leftarrow 1$ **to** o **do**
    $\bar{p}_i \leftarrow 0$
    foreach $k \in F_i$ **do** $\bar{p}_i \leftarrow \bar{p}_i + p_{ik}$
    $\bar{p}_i \leftarrow \bar{p}_i / |F_i|$
  $\mathcal{Q} \leftarrow \{i \mid |\mathcal{S}_i| = 0\}$
  for $i \leftarrow 1$ **to** o **do**
    if $i \in \mathcal{Q}$ **then** $RW_i \leftarrow \bar{p}_i$ **else** $RW_i \leftarrow 0$
  while $\mathcal{Q} \neq \emptyset$ **do**
    Let $z \in \mathcal{Q}$, $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{z\}$
    foreach $i \in \mathcal{P}_z$ **do**
      if $RW_i < RW_z + \bar{p}_i$ **then**
        $RW_i \leftarrow RW_z + \bar{p}_i$
        if $i \notin \mathcal{Q}$ **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{i\}$
  $C'_{\max} \leftarrow 0$, $u'[1 \ldots o] \leftarrow 0$, $v'[1 \ldots m] \leftarrow 0$, $L'[1 \ldots m] \leftarrow 0$
  for $i \leftarrow 1$ **to** o **do**
    foreach $k \in F_i$ **do** $L'[k] \leftarrow L'[k] + p_{ik}$
  for $i \leftarrow 1$ **to** o **do** $\eta'[i] \leftarrow |\mathcal{P}_i|$
  $\Omega' \leftarrow \{i \mid \eta'[i] = 0\}$, $\mathcal{N}' \leftarrow \emptyset$
  foreach $\lambda \in \Omega$ **do**
    $L, \eta, \Omega \leftarrow L', \eta', \Omega'$
    foreach $k \in F_\lambda$ **do** $L[k] \leftarrow L[k] - p_{\lambda k}$
    foreach $i \in \mathcal{S}_\lambda$ **do**
      $\eta[i] \leftarrow \eta[i] - 1$
      if $\eta[i] = 0$ **then** $\Omega \leftarrow \Omega \cup \{i\}$
    $\Omega \leftarrow \Omega \setminus \{\lambda\}$
    foreach $\theta \in F_\lambda$ **do**
      $s, w, C_{\max}, u, v \leftarrow s', w', C'_{\max}, u', v'$
      $s[\lambda] \leftarrow \max(u[\lambda], v[\theta])$, $w[\lambda] \leftarrow \theta$, $C_{\max} \leftarrow \max\{C_{\max}, s[\lambda] + p_{\lambda\theta}\}$, $v[\theta] \leftarrow s[\lambda] + p_{\lambda\theta}$
      foreach $i \in \mathcal{S}_\lambda$ **do** $u[i] \leftarrow \max(u[i], s[\lambda] + p_{\lambda\theta})$
      $C^{\text{est}}_{\max} \leftarrow$ CmaxEst$(m, o, F, p, \mathcal{S}, RW, 1, s, w, C_{\max}, u, v, L, \eta, \Omega)$
      $\mathcal{N}' \leftarrow \mathcal{N}' \cup \{(1, s, w, C_{\max}, u, v, L, \eta, \Omega, C^{\text{est}}_{\max})\}$
  $\hat{\beta} \leftarrow \lceil \beta \, |\mathcal{N}'| \rceil$ and let $\widehat{C}^{\text{est}}_{\max}$ be the $\hat{\beta}$-th smallest $C^{\text{est}}_{\max}$ such that $(\cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, C^{\text{est}}_{\max}) \in \mathcal{N}'$
  $\mathcal{N} \leftarrow \{(1, s, w, C_{\max}, u, v, L, \eta, \Omega) \mid (1, s, w, C_{\max}, u, v, L, \eta, \Omega, C^{\text{est}}_{\max}) \in \mathcal{N}' \text{ and } C^{\text{est}}_{\max} \leq \widehat{C}^{\text{est}}_{\max}\}$
  $s^*, w^*, C^*_{\max} \leftarrow$ BeamSearch$(m, o, F, p, \mathcal{S}, RW, \alpha, \xi, \mathcal{N})$
  return $s^*[i], w^*[i]$ $(i = 1, \ldots, o), C^*_{\max}$
end

Algorithm 3: Completion time estimation procedure (global evaluation).

Input: m, o, ℓ, $C_{\max}$, Ω, $u[i]$, RW_i, $\mathcal{S}_i$, F_i, $\eta[i]$, $s[i]$, $w[i]$ $(i = 1, \ldots, o)$, p_{ik} $(i = 1, \ldots, o, k \in F_i)$, $v[k]$, $L[k]$ $(k = 1, \ldots, m)$

Output: $C_{\max}$

CmaxEst$(m, o, F, p, \mathcal{S}, RW, \ell, s, w, C_{\max}, u, v, L, \eta, \Omega)$

begin
  do $o - \ell$ **times**
    $\lambda, \theta \leftarrow$ Select$(m, o, F, p, RW, u, v, L, \Omega, \emptyset)$
    $s[\lambda] \leftarrow \max(u[\lambda], v[\theta])$, $w[\lambda] \leftarrow \theta$, $C_{\max} \leftarrow \max\{C_{\max}, s[\lambda] + p_{\lambda\theta}\}$
    $v[\theta] \leftarrow s[\lambda] + p_{\lambda\theta}$, $\Omega \leftarrow \Omega \setminus \{\lambda\}$
    foreach $k \in F_\lambda$ **do** $L[k] \leftarrow L[k] - p_{\lambda k}$
    foreach $i \in \mathcal{S}_\lambda$ **do**
      $\eta[i] \leftarrow \eta[i] - 1$, $u[i] \leftarrow \max(u[i], s[\lambda] + p_{\lambda\theta})$
      if $\eta[i] = 0$ **then** $\Omega \leftarrow \Omega \cup \{i\}$
  return $C_{\max}$
end

Algorithm 4: Selection procedure (local evaluation).

Input: m, o, $u[i]$, RW_i, F_i $(i = 1, \ldots, o)$, p_{ik} $(i = 1, \ldots, o,\ k \in F_i)$, $v[k]$, $L[k]$ $(k = 1, \ldots, m)$, Ω, $\mathcal{F}$

Output: λ, θ

Select$(m, o, F, p, RW, u, v, L, \Omega, \mathcal{F})$

begin
  $\widehat{st} \leftarrow \infty$, $\widehat{RW} \leftarrow 0$
  foreach $i \in \Omega$ **do**
    $\widetilde{st} \leftarrow \infty$, $\widetilde{pt} \leftarrow \infty$
    foreach $k \in F_i$ **do**
      if $(i, k) \notin \mathcal{F}$ **then**
        $st \leftarrow \max(u[i], v[k])$
        if $st < \widetilde{st}$ or ($st = \widetilde{st}$ and $p_{ik} < \widetilde{pt}$)
          or ($st = \widetilde{st}$ and $p_{ik} = \widetilde{pt}$ and $L[k] < L[\widetilde{k}]$)
          or ($st = \widetilde{st}$ and $p_{ik} = \widetilde{pt}$ and $L[k] = L[\widetilde{k}]$ and $k < \widetilde{k}$) **then**
          $\widetilde{st} \leftarrow st$, $\widetilde{pt} \leftarrow p_{ik}$, $\widetilde{k} \leftarrow k$
    if $\widetilde{st} < \widehat{st}$ or ($\widetilde{st} = \widehat{st}$ and $RW_i > \widehat{RW}$)
      or ($\widetilde{st} = \widehat{st}$ and $RW_i = \widehat{RW}$ and $L[\widetilde{k}] > L[\theta]$)
      or ($\widetilde{st} = \widehat{st}$ and $RW_i = \widehat{RW}$ and $L[\widetilde{k}] = L[\theta]$ and $i < \lambda$) **then**
      $\widehat{st} \leftarrow \widetilde{st}$, $\widehat{RW} \leftarrow RW_i$, $\theta \leftarrow \widetilde{k}$, $\lambda \leftarrow i$
  return λ, θ
end

Algorithm 5: Beam Search body.

