A filtered beam search method for the m-machine permutation flowshop scheduling problem minimizing the earliness and tardiness penalties and the waiting time of the jobs*

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

April 2, 2019§

Abstract

This paper addresses the minimization of the absolute deviation of job completion times from a common due date in a flowshop scheduling problem. Besides this main objective, the minimization of the waiting time of the jobs in the production environment, that can be seen as an intermediate inventory cost, is also considered. Initially, a mixed integer programming model for this problem is proposed and, due to its complexity, heuristic approaches are developed. A list-scheduling algorithm for the approached problem is introduced. Moreover, a filtered beam search method that explores specific characteristics of the considered environment is proposed. Numerical experiments show that the presented methods can be successfully applied to this problem.

Key words: Scheduling, flowshop, earliness and tardiness, common due date, waiting time, heuristics, beam search.

1 Introduction

This paper addresses the flowshop scheduling problem. In this environment, there are n jobs and m machines. Every job must be processed in the m machines and the kth operation of every job must be conducted on machine k. We consider the case in which all machines must process the jobs in the same order, known as permutation schedule. The importance of permutation schedules should not be underestimated, since only permutation schedules are feasible in most real-life situations [16]. A static and deterministic environment is assumed where all jobs are available for processing since the beginning. Each job i has a known processing time p_{ik} on each machine k and there is a common due date d. Preemption is not allowed. When a job i is completed before its due date,

*This work has been partially supported by FAPESP (grants 2013/07375-0, 2016/01860-1, and 2018/24293-0) and CNPq (grants 309517/2014-1 and 306083/2016-7).

†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. Luciano Gualberto, 1380, Cidade Universitária, 05508-010, São Paulo SP, Brazil. e-mail: dronconi@usp.br

§Revision made on August 26, 2019.

its earliness is given by $E_i = d - C_{im}$, where C_{im} is its completion time on the last machine. Conversely, if job i is finished after the desired date, its tardiness is given by $T_i = C_{im} - d$. The objective is to find a schedule that minimizes the sum of tardiness and earliness of the jobs. As a second objective, among the solutions that minimize the sum of earliness and tardiness (E/T) of the jobs, we seek a solution that minimizes the waiting time of the jobs between the machines. This secondary objective can be seen as the intermediate inventory cost, i.e. the cost of carrying jobs between two adjacent machines (see [20, 24]). This goals' hierarchy fits the tackled problem in the bilevel optimization framework [9], in opposition to the multiobjective approach [18].

Practical examples where the minimization of the waiting time of the jobs between the machines is a relevant aspect of the problem can be found in the biotechnology industry; where delays between stages increase the chances of contamination or deterioration. As pointed out in [7], there are products that require refrigeration or storage in other types of controlled facilities that might be expensive to build or maintain.

Meeting due dates is a common objective for many manufacturing processes. Tardy jobs may generate contractual penalties and loss of credibility, causing damages to the company's image and loss of clients [32]. Early jobs were discouraged since the advent of Just-in-Time approaches due to the costs generated by those jobs, such as tied-up capital and inventory costs [6]. A particular case occurs when all jobs have the same due date. Such scenarios can be found in practice, for example, at a chemical company where different chemicals are manufactured using the same process and have to be combined as close as possible to a common due date to avoid deterioration [31].

There are two main scenarios in the presence of a common due date. In the unrestrictive case, the common due date is a decision variable or, if its value is given, it has no influence over the optimal sequence [10]. This happens, for example, when the due date is greater than or equal to the sum of all processing times on all machines (see [19]). However, if the due date is given and it influences the optimal sequence of jobs then it is considered restrictive. The single-machine problem with restrictive due date is NP-hard [13], and so is the m-machine case.

A wide variety of studies involving scheduling problems with E/T penalties and common due date can be found in the literature, such as those described in the reviews presented by Baker and Scudder [3] and Gordon *et al.* [12]. Nonetheless, both papers only addressed scenarios with single or parallel machines. The survey of Lauff and Werner [20] tackles this non-regular measure for multi-machine problems. These authors mention that the development of general strategies for solving m-machine flowshop problems is of interest. They also mention that the elaboration of heuristics for solving multi-stage scheduling problems is a worthy contribution for the applicability of scheduling theory to real world problems [19].

Due to the complexity of the considered problem, heuristic methods for the flowshop environment have been developed by several authors [31, 36, 20, 21, 38, 30, 15, 14]. However, these papers deal with the two-machine flowshop scheduling problem with different characteristics. Considering the m-machine flowshop problems with a common due date, procedures that are applicable to a wide range of possible due date settings were introduced in [8]. For the restrictive due date scenario, a heuristic based on the bottleneck machine followed by myopic perturbations and a Tabu Search procedure, that uses the pairwise interchange of jobs, were proposed. In the numerical experiments, for each instance, the authors considered two alternative restrictive common due dates to evaluate the performance of their approach. For small problems ($n \in \{5, 10\}$ and $m = 5$), the average deviation from the optimal solution in each set was less than 1.5%. For larger problems

($5 \leq n \leq 100$ and $5 \leq m \leq 20$), a comparison with randomly generated sequences was conducted and the improvements were in the range of 14% to 38%.

According to Yenisey and Yagmahan [39], that presented a review about multi-objective permutation flowshop scheduling problems, there are three groups of objectives for scheduling problems: (i) based on the completion time, (ii) based on due date, and (iii) based on inventory and utilization costs. Although the first and second groups of objectives are more commonly used in the flowshop scheduling problems, studies based on the third objectives' group can be found in the literature. Since our secondary objective is the minimization of the waiting time of the jobs between the machines, which can be seen as the intermediate storage cost for each job, the study of this criterion associated with the E/T penalties will be briefly reviewed next. In [19], the complexity and other properties of the two-machine flowshop problem were investigated. Among other results, the authors showed that the problem is NP-hard even with a nonrestrictive common due date. In [7], a non-permutation flowshop scheduling problem where the jobs have different ready times and due dates was considered. An integer model and lower bounds were presented and the problem was heuristically solved by applying Dantzig-Wolfe reformulation and Lagrangian relaxation methods. In [24], the minimization of the work-in-process inventory considering the machine set up times was approached. However, only penalties for the tardiness of the batches were taken into account in this work. In this research, two new recently proposed metaheuristic, namely Teaching-Learning Based Optimization Algorithm and Jaya, were applied and compared with two known metaheuristics: Particle Swarm Optimization and Simulated Annealing.

The minimization of the E/T penalties in a *no-wait* flowshop environment with a common due date is a problem closely related to the problem considered in the present work, in which there is no no-wait constraint but the sum of the waiting times of the jobs between the machines is minimized as a secondary objective. There are only a few papers in the literature that deal with the minimization of earliness and tardiness penalties in a *no-wait* flowshop environment. The two-machines case with learning effect and convex resource-dependent processing times is considered in [35], where two objectives are seek: (i) the weighted sum of earliness, tardiness, and flow allowance costs and (ii) the resource consumption cost. The problem, with two machines, is shown to be solvable in polynomial time. The case in which setup times are sequence-dependent is considered in [2]. Heuristic methods based on tabu search and particle swarm optimization, that use a timing algorithm to find an optimal schedule of a given sequence, are proposed.

Considering the reduced amount of works that tackle the minimization of the E/T penalties associated with the intermediate inventory cost in the m-machine flowshop problem, the objective of this paper is to contribute to the development of heuristic techniques able to produce reasonable results for this problem in acceptable time. Initially, a mixed integer linear programming model that represents the problem is presented. Next, a list scheduling algorithm is proposed, motivated by its simplicity and applicability to job scheduling in production environments (see, for example, [22, 23, 4]). The natural extension of the list scheduling algorithm to a filtered beam search method is also investigated. 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 [26]. 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. 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.

After the pioneer work of Ow and Morton [25], other authors addressed multimachine problems with this approach. The jobshop scheduling problem minimizing the makespan was tackled in [29], where several numerical experiments were conducted in order 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. Considering multiple objectives, a filtered beam search approach for the flexible job shop minimizing the weighted sum of the makespan, the total workload of machines, and the workload of the most loaded machine was introduced in [33]. An extended version of the flexible job shop problem that allows the precedences between the operations to be given by an arbitrary directed acyclic graph instead of a linear order was considered in [4]. The goal in this research was the minimization of the makespan. A list scheduling and a beam search method were introduced. More recently, considering the permutation flowshop scheduling problem, a beam-search-based constructive heuristic to minimize the total flowtime was presented in [11]. The proposed algorithm was inspired in the logic of the beam search, although it remained a fast constructive heuristic. The results obtained by the proposed algorithm outperformed those obtained by other constructive heuristics in the literature for the problem.

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 Model

Let n and m be the number of jobs and machines, respectively; let p_{ik} be the processing time of job i on machine k; and let d be the due date, common to all jobs. The mixed integer linear programming (MILP) model below [27] (see also [17]) formulates the problem of minimizing the sum of earliness and tardiness of the jobs in a flowshop environment. In the model, the continuous variables C_{ik} $(i = 1, \ldots, n,\ k = 1, \ldots, m)$ represent the completion time of job i on machine k; the continuous variables E_i and T_i $(i = 1, \ldots, n)$ represent the earliness and tardiness of job i, respectively; and the binary variables z_{ij} $(i = 1, \ldots, n,\ j = i+1, \ldots, n)$ say whether job i precedes job j in the sequence $(z_{ij} = 1)$ or not $(z_{ij} = 0)$.

$$\text{Minimize} \quad \sum_{i=1}^{n}(E_i + T_i) \tag{1}$$

$$\text{subject to} \quad T_i \geq C_{im} - d, \quad i = 1, \ldots, n, \tag{2}$$

$$E_i \geq d - C_{im}, \quad i = 1, \ldots, n, \tag{3}$$

$$C_{i1} \geq p_{i1}, \quad i = 1, \ldots, n, \tag{4}$$

$$C_{i,k+1} \geq p_{i,k+1} + C_{ik}, \quad i = 1, \ldots, n, k = 1, \ldots, m-1, \tag{5}$$

$$C_{ik} \geq p_{ik} + C_{jk} - Mz_{ij}, \quad i = 1, \ldots, n,\ j = i+1, \ldots, n,\ k = 1, \ldots, m, \tag{6}$$

$$C_{jk} \geq p_{jk} + C_{ik} - M(1 - z_{ij}), \quad i = 1, \ldots, n,\ j = i+1, \ldots, n,\ k = 1, \ldots, m, \tag{7}$$

$$z_{ij} \in \{0, 1\}, \quad i = 1, \ldots, n,\ j = i+1, \ldots, n, \tag{8}$$

$$T_i, E_i \geq 0, \quad i = 1, \ldots, n. \tag{9}$$

Constraints (2), (3), and (9) correspond to the linearization of $T_i \geq \max\{C_{im} - d, 0\}$ and $E_i \geq \max\{d - C_{im}, 0\}$. Thus, they impose lower bounds on the earliness and tardiness of each job, respectively. Constraints (4) and (5) deal with the completion times of individual jobs. Constraint (4) says that the completion time of a job on the first machine must be greater than or equal to the processing time of the job on that machine. Constraint (5) says that between the completion time of a job on two consecutive machines there must be enough time to process the job on the second of the two machines. Constraints (6) and (7) ensure that only one operation is processed on each machine at any given time. In these two constraints, M is a sufficiently large positive constant.

It is easy to see in the model that, due to constraint (5), the spare or waiting time of job i between two consecutive machines k and $k+1$ is given by $C_{i,k+1} - p_{i,k+1} - C_{ik}$, for $i = 1, \ldots, n$ and $k = 1, \ldots, m-1$. In the present work, we are interested in minimizing the sum of earliness and tardiness, as already represented by the objective function (1); and, among the solutions that minimize it, choosing one that minimizes the sum of the waiting times of the jobs between consecutive machines. (Note that the time between the beginning of the time horizon and the instant in which each job starts to be processed in the first machine *is not* considered in the quantity we desire to minimize.)

