Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 273567, 14 pages
http://dx.doi.org/10.1155/2014/273567

Research Article

The Nonpermutation Flowshop Scheduling Problem: Adjustment and Bounding Procedures

Anis Gharbi, Mohamed Labidi, and Mohamed Aly Louly

Department of Industrial Engineering, College of Engineering, King Saud University, P.O. Box 800, Riyadh 11421, Saudi Arabia

Correspondence should be addressed to Anis Gharbi; a.gharbi@ksu.edu.sa

Received 13 April 2014; Revised 8 October 2014; Accepted 14 October 2014; Published 26 November 2014

Academic Editor: Yuri N. Sotskov

We consider the makespan minimization in a flowshop environment where the job sequence does not have to be the same for all the machines. Contrarily to the classical permutation flowshop scheduling problem, this strongly NP-hard problem received very scant attention in the literature. In this paper, some improved single-machine-based adjustment procedures are proposed, and a new two-machine-based one is introduced. Based on these adjustments, new lower and upper bounding schemes are derived. Our experimental analysis shows that the proposed procedures provide promising results.

1. Introduction

In this paper, we focus on the following scheduling problem: a set of n jobs $(1, \ldots, n)$ has to be processed on a set of m machines $M_1, M_2, \ldots, M_m$ in that order. That is, each job has to be processed first on machine M_1, then on machine M_2, and so on until performing its last operation on machine M_m. Each operation O_{ij} $(i = 1, \ldots, m$ and $j = 1, \ldots, n)$ requires an integer and deterministic processing time p_{ij}. The objective is to find a feasible schedule which minimizes the makespan. We also make the following common assumptions.

(i) Each job can be processed at most on one machine at the same time.

(ii) Each machine can process only one job at a time.

(iii) No preemption is allowed; that is, the processing of an operation cannot be interrupted.

(iv) All jobs are independent and are available for processing at time zero.

(v) The machines are continuously available.

In studying flowshop scheduling problems, it is usually assumed that the sequence in which each machine processes the jobs is identical on all machines. A sequence of this type is called a *permutation* sequence. Almost all of the research has been focused on the development of procedures to obtain permutation schedules. The main reason is probably that, in the general case involving m machines and n jobs, the total number of feasible schedules tends to $(n!)^m$, whereas with the assumption of no job passing, the number of feasible solutions is reduced to $n!$. However, identifying the best permutation schedule itself becomes very difficult, as the problem size grows bigger. Obviously, finding an optimal solution when sequence changes are permitted is more complex. Figures 1 and 2 depict the optimal solutions of the permutation and the nonpermutation flowshop instance with 3 jobs and 4 machines whose data are provided in Table 1.

Several authors emphasized the worth of considering nonpermutation schedules in real life flowshop environments [1, 2]. In particular, Potts et al. [3] exhibit a family of instances for which the value of the optimal permutation schedule is worse than that of the optimal nonpermutation schedule by a factor of more than $(1/2)\lceil\sqrt{m} + 1/2\rceil$. This is a nonnegligible gap since it can reach 50% for a 4-machine flowshop instance. Also, Sviridenko [4] proposed a new approximation algorithm which delivers a permutation schedule with makespan at most $O(\sqrt{m \log m})$ times of the optimal nonpermutation schedule. In this paper, we consider the nonpermutation case where the job sequence is not necessarily identical on all the machines. Using the notation of Graham et al. [5], this problem is denoted by $F||C_{\max}$.

Although the $F||C_{\max}$ is solvable to optimality in polynomial time when $m = 2$ [6], it is known to be $\mathcal{NP}$-hard in the strong sense when $m \geq 3$ [7]. The $F||C_{\max}$

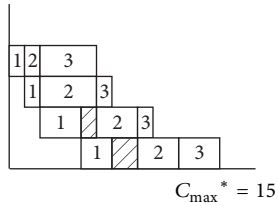

Figure 1: Optimal permutation schedule.

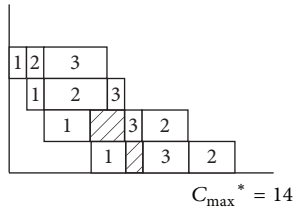

Figure 2: Optimal nonpermutation schedule.

Table 1: Data of a 3-job-4-machine flowshop instance.

	M_1	M_2	M_3	M_4
1	1	1	3	2
2	1	4	3	3
3	4	1	1	3

has several interesting practical applications since in several manufacturing environments (such as in glass industry, textile, and microelectronic chip), jobs have unidirectional flow with identical flow pattern.