Input: m, o, RW_i, $\mathcal{S}_i$, F_i $(i = 1, \ldots, o)$, p_{ik} $(i = 1, \ldots, o,\ k \in F_i)$, α, ξ, $\mathcal{N}$
Output: $s^*[i]$, $w^*[i]$ $(i = 1, \ldots, o)$, $C^*_{\max}$
BeamSearch(m, o, F, p, $\mathcal{S}$, RW, α, ξ, $\mathcal{N}$)

begin
  do $o - 1$ **times**
    nid $\leftarrow 0$
    foreach $(\ell, s, w, C_{\max}, u, v, L, \eta, \Omega) \in \mathcal{N}$ **do**
      nid $\leftarrow$ nid $+ 1$
      $\hat{\alpha}_1 \leftarrow \lceil\ \alpha\ |\{(i,k) \mid i \in \Omega \text{ and } k \in F_i\}|\ \rceil$
      $st_{\min} \leftarrow \infty,\ pt_{\max} \leftarrow 0$
      foreach $i \in \Omega$ **do**
        foreach $k \in F_i$ **do**
          $st \leftarrow \max(u[i], v[k])$
          $st_{\min} \leftarrow \min\{st_{\min}, st\},\ pt_{\max} \leftarrow \max\{pt_{\max}, p_{ik}\}$
      $\hat{\alpha}_2 \leftarrow 0$
      foreach $i \in \Omega$ **do**
        foreach $k \in F_i$ **do**
          $st \leftarrow \max(u[i], v[k])$
          if $st \leq st_{\min} + \xi\ pt_{\max}$ **then** $\hat{\alpha}_2 \leftarrow \hat{\alpha}_2 + 1$
      $\mathcal{D}_{\text{nid}} \leftarrow \emptyset,\ \mathcal{F} \leftarrow \emptyset$
      do $\min\{\hat{\alpha}_1, \hat{\alpha}_2\}$ **times**
        $s', w', C'_{\max}, u', v', L', \eta', \Omega' \leftarrow s, w, C_{\max}, u, v, L, \eta, \Omega$
        $\lambda, \theta \leftarrow$ Select($m, o, F, p, RW, u', v', L', \Omega', \mathcal{F}$)
        $s'[\lambda] \leftarrow \max(u'[\lambda], v'[\theta]),\ w'[\lambda] \leftarrow \theta,\ C'_{\max} \leftarrow \max\{C'_{\max}, s'[\lambda] + p_{\lambda\theta}\}$
        $v'[\theta] \leftarrow s'[\lambda] + p_{\lambda\theta},\ \Omega' \leftarrow \Omega' \setminus \{\lambda\}$
        foreach $k \in F_\lambda$ **do** $L'[k] \leftarrow L'[k] - p_{\lambda k}$
        foreach $i \in \mathcal{S}_\lambda$ **do**
          $\eta'[i] \leftarrow \eta'[i] - 1,\ u'[i] \leftarrow \max(u'[i], s'[\lambda] + p_{\lambda\theta})$
          if $\eta'[i] = 0$ **then** $\Omega' \leftarrow \Omega' \cup \{i\}$
        $C^{\text{est}}_{\max} \leftarrow$ CmaxEst($m, o, F, p, \mathcal{S}, RW, \ell + 1, s', w', C'_{\max}, u', v', L', \eta', \Omega'$)
        $\mathcal{D}_{\text{nid}} \leftarrow \mathcal{D}_{\text{nid}} \cup \{(\lambda, \theta, \ell + 1, s', w', C'_{\max}, u', v', L', \eta', \Omega', C^{\text{est}}_{\max})\},\ \mathcal{F} \leftarrow \mathcal{F} \cup \{(\lambda, \theta)\}$
      $\mathcal{N}' \leftarrow \emptyset$
      for $\text{nid}_1 \leftarrow 1$ **to** $|\mathcal{N}|$ **do**
        foreach $(\lambda_1, \theta_1, \ell + 1, s, w, C_{\max}, u, v, L, \eta, \Omega, C^{\text{est}}_{\max}) \in \mathcal{D}_{\text{nid}_1}$ **do**
          for $\text{nid}_2 \leftarrow \text{nid}_1 + 1$ **to** $|\mathcal{N}|$ **do**
            foreach $(\lambda_2, \theta_2, \ell + 1, s', w', C'_{\max}, u', v', L', \eta', \Omega', C^{\text{est}\,\prime}_{\max}) \in \mathcal{D}_{\text{nid}_2}$ **do**
              if ($s = s'$ and $w = w'$) or ($\ell + 1 = o$ and $C_{\max} = C'_{\max}$) **then**
                if $\lambda_1 < \lambda_2$ or ($\lambda_1 = \lambda_2$ and $\theta_1 < \theta_2$) **then** $C^{\text{est}\,\prime}_{\max} \leftarrow \infty$
                else $C^{\text{est}}_{\max} \leftarrow \infty$
        Remove from $\mathcal{D}_{\text{nid}_1}$ the elements with $C^{\text{est}}_{\max} = \infty$.
        if $\mathcal{D}_{\text{nid}_1} \neq \emptyset$ **then**
          Let $(\lambda, \theta, \ell + 1, s, w, C_{\max}, u, v, L, \eta, \Omega, C^{\text{est}}_{\max}) \in \mathcal{D}_{\text{nid}_1}$ with smallest $C^{\text{est}}_{\max}$. In case of ties, consider the one with smallest λ and in case of ties consider the one with smallest θ. $\mathcal{N}' \leftarrow \mathcal{N}' \cup \{(\ell + 1, s, w, C_{\max}, u, v, L, \eta, \Omega)\}$.
      $\mathcal{N} \leftarrow \mathcal{N}'$
  Let $(o, s^*, w^*, C^*_{\max}, u, v, L, \eta, \Omega) \in \mathcal{N}$ with smallest $C^*_{\max}$.
  return $s^*[i], w^*[i]$ $(i = 1, \ldots, o), C^*_{\max}$
end

5 Numerical experiments

We implemented the list scheduling algorithm (Algorithm 1) and the beam search method (Algorithms 2–5) in order to be able to evaluate their efficiency and efficacy. Codes, fully written in C/C++ and available for download at *http://www.ime.usp.br/∼egbirgin/*, were compiled with the g++ compiler of GCC (version 4.7.2, Debian 4.7.2-5) using the optimization option "-O3". All tests were conducted on a 2.40GHz Intel(R) Xeon(R) E5645 with 132GB of RAM memory and running GNU/Linux operating system (Debian 7, kernel 3.7.6 SMP x86_64). Detailed descriptions of the numerical results (including a complete description of the obtained solutions) are also available for download together with the source codes of the implemented methods.

In order to have a way to evaluate the behavior of the introduced methods when applied to small and medium-sized instances, we applied an exact solver to the 50 instances of model (1–8) introduced in [3]. Instances YFJS01–YFJS20 correspond to instances composed by two independent sequences of operations followed by an assembling operation that joins the previously processed components (named Y-jobs in [3]); while instances DAFJS01–DAFJS30 correspond to instances composed by jobs whose precedences are given by arbitrary directed acyclic graphs. Tables 1 and 2 summarize the main characteristics of each instance. Figure 3 and 4 illustrate the kind of operations' precedence constraints that are present in each set of instances. It is worth noting that other complicating issues of the instances (like the routing flexibility given by the fact that each operation can be performed by a subset of machines) are not being displayed in the pictures. As exact solver, we used the IBM ILOG CPLEX 12.1 solver with the following settings: 1 for the maximum number of threads and 2048MB for working memory. All other parameters were left at their default values. In a first run, the CPU time limit was set to 1 hour and, in a second run, considering only the instances for which the optimal solutions was not found in the first run, the CPU time limit was set to 10 hours. Tables 3 and 4 show the results for the sets of instances YFJS01–YFJS20 and DAFJS01–DAFJS30, respectively. In the tables, "mks" stands for makespan and "CPU(s)" stands for the elapsed CPU time in seconds. In the cases in which the exact solver achieved the CPU time limit without finding an optimal solution, the tables report the obtained lower and upper bounds and the relative gap (given by the difference between the bounds divided by the upper bound). It is worth noting (in Table 3) that the exact solver was able to find the optimal solution in 14 (YFJS01–YFJS14) out of the 20 instances YFJS01–YFJS20, while a gap smaller than 1% was also obtained in 2 other instances (YFJS15 and YFJS16). On the other hand, Table 4 shows that the exact solver found the optimal solution in only 4 instances out of 30 and a gap smaller than 1% in a single instance when considering the set of instances DAFJS01–DAFJS30. These results will explain the "improvements" obtained by the introduced heuristic methods when compared against the solutions obtained by the exact solver (with a CPU time limit).