The *weighted* sum of the waiting times of the jobs between consecutive machines is given by

$$\sum_{i=1}^{n}\sum_{k=1}^{m-1} \omega_{ik}\left(C_{i,k+1} - p_{i,k+1} - C_{ik}\right), \tag{10}$$

where $\omega_{ik} \geq 0$ ($i = 1, \ldots, n,\ k = 1, \ldots, m-1$), not all null, would represent given weights to penalize the waiting time of job i between consecutive machines k and $k+1$. In the particular case in which $\omega_{ik} \equiv 1$ for all i and k, since $\sum_{k=1}^{m-1}\left(C_{i,k+1} - p_{i,k+1} - C_{ik}\right) = C_{im} - C_{i1} - \sum_{k=2}^{m} p_{ik}$, we have that (10) coincides with

$$\sum_{i=1}^{n}\left(C_{im} - C_{i1}\right) - \sum_{i=1}^{n}\sum_{k=2}^{m} p_{ik}.$$

Thus, minimizing the *unweighted* sum of the waiting times of the jobs between consecutive machines

is equivalent to minimizing

$$\sum_{i=1}^{n} (C_{im} - C_{i1}) \,. \tag{11}$$

If we assume that the problem data, i.e. the processing times p_{ik} and the common due date d, are integer values then it is true that every optimal solution of problem (1–9) is such that T_i and E_i are integer values[1]. In consequence, if p_{ik} and d are all integers, the optimal value of problem (1–9) assumes an integer value. On the other hand, in an optimal solution, it is also true[2] that $\sum_{i=1}^{n} C_{im} < n(d + P)$, where

$$P = \sum_{i=1}^{n} \sum_{k=1}^{m} p_{ik}.$$

Therefore,

$$\sum_{i=1}^{n} (C_{im} - C_{i1}) \leq \sum_{i=1}^{n} C_{im} < n(d + P),$$

or, equivalently

$$\frac{\sum_{i=1}^{n} (C_{im} - C_{i1})}{n(d + P)} < 1. \tag{12}$$

This means that, from the point of view of minimizing the sum of earliness and tardiness, it is innocuous to add to the objective function the fraction on the left hand side of (12). Moreover, the minimization of this combined objective has the desired effect of minimizing the sum of earliness and tardiness and, among solutions with minimum value, choosing one that minimizes the unweighted sum of the waiting times between consecutive machines. Thus, summing up, the problem considered in the present work consists in minimizing

$$\sum_{i=1}^{n} (E_i + T_i) + \frac{\sum_{i=1}^{n} (C_{im} - C_{i1})}{n(d + P)},$$

or, equivalently,

$$n(d + P) \sum_{i=1}^{n} (E_i + T_i) + \sum_{i=1}^{n} (C_{im} - C_{i1}) \tag{13}$$

subject to (2–9).

It is worth noticing that the described problem is *not* equivalent to the problem of finding *a single sequence* that minimizes the earliness and tardiness and, in a second stage, while preserving the same sequence and its sum of earliness and tardiness, minimizing the waiting times. A solution

[1] Given a feasible solution in which there exist i such that E_i or T_i is not integer, it is possible to reduce the value of E_i or T_i and, in consequence, the objective function value, by pushing job i toward the due date d (pushing other jobs may be needed as well).

[2] Consider the feasible solution in which jobs are all processed in the order of their indices; job 1 starts to be processed at instant d and has completion time $C_{1m} = d + \sum_{k=1}^{m} p_{1k}$; and, for $i = 2, \ldots, n$, job i starts to be processed at instant $C_{i-1,m}$ and has completion time $C_{im} = C_{i-1,m} + \sum_{k=1}^{m} p_{ik}$. In this feasible solution, $\sum_{i=1}^{n} C_{im} = nd + \sum_{i=1}^{n} \left((n - i + 1) \sum_{k=1}^{m} p_{ik} \right) < nd + n \sum_{i=1}^{n} \sum_{k=1}^{m} p_{ik} = n(d + P)$. Note that tighter strict bounds exist, like, for example, the one obtained when jobs are ordered not by increasing order of their indices but by the sum of their processing times $P_i = \sum_{k=1}^{m} p_{ik}$.

like this one could be obtained as follows. First, obtain a scheduling that minimizes the sum of earliness and tardiness of the jobs by solving the MILP model (1–9). Assume that z^*_{ij} ($i = 1, \ldots, n$, $j = i+1, \ldots, n$) represent the optimal sequence and that the optimal sum of earliness and tardiness, given by (1), is equal to ξ^*. In the second stage, consider the constraints (2–9) in which $z_{ij} = z^*_{ij}$ are fixed as constants plus the constraint $\sum_{i=1}^{n} E_i + T_i = \xi^*$ and minimize the sum of the waiting times given by (11). This problem is a linear programming (LP) problem. Figure 1a shows a solution obtained by applying this two-stages method to the instance given by $n = m = 3$, $p_{11} = 1$, $p_{12} = 2$, $p_{13} = 3$, $p_{21} = 1$, $p_{22} = 2$, $p_{23} = 1$, $p_{31} = 2$, $p_{32} = 1$, $p_{33} = 1$, and $d = 7$. This solution has a sum of earliness and tardiness equal to two and a sum of waiting times of the jobs between the machines equal to one. On the other, when the proposed model (that consists in minimizing (13) subject to (2–9)) is applied to that instance, a solution with the same sum of earliness and tardiness (equal to two) but no waiting times between the machines is found (see Figure 1b). If we had not fixed the sequence $z_{ij} = z^*_{ij}$ in the second stage of the two-stages method then the second stage would consist in solving an MILP instead of an LP problem. Thus, the two-stages approach would consist in solving two MILP problems; the first one to find the optimal earliness and tardiness and the second one, restricted to the optimality of the earliness and tardiness, to minimize the waiting times. This would be equivalent to solving the single proposed model.

3 List scheduling algorithm

In this section a list scheduling algorithm is introduced. The algorithm is composed by three stages. In the first stage, a sequence possessing a compact scheduling, that hopefully minimizes the waiting times of the jobs between the machines, is computed. The second stage determines the completion times of the jobs on the last machine aiming to minimize the sum of earliness and tardiness of the jobs. In the third stage, the compactness of the scheduling is reestablished by computing the completion times of the jobs in all the other machines.

The first stage of the list scheduling algorithm determines a schedule starting with an empty sequence and adding one job at a time at the end of the sequence. When a job is added, it is scheduled as soon as possible. The first job to be added to the sequence is the job with the smallest sum of processing times. In case of ties, the job with the smallest index is chosen. Let $\sigma_1, \sigma_2, \ldots, \sigma_{i-1}$ be the jobs in a partially build schedule with $i-1$ jobs and let $C_{\sigma_{i-1},k}$ ($k = 1, \ldots, m$) be the completion time of job σ_{i-1} in each machine k. For each job $j \in \{1, 2, \ldots, n\} \setminus \{\sigma_1, \sigma_2, \ldots, \sigma_{i-1}\}$, we consider the merit function

$$\Gamma_j = (1-\gamma) \sum_{k=1}^{m} p_{jk} + \gamma \sum_{k=1}^{m-1} \left| C_{\sigma_{i-1},k+1} - C_{\sigma_{i-1},k} - p_{jk} \right|, \tag{14}$$

where $\gamma \in [0, 1]$ is a parameter. The first term of the merit function is the sum of the processing times of job j in all the machines. The second term, introduced in [28], computes the fitting of job j in the "windows" left by job σ_{i-1} between the machines; and it aims to measure the waiting time of job j between the machines or the idle time of the machines produced by adding job j to the partial sequence. The job j with smallest Γ_j is the one added to the sequence, i.e. $\sigma_i = j$. In case of ties, the job with the smallest index is chosen. As already mentioned, the new job σ_i is added to the end of the sequence and scheduled as soon as possible. This means that we will

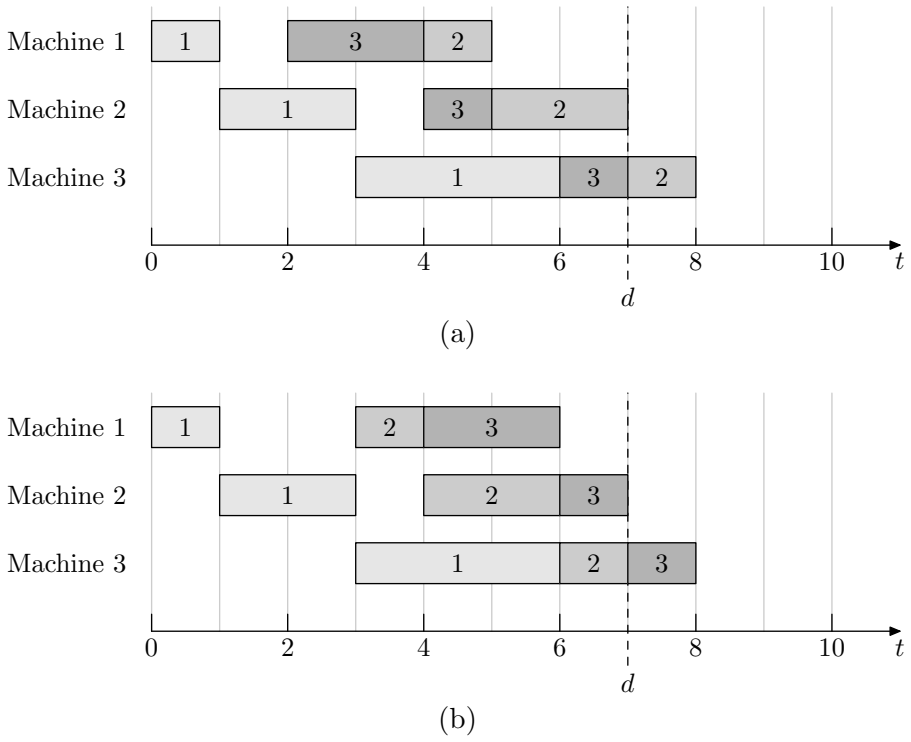

Figure 1: Both Gantt's diagrams represent optimal solutions to the problem of minimizing the sum of earliness and tardiness in a flowshop environment. Solution (a) was found by applying a two-stages approach that is sub-optimal as a solution method for the proposed problem. Solution (b) was found by solving the proposed MILP model that, among solutions that minimize earliness and tardiness, seeks one that minimizes the waiting time of the jobs between the machines.

have $C_{\sigma_i,1} = C_{\sigma_{i-1},1} + p_{\sigma_i,1}$ and $C_{\sigma_i,k} = \max\{C_{\sigma_i,k-1}, C_{\sigma_{i-1},k}\} + p_{\sigma_i,k}$ for $k = 2, \ldots, m$. Since the merit function (14) is computed, for each "non-fixed" job, every time a new job is added to the partially build schedule, the described first stage of the list scheduling algorithm can be considered as a dynamic dispatching rule.

The first stage of the list scheduling algorithm seeks producing a sequence with a "compact" scheduling; but, when doing that, the common due date d is ignored. Thus, in the second stage, the sequence produced in the first stage is preserved but the completion time of the jobs in the last machine is recalculated. In this second stage, the completion times of the jobs in machine $m-1$ are seen as if they were release times, i.e. $r_{\sigma_i} \equiv C_{\sigma_i,m-1}$ $(i = 1, \ldots, n)$; and a new completion time $C_{\sigma_i,m}$ $(i = 1, \ldots, n)$ for each job in the last machine is computed with the timing algorithm introduced in [30]. This timing algorithm applies to the single machine environment in which there are n jobs with release dates and a common due date; and, for a given sequence, it computes the completion times of the jobs that minimizes the sum of earliness and tardiness. Finally, in the third stage of the list scheduling algorithm, while preserving the completion times of the jobs in the last machine, the completion times $C_{\sigma_i,k}$ $(i = 1, \ldots, n,\ k = 1, \ldots, m-1)$ of the jobs in all the other machines are recomputed in order to recover a compact scheduling.

The proposed list scheduling algorithm is fully described in Algorithm 1. The first stage corresponds to lines 2–14. The second stage that simply uses the timing algorithm corresponds to line 15; while the third stage corresponds to lines 16–19. In the algorithm, the symbol $\prec$ stands for lexicographical order; i.e. for any two q-uples v and w, $(v_1, v_2, \ldots, v_q) \prec (w_1, w_2, \ldots, w_q)$ means that there exists $1 \leq q_0 \leq q$ such that $v_\ell = w_\ell$ for all $\ell < q_0$ and $v_{q_0} < w_{q_0}$. This notation is used, for example, in line 3 to concisely express that we consider the jobs by increasing order of the sum of their processing times and that the smallest index rule is used as a tie-breaking rule. Note that mentioning the tie-breaking rule allows reproducibility. Notation $\{a_1, a_2, \ldots\} = \{b_1, b_2, \ldots\}$ is used in the algorithm to say that there is a set with elements $a_1, a_2, \ldots$, a set with elements $b_1, b_2, \ldots$, and that these two sets are equal. In particular, we always use this notation with sets of indices to say that the indices to the left are a permutation of the indices to the right. For example, line 3 of the algorithm says that the indices $i_1, i_2, \ldots, i_n$ correspond to the jobs $1, 2, \ldots, n$ ordered in increasing order of the sum of their processing times, with the smallest index rule as a tie-breaking rule. Notation swap(a, b) means that the values of a and b are interchanged.