The paper is organized as follows. In Section 2, we provide a literature survey of the nonpermutation flowshop as well as two closely related problems: the permutation flowshop and the jobshop scheduling problems. In Section 3, we recall several adjustment procedures from the literature and we investigate new ones. The main purpose of adjustments is to reduce the time windows of operations, mainly to achieve a better efficiency for enumerative approaches. New bounding strategies based on these adjustments are introduced in Section 4. Our experimental results are presented in Section 5. Finally, we conclude our paper by providing a synthesis of our research and indicating some directions for future research.

2. Literature Survey on Shop Scheduling Problems

In this section, we provide a brief overview of the existing research on the nonpermutation flowshop scheduling problem together with two closely related and much more studied problems: the permutation flowshop and the jobshop scheduling problems. Furthermore, we emphasize the most important relationships between the permutation and the nonpermutation flowshop scheduling problems.

2.1. The Flowshop Scheduling Problem. Since the nonpermutation flowshop scheduling problem received very scant attention in the literature, we devote this section to the case where we have identical job sequence on each machine. This problem is commonly referred to as $F_m|\text{prmu}|C_{\max}$.

Johnson [6] demonstrated that the two-machine problem can be solved in $O(n \log n)$ time by the following sequencing rule. First schedule the jobs with $p_{1j} \leq p_{2j}$ in order of nondecreasing p_{1j}; then schedule the remaining jobs in order of nonincreasing p_{2j}. Unfortunately, the problem is no longer polynomial for larger values of m. Indeed, Garey et al. [7] showed that $F_3|\text{prmu}|C_{\max}$ is strongly $\mathcal{NP}$-hard.

The $F|\text{prmu}|C_{\max}$ has received an impressive interest by scheduling researchers. Indeed, the first proposed branch-and-bound algorithms have been developed by Ignall and Schrage [8] and Lomnicki [9]. In addition, several exact approaches have been devised by Grabowski [10], Carlier and Rebaï [11], and Cheng et al. [12]. All these algorithms, except the latter, can solve only instances of very limited size. Also, an effective branch-and-bound algorithm is proposed by Haouari and Ladhari [13].

Many algorithms have been designed to find near optimal schedule in reasonable time. These algorithms can be classified as either constructive or based on local search. There are several constructive heuristics available (for a comprehensive review, see [14]). Nawaz et al.'s algorithm [15] (NEH) is currently considered as the most efficient constructive heuristic among others for the permutation flowshop problem. In addition, a new local search paradigm based on a truncated branch-and-bound strategy, and called *branch-and-bound-based local search*, has been implemented for the $F|\text{prmu}|C_{\max}$ by Haouari and Ladhari [16] and shown to yield approximate solutions of excellent quality.

The permutation flowshop problem has been tackled by several metaheuristics like simulated annealing (SA) (see [17]) and tabu search (see [18, 19]).

2.2. The Jobshop Scheduling Problem. In jobshop environment, each job has its own routing on machines. The jobshop problem, denoted by $J||C_{\max}$, is not only $\mathcal{NP}$-hard, but even among the members of this class, it belongs to the worst in practice. A notorious problem with 10 jobs and 10 machines given by Fisher and Thompson [20] remained unsolved for more than 25 years. Interested readers can find various jobshop-related complexity results in Brucker et al. [21–23].

Several exact approaches have been proposed for the jobshop problem. In particular, an interesting branch-and-bound algorithm was proposed by Carlier and Pinson [24]. It was the first algorithm to solve optimally the notorious 10-job-10-machine jobshop problem of Fisher and Thompson [20]. This algorithm has been followed by Brucker et al. [25] and Brinkkötter and Brucker [26]. The efficiency of these algorithms relies on the concept of immediate selections leading to effective adjustments.

Among the heuristics, one of the most successful approaches is the shifting bottleneck proposed by Adams et al. [27]. Numerous enhancements of this method have been proposed by Dauzere-Peres and Lasserre [28], Balas et al. [29], and Wenqi and Aihua [30].

Several tabu search approaches have been designed for the jobshop problem such as Nowicki and Smutnicki [19] and Armentano and Scrich [31]. Nowicki and Smutnicki [32] presented a new algorithm for the jobshop problem using tabu search. The computational experiments showed that the algorithm not only finds shorter makespan than the best approximation approaches, but also runs in shorter time.

2.3. Permutation versus Nonpermutation Flowshop Scheduling Problems. In this section, we recall the main relationships and differences between permutation and nonpermutation flowshops. First, it is worth noting that both problems are symmetric. That is, the value of the optimal makespan of the original problem (denoted by the forward problem) is equal to that of its symmetric problem (denoted by the backward problem) that is obtained by reversing the routing of the jobs; that is, the jobs are first processed on M_m, then on M_{m-1}, and so on. Now, we state the following property which is proposed by Conway et al. [33].

Property 1. For $F||C_{\max}$, there exists an optimal solution having the same processing permutation on the first two machines.