In a first set of experiments, we aim to evaluate the numerical performance of the list scheduling algorithm. Tables 5 and 6 show the results of applying the list scheduling algorithm to the sets of instances YFJS01–YFJS20 and DAFJS01–DAFJS30, respectively. For each instance, the tables display the makespan of the solution obtained by the list scheduling algorithm. A comparison with the makespan obtained by the EST heuristic presented in [3] and the makespan obtained by the exact solver with CPU time limits of 1 and 10 hours is also presented. In each case, if v_1 is the value of the makespan found by the list scheduling algorithm and v_2 is the

Name	n	m	# of operations			$\lvert F_i \rvert$			# of precedences		
			min	max	$\sum$	min	max	$\sum$	min	max	$\sum$
YFJS01	4	7	10	10	40	1	3	104	0	2	36
YFJS02	4	7	10	10	40	1	3	104	0	2	36
YFJS03	6	7	4	4	24	2	3	63	0	2	18
YFJS04	7	7	4	4	28	2	3	71	0	2	21
YFJS05	8	7	4	4	32	2	3	81	0	2	24
YFJS06	9	7	4	4	36	2	3	95	0	2	27
YFJS07	9	7	4	4	36	2	3	93	0	2	27
YFJS08	9	12	4	4	36	2	3	100	0	2	27
YFJS09	9	12	4	4	36	4	8	219	0	2	27
YFJS10	10	12	4	4	40	1	3	113	0	2	30
YFJS11	10	10	5	5	50	1	3	134	0	2	40
YFJS12	10	10	5	5	50	2	3	133	0	2	40
YFJS13	10	10	5	5	50	1	3	137	0	2	40
YFJS14	13	26	17	17	221	2	3	641	0	2	208
YFJS15	13	26	17	17	221	2	3	648	0	2	208
YFJS16	13	26	17	17	221	2	3	633	0	2	208
YFJS17	17	26	17	17	289	3	5	1328	0	2	272
YFJS18	17	26	17	17	289	3	5	1362	0	2	272
YFJS19	17	26	17	17	289	3	5	1347	0	2	272
YFJS20	17	26	17	17	289	3	5	1343	0	2	272

Table 1: Description of the instances with Y-jobs.

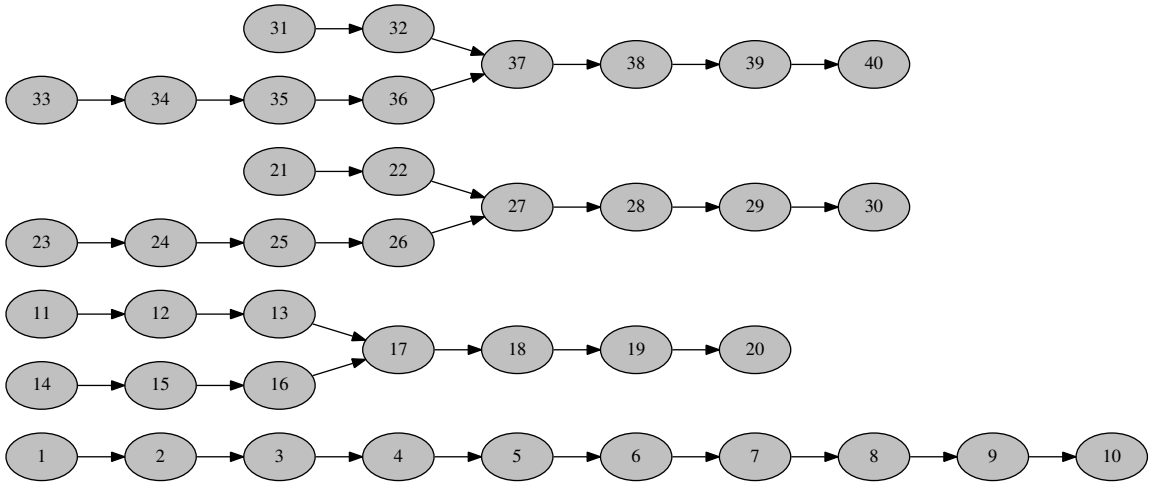


Figure 3: Graphical representation of the operations' precedence constraints of instance YFJS02.

Name	n	m	# of operations			$\lvert F_i \rvert$			# of precedences		
			min	max	$\sum$	min	max	$\sum$	min	max	$\sum$
DAFJS01	4	5	5	9	26	2	4	82	0	3	26
DAFJS02	4	5	5	7	25	2	4	79	0	3	23
DAFJS03	4	10	10	17	55	3	7	279	0	2	52
DAFJS04	4	10	9	14	43	3	7	220	0	2	40
DAFJS05	6	5	5	13	39	2	4	104	0	3	34
DAFJS06	6	5	5	13	44	2	4	136	0	3	41
DAFJS07	6	10	7	23	85	3	7	431	0	3	82
DAFJS08	6	10	6	23	85	3	7	403	0	3	82
DAFJS09	8	5	4	9	45	2	4	135	0	3	42
DAFJS10	8	5	4	11	58	2	4	168	0	3	52
DAFJS11	8	10	10	23	113	3	7	534	0	3	108
DAFJS12	8	10	9	22	117	3	7	603	0	3	114
DAFJS13	10	5	5	11	62	2	4	193	0	3	55
DAFJS14	10	5	4	10	69	2	4	206	0	3	62
DAFJS15	10	10	8	19	120	3	7	595	0	3	117
DAFJS16	10	10	6	20	120	3	7	602	0	3	114
DAFJS17	12	5	4	11	82	2	4	246	0	3	77
DAFJS18	12	5	5	9	74	2	4	231	0	2	64
DAFJS19	8	7	7	13	70	3	5	283	0	3	66
DAFJS20	10	7	6	17	92	3	5	361	0	3	87
DAFJS21	12	7	5	16	107	3	5	425	0	3	102
DAFJS22	12	7	5	17	116	3	5	450	0	3	109
DAFJS23	8	9	6	17	76	4	6	367	0	3	71
DAFJS24	8	9	6	25	92	4	6	463	0	2	87
DAFJS25	10	9	9	19	123	4	6	619	0	3	119
DAFJS26	10	9	8	17	119	4	6	606	0	3	116
DAFJS27	12	9	7	22	127	4	6	625	0	3	118
DAFJS28	8	10	8	15	91	3	7	457	0	3	89
DAFJS29	8	10	7	19	95	3	7	468	0	3	94
DAFJS30	10	10	8	19	98	3	7	509	0	3	94

Table 2: Description of the instances for which the precedence relations between the operations are given by an arbitrary directed acyclic graph.

value of the makespan found by the other method, "diff", that stands for relative difference, is given by $(v_1 - v_2)/v_2$. This means that negative values of "diff" indicate that the list scheduling algorithm found a solution of better quality. Both tables show that the introduced list scheduling method improves the solutions obtained by the heuristic EST introduced in [3]. When compared against the exact solver, we must consider the two sets of instances in separate. In the set of instances YFJS01–YFJS20 for which the exact solver found optimal solutions in most of the