The first stage of the algorithm (lines 2–14) has time complexity $O(n^2m + n^2 \log n)$. (Line 2 is $O(nm)$, line 3 is $O(n \log n)$, lines 5–7 are $O(m)$, and lines 8–14 are $O(n^2m + n^2 \log n)$. Since the timing algorithm introduced in [30] has time complexity $O(n)$ then the second stage of the the algorithm is $O(n)$. The third stage of the algorithm is $O(nm)$. Thus, Algorithm 1 has time complexity $O(n^2m + n^2 \log n)$.

We end this section by illustrating the application of the list scheduling algorithm to the example given in Section 2, i.e. a small instance with $n = m = 3$, $p_{11} = 1$, $p_{12} = 2$, $p_{13} = 3$, $p_{21} = 1$, $p_{22} = 2$, $p_{23} = 1$, $p_{31} = 2$, $p_{32} = 1$, $p_{33} = 1$, and $d = 7$. In line 2, the sums of the processing times of the jobs are computed. Thus, we have $P_1 = 6$, $P_2 = P_3 = 4$. In line 3, the indices i_1, i_2, and i_3 such that $(P_{i_1}, i_1) \prec (P_{i_2}, i_2) \prec (P_{i_3}, i_3)$ are computed. Since $(4, 2) \prec (4, 3) \prec (6, 1)$, we have $i_1 = 2$, $i_2 = 3$, and $i_3 = 1$. Note that the tie-breaking rule is used to break the tie between P_2 and P_3 (both equal to 4). In line 4, we set $\sigma = (2, 1, 3)$. It corresponds to the sequence $(1, 2, 3)$ in which the job $i_1 = 2$, that is the job with the smallest sum of processing times, was swapped with job 1. In the sequence σ, job $\sigma_1 = 2$ is fixed (at the first position of the sequence); while

Algorithm 1: List scheduling algorithm.

Data: n, m, d, p_{ik} $(i = 1, \ldots, n; k = 1, \ldots, m), \gamma$
Result: σ_i $(i = 1, \ldots, n), C_{ik}$ $(i = 1, \ldots, n; k = 1, \ldots, m)$

1 **begin**
2 **for** $i \leftarrow 1$ **to** n **do** $P_i \leftarrow \sum_{k=1}^{m} p_{ik}$
3 let $\{i_1, \ldots, i_n\} = \{1, \ldots, n\}$ be such that $(P_{i_1}, i_1) \prec \cdots \prec (P_{i_n}, i_n)$
4 let $\sigma = (1, 2, \ldots, n)$ and swap(σ_1, σ_{i_1})
5 $C_{\sigma_1,1} \leftarrow p_{\sigma_1,1}$
6 **for** $k \leftarrow 2$ **to** m **do**
7 $C_{\sigma_1,k} \leftarrow C_{\sigma_1,k-1} + p_{\sigma_1,k}$
8 **for** $i \leftarrow 2$ **to** n **do**
9 **for** $j \leftarrow i$ **to** n **do** $\Gamma_{\sigma_j} \leftarrow (1-\gamma) P_{\sigma_j} + \gamma \sum_{k=1}^{m-1} \left| C_{\sigma_{i-1},k+1} - C_{\sigma_{i-1},k} - p_{\sigma_j,k} \right|$
10 let $\{j_1, \ldots, j_{n-i+1}\} = \{i, \ldots, n\}$ be such that $(\Gamma_{\sigma_{j_1}}, \sigma_{j_1}) \prec \cdots \prec (\Gamma_{\sigma_{j_{n-i+1}}}, \sigma_{j_{n-i+1}})$
11 swap(σ_i, σ_{j_1})
12 $C_{\sigma_i,1} \leftarrow C_{\sigma_{i-1},1} + p_{\sigma_i,1}$
13 **for** $k \leftarrow 2$ **to** m **do**
14 $C_{\sigma_i,k} \leftarrow \max\{C_{\sigma_i,k-1}, C_{\sigma_{i-1},k}\} + p_{\sigma_i,k}$
15 Considering the common due date d, the processing times $p_{\sigma_i,m}$ $(i = 1, \ldots, n)$, and the release dates $r_{\sigma_i} \equiv C_{\sigma_i,m-1}$ $(i = 1, \ldots, n)$, recompute $C_{\sigma_i,m}$ $(i = 1, \ldots, n)$ by calling the Timing algorithm introduced in [30, p.65].
16 **for** $k \leftarrow m-1$ **to** 1 *with step* -1 **do**
17 $C_{\sigma_n,k} \leftarrow C_{\sigma_n,k+1} - p_{\sigma_n,k+1}$
18 **for** $i \leftarrow n-1$ **to** 1 *with step* -1 **do**
19 $C_{\sigma_i,k} \leftarrow \min\{C_{\sigma_{i+1},k} - p_{\sigma_{i+1},k}, C_{\sigma_i,k+1} - p_{\sigma_i,k+1}\}$

jobs $\sigma_2 = 1$ and $\sigma_3 = 3$ are unfixed jobs. In lines 5–7, the first job of the sequence is scheduled as soon as possible, that translates into $C_{21} = 1$, $C_{22} = 3$, and $C_{23} = 4$. Figure 3a shows this partial scheduling highlighting the "windows" left in machines 1 and 2 by the last fixed/scheduled job (the only one in this case).

We now consider the main loop of the algorithm (lines 8–14) for $i = 2$, i.e. for the task of fixing (and scheduling as soon as possible) the second job in the sequence. With that purpose, in line 9, the merit function (14) is computed for the two unfixed jobs: Γ_{σ_2} corresponding to $\sigma_2 = 1$ and Γ_{σ_3} corresponding to $\sigma_3 = 3$. The first term in the merit function, multiplied by $(1 - \gamma)$, is simple the sum of the processing times of the job. The second term, multiplied by γ, measures the fitting of the unfixed job to the windows left by the last fixed job. Consider first job $\sigma_2 = 1$, that has $p_{11} = 1$ and $p_{12} = 2$. Since the windows in machines 1 and 2 have length 2 and 1, respectively, job 1 is one time unit smaller on the first machine and one time unit larger in the second machine. This means that its "fitting score" is 2. Consider now job $\sigma_3 = 3$, that has $p_{31} = 2$ and $p_{32} = 1$. Job 3 has a perfect fitting in both windows and, therefore, its "fitting score" is 0. Thus, we have, $\Gamma_{\sigma_2} = (1 - \gamma)\, 6 + \gamma\, 2$ and $\Gamma_{\sigma_3} = (1 - \gamma)\, 4 + \gamma\, 0$; and for any value of the parameter $\gamma \in [0, 1]$, Γ_{σ_3} is smaller than Γ_{σ_2}. Thus, in line 10 of the algorithm, we have $j_1 = 3$ and $j_2 = 2$, meaning that job $\sigma_3 = 3$ is being selected to be fixed as the second job of the sequence. Thus, in line 11, we swap σ_2 and σ_3 obtaining $\sigma = (2, 3, 1)$. Then, in lines 12–14, job 3 is scheduled as soon as possible. This

ends the main loop for $i = 2$. Since there is only one job left unfixed, when $i = 3$, job 1 is fixed in the third position of the sequence and scheduled as soon as possible. This ends the first stage of the list scheduling algorithm with the schedule depicted in Figure 3b. In this stage, the computed schedule is "compact" in the sense that using the merit function (14) at each step aims to minimize the waiting times of the jobs between the machines as well as the idle times of the machines.

The second stage of the list scheduling algorithm consists in using the Timing algorithm introduced in [30, p.65] to recompute the completion times of the jobs in the last (third) machine. With this purpose, the completion times of the jobs in the second machine are used as if they were release times. So we consider $r_{\sigma_1} = C_{\sigma_1,2} = 3$, $r_{\sigma_2} = C_{\sigma_2,2} = 4$, and $r_{\sigma_3} = C_{\sigma_3,2} = 6$. For the sequence $\sigma = (2, 3, 1)$ with these release times and the due date $d = 7$, the scheduling that minimizes the E/T measure is $C_{\sigma_1,3} = 6$, $C_{\sigma_2,3} = 7 = d$, and $C_{\sigma_3,3} = 10$; and so this is the output of the second stage of the list scheduling algorithm. Note that, in this stage, only the completion times in the last machine are modified, potentially increasing the waiting times of the jobs between the last but one machine and the last machine. See Figure 3c. The E/T measure of the scheduling being constructed is now determined and its value is 4. (Job 2 is one unit of time early; job 3 ends at the due date; and job 1 is three units of time late.)

In the third and last stage of the list scheduling algorithm (lines 16–19), the completion times in all machines other than the last one are recomputed so as to reduce the waiting times that might be enlarged in the second stage. This is the meaning of trying to recover the "compactness" of the schedule in the third stage. This goal is achieved introducing each job to the system as late as possible, that reduces the aggregated waiting times. See Figure 3d. The final scheduling has E/T measure equal to 4 (as already determined at the second stage) and sum of the waiting times equal to 2 (job $\sigma_1 = 2$ and $\sigma_2 = 3$ have a unit of waiting time between machines 2 and 3 each; while job $\sigma_3 = 1$ has no waiting time). The constructed scheduling is not optimal, since, as it was shown in Section 2, the optimal schedule has E/T measure equal to 2 and no waiting times at all.

4 Beam search method

In this section, a filtered beam search algorithm that applies to the tackled problem is introduced. A beam search algorithm is a semi-enumerative deterministic approach that consists in constructing, simultaneously, a "small" population of solutions. The amount of solutions that are constructed simultaneously is usually named β and it is known as "width of the beam". Assume that constructing a solution involves L steps and that, at a given stage $\ell < L$, we have β partial solution. There may be a huge number of possibilities for transforming a partial solution at stage ℓ into a partial solution at stage $\ell + 1$; and choosing among a huge number of possibilities may be very expensive. Thus, in a filtered beam search method, possibilities are filtered with a cheap procedure. For each solution at stage ℓ, α partial solutions that will potentially populate stage $\ell + 1$ are selected with a *local search*. Then, a more expensive procedure named *global search* is applied to the $\alpha \times \beta$ partial solutions; and the best β are chosen to constitute the set of partial solutions of stage $\ell + 1$. When the last stage is reached, partial solutions are in fact complete solutions and the best among them is returned. In the filtered beam search method being introduced in this work, we have $L = n$ since, as well as in the list scheduling algorithm, jobs are added one at a time, starting from an empty sequence until reaching a sequence with the n considered jobs.