To see Property 1, consider any solution where different orders exist on the first two machines. Then, there must be a pair of adjacent jobs, say a and b, on the first machine permutation that appear in reverse order in the permutation on the second. But these two jobs can be reversed on the first machine without increasing the start time (and thus the completion time) of any job on the second machine. Inductively, we can repeat this pairwise switching process until the permutation on the first machine is made to agree with the (original) order on the second. An immediate consequence of Property 1 is that the $F2||C_{\max}$ and the $F2|\text{perm}|C_{\max}$ are equivalent. That is, in the case of two-machine flowshop, there exists an optimal schedule which is a permutation. Consequently, the $F2||C_{\max}$ is solvable in polynomial time using Johnson's algorithm [6]. Moreover, due to the symmetry of the $F||C_{\max}$, the following property holds.

Property 2. For $F||C_{\max}$, there exists an optimal solution having the same processing permutation on the last two machines.

According to the two above properties, one can deduce that there is an optimal solution which has the same permutation on the machines for the problem $F3||C_{\max}$. In other terms, the $F3||C_{\max}$ and $F3|\text{perm}|C_{\max}$ are equivalent.

Garey et al. [7] showed that the problems of nonpermutation and permutation flowshop become $\mathcal{NP}$-hard in the strong sense when the number of machines is larger than 3. However, there are several polynomially solvable special cases of $F_m||C_{\max}$ that result from imposing certain inequalities on the processing times. For instance, Johnson [6] observed that if $\max_j p_{2j} \leq \max\{\min_j p_{1j}, \min_j p_{3j}\}$ holds in an $F3||C_{\max}$ instance, then the second machine is nonbottleneck, and the optimal algorithm for problem $F2||C_{\max}$ can be suitably adapted. Monma and Rinnooy Kan [34] provide a survey of these types of results.

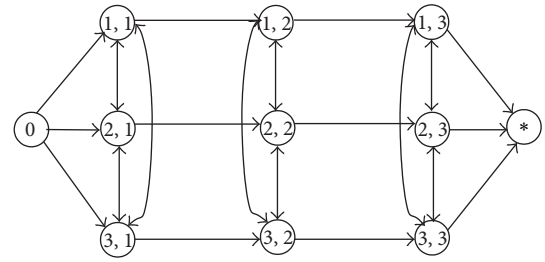

Figure 3: Disjunctive graph of a 3×3 flowshop.

Now, consider the two-machine flowshop and assume that a time delay (or time lag) is incurred by the transfer of the jobs from the first machine to the second one. The obtained problem, denoted by $F2|l_j|C_{\max}$, is proven to be solvable in $O(n \log n)$ time in the permutation case [35], whereas it turns to be $\mathcal{NP}$-hard in the general case [36]. Dell'Amico [37] proved that the nonpermutation case remains $\mathcal{NP}$-hard even if preemption is allowed. It is worth noting that Rebaine [38] evaluated the worst-case performance ratio between the optimal permutation and nonpermutation schedule. He observed that, even in the restricted case of two machines and unit execution time operations, the two models may generate different optimal values for the makespan. More specifically, he showed that, in the two-machine case, the performance ratio between the two optimal solutions is bounded by 2. When the operations of the n jobs are restricted to be unit execution time, this ratio is reduced to $2 - (3/(n+2))$ for the two-machine case and is m for the m-machine case.

If we consider the case where $n = 2$, the optimal permutation schedule is trivially computed by taking the best permutation among the two possible ones. The problem turns to be slightly more complicated but is still polynomially solvable, if a nonpermutation optimal schedule is to be determined. Indeed, Benson and Dror [39] proved that the two-job nonpermutation flowshop is solved in $O(n)$ time.

3. Adjustment Procedures

The nonpermutation flowshop problem may be represented using a disjunctive graph: two operations i and j, executed by the same machine, cannot be simultaneously processed. Therefore with each pair (i, j) of operations, we associate a pair of disjunctive arcs $[i, j] = \{(i, j), (j, i)\}$. The problem is then modelled by a disjunctive graph $\mathcal{G} = (\mathcal{C}, \mathcal{D})$, where $\mathcal{C} = (X, U)$ is a conjunctive graph (two operations belonging to the same job are represented by a conjunctive arc) and $\mathcal{D}$ is a set of disjunctions.

Figure 3 depicts a disjunctive graph of a 3-job-3-machine instance. The pair (a, b) denotes the processing of job a on machine b. Two dummy operations 0 and $*$ are included which represent the source and the sink node, respectively. The minimum starting time denoted by r_j of an operation j is equal to the longest weighted path from 0 to j in the disjunctive graph G. Similarly for each operation j the tail q_j is the length of the longest weighted path from j to $*$ in G.

In the sequel, we assume that the makespan is fixed to a trial value C. The problem amounts to checking whether a feasible schedule