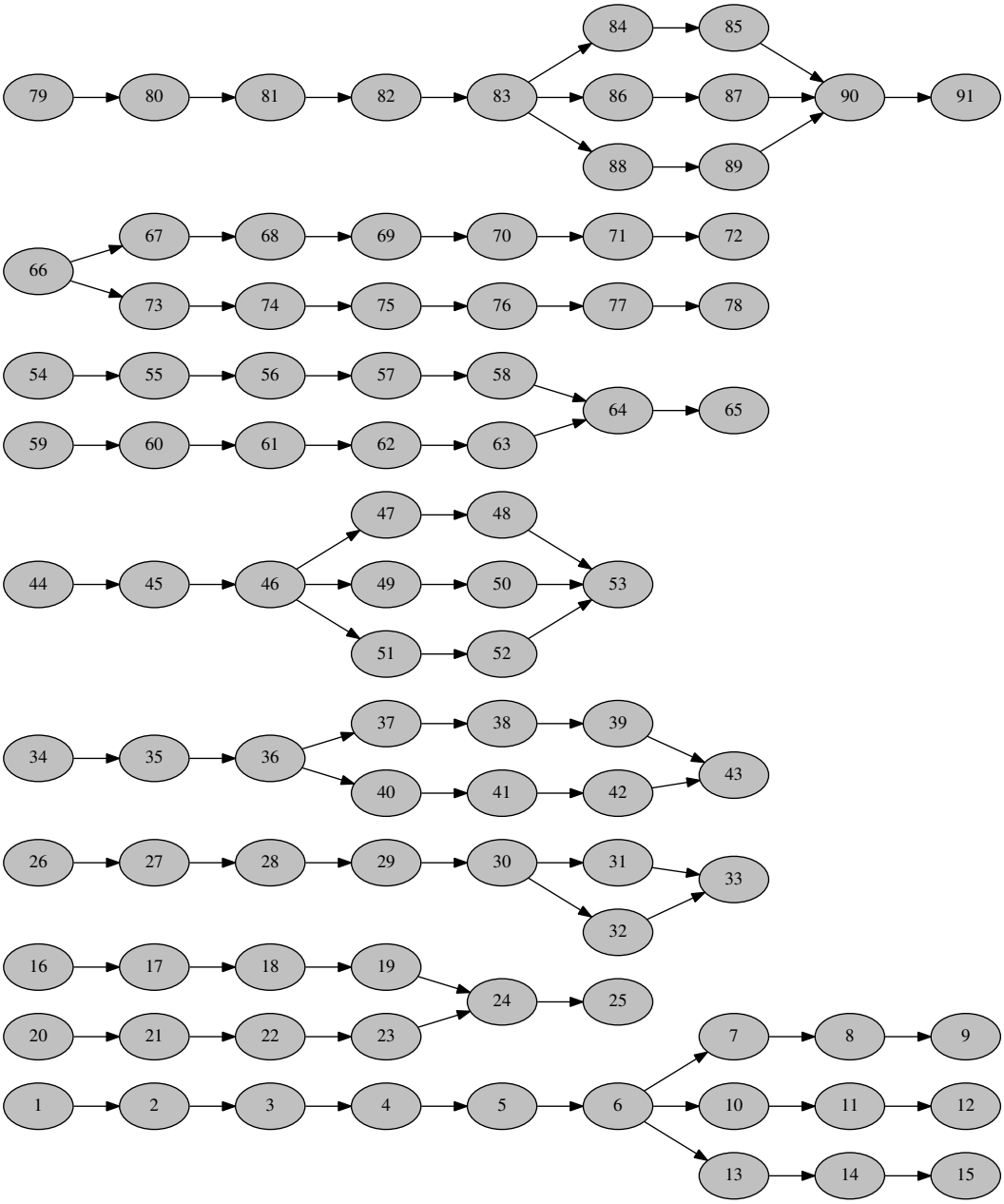


Figure 4: Graphical representation of the operations' precedence constraints of instance DAFJS28.

cases, the difference is approximately 35%. In the set of instances DAFJS01–DAFJS30, in which the exact solver was unable to find optimal solutions in most of the cases, the distance is approximately 7%. Needless to say, the list scheduling algorithm runs, for every instance, in

Instance	CPLEX with 1h of time limit		CPLEX with 10h of time limit	
	mks	CPU(s)	mks	CPU(s)
YFJS01	773	10.83	–	–
YFJS02	825	9.25	–	–
YFJS03	347	3.50	–	–
YFJS04	390	7.31	–	–
YFJS05	445	341.38	–	–
YFJS06	[427;447] 4.47%	3600	446	15072.02
YFJS07	444	1277.35	–	–
YFJS08	353	0.59	–	–
YFJS09	242	13.09	–	–
YFJS10	399	3.83	–	–
YFJS11	526	168.72	–	–
YFJS12	512	2843.79	–	–
YFJS13	405	1427.47	–	–
YFJS14	1317	3378.27	–	–
YFJS15	[1239;1244] 0.40%	3600	[1239;1244] 0.40%	36000
YFJS16	[1222;1243] 1.69%	3600	[1222;1231] 0.73%	36000
YFJS17	[1133;1622] 30.15%	3600	[1133;1290] 12.17%	36000
YFJS18	[1220;2082] 41.40%	3600	[1220;1499] 18.61%	36000
YFJS19	[926;1525] 39.28%	3600	[926;1333] 30.53%	36000
YFJS20	[968;2020] 52.08%	3600	[968;1279] 24.32%	36000

Table 3: Numerical results of applying an exact MIP solver (CPLEX) to instances YFJS01–YFJS20 of model (1-8). Following [3], an initial feasible solution computed with the heuristic EST [3] was given to CPLEX.

less than a fraction of a second. In any case, it is not our intention to compare our heuristic method against the exact solver but simple to evaluate the quality of the obtained solutions.

In a second set of experiments, we aim to evaluate the numerical performance of the introduced beam search method. As every heuristic method, it is also relevant to evaluate the influence of the method's parameters in its performance. With that purpose, we considered all 80 combinations of $\alpha \in \{0.25, 0.5, 0.75, 1\}$, $\beta \in \{0.25, 0.5, 0.75, 1\}$, and $\xi \in \{0, 0.25, 0.5, 0.75, 1\}$. For each combination of parameters, we run the beam search method and we computed the average difference against the solutions obtained by the exact solver running with a CPU time limit of 1 hour and 10 hours. Tables 7 and 8 show those average distances for the sets of instances YFJS01–YFJS20 and DAFJS01–DAFJS30, respectively. The same information is graphically presented in Figures 5a and 5b, respectively. Additionally, Figures 6a and 6b present the average elapsed CPU time (in seconds) of the beam search method for each combination of parameters.

A few comments are in order. We will focus in the comparison against the solutions obtained by the exact solver with a CPU time limit of 1 hour (the other comparison is similar). Table 7 shows that the beam search method with the combination $(\alpha, \beta, \xi) = (1, 1, 1)$ achieves an average