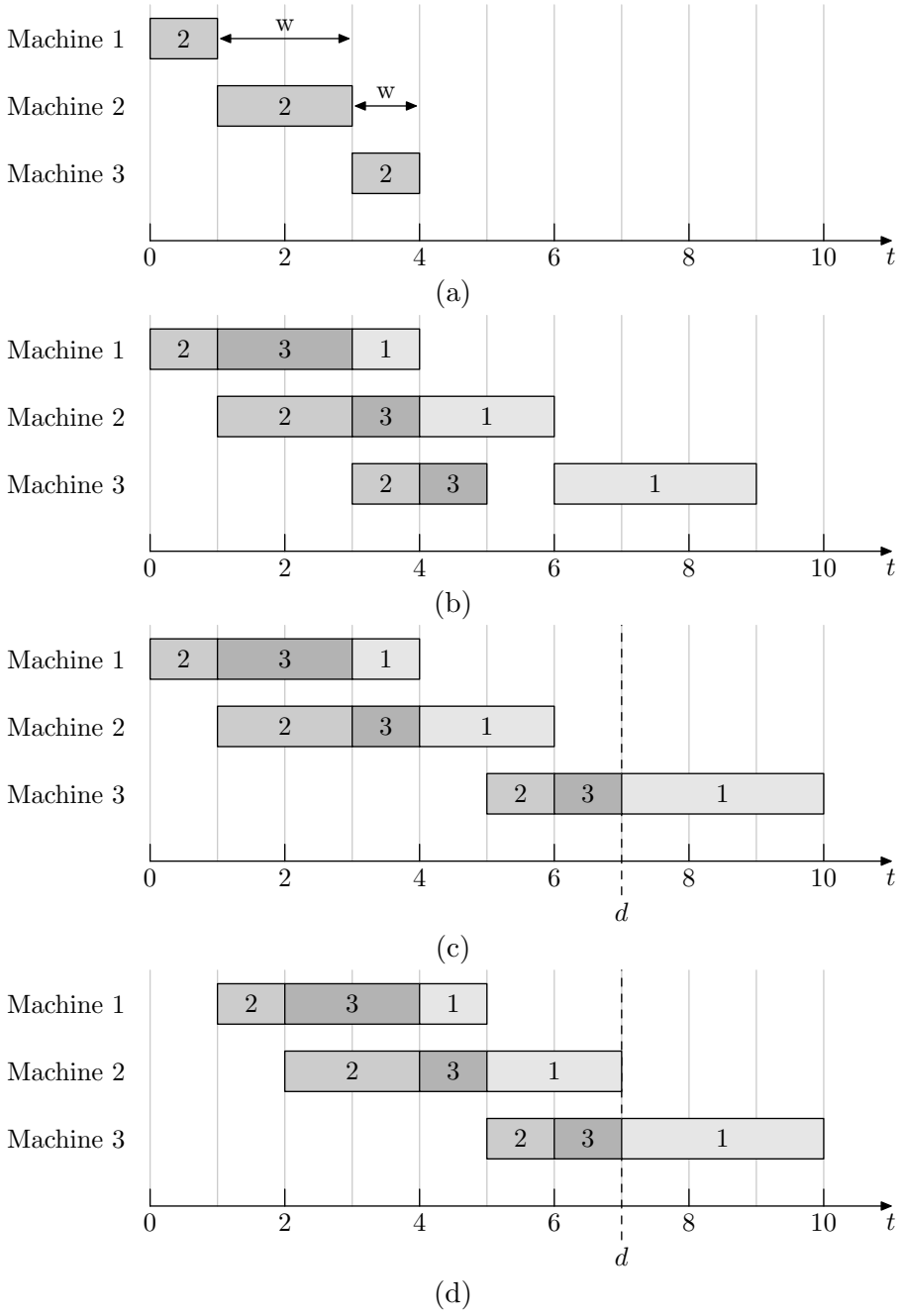

Figure 2: Illustration of the application of the list scheduling algorithm to a simple small instance with three jobs and three machines. (a) Partial schedule with a single job. The picture highlights the "windows" left by the job on machines 1 and 2. Windows are denoted by "w" in the picture. (b) Schedule obtained at the first stage, in which a sequence is constructed and jobs are scheduled as soon as possible. (c) Schedule obtained after the second stage, in which the due date is considered and the completion times in the last machine are recomputed in order to minimize the E/T measure for the already computed sequence. (d) Final schedule.

A beam search method is a natural way of extending a list scheduling algorithm like the one introduced in the previous section. In the list scheduling, at each iteration (stage or step) ℓ, we have a partial solution with $\ell - 1$ "fixed" jobs and the ℓth job of the sequence must be chosen. For each "unfixed" job j, the merit function Γ_j given by (14) is computed and the job that minimizes it is chosen. Of course, this is a local greedy choice. In the beam search, for each one of the β partial solution with $\ell - 1$ jobs, the α unfixed jobs with smallest value of the merit function are considered as candidates to occupy the ℓth position in the sequence. Then, each one of the $\alpha \times \beta$ partial solutions with ℓ jobs are evaluated in a more expensive way (global search) by completing them using, again, the list scheduling algorithm. Among the $\alpha \times \beta$ partial solutions with ℓ jobs, the β ones that, when completed, posses the smallest value of the objective function (13) are selected to constitute the new population of solutions (now with ℓ fixed jobs).

The filtered beam search algorithm is described in Algorithm 2. The set $\mathcal{L}$ represents the population with β partial solutions that, in line 3, is initialized as an empty set. Lines from 4 to 10 build the β partial solutions of the first level. These partial solutions have a single job that corresponds to one of the β jobs with smallest sum of processing times. In case of ties, the smallest index rule is used as a tie-breaking rule. Each partial solution is represented by a permutation $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ of the sequence $(1, 2, \ldots, n)$. If the partial solution is a solution with $\ell - 1$ fixed jobs then $\sigma_1, \ldots, \sigma_{\ell-1}$ represent the fixed jobs; while $\sigma_\ell, \ldots, \sigma_n$ correspond to the unfixed jobs. This is why, if $j_1, \ldots, j_\beta$ represent the indices of the β jobs with smallest sum of their processing times (computed at line 5), the β partial solutions with a single fixed job are constructed by considering the sequence $\sigma = (1, 2, \ldots, n)$ and interchanging σ_1 and σ_j for each $j \in \{j_1, \ldots, j_\beta\}$ (see line 8). In addition to the partial sequence, a partial solution also saves the completion times of the fixed jobs, that, in the first level of the method, corresponds to scheduling the fixed job in an optimal way, i.e. with its completion time equal to the due date if possible or with the smallest tardiness otherwise, plus no waiting time between the machines (see line 9). In line 10, it can be seen that pairs of the form (σ, C) are added to $\mathcal{L}$, σ representing the partial sequence and C representing the completion times of the fixed jobs. The portion of the algorithm that we are describing mentions $\bar{\beta} = \min\{\beta, n\}$ instead of β, so it is well defined even when the parameter β is larger than n. The description of the way the first level is built ends here. The large loop from lines 11 to 39 corresponds to the construction of the remaining $n - 1$ stages and it is described below.

The construction of a new set of partial solutions with one more fixed job starts defining $\mathcal{L}'$ as an empty set (see line 12). This set will contain a set of $\alpha \times \beta$ partial solutions that will potentially constitute the new set $\mathcal{L}$. Most of them will be discarded to reduce its cardinality to β (see lines from 35 to 38) and the iteration will end by setting $\mathcal{L} \leftarrow \mathcal{L}'$ (see line 39). For each partial solution (σ, C) in $\mathcal{L}$, we consider $\bar{\alpha}$ possibilities for the job that might occupy the ℓth position in σ, where $\bar{\alpha}$ is the minimum between the given parameter α and the remaining quantity of unfixed jobs (see line 13). The selection of the jobs starts by discarding the completion times C and recomputing $C_{\sigma_i,k}$ $(i = 1, \ldots, \ell - 1,\ k = 1, \ldots, m)$ by scheduling the jobs $\sigma_1, \ldots, \sigma_{\ell-1}$ as soon as possible. With this scheduling, for each unfixed job $j \in \{\sigma_\ell, \ldots, \sigma_n\}$, it is computed the merit function Γ_{σ_j}, and the $\bar{\alpha}$ unfixed jobs with the smallest value of the merit function are selected. The indices of these unfixed jobs are represented by $j_1, \ldots, j_{\bar{\alpha}}$ (see lines from 18 to 21). Then, $\bar{\alpha}$ copies of the partial sequence with $\ell - 1$ jobs are done. In each one of them, a job $j \in \{j_1, \ldots, j_{\bar{\alpha}}\}$ is located at the ℓth position of the sequence; and, aiming to evaluate the quality of the sequence, it is completed

using the list scheduling algorithm (lines 25 to 28). Then, for each sequence, the completion times of the jobs are computed as it is done in the list scheduling algorithm, i.e. the timing procedure is used to compute the completion time of all the jobs in the last machine (line 29) and, then, the completion times in the other machines are computed aiming to obtain a compact sequence with small waiting times between the machines (lines 30 to 33). At this point, we have a set of partial solutions with ℓ jobs that will potentially constitute the new set $\mathcal{L}$. The objective function (13) is evaluated (line 38) and the best β solutions are preserved, while to others are discarded. It should be stressed that the sequences were completed (with the list scheduling procedure) with the only purpose of evaluating their quality. This means that, for the selected ones, only the first ℓ jobs will be keep fixed. The algorithm ends the main loop after n steps with β solutions. The best one among these β solutions is chosen (line 41) as the one to be returned and the algorithm ends.

Algorithm 2 has two additional parameters that were not mentioned yet, namely, ℓ_{small} and β_{large}. They are both related to the expansion of the first levels of the search tree, that strongly affects the quality of the delivered solutions. At levels $\ell \leq \ell_{\text{small}}$, the filtering strategy is inhibited (see line 13). Moreover, the number of partial solutions at any level $\ell \leq \ell_{\text{small}}$ is not β but β_{large} (see line 35). The combination of these two features aims to construct, with an extra effort, a level $\ell = \ell_{\text{small}}$ with a more diverse set of good quality partial solutions.

Algorithm 2: Filtered Beam Search algorithm.

Data: n, m, d, p_{ik} $(i = 1, \ldots, n; k = 1, \ldots, m), \alpha, \beta, \beta_{\text{large}}, \ell_{\text{small}}, \gamma$.
Result: σ_i $(i = 1, \ldots, n), C_{ik}$ $(i = 1, \ldots, n; k = 1, \ldots, m)$.