Instance	CPLEX with 1h of time limit		CPLEX with 10h of time limit	
	mks	CPU(s)	mks	CPU(s)
DAFJS01	257	50.17	–	–
DAFJS02	289	794.66	–	–
DAFJS03	576	10.22	–	–
DAFJS04	606	0.76	–	–
DAFJS05	[351.57;402] 12.54%	3600	[381;394] 3.30%	36000
DAFJS06	[326;431] 24.36%	3600	[326;410] 20.49%	36000
DAFJS07	[497.90;565] 11.88%	3600	[498.22;547] 8.92%	36000
DAFJS08	[628;631] 0.48%	3600	[628;631] 0.48%	36000
DAFJS09	[315;484] 34.92%	3600	[316.74;471] 32.75%	36000
DAFJS10	[336;569] 40.95%	3600	[336;538] 37.55%	36000
DAFJS11	[658;708] 7.06%	3600	[658;701] 6.13%	36000
DAFJS12	[530;720] 26.39%	3600	[530;720] 26.39%	36000
DAFJS13	[304;710] 57.18%	3600	[304;683] 55.49%	36000
DAFJS14	[358.95;838] 57.17%	3600	[358.95;775] 53.68%	36000
DAFJS15	[512;818] 37.41%	3600	[512;796] 35.68%	36000
DAFJS16	[640;831] 22.98%	3600	[640;798] 19.80%	36000
DAFJS17	[300;904] 66.81%	3600	[300;902] 66.74%	36000
DAFJS18	[322;951] 66.14%	3600	[322;878] 63.33%	36000
DAFJS19	[512;595] 13.95%	3600	[512;585] 12.48%	36000
DAFJS20	[434;815] 46.75%	3600	[434;810] 46.42%	36000
DAFJS21	[504;965] 47.77%	3600	[504;959] 47.45%	36000
DAFJS22	[464;902] 48.56%	3600	[464;896] 48.21%	36000
DAFJS23	[450;541] 16.82%	3600	[450;537] 16.20%	36000
DAFJS24	[476;660] 27.88%	3600	[476;648] 26.54%	36000
DAFJS25	[584;897] 34.89%	3600	[584;879] 33.56%	36000
DAFJS26	[565;903] 37.43%	3600	[565;898] 37.08%	36000
DAFJS27	[503;981] 48.73%	3600	[503;981] 48.73%	36000
DAFJS28	[535;662] 19.18%	3600	[535;572] 6.47%	36000
DAFJS29	[609;720] 15.42%	3600	[609;710] 14.23%	36000
DAFJS30	[467;637] 26.69%	3600	[467;615] 24.07%	36000

Table 4: Numerical results of applying an exact MIP solver (CPLEX) to instances DAFJS01–DAFJS30 of model (1-8). Following [3], an initial feasible solution computed with the heuristic EST [3] was given to CPLEX.

distance of 3.5% in the set of instances YFJS01–YFJS20. This is an interesting result if we recall that for that set of instances the exact solver found optimal solutions in most of the cases. On the other extreme of the table, with the choice of parameters $(\alpha, \beta, \xi) = (0.25, 0.25, 0)$, the difference is 8.86%. On the other hand, Figure 6a shows that with the former combination of parameters the beam search method requires, almost 2 650 seconds per instance in average,

Instance	mks	EST heuristic [3]		CPLEX (1h limit)		CPLEX (10h limit)	
		mks	diff	mks	diff	mks	diff
YFJS01	1130	1318	-14.26	773	46.18	773	46.18
YFJS02	1133	1243	-8.85	825	37.33	825	37.33
YFJS03	575	439	30.98	347	65.71	347	65.71
YFJS04	576	569	1.23	390	47.69	390	47.69
YFJS05	608	566	7.42	445	36.63	445	36.63
YFJS06	633	633	0.00	447	41.61	446	41.93
YFJS07	628	628	0.00	444	41.44	444	41.44
YFJS08	485	531	-8.66	353	37.39	353	37.39
YFJS09	402	506	-20.55	242	66.12	242	66.12
YFJS10	513	541	-5.18	399	28.57	399	28.57
YFJS11	745	740	0.68	526	41.63	526	41.63
YFJS12	744	813	-8.49	512	45.31	512	45.31
YFJS13	553	717	-22.87	405	36.54	405	36.54
YFJS14	1555	2055	-24.33	1317	18.07	1317	18.07
YFJS15	1690	2296	-26.39	1244	35.85	1244	35.85
YFJS16	1769	2006	-11.81	1243	42.32	1231	43.70
YFJS17	1734	2408	-27.99	1622	6.91	1290	34.42
YFJS18	1735	2082	-16.67	2082	-16.67	1499	15.74
YFJS19	1604	2038	-21.30	1525	5.18	1333	20.33
YFJS20	1700	2369	-28.24	2020	-15.84	1279	32.92
Average			-10.26		32.40		38.68

Table 5: Results of applying the list scheduling algorithm to the instances YFJS01–YFJS20.

while with the latter combination it requires approximately 5 seconds per instance in average. Other than that, the table shows that, considering all the 80 combinations of parameters, the differences range from 3.5% to 9.03%, showing a nice feature: the performance of the method *does not* strongly depend on a precise choice of parameters. The relation between the choice of parameters and the performance of the method can be easily seen in Figures 5a and 5b. For fixed values of α and β, the optimal value of ξ is always greater than or equal to 0.5. Moreover, as expected, the larger the value of α and β, the larger the search space and, in consequence, the better the solutions' quality and the larger the required CPU time.

If we now focus in Table 8, we can see that, in the set of instances DAFJS01–DAFJS30, for which the exact solver was unable to find optimal solutions in most of the cases, the beam search method improves the exact solver's solutions displaying differences that range from -4.94% (for the combination of parameters $(\alpha, \beta, \xi) = (0.5, 0.25, 0)$) up to -6.36% (for the combination of parameters $(\alpha, \beta, \xi) = (1, 1, 0.5)$). In the former case, considering instances DAFJS01–DAFJS30 individually, differences ranges from 12.94% up to -17.18% with an average CPU time of 1.18 seconds per instance; while in the latter case, differences ranges from 8.95% up to -17.52% with an average CPU time of 115.04 seconds per instance. Recalling that this comparison is being

Instance	mks	EST heuristic [3]		CPLEX (1h limit)		CPLEX (10h limit)	
		mks	diff	mks	diff	mks	diff
DAFJS01	321	327	-1.83	257	24.90	257	24.90
DAFJS02	350	382	-8.38	289	21.11	289	21.11
DAFJS03	631	710	-11.13	576	9.55	576	9.55
DAFJS04	607	653	-7.04	606	0.17	606	0.17
DAFJS05	505	482	4.77	402	25.62	394	28.17
DAFJS06	497	489	1.64	431	15.31	410	21.22
DAFJS07	632	717	-11.85	565	11.86	547	15.54
DAFJS08	706	847	-16.65	631	11.89	631	11.89
DAFJS09	533	535	-0.37	484	10.12	471	13.16
DAFJS10	621	629	-1.27	569	9.14	538	15.43
DAFJS11	767	708	8.33	708	8.33	701	9.42
DAFJS12	727	720	0.97	720	0.97	720	0.97
DAFJS13	768	766	0.26	710	8.17	683	12.45
DAFJS14	888	871	1.95	838	5.97	775	14.58
DAFJS15	788	818	-3.67	818	-3.67	796	-1.01
DAFJS16	808	831	-2.77	831	-2.77	798	1.25
DAFJS17	935	910	2.75	904	3.43	902	3.66
DAFJS18	939	951	-1.26	951	-1.26	878	6.95
DAFJS19	598	601	-0.50	595	0.50	585	2.22
DAFJS20	854	815	4.79	815	4.79	810	5.43
DAFJS21	937	965	-2.90	965	-2.90	959	-2.29
DAFJS22	826	902	-8.43	902	-8.43	896	-7.81
DAFJS23	548	632	-13.29	541	1.29	537	2.05
DAFJS24	687	674	1.93	660	4.09	648	6.02
DAFJS25	885	897	-1.34	897	-1.34	879	0.68
DAFJS26	915	903	1.33	903	1.33	898	1.89
DAFJS27	982	981	0.10	981	0.10	981	0.10
DAFJS28	633	703	-9.96	662	-4.38	572	10.66
DAFJS29	800	760	5.26	720	11.11	710	12.68
DAFJS30	640	657	-2.59	637	0.47	615	4.07
Average			-2.20		5.73		8.40

Table 6: Results of applying the list scheduling algorithm to the instances DAFJS01–DAFJS30.

done with the exact solver that run with a CPU time limit of 1 hour (that was achieved in most of the cases; see Table 4), this means the beam search method is able to find high quality solutions in a small elapsed time.

In a final experiment, we evaluate the performance of the introduced beam search method (for the combination of parameters $(\alpha, \beta, \xi) = (1, 1, 1)$) when applied to the classical FJSP (without sequencing flexibility). We considered the sets of instances introduced in [2, 4, 6, 13].