begin
  $P_i \leftarrow \sum_{k=1}^{m} p_{ik}$ for $i = 1, \ldots, n$
  $\mathcal{L} \leftarrow \emptyset$
  $\bar{\beta} \leftarrow \min\{\beta, n\}$
  if $\bar{\beta} < n$ **then** let $\{j_1, \ldots, j_n\} = \{1, \ldots, n\}$ be such that $(P_{j_1}, j_1) \prec \cdots \prec (P_{j_n}, j_n)$
  else let $j_1 = 1, j_2 = 2, \ldots, j_n = n$
  foreach $j \in \{j_1, \ldots, j_{\bar{\beta}}\}$ **do**
    let $\sigma = (1, 2, \ldots, n)$ and swap(σ_1, σ_j)
    $C_{\sigma_1,m} \leftarrow \max\{d, P_{\sigma_1}\} - p_{\sigma_1,m}$, **for** $k \leftarrow m-1$ **to** 1 *with step* -1 **do** $C_{\sigma_1,k} \leftarrow C_{\sigma_1,k+1} - p_{\sigma_1,k+1}$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\sigma, C)\}$
  for $\ell \leftarrow 2$ **to** n **do**
    $\mathcal{L}' \leftarrow \emptyset$
    if $\ell \leq \ell_{\text{small}}$ **then** $\bar{\alpha} = n - \ell + 1$ **else** $\bar{\alpha} \leftarrow \min\{\alpha, n - \ell + 1\}$
    foreach $(\sigma, C) \in \mathcal{L}$ **do**
      $C_{\sigma_1,1} \leftarrow p_{\sigma_1,1}$, **for** $k \leftarrow 2$ **to** m **do** $C_{\sigma_1,k} \leftarrow C_{\sigma_1,k-1} + p_{\sigma_1,k}$
      for $i \leftarrow 2$ **to** $\ell - 1$ **do**
        $C_{\sigma_i,1} \leftarrow C_{\sigma_{i-1},1} + p_{\sigma_i,1}$, **for** $k \leftarrow 2$ **to** m **do** $C_{\sigma_i,k} \leftarrow \max\{C_{\sigma_i,k-1}, C_{\sigma_{i-1},k}\} + p_{\sigma_i,k}$
      if $\bar{\alpha} < n - \ell + 1$ **then**
        for $j \leftarrow \ell$ **to** n **do** $\Gamma_{\sigma_j} \leftarrow (1-\gamma)P_{\sigma_j} + \gamma \sum_{k=1}^{m-1} \left|C_{\sigma_{\ell-1},k+1} - C_{\sigma_{\ell-1},k} - p_{\sigma_j,k}\right|$
        let $\{j_1, \ldots, j_{n-\ell+1}\} = \{\ell, \ldots, n\}$ be such that $(\Gamma_{\sigma_{j_1}}, \sigma_{j_1}) \prec \cdots \prec (\Gamma_{\sigma_{j_{n-\ell+1}}}, \sigma_{j_{n-\ell+1}})$
      else let $j_1 = \ell, j_2 = \ell + 1, \ldots, j_{n-\ell+1} = n$
      foreach $j \in \{j_1, \ldots, j_{\bar{\alpha}}\}$ **do**
        let $\bar{\sigma}$ be a copy of σ and swap$(\bar{\sigma}_\ell, \bar{\sigma}_j)$
        $C_{\bar{\sigma}_\ell,1} \leftarrow C_{\bar{\sigma}_{\ell-1},1} + p_{\bar{\sigma}_\ell,1}$, **for** $k \leftarrow 2$ **to** m **do** $C_{\bar{\sigma}_\ell,k} \leftarrow \max\{C_{\bar{\sigma}_\ell,k-1}, C_{\bar{\sigma}_{\ell-1},k}\} + p_{\bar{\sigma}_\ell,k}$
        for $i \leftarrow \ell + 1$ **to** n **do**
          for $t \leftarrow i$ **to** n **do** $\Gamma_{\bar{\sigma}_t} \leftarrow (1-\gamma)P_{\bar{\sigma}_t} + \gamma \sum_{k=1}^{m-1} \left|C_{\bar{\sigma}_{i-1},k+1} - C_{\bar{\sigma}_{i-1},k} - p_{\bar{\sigma}_t,k}\right|$
          swap$(\bar{\sigma}_\ell, \bar{\sigma}_{t_1})$, where t_1 is the index corresp. to the smallest $\{(\Gamma_{\bar{\sigma}_t}, \bar{\sigma}_t) \mid t = i, \ldots, n\}$
          $C_{\bar{\sigma}_i,1} \leftarrow C_{\bar{\sigma}_{i-1},1} + p_{\bar{\sigma}_i,1}$, **for** $k \leftarrow 2$ **to** m **do** $C_{\bar{\sigma}_i,k} \leftarrow \max\{C_{\bar{\sigma}_i,k-1}, C_{\bar{\sigma}_{i-1},k}\} + p_{\bar{\sigma}_i,k}$
        Considering the common due date d, the processing times $p_{\bar{\sigma}_i,m}$ $(i = 1, \ldots, n)$, and the release dates $r_{\bar{\sigma}_i} \equiv C_{\bar{\sigma}_i,m-1}$ $(i = 1, \ldots, n)$, compute $\bar{C}_{\bar{\sigma}_i,m}$ $(i = 1, \ldots, n)$ by calling the Timing algorithm introduced in [30, p.65].
        for $k \leftarrow m - 1$ **to** 1 *with step* -1 **do**
          $\bar{C}_{\bar{\sigma}_n,k} \leftarrow \bar{C}_{\bar{\sigma}_n,k+1} - p_{\bar{\sigma}_n,k+1}$
          for $i \leftarrow n - 1$ **to** 1 *with step* -1 **do**
            $\bar{C}_{\bar{\sigma}_i,k} \leftarrow \min\{\bar{C}_{\bar{\sigma}_{i+1},k} - p_{\bar{\sigma}_{i+1},k}, \bar{C}_{\bar{\sigma}_i,k+1} - p_{\bar{\sigma}_i,k+1}\}$
        $\mathcal{L}' \leftarrow \mathcal{L}' \cup \{(\bar{\sigma}, \bar{C})\}$
    if $\ell \leq \ell_{\text{small}}$ **then** $\bar{\beta} \leftarrow \beta_{\text{large}}$ **else** $\bar{\beta} \leftarrow \beta$
    if $|\mathcal{L}'| > \bar{\beta}$ **then**
      foreach $(\bar{\sigma}, \bar{C}) \in \mathcal{L}'$ **do**
        By setting $E_j = \max\{0, d - \bar{C}_{\bar{\sigma}_j}\}$ and $T_j = \max\{0, \bar{C}_{\bar{\sigma}_j} - d\}$, for $j = 1, \ldots, n$, compute the objective function value $f(\bar{\sigma}, \bar{C})$ given by (13).
      Keep the $\bar{\beta}$ pairs $(\bar{\sigma}, \bar{C}) \in \mathcal{L}'$ with smallest $(f(\bar{\sigma}, \bar{C}), \bar{\sigma}_\ell)$ and discard the others.
    $\mathcal{L} \leftarrow \mathcal{L}'$
  Let σ and C be such that $(\sigma, C) \in \mathcal{L}$ is the element with smallest $(f(\sigma, C), \sigma_n)$.

5 Numerical experiments

Algorithms 1 and 2 were implemented in C/C++ and are freely avaliable for download at `http://www.ime.usp.br/~egbirgin/`. Codes were compiled with the g++ compiler of GCC (version 6.3.0 20170516, Debian 6.3.0-18+deb9u1) using the optimization option "-O3". All tests were conducted on a 2.40GHz Intel(R) Xeon(R) CPU E5645 with 216GB of RAM memory and running GNU/Linux operating system (Debian 9, kernel 4.9.0-6-amd64 x86_64).

In the numerical experiments, we considered a set composed by 2,400 instances introduced in [8] and available at `http://home.iitk.ac.in/~pmehta/flowshop.htm`[3], that follows the experimental design suggested in [37]. First, the twenty four combinations of the number of jobs n and the number of machines m with $n \in \{5, 10, 20, 50, 80, 100\}$ and $m \in \{5, 10, 15, 20\}$ are considered. For each combination, fifty instances are generated choosing the processing times p_{ik} ($i = 1, \ldots, n$, $k = 1, \ldots, m$) as random integer numbers in the range $[1, 99]$ with discrete uniform distribution. Finally, for each of the 1,200 instances, two different restrictive due dates $d' = \lfloor \frac{1}{2}(d_1 + d_2) \rfloor$ and a random integer d'' in the range $[d_2, d_1]$ with discrete uniform distribution are considered, where $d_2 = \min_{1 \leq i \leq n} \{\sum_{k=1}^{m} p_{ik}\}$ and d_1 satisfying $d_1 \geq d_2$ is an integer unrestrictive due date. (See [8, pp.5596-5597] for details in the way d_1 is computed.) From now on, we refer to the 1,200 instances with due date d' and to the 1,200 instances with due date d'' as Set d' and Set d'', respectively.

5.1 Evaluation of the list scheduling algorithm's parameter γ

In this section, the experiments aim to evaluate the performance of the list scheduling algorithm for variations of its parameter $\gamma \in \{0, 0.1, \ldots, 1\}$. Note that, when $\gamma = 0$, the list scheduling algorithm generates the same sequence generated by the well-known SPT (smallest processing time) dispatching rule using the total processing time. In [30], the problem of minimizing the mean absolute deviation from a common due date in a flowshop environment with two machines (i.e. problem (1–9) with $m = 2$) was considered; and some heuristic approaches were introduced. Nine different dispatching rules were considered for the process of constructing an initial guess for the heuristic approaches and, in that context, the SPT rule was the most successful one. This means that the present comparison includes a comparison against a dispatching rule that showed to be very effective in a similar problem.

Tables 1 and 2 show the results of the list scheduling algorithm applied to the instances in Sets d' and d'', respectively. For each instance $\mathcal{I}$, the list scheduling algorithm is being applied with eleven different values of $\gamma \in \{0, 0.1, \ldots, 1\}$. Let $f_{\bar{\gamma}}^{\mathcal{I}}$ be the value of the objective function (13) associated with the solution obtained by the list scheduling algorithm when applied to instance $\mathcal{I}$ with $\gamma = \bar{\gamma}$ and let

$$f_{\min}^{\mathcal{I}} = \min_{\gamma \in \{0, 0.1, \ldots, 1\}} \{f_{\gamma}^{\mathcal{I}}\} \quad \text{and} \quad f_{\max}^{\mathcal{I}} = \max_{\gamma \in \{0, 0.1, \ldots, 1\}} \{f_{\gamma}^{\mathcal{I}}\}.$$

The relative index deviation (RDI) associated with the performance of the list scheduling algorithm with $\gamma = \bar{\gamma}$ when applied to instance $\mathcal{I}$ is given by

$$RDI_{\bar{\gamma}}^{\mathcal{I}} = \frac{f_{\bar{\gamma}}^{\mathcal{I}} - f_{\min}^{\mathcal{I}}}{f_{\max}^{\mathcal{I}} - f_{\min}^{\mathcal{I}}}.$$

[3] A copy is also available at `http://www.ime.usp.br/~egbirgin/`

It is clear from the definition that the index goes from zero to one and that, the smaller the value of the index, the better the relative performance of the method. Each cell in the tables corresponds to the average of the $RDI_\gamma^{\mathcal{I}}$ computed over each subset of 50 instances with the same number of jobs n and machines m. The last two rows in the tables also show, for each γ, the average of the $RDI_\gamma^{\mathcal{I}}$ over the whole set of 1,200 instances in Sets d' and d'' and the number of times the list scheduling algorithm found the best solution using this value of γ. In both tables, it can be observed that the smallest average $RDI_\gamma^{\mathcal{I}}$ and the largest number of best solutions are obtained when $\gamma = \frac{1}{2}$. It is worth noticing that, in both tables, the number of best solutions as a function of γ exhibits a nice bell-shaped curve.

Size		γ										
n	m	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
5	5	0.46	0.39	0.32	0.26	0.27	0.23	0.26	0.27	0.30	0.39	0.48
5	10	0.20	0.23	0.22	0.24	0.30	0.36	0.42	0.49	0.62	0.72	0.78
5	15	0.44	0.39	0.38	0.37	0.42	0.37	0.33	0.35	0.44	0.46	0.51
5	20	0.32	0.31	0.29	0.30	0.29	0.36	0.44	0.45	0.51	0.67	0.68
10	5	0.67	0.58	0.50	0.48	0.41	0.32	0.24	0.26	0.35	0.51	0.60
10	10	0.55	0.47	0.45	0.35	0.40	0.29	0.28	0.39	0.49	0.59	0.71
10	15	0.50	0.46	0.43	0.38	0.39	0.33	0.25	0.35	0.37	0.56	0.68
10	20	0.46	0.39	0.32	0.28	0.28	0.33	0.32	0.39	0.49	0.60	0.75
20	5	0.66	0.59	0.52	0.41	0.36	0.30	0.26	0.37	0.48	0.62	0.78
20	10	0.63	0.47	0.44	0.40	0.32	0.32	0.30	0.39	0.48	0.57	0.70
20	15	0.64	0.59	0.47	0.42	0.38	0.27	0.38	0.39	0.57	0.57	0.68
20	20	0.62	0.52	0.44	0.39	0.34	0.31	0.31	0.37	0.44	0.56	0.68
50	5	0.55	0.39	0.27	0.20	0.13	0.13	0.14	0.25	0.47	0.72	0.94
50	10	0.70	0.55	0.42	0.29	0.19	0.18	0.21	0.33	0.58	0.77	0.89
50	15	0.74	0.55	0.38	0.30	0.21	0.19	0.26	0.33	0.54	0.74	0.80
50	20	0.80	0.58	0.47	0.36	0.28	0.23	0.24	0.39	0.55	0.68	0.79
80	5	0.45	0.27	0.18	0.13	0.09	0.10	0.14	0.29	0.47	0.78	0.99
80	10	0.61	0.42	0.34	0.21	0.14	0.10	0.17	0.32	0.64	0.84	0.90
80	15	0.74	0.52	0.33	0.27	0.17	0.17	0.20	0.29	0.57	0.76	0.84
80	20	0.74	0.53	0.36	0.27	0.21	0.16	0.17	0.37	0.55	0.71	0.84
100	5	0.37	0.23	0.15	0.09	0.05	0.06	0.12	0.25	0.45	0.72	1.00
100	10	0.61	0.42	0.29	0.21	0.15	0.12	0.14	0.39	0.69	0.87	0.94
100	15	0.74	0.50	0.34	0.23	0.19	0.10	0.19	0.35	0.62	0.84	0.86
100	20	0.76	0.49	0.32	0.25	0.18	0.14	0.17	0.39	0.61	0.79	0.86
Average		0.58	0.45	0.36	0.30	0.26	0.23	0.25	0.35	0.51	0.67	0.78
# of best solutions		124	153	180	234	302	357	298	188	133	99	77

Table 1: Analysis of the influence of the weight γ in the performance of the list scheduling algorithm when applied to the 1,200 instances in Set d'.

Size		γ										
n	m	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
5	5	0.46	0.40	0.34	0.30	0.33	0.29	0.32	0.32	0.33	0.43	0.47
5	10	0.26	0.26	0.22	0.20	0.30	0.37	0.44	0.50	0.62	0.71	0.77
5	15	0.43	0.39	0.42	0.42	0.41	0.38	0.35	0.38	0.45	0.48	0.51
5	20	0.29	0.36	0.36	0.36	0.35	0.36	0.47	0.49	0.56	0.69	0.69
10	5	0.66	0.60	0.50	0.43	0.35	0.34	0.26	0.30	0.37	0.51	0.58
10	10	0.61	0.53	0.51	0.39	0.41	0.35	0.37	0.44	0.50	0.61	0.65
10	15	0.51	0.46	0.41	0.36	0.42	0.35	0.30	0.39	0.41	0.57	0.63
10	20	0.46	0.43	0.37	0.36	0.34	0.35	0.42	0.48	0.55	0.60	0.73
20	5	0.66	0.57	0.47	0.40	0.36	0.31	0.34	0.40	0.49	0.61	0.75
20	10	0.66	0.53	0.49	0.44	0.35	0.36	0.33	0.36	0.47	0.59	0.65
20	15	0.62	0.58	0.49	0.42	0.37	0.31	0.42	0.42	0.54	0.59	0.67
20	20	0.62	0.51	0.46	0.44	0.36	0.32	0.32	0.36	0.42	0.54	0.65
50	5	0.58	0.45	0.35	0.22	0.20	0.20	0.21	0.23	0.51	0.69	0.88
50	10	0.72	0.57	0.46	0.35	0.31	0.17	0.27	0.34	0.55	0.69	0.79
50	15	0.71	0.56	0.42	0.33	0.28	0.27	0.27	0.31	0.56	0.70	0.74
50	20	0.76	0.59	0.50	0.43	0.29	0.24	0.23	0.41	0.54	0.64	0.77
80	5	0.53	0.38	0.30	0.22	0.16	0.12	0.16	0.28	0.49	0.75	0.91
80	10	0.64	0.49	0.41	0.27	0.20	0.14	0.21	0.28	0.61	0.77	0.83
80	15	0.73	0.54	0.39	0.35	0.28	0.20	0.20	0.30	0.54	0.72	0.80
80	20	0.69	0.52	0.36	0.31	0.23	0.19	0.15	0.36	0.58	0.71	0.81
100	5	0.48	0.37	0.29	0.21	0.16	0.13	0.17	0.26	0.44	0.71	0.89
100	10	0.64	0.49	0.37	0.31	0.25	0.14	0.17	0.34	0.67	0.85	0.86
100	15	0.71	0.51	0.39	0.28	0.23	0.15	0.21	0.37	0.56	0.76	0.82
100	20	0.72	0.53	0.37	0.30	0.21	0.22	0.22	0.38	0.62	0.72	0.81
Average		0.59	0.48	0.40	0.34	0.30	0.26	0.28	0.36	0.52	0.65	0.74
# of best solutions		126	144	167	218	267	362	287	202	148	119	104

Table 2: Analysis of the influence of the weight γ in the performance of the list scheduling algorithm when applied to the 1,200 instances in Set d''.

5.2 Evaluation of the beam search algorithm's parameters α and β

In the experiment of the present section, we aim to evaluate the influence of the beam search parameters α and β in the method's performance. Parameter ℓ_{small}, that inhibits the filtering at levels $\ell \leq \ell_{\text{small}}$, and parameter $\beta_{\text{large}} \geq \beta$, that determines the beam width when $\ell \leq \ell_{\text{small}}$, have a twofold nature. On the one hand, for medium- and large-sized instances, they intensify the search for better quality partial solutions at the beginning of the search tree. On the other hand, for small-sized instances, they allow all possible sequences to be tested. Based on the latter objective, we arbitrarily set $\ell_{\text{small}} = 4$ and $\beta_{\text{large}} = 120$; since using these values implies that all possible sequences are considered for instances with up to 5 jobs. Based on the numerical experimentation

with the list scheduling algorithm in the previous section, we also set $\gamma = 0.5$.

Table 3 shows the results of applying the beam search algorithm with the sixteen combinations of α and $\beta \in \mathcal{S} \equiv \left\{\left\lceil \frac{1}{4}n \right\rceil, \left\lceil \frac{1}{2}n \right\rceil, \left\lceil \frac{3}{4}n \right\rceil, n\right\}$ to 20% of the instances in Sets d' and d'', i.e. 10 instances (over a total of 50) for each combination of n and m. The left-hand-side of the table displays, in each cell associated with a pair $(\bar{\alpha}, \bar{\beta}) \in \mathcal{S} \times \mathcal{S}$, the average of $RDI^{\mathcal{I}}_{\bar{\alpha},\bar{\beta}}$ over all considered instances $\mathcal{I}$, where

$$RDI^{\mathcal{I}}_{\bar{\alpha},\bar{\beta}} = \frac{f^{\mathcal{I}}_{\bar{\alpha},\bar{\beta}} - f^{\mathcal{I}}_{\min}}{f^{\mathcal{I}}_{\max} - f^{\mathcal{I}}_{\min}},$$

$f^{\mathcal{I}}_{\bar{\alpha},\bar{\beta}}$ is the value of the objective function (13) associated with the solution obtained by the beam search algorithm when applied to instance $\mathcal{I}$ with $\alpha = \bar{\alpha}$ and $\beta = \bar{\beta}$ (other than $\ell_{\text{small}} = 4$, $\beta_{\text{large}} = 120$, and $\gamma = 0.5$), $f^{\mathcal{I}}_{\min} = \min_{\alpha,\beta \in \mathcal{S}} \left\{f^{\mathcal{I}}_{\alpha,\beta}\right\}$, and $f^{\mathcal{I}}_{\max} = \max_{\alpha,\beta \in \mathcal{S}} \left\{f^{\mathcal{I}}_{\alpha,\beta}\right\}$. The right-hand-side of the table shows, for each pair $(\bar{\alpha}, \bar{\beta}) \in \mathcal{S} \times \mathcal{S}$, the number of best solutions obtained. Every column in the table shows that, the larger the width of the beam β, the smaller the relative deviation index and the larger the number of best solutions found. On the other hand, the behavior of the method as a function of the filtering parameter α is not very clear. A big improvement is obtained from $\alpha = \left\lceil \frac{1}{4}n \right\rceil$ to $\alpha = \left\lceil \frac{1}{2}n \right\rceil$, but better results are *not* always obtained for larger values of α. Overall, the best combination (among the tested combinations) is given by $\alpha = \left\lceil \frac{1}{2}n \right\rceil$ and $\beta = n$. Of course, the larger the values of α and β, the larger the computational effort of the beam search algorithm. Figure 3 displays the average CPU time in seconds, computed over all the instances being considered in the present experiment, as a function of α and β.

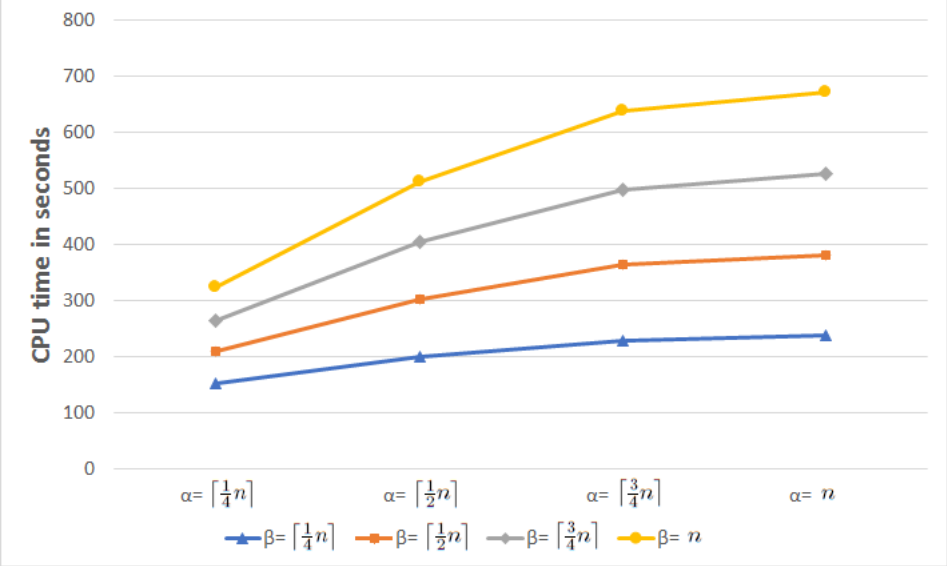

Figure 3: Average CPU time in seconds of the beam search algorithm as a function of its parameters α and β.

	RDI				Number of best solutions			
	$\alpha = \lceil \frac{1}{4}n \rceil$	$\alpha = \lceil \frac{1}{2}n \rceil$	$\alpha = \lceil \frac{3}{4}n \rceil$	$\alpha = n$	$\alpha = \lceil \frac{1}{4}n \rceil$	$\alpha = \lceil \frac{1}{2}n \rceil$	$\alpha = \lceil \frac{3}{4}n \rceil$	$\alpha = n$
$\beta = \lceil \frac{1}{4}n \rceil$	0.58	0.51	0.49	0.49	73	78	79	80
$\beta = \lceil \frac{1}{2}n \rceil$	0.39	0.33	0.33	0.33	86	90	88	90
$\beta = \lceil \frac{3}{4}n \rceil$	0.29	0.23	0.22	0.23	101	108	106	108
$\beta = n$	0.21	0.13	0.15	0.14	114	139	125	133

(a) Set d'.

	RDI				Number of best solutions			
	$\alpha = \lceil \frac{1}{4}n \rceil$	$\alpha = \lceil \frac{1}{2}n \rceil$	$\alpha = \lceil \frac{3}{4}n \rceil$	$\alpha = n$	$\alpha = \lceil \frac{1}{4}n \rceil$	$\alpha = \lceil \frac{1}{2}n \rceil$	$\alpha = \lceil \frac{3}{4}n \rceil$	$\alpha = n$
$\beta = \lceil \frac{1}{4}n \rceil$	0.60	0.49	0.50	0.50	72	73	72	75
$\beta = \lceil \frac{1}{2}n \rceil$	0.37	0.33	0.34	0.35	80	77	80	75
$\beta = \lceil \frac{3}{4}n \rceil$	0.27	0.20	0.25	0.24	98	105	96	98
$\beta = n$	0.21	0.12	0.16	0.16	110	136	118	128

(b) Set d''.

Table 3: Evaluation of the beam search method for different combinations of its parameters α and $\beta \in \left\{ \lceil \frac{1}{4}n \rceil, \lceil \frac{1}{2}n \rceil, \lceil \frac{3}{4}n \rceil, n \right\}$ applied to 20% of the instances in Sets d' and d''.

5.3 Beam search algorithm versus list scheduling algorithm

Having fixed $\alpha = \lceil \frac{1}{2}n \rceil$, $\beta = n$, $\ell_{\text{small}} = 4$, $\beta_{\text{large}} = 120$, and $\gamma = 0.5$, the beam search algorithm was applied to all the 2,400 instances in Sets d' and d''; and we now evaluate the improvement obtained by the beam search algorithm over the list scheduling algorithm. Table 4 shows the results. For each pair (n, m), the table shows the average, over all the fifty instances $\mathcal{I}$ with n jobs and m machines, of the gap given by

$$100\% \left(\frac{f_{\text{BS}}^{\mathcal{I}} - f_{\text{LS}}^{\mathcal{I}}}{f_{\text{LS}}^{\mathcal{I}}} \right),$$

where $f_{\text{BS}}^{\mathcal{I}}$ and $f_{\text{LS}}^{\mathcal{I}}$ are the values of the objective function (13) associated with the solution computed by the beam search and the list scheduling algorithms, respectively, when applied to instance $\mathcal{I}$. The table shows an average gap of approximately -22%, meaning that the beam search algorithm improves the solutions found by the list scheduling algorithm in 22% in average. The table also shows the average CPU time in seconds used by the beam search algorithm. The list scheduling algorithm never uses more than 0.02 seconds of CPU time to solve any of the considered

instances.