α	ξ	diff vs. CPLEX (1h limit)				diff vs. CPLEX (10h limit)			
		β				β			
		0.25	0.50	0.75	1.00	0.25	0.50	0.75	1.00
0.25	0.00	8.86	8.87	8.42	8.28	13.94	13.91	13.47	13.32
	0.25	7.73	7.17	6.51	6.25	12.72	12.15	11.45	11.18
	0.50	7.17	6.42	5.70	5.68	12.09	11.32	10.59	10.56
	0.75	7.76	7.02	6.54	6.22	12.74	11.98	11.49	11.17
	1.00	8.32	7.68	6.80	6.46	13.31	12.65	11.75	11.41
0.50	0.00	8.76	7.45	7.41	7.24	13.80	12.46	12.41	12.24
	0.25	7.14	6.00	5.22	5.05	12.05	10.88	10.07	9.90
	0.50	6.97	5.80	5.69	5.23	11.93	10.74	10.62	10.17
	0.75	6.82	6.11	5.65	5.21	11.72	11.00	10.54	10.09
	1.00	6.51	5.80	5.46	5.50	11.46	10.75	10.40	10.43
0.75	0.00	9.03	6.86	6.87	6.51	14.04	11.87	11.88	11.54
	0.25	6.44	4.80	4.24	4.19	11.35	9.69	9.09	9.04
	0.50	6.01	5.11	4.80	4.48	10.91	9.99	9.68	9.35
	0.75	5.50	4.53	4.32	3.81	10.44	9.45	9.24	8.73
	1.00	4.99	4.08	4.00	3.82	9.93	8.99	8.91	8.73
1.00	0.00	9.03	6.85	7.11	6.86	14.05	11.86	12.09	11.87
	0.25	6.53	4.63	4.59	4.42	11.43	9.48	9.44	9.27
	0.50	5.41	4.66	4.43	4.00	10.30	9.54	9.31	8.86
	0.75	5.12	4.32	4.01	3.87	9.99	9.19	8.84	8.71
	1.00	4.56	3.79	3.92	3.50	9.49	8.70	8.80	8.38

Table 7: Comparison of applying the Beam Search method, with different choices for parameters α, β, and ξ, to instances YFJS01–YFJS20 against the CPLEX solver with two different CPU time limits (1 hour and 10 hours).

Table 9 reports the relative error (RE) of the makespan mks found by our beam search method with respect to the lower bounds mks_{LB} available at [41] given by

$$RE = 100\% \times (mks - mks_{LB})/mks_{LB}.$$

In the table, "# inst." is the number of instances in each set. If these results are compared to those associated with the state-of-the-art GA method introduced in [27, see Table 7 on p.3210], it can be seen that the beam search method obtains competitive results; outperforming all the three GA methods whose results are reported in [27] in the set "Dauzére-Pérés and Paulli" and the set "Hurink VData" that is the set in which instances present the largest machine flexibility. However, all possible warning applies to this comparison: (a) the beam search method is a deterministic method, with no randomness and with finite termination; while the other methods are run five times and the best makespan is used in the comparison; (b) in order to obtain a comparison-at-a-glance, we run the beam search method fixing its parameters α, β, and ξ all equal to 1 (while 80 different combinations were evaluated when the numerical experiments with

α	ξ	diff vs. CPLEX (1h limit)				diff vs. CPLEX (10h limit)			
		β				β			
		0.25	0.50	0.75	1.00	0.25	0.50	0.75	1.00
0.25	0.00	-5.07	-5.57	-5.75	-5.72	-2.69	-3.20	-3.38	-3.35
	0.25	-5.18	-5.74	-5.95	-6.01	-2.80	-3.37	-3.60	-3.65
	0.50	-5.24	-5.70	-5.81	-5.91	-2.86	-3.33	-3.44	-3.55
	0.75	-5.20	-5.67	-5.85	-5.90	-2.81	-3.30	-3.49	-3.54
	1.00	-5.19	-5.66	-5.87	-6.03	-2.80	-3.28	-3.50	-3.68
0.50	0.00	-4.94	-5.56	-5.75	-5.77	-2.55	-3.19	-3.39	-3.41
	0.25	-5.53	-6.01	-6.14	-6.19	-3.16	-3.65	-3.79	-3.84
	0.50	-5.71	-6.07	-6.13	-6.19	-3.36	-3.72	-3.78	-3.85
	0.75	-5.43	-5.91	-6.06	-6.11	-3.05	-3.54	-3.70	-3.75
	1.00	-5.11	-5.86	-5.98	-5.96	-2.72	-3.49	-3.62	-3.60
0.75	0.00	-5.19	-5.62	-5.70	-5.77	-2.81	-3.25	-3.33	-3.40
	0.25	-5.88	-6.02	-6.07	-6.25	-3.52	-3.66	-3.72	-3.90
	0.50	-5.82	-6.06	-6.27	-6.32	-3.46	-3.71	-3.93	-3.98
	0.75	-5.68	-5.93	-6.16	-6.25	-3.32	-3.58	-3.81	-3.90
	1.00	-5.74	-6.07	-6.19	-6.34	-3.37	-3.71	-3.83	-3.99
1.00	0.00	-5.19	-5.65	-5.70	-5.77	-2.81	-3.27	-3.33	-3.40
	0.25	-5.90	-6.10	-6.30	-6.33	-3.53	-3.74	-3.95	-3.98
	0.50	-6.01	-6.28	-6.36	-6.36	-3.65	-3.94	-4.02	-4.02
	0.75	-5.72	-6.04	-6.19	-6.29	-3.36	-3.69	-3.84	-3.95
	1.00	-5.65	-6.01	-6.13	-6.22	-3.28	-3.65	-3.77	-3.86

Table 8: Comparison of applying the Beam Search method, with different choices for parameters α, β, and ξ, to instances DAFJS01–DAFJS30 against the CPLEX solver with two different CPU time limits (1 hour and 10 hours).

the FJSP with sequencing flexibility were done); and (c) methods being compared were run on different machines and the CPU times of the GA methods considered in [27] were not reported.

6 Conclusions

An extension of the classical flexible job shop problem, in which precedences between the operations can be given by an arbitrary directed acyclic graph instead of a linear order, was considered in this work. A list scheduling algorithm that fully exploits the characteristics of the problem was introduced. Then, substituting the choice of a single operation to be scheduled and sequenced at each step of the method by a set of operations, a beam search method was naturally developed. Numerical results assessed the effectiveness and efficiency of both approaches. The precise description and full availability of methods and results allows them to be used as benchmark for future developments. There is vast range of extensions of the classical flexible job shop problem that might be considered in order to deal with more realistic situations. Redesigning

Data Set	# inst.	RE
Brandimarte	10	26.54
Dauzére-Pérés and Paulli	18	6.21
Barnes and Chambers	21	32.09
Hurink EData	43	8.52
Hurink RData	43	4.75
Hurink VData	43	0.26

Table 9: Performance of the beam search method when applied to *all* well-known instances of the classical FJSP.

the presented methods to deal with those extensions would be the subject of future research.

Acknowledgements. This work was supported by PRONEX-CNPq/FAPERJ E-26/111.449/2010-APQ1, FAPESP (2010/10133-0, 2013/03447-6, 2013/05475-7, and 2013/07375-0), and CNPq. The authors are thankful to Marcio T. I. Oshiro that made it possible the numerical comparisons included in the present work by re-running the numerical experiments in [3] on the same computational environment considered in the present work. The authors are also thankful to Jun Zeng (Hewlett-Packard Laboratories, Hewlett-Packard Co. Palo Alto, California, USA) for fruitful discussions and suggestions and to three anonymous referees whose comments helped to improve the quality of this work.

References

[1] R. Alvarez-Valdés, A. Fuertes, J. M. Tamarit, G. Giménez, and R. Ramos, A heuristic to schedule flexible job-shop in a glass factory, *European Journal of Operational Research* 165, pp. 525–534, 2005.

[2] J. W. Barnes and J. B. Chambers, Flexible Job Shop Scheduling with tabu search, Graduate program in operations research and industrial engineering, *Technical Report* ORP9609, University of Texas, Austin, 1996.