Size		Set d'		Set d''	
n	m	gap (%)	time	gap (%)	time
5	5	-20.00	0.08	-20.71	0.08
5	10	-24.13	0.10	-23.41	0.09
5	15	-25.37	0.10	-25.79	0.10
5	20	-30.13	0.11	-29.55	0.11
10	5	-20.80	1.67	-20.14	1.67
10	10	-25.37	1.94	-24.29	1.95
10	15	-28.79	2.19	-25.55	2.17
10	20	-29.00	2.42	-26.93	2.44
20	5	-20.15	14.08	-19.43	14.01
20	10	-24.31	16.56	-22.01	16.75
20	15	-24.58	18.89	-22.85	18.94
20	20	-26.08	21.81	-24.24	21.72
50	5	-17.72	235.47	-17.68	239.48
50	10	-21.20	294.88	-19.59	295.64
50	15	-21.63	342.63	-21.03	341.75
50	20	-22.42	401.70	-21.54	400.60
80	5	-16.59	1227.46	-16.82	1200.81
80	10	-18.70	1543.24	-19.19	1552.12
80	15	-20.09	1843.45	-19.78	1856.81
80	20	-20.29	2172.28	-19.90	2170.82
100	5	-15.66	2786.91	-16.26	2809.85
100	10	-18.34	3604.56	-18.24	3607.43
100	15	-17.98	4296.59	-18.23	4334.85
100	20	-19.44	5184.46	-19.44	5176.73
Average		-22.03	1000.57	-21.36	1002.79

Table 4: Improvement of the beam search algorithm over the list scheduling algorithm.

5.4 Beam search and list scheduling algorithms against upper bounds computed by a commercial solver

In the next experiment, we consider instances with up to 50 jobs and 20 machines and we compare the solutions obtained by the list scheduling algorithm (with $\gamma = 0.5$) and the beam search algorithm (with $\alpha = \lceil \frac{1}{2}n \rceil$, $\beta = n, \ell_{\text{small}} = 4$, $\beta_{\text{large}} = 120$, and $\gamma = 0.5$) against solutions obtained with CPLEX. The model was implemented in C/C++ using the ILOG Concert Technology and solved using IBM ILOG CPLEX 12.8.0. As described, for example, in [1] and [5], by default, a solution is reported as optimal by the solver when

$$\text{absolute gap} = \text{best feasible solution} - \text{best lower bound} \leq \varepsilon_{\text{abs}}$$

or

$$\text{relative gap} = \frac{|\text{ best feasible solution} - \text{best lower bound }|}{10^{-10} + |\text{ best feasible solution }|} \leq \varepsilon_{\text{rel}},$$

with $\varepsilon_{\text{abs}} = 10^{-6}$ and $\varepsilon_{\text{rel}} = 10^{-4}$, where "best feasible solution" corresponds to the smallest value of the objective function related to a feasible solution generated by the solver. The objective function (13) has the particular property of assuming relatively large integer values at feasible points. Thus, a stopping criterion based on a relative error less than or equal to $\varepsilon_{\text{rel}} = 10^{-4}$ may stop the solver prematurely. On the other hand, due to the integrality of the objective function values, an absolute error strictly smaller than 1 is enough to prove optimality. Therefore, in the numerical experiments, we considered $\varepsilon_{\text{abs}} = 1 - 10^{-6}$ and $\varepsilon_{\text{rel}} = 0$. All other parameters of CPLEX were set with their default values. A CPU time limit of one hour was imposed.

As a whole, 1,600 instances were considered, namely, instances in the Sets d' and d'' with $n \in \{5, 10, 20, 50\}$ and $m \in \{5, 10, 15, 20\}$. The comparison splits the instances into two sets: 800 small-sized instances with $n \in \{5, 10\}$, for which it is expected CPLEX to find optimal solutions, and 800 medium-sized instances with $n \in \{20, 50\}$.

In the small-sized instances, CPLEX was run in two different ways – considering and not considering the solution found by the list scheduling algorithm as a warm start. Table 5 shows the results. For each subset of fifty instances, the table shows in column #Opt the number of instances for which an optimal solution was found and the average CPU time in seconds used by CPLEX in those instances in which an optimal solution was found. (In the remaining instances, the CPU time limit of one hour was reached and that time was not included in the average.) As it can be seen in the table, CPLEX without the warm start was unable to find an optimal solution (within the CPU time limit) in eight instances; while the same happened with the warm-started CPLEX in six instances. As a whole, none of the variants was able to find an optimal solution in five instances only and, therefore, optimal solutions for 795 instances (out of 800) were found. Although, for the remaining five instances, small gaps were obtained, the comparison will consider the 795 instances for which an optimal solution is available. (Details can be found in `http://www.ime.usp.br/~egbirgin/`.)

Tables 6 and 7 show the performances of the list scheduling and the beam search method, respectively, in the 795 small-sized instances with known optimal solution. For each pair (n, m), column #Inst says how many instances with known optimal solution are being considered; while column #Opt says in how many instances the method (list scheduling or beam search) was able to find an optimal solution. For those #Inst − #Opt instances $\mathcal{I}$ for which the method was *not* able to find an optimal solutions, column "gap" shows the average of the relative gap given by

$$100\% \left(\frac{f^{\mathcal{I}}_{\text{M}} - f^{\mathcal{I}}_{\text{OPT}}}{f^{\mathcal{I}}_{\text{OPT}}} \right),$$

where $f^{\mathcal{I}}_{\text{M}}$ and $f^{\mathcal{I}}_{\text{OPT}}$ are the values of the objective function (13) associated with the solution computed by the method being evaluated and the optimal solution computed by CPLEX, respectively, when applied to instance $\mathcal{I}$. Column "time" shows the average CPU time in seconds. (For the case of the list scheduling algorithm, CPU times are omitted since it never uses more than 10^{-5} seconds of CPU time.) For example, Table 7 shows that, in Set d' with $(n, m) = (5, 5)$, 50 instances are being considered; an optimal solution was found by the beam serach method in 32 of them; and, in

the 18 cases in which an optimal solution was not found, the average relative gap of the objective function (13) is 0.0005%. These figures suggests the beam search method performed very well in the small-sized instances, finding good quality solutions in a small fraction of a second. On the other hand, Table 6 shows that, as expected, the list scheduling algorithm rarely finds an optimal solution; solutions being, in average, approximately 40% far from an optimal solution.

The small average relative gaps in column "gap" of Table 7 deserve further explanations. The objective function (13) consists in the earliness and tardiness penalty (1) of the jobs multiplied by a "large constant" plus the term (11) related to the waiting times of the jobs. Thus, the larger the constant, the smaller the gap, whenever the solution found by the beam search methods is optimal for the earliness and tardiness measure (1). Column #Opt(E/T) says in how many of the instances for which an optimal solution was not found, the solution is optimal for the earliness and tardiness measure (1). For example, comparing #Inst against #Opt + #Opt(E/T), it is possible to see that, as expected, the beam search method found solutions that are optimal for the earliness and tardiness measure (1) whenever $n = 5$. For the instances in which the solutions found *are* optimal only for the earliness and tardiness measure (1), column gap(W) shows the average relative gap in the waiting measure (11); and for the instances in which the solutions found are *not* optimal for the earliness and tardiness measure, column gap(E/T) shows the average relative gap of the earliness and tardiness measure (1). Summing up, Table 7 shows that (a) the beam search method found optimal solutions in approximately half of the instances; (b) when an optimal solutions is not found, the solution found is optimal for the earliness and tardiness measure in approximately half of the cases; and (c) in the remaining one quarter of the instances, the gap in the earliness and tardiness measure is small. Moreover, the overall gap is also small and the method never uses more than five hundredths of a second of CPU time in average.

		CPLEX				CPLEX with warm start			
Size		Set d'		Set d''		Set d'		Set d''	
n	m	#Opt	time	#Opt	time	#Opt	time	#Opt	time
5	5	50	26.91	50	23.75	50	20.56	50	20.11
5	10	50	22.80	50	23.27	50	19.85	50	20.54
5	15	50	23.39	50	23.47	50	19.70	50	20.22
5	20	50	20.84	50	21.35	50	19.48	50	19.13
10	5	50	800.47	50	582.96	50	799.85	50	602.62
10	10	49	951.29	48	966.58	49	1033.03	49	1069.88
10	15	49	1303.49	47	1016.49	49	1257.09	49	1128.52
10	20	50	1693.98	49	1522.34	50	1719.58	48	1423.80

Table 5: Performance of CPLEX in 800 small-sized instances.

In the medium-sized instances, CPLEX was also run in two different ways – considering and not considering the solution found by the list scheduling algorithm as a warm start. Within the imposed CPU time limit (one hour), in none of the instances the solver was able to achieve the required absolute or relative gap in order to report a solution as optimal. Table 8 compares the solutions found by the beam search algorithm against the best feasible solutions found by CPLEX. The table shows, for each set of 50 instances with the same number of jobs and machines, the average

Size		#Inst	#Opt	#Opt–E/T	gap–W (%)	gap–E/T (%)	gap (%)
n	m						
5	5	50	2	1	72.40	30.65	28.81
5	10	50	1	1	25.53	43.87	42.11
5	15	50	3	0	–	42.19	39.65
5	20	50	0	0	–	59.79	59.79
10	5	50	0	0	–	28.77	28.77
10	10	49	0	0	–	37.70	38.16
10	15	49	0	0	–	47.02	47.10
10	20	50	0	0	–	47.01	47.01
Average			0.75	0.25	48.97	42.21	41.50

(a) Set d'

Size		#Inst	#Opt	#Opt–E/T	gap–W (%)	gap–E/T (%)	gap (%)
n	m						
5	5	50	2	2	107.14	34.65	31.87
5	10	50	1	0	–	38.32	37.56
5	15	50	2	0	–	44.15	42.38
5	20	50	0	1	31.58	57.91	56.75
10	5	50	0	0	–	28.74	28.74
10	10	49	0	0	–	38.36	38.84
10	15	49	0	0	–	40.35	40.52
10	20	49	0	0	–	46.47	46.93
Average			0.63	0.38	81.95	41.13	40.48

(b) Set d''

Table 6: Evaluation of the list scheduling algorithm in 795 small-sized instances with known optimal solution.

improvement of the beam search algorithm over the best feasible solution found by CPLEX. The table also shows the average CPU time per instance of the beam search algorithm. It should be noted that the larger the instances, the larger the average improvement. Overall, using no more than 10 seconds of CPU time per instance, the beam search algorithm improved the best feasible solution found by CPLEX (with the CPU time limit of one hour) in %15.70 for instances in Set d' and in %12.03 for instances in Set d''. It should be noted that, due to the nature of the objective function of the problem under consideration, and considering that, very likely, solutions being compared are not optimal, the reported gaps of improvement correspond to improvements in the E/T measure.

Size		#Inst	#Opt	#Opt–E/T	gap–W (%)	gap–E/T (%)	gap (%)	time
n	m							
5	5	50	32	18	28.61	–	0.0005	0.0017
5	10	50	32	18	18.90	–	0.0003	0.0019
5	15	50	41	9	16.06	–	0.0001	0.0021
5	20	50	36	14	17.46	–	0.0002	0.0022
10	5	50	27	5	2.75	1.61	0.6722	0.0334
10	10	49	17	11	4.14	1.70	0.9804	0.0389
10	15	49	24	5	1.28	2.06	1.1634	0.0437
10	20	50	14	8	1.10	2.39	2.0858	0.0485
Average			28	11	14.98	1.99	0.6105	0.0215

(a) Set d'

Size		#Inst	#Opt	#Opt–E/T	gap–W (%)	gap–E/T (%)	gap (%)	time
n	m							
5	5	50	30	20	29.21	–	0.0004	0.0017
5	10	50	39	11	17.79	–	0.0001	0.0019
5	15	50	40	10	15.92	–	0.0001	0.0020
5	20	50	36	14	20.67	–	0.0002	0.0022
10	5	50	12	11	9.29	1.09	1.0170	0.0335
10	10	49	13	10	3.04	1.68	1.4601	0.0390
10	15	49	18	11	0.54	2.49	2.1593	0.0433
10	20	49	10	6	1.71	2.52	2.1931	0.0488
Average			25	12	14.81	1.94	0.8456	0.0215

(b) Set d''

Table 7: Evaluation of the beam search method in 795 small-sized instances with known optimal solution.