[3] E. G. Birgin, P. Feofiloff, C. G. Fernandes, E. L. de Melo, M. T. I. Oshiro, and D. P. Ronconi, A MILP model for an extended version of the Flexible Job Shop Problem, *Optimization Letters* 8, pp. 1417–1431, 2014.

[4] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, *Annals of Operations Research* 41, pp. 157–183, 1993.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., The MIT Press, Cambridge, MA, 2009.

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

[7] P. Fattahi, M. Mehrabad, and F. Jolai, Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, *Journal of Intelligent Manufacturing* 18, pp. 331–342, 2007.

[8] D. R. Fulkerson, Note on Dilworth's decomposition theorem for partially ordered sets, *Proceedings of the American Mathematical Society* 7, pp. 701–702, 1956.

[9] P. Y. Gan and K. S. Lee, Scheduling of flexible-sequenced process plans in a mould manufacturing shop, *International Journal of Advanced Manufacturing Technology* 20, pp. 214–222, 2002.

[10] M. Garey, D. Johnson, and R. Sethi, The Complexity of Flowshop and Jobshop Scheduling, *Mathematics of Operations Research* 1, pp. 117–129, 1976.

[11] C. Gutiérrez and I. García-Magariño, Modular design of a hybrid genetic algorithm for a flexible job-shop scheduling problem, *Knowledge-Based Systems* 24, pp. 102–112, 2011.

[12] N. B. Ho, J. C. Tai, and E. M.-K. Lai, An effective architecture for learning and evolving flexible job-shop schedules, *European Journal of Operational Research* 179, pp. 316–333, 2006.

[13] J. Hurink, B. Jurish, and M. Thole, Tabu search for the job shop scheduling problem with multi-purpose machines, *OR-Spektrum* 15, pp. 205–215, 1994.

[14] O. A. Joseph and R. Sridharan, Effects of routing flexibility, sequencing flexibility and scheduling decision rules on the performance of a flexible manufacturing system, *International Journal of Advanced Manufacturing Technology* 56, pp. 291–306, 2011.

[15] I. Kacem, S. Hammadi, and P. Borne, Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic, *Mathematics and Computers in Simulation* 60, pp. 245–276, 2002.

[16] Y.-D. Kim, A backward approach in list scheduling algorithms for multi-machine tardiness problems, *Computers & Operations Research* 22, pp. 307–319, 1995.

[17] Y. K. Kim, K. Park, and J. Ko, A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling, *Computers & Operations Research* 30, pp. 1151–1171, 2003.

[18] G.-C. Lee, Y.-D. Kim, and S.-W. Choi, Bottleneck-focused scheduling for a hybrid flowshop, *International Journal of Production Research* 42, pp. 165–181, 2004.

[19] Sanghyup Lee , Ilkyeong Moon , Hyerim Bae, and Jion Kim, Flexible job-shop scheduling problems with AND/OR precedence constraints, *International Journal of Production Research* 50, pp, 1979-2001, 2012.

[20] J. Li, Q. Pan, and Y. Liang, An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems, *Computers & Industrial Engineering* 59, pp. 647–662, 2010.

[21] G. Y-.J. Lin and J. J. Solberg, Effectiveness of flexible routing control, *International Journal of Flexible Manufacturing Systems* 3, pp. 189–211, 1991.

[22] A. Manne, On the job-shop scheduling problem, *Operations Research* 8. pp. 219–223, 1960.

[23] G. Mainieri and D. P. Ronconi, New heuristics for total tardiness minimization in a flexible flowshop, *Optimization Letters* 7, pp. 665–684, 2013.

[24] M. Mastrolilli and L. M. Gambardella, Effective neighbourhood functions for the flexible job shop problem, *Journal of Scheduling* 3, pp. 3–20, 2000.

[25] P. S. Ow and T. E Morton, Filtered beam search in scheduling, *International Journal of Production Research* 26, pp. 35–62, 1988.

[26] C. Özgüven, L. Özbakır, and Y. Yavuz, Mathematical models for job-shop scheduling problems with routing and process plan flexibility, *Applied Mathematical Modeling* 34, pp. 1539–1548, 2010.

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

[28] M. Pinedo, *Scheduling: theory, algorithms, and systems*, third edition, Springer, New York, NY, 2008.

[29] R. Rachamadugu, U. Nandkeolyar, and T. Schriber, Scheduling with sequencing flexibility, *Decision Sciences* 24, pp. 315–342, 1993.

[30] I. Sabuncuoglu and M. Bayiz, Job shop scheduling with beam search, *European Journal of Operational Research* 118, pp. 390–412, 1999.

[31] I. Sabuncuoglu and S. Karabuk, A beam search-based algorithm and evaluation of scheduling approaches for flexible manufacturing systems, *IIE Transaction* 30, pp. 179–191, 1998.

[32] X. Shao, W. Liu, Q. Liu, and C. Zhang, Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem, *International Journal of Advanced manufacturing Technology* 67, pp. 2885–2901, 2013.

[33] W. Shi-Jin, Z. Bing-Hai, and X. Li-Feng, A filtered-beam-search-based heuristic algorithm for flexible job-shop scheduling problem, *International Journal of Production Research* 46, pp. 3027–3058, 2008.

[34] K. Sörensen, Metaheuristics – the metaphor exposed, *International Transactions in Operational Research* 22, pp. 3–18, 2015.

[35] J. C. Tay and N. B. Ho, Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems, *Computers & Industrial Engineering* 54, pp. 453–473, 2008.

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

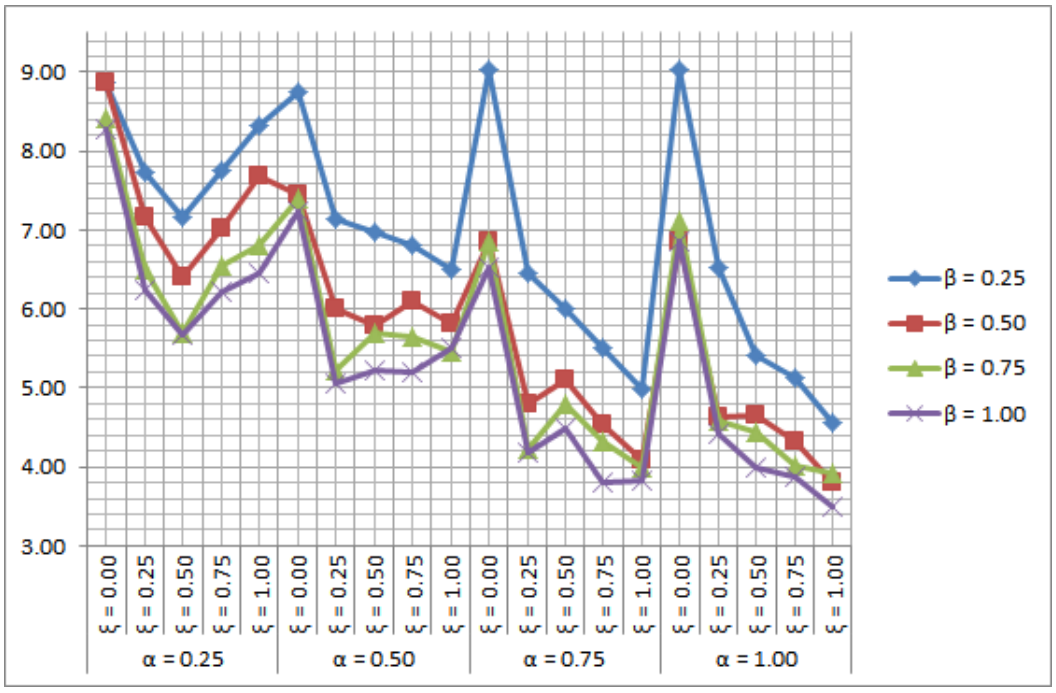
[37] Y. Yuan and H. Xu, An integrated search heuristic for large-scale flexible job shop scheduling problems, *Computers & Operations Research* 40, pp. 2864–2877, 2013.

[38] J. Zeng, S. Jackson, I.-J. Lin, M. Gustafson, E. Hoarau, and R. Mitchell, On-demand digital print operations: A simulation based case study, *Technical Report*, Hewlett-Packard, 2010.

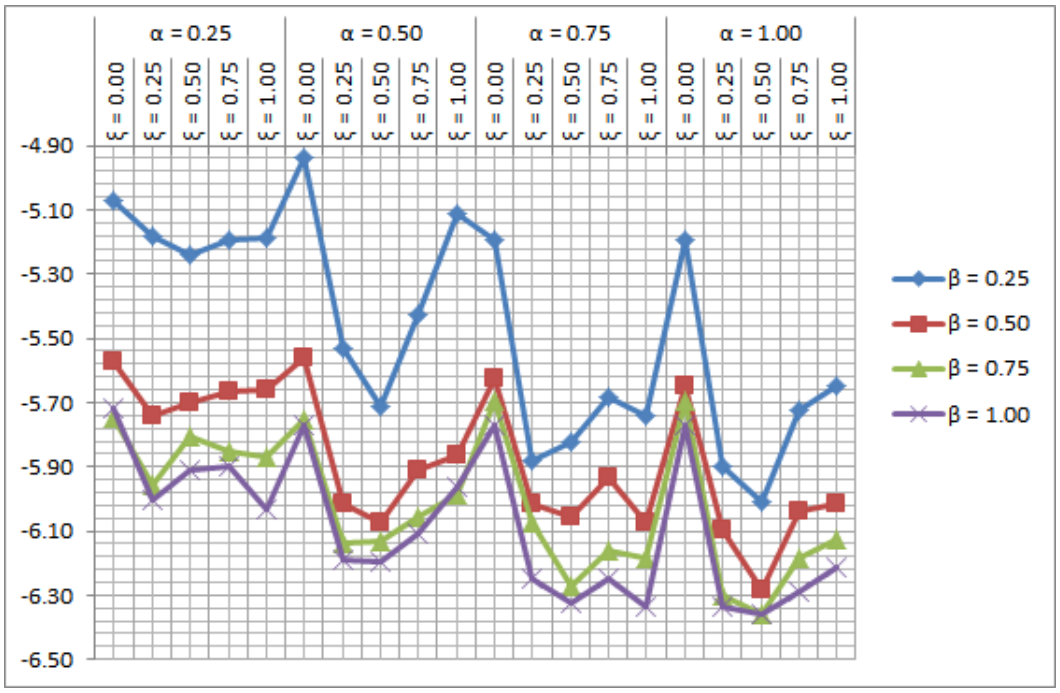
[39] G. Zhang, X. Shao, P. Li, and L. Gao, An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem, *Computers & Industrial Engineering* 56, pp. 1309–1318, 2009.

[40] http://www.pneac.org, web page of the Printers National Environmental Assistance Center (PNEAC), accessed on February 9, 2015.

[41] http://people.idsia.ch/∼monaldo/fjspresults/fjsp_result.ps, accessed on February 9, 2015.

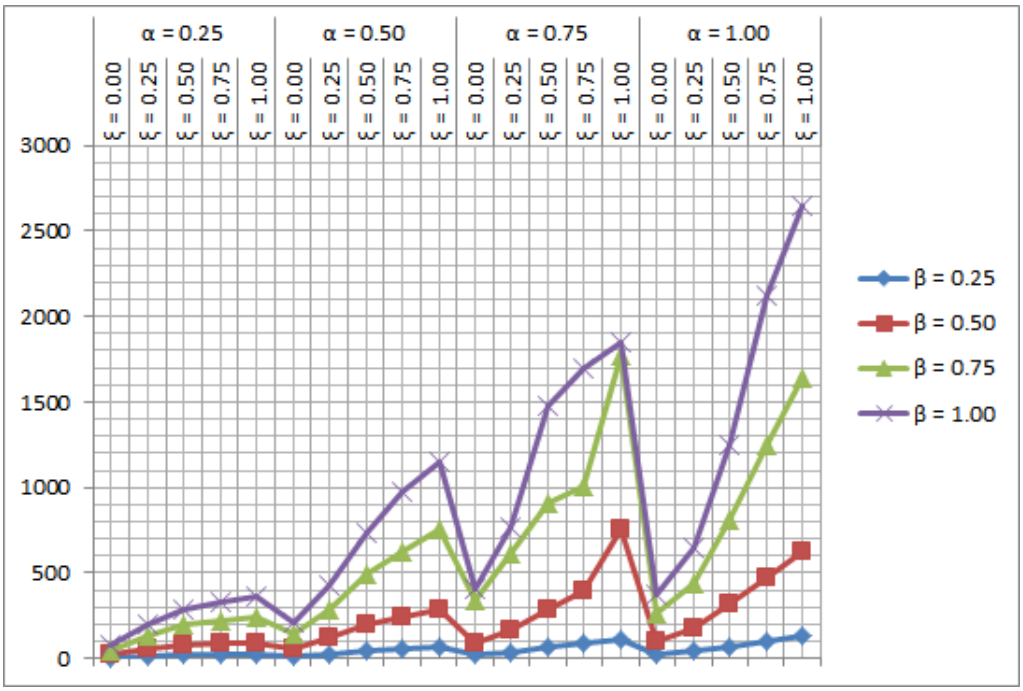


(a)

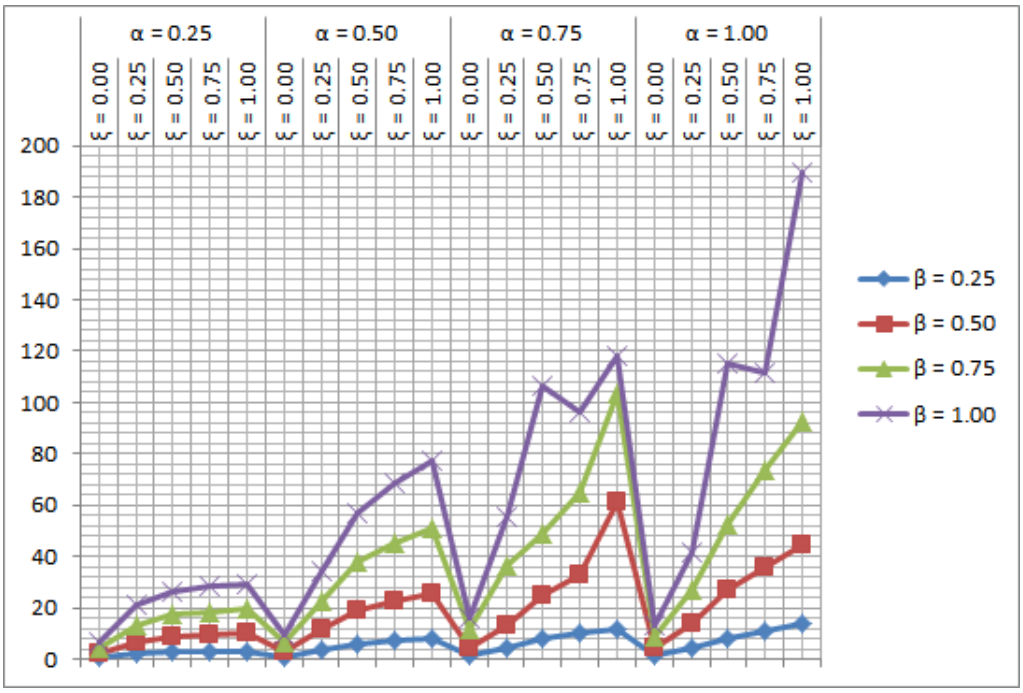


(b)

Figure 5: Graphical representation of the performance of the beam search method for different choices of parameters α, β, and ξ (when compared against the results obtained by running the exact solver with a CPU time limit of 1 hour). (a) Instances YFJS01–YFJS20. (b) Instances DAFJS01–DAFJS30.



(a)



(b)

Figure 6: Graphical representation of the elapsed CPU time required by the beam search method for different choices of parameters α, β, and ξ. (a) Instances YFJS01–YFJS20. (b) Instances DAFJS01–DAFJS30.