5.5 Comparison of the beam search algorithm for the minimization of the earliness and tardiness of the jobs against the method introduced in [8]

The method introduced in [8] seeks the minimization of the sum of the earliness and tardiness in a flowshop environment with a common due date; while ignoring the waiting times between the machines, i.e. it applies to problem (1–9). Therefore, the comparison presented in this section considers the sum of earliness and tardiness only, given by the objective function (1). Table 9 presents a comparison of the beam search algorithm introduced in the present work (with $\alpha = \lceil 0.5n \rceil$, $\beta = n$, $\ell_{\text{small}} = 4$, $\beta_{\text{large}} = 120$, and $\gamma = 0.5$) against the tabu search algorithm introduced in [8]. For each pair (n, m), the table shows the average, over all the fifty instances $\mathcal{I}$ with n jobs

Size		Set d'		Set d''	
n	m	gap (%)	time	gap (%)	time
20	5	-3.53	0.28	-1.23	0.28
20	10	-9.12	0.33	-4.48	0.33
20	15	-12.46	0.38	-5.49	0.38
20	20	-13.33	0.43	-6.69	0.43
50	5	-18.10	4.44	-14.70	4.80
50	10	-22.42	5.79	-20.18	5.91
50	15	-22.96	6.76	-21.76	6.84
50	20	-23.72	7.87	-21.71	8.02
Average		-15.70	3.29	-12.03	3.37

Table 8: Evaluation of the beam search method in 800 medium-sized instances for which an upper bound was computed using CPLEX.

and m machines, of the gap given by

$$100\% \left(\frac{\hat{f}^{\mathcal{I}}_{\text{BS}} - \hat{f}^{\mathcal{I}}_{\text{Chandra}}}{\hat{f}^{\mathcal{I}}_{\text{Chandra}}} \right),$$

where $\hat{f}^{\mathcal{I}}_{\text{BS}}$ and $\hat{f}^{\mathcal{I}}_{\text{Chandra}}$ are the values of the objective function (1) associated with the solution computed by the beam search method being introduced in the present work and the tabu search method introduced in [8], respectively, when applied to instance $\mathcal{I}$. The table shows average gaps equal to -3.73% and -1.97% for the instances in Sets d' and d'', respectively. This means that the beam search method improves the solutions found by the tabu search method in 3.73% and 1.97% in average for the instances in Sets d' and d'', respectively. Note also that, in general, the larger the values of n and m, the larger the gaps. The table also shows, in columns "W", "T", and "L", for each subset of 50 instances, the number of times the solution found by the beam search method was better than (win), equal to (tie), and worst than (loss) the solution found by the tabu search method. The CPU time in seconds required by the beam search method was already depicted in Table 4. The solutions obtained by the tabu search method were taken from `http://home.iitk.ac.in/~pmehta/flowshop.htm`. Since the CPU time of the tabu search method when applied to the considered instances was not reported in [8], the present comparison is limited to the quality of the solutions obtained by the methods and a comparison of the corresponding efforts is not possible.

6 Conclusions

The minimization of the earliness and tardiness penalties in a flowshop environment with a common due date is a problem that has already been addressed in the literature. As a byproduct of the minimization of the deviation from the common due date, many methods deliver a compact scheduling. However, the minimization of the waiting times between the machines has never been

Size		Set d'				Set d''			
n	m	gap (%)	W	T	L	gap (%)	W	T	L
5	5	-0.86	23	27	0	-0.69	11	39	0
5	10	-1.02	21	29	0	-1.59	17	33	0
5	15	-1.02	18	32	0	-1.14	15	35	0
5	20	-0.93	20	30	0	-0.74	13	37	0
10	5	-0.72	35	7	8	-0.24	23	17	10
10	10	-1.45	39	3	8	-1.12	26	7	17
10	15	-1.41	36	5	9	-0.14	20	17	13
10	20	-1.16	32	5	13	0.18	19	8	23
20	5	-2.18	35	1	14	-0.68	29	0	21
20	10	-2.82	45	0	5	-0.45	29	0	21
20	15	-3.54	43	0	7	-0.53	31	0	19
20	20	-4.33	44	0	6	-1.01	32	0	18
50	5	-2.53	48	0	2	-0.96	35	0	15
50	10	-5.35	50	0	0	-2.86	40	0	10
50	15	-5.85	50	0	0	-3.79	41	0	9
50	20	-5.71	49	0	1	-3.51	42	0	8
80	5	-2.92	48	0	2	-1.42	36	0	14
80	10	-6.15	50	0	0	-4.02	42	0	8
80	15	-7.01	50	0	0	-3.88	42	0	8
80	20	-7.35	50	0	0	-4.51	43	0	7
100	5	-3.59	49	0	1	-1.40	34	0	16
100	10	-6.39	50	0	0	-3.42	41	0	9
100	15	-7.31	50	0	0	-4.64	43	0	7
100	20	-7.87	50	0	0	-4.84	44	0	6
Average		-3.73	41.04	5.79	3.17	-1.97	31.17	8.04	10.79

Table 9: Comparison against the results obtained in [8] for the problem of minimizing the sum of earliness and tardiness only.

formally included in the problem as it was done in the bilevel approach considered in the present work then formulated as an MILP problem. The model is based on precedence variables. While positional variable would also have been used (see [27]), the proposed model can easily incorporate different weights ω_{ik} to penalize the waiting time of each job i between each pair of consecutive machines k and $k+1$ ($i = 1, \ldots, n$, $k = 1, \ldots, m-1$). Since the weighted sum of waiting times may be more adequate to represent real situations, it might be a line for future research. In addition to the MILP formulation of the problem, a list scheduling and its natural extension as a filtered beam search algorithm were introduced. The beam search algorithm is deterministic, has finite termination, and it has two main parameters only. Moreover, its computational cost can be easily adjusted varying these two parameters. In the present work, the introduced methods were meticulously described, including tie-breaking rules, in order to allow reproducibility. Perhaps, one

of the most important contributions of the present work is to show that a simple method with several positive characteristics can be successfully used to address a difficult problem like the one being considered in the present work. The minimization of the earliness and tardiness penalties in a *no-wait* flowshop environment with a common due date is a problem closely related to the problem considered in the present work, in which there is no no-wait constraint but the sum of the waiting times of the jobs between the machines is minimized as a secondary objective. Adapting the introduced methods to tackle this related problem might be a line for future research.

References

[1] R. Andrade, E. G. Birgin, and R. Morabito, Two-stage two-dimensional guillotine cutting stock problems with usable leftovers, *International Transactions in Operational Research* 23, pp. 121–145, 2016.

[2] S. Arabameri and N. Salmasi, Minimization of weighted earliness and tardiness for no-wait sequence-dependent setup times flowshop scheduling problem, *Computers & Industrial Engineering* 64, pp. 902–916, 2013.

[3] K. R. Baker and G. D. Scudder, Sequencing with earliness and tardiness penalties: a review, *Operations Research* 38, 22–36, 1990.

[4] E. G. Birgin, J. E. Ferreira, and D. P. Ronconi, List scheduling and beam search methods for an extended version of the flexible job shop scheduling problem, *European Journal of Operational Research* 247, pp. 421–440, 2015.

[5] E. G. Birgin, O. C. Romão, and D. P. Ronconi, The multiperiod two-dimensional non-guillotine cutting stock problem with usable leftovers, *International Transactions in Operational Research*, to appear (DOI: 10.1111/itor.12648).

[6] D. Biskup and M. Feldmann, Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates, *Computers & Operations Research* 28, pp. 787–801, 2001.

[7] K. Bülbül, P. Kaminsky, and C. Yano, Flow shop scheduling with earliness, tardiness, and intermediate inventory holding costs, *Naval Research Logistics* 51, pp. 407–445, 2004.

[8] P. Chandra, P. Mehta, and D. Tirupati, Permutation flow shop scheduling with earliness and tardiness penalties, *International Journal of Production Research* 47, pp. 5591–5610, 2009.

[9] S. Dempe, *Foundations of Bilevel Programming*, Kluwer Academic Publishers, The Netherlands, 2002.

[10] M. Feldmann and D. Biskup, Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristics approaches, *Computers & Industrial Engineering* 44, pp. 307–323, 2003.

[11] V. Fernandez-Viagas and J. M. Framinan, A beam-search-based constructive heuristic for the PFSP to minimise total flowtime, *Computers & Operations Research* 81, pp. 167–177, 2017.

[12] V. Gordon, J. Proth, and C. Chu, A survey of the state-of-the-art of common due date assignment and scheduling research, *European Journal of Operational Research* 139, pp. 1–25, 2002.

[13] N. G. Hall, W. Kubiak, and S. P. Sethi, Earliness-tardiness scheduling problems II: Deviation of completion times about a restrictive common due date, *Operations Research* 39, pp. 847–856, 1991.

[14] N. Hosseini and R. Tavakkoli-Moghaddam, Two meta-heuristics for solving a new two-machine flowshop scheduling problem with the learning effect and dynamic arrivals, *The International Journal of Advanced Manufacturing Technology* 65, pp. 771–786, 2013.

[15] M. C. Işler, B. Toklu, and V. Çelik, Scheduling in a two-machine flow-shop for earliness/tardiness under learning effect, *The International Journal of Advanced Manufacturing Technology* 61, pp. 1129–1137, 2012.

[16] Y. D. Kim, Minimizing total tardiness in permutation flowshops, *European Journal of Operational Research* 85, pp. 541–555, 1985.

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

[18] K. Miettinen, *Nonlinear Multiobjective Optimization*, Kluwer Academic Publishers, Boston, London, Dordrecht, 1998.

[19] V. Lauff and F. Werner, On the complexity and some properties of multi-stage scheduling problems with earliness and tardiness penalties, *Computers & Operations Research* 31, pp. 317–345, 2004.

[20] V. Lauff and F. Werner, Scheduling with common due date, earliness and tardiness penalties for multimachine problems: a survey, *Mathematical and Computer Modelling* 40, pp. 637–655, 2004.

[21] V. Lauff and F. Werner, Heuristics for two-machine flow shop problems with earliness and tardiness penalties, *International Journal of Operations and Quantitative Management* 10, pp. 125–144, 2004.

[22] 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.

[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] A. Mishra and D. Shrivastava, A TLBO and a Jaya heuristics for permutation flow shop scheduling to minimize the sum of inventory holding and batch delay costs, *Computers & Industrial Engineering* 124, pp. 509–522, 2018.

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

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

[27] D. P. Ronconi and E. G. Birgin, Mixed-integer programming models for flowshop scheduling problems minimizing the total earliness and tardiness, in *Just-in-Time Systems*, R. Z. Ríos-Mercado and Y. A. Ríos-Solís (Eds.), *Springer Series on Optimization and its Applications* 60, 2012, pp. 91–105.

[28] D. P. Ronconi and L. R. S. Henriques, Some heuristic algorithms for total tardiness minimization in a flowshop with blocking, *Omega* 37, pp. 272–281, 2009.

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

[30] C. S. Sakuraba, D. P. Ronconi, and F. Sourd, Scheduling in a two-machine flowshop for the minimization of the mean absolute deviation from a common due date, *Computers & Operations Research* 36, pp. 60–72, 2009.

[31] H. Sarper, Minimizing the sum of absolute deviations about a common due date for the two-machine flow shop problem, *Applied Mathematical Modelling* 19, pp. 153–161, 1995.

[32] T. Sen and S. K. Gupta, A state-of-art survey of scheduling research involving due dates, *Omega* 12, pp. 63–76, 1984.

[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] X. Y. Sun, X. N. Geng, J. B. Wang, and F. Liu, Convex resource allocation scheduling in the no-wait flowshop with common flow allowance and learning effect, *International Journal of Production Research* 57, pp. 1873–1891, 2019.

[36] C. S. Sung and J. I. Min, Scheduling in a two-machine flowshop with batch processing machine(s) for earliness/tardiness measure under a common due date, *European Journal of Operational Research* 131, pp. 95–106, 2001.

[37] E. Taillard, Benchmarks for basic scheduling problems, *European Journal of Operational Research* 64, pp. 278–285, 1993.

[38] W. K. Yeung, C. Oguz, and T. C. E. Cheng, Two-stage flowshop earliness and tardiness machine scheduling involving a common due window, *International Journal of Production Economics* 90, pp. 421–434, 2004.

[39] M. M. Yenisey and B. Yagmahan, Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends, *Omega* 45, pp. 119–135, 2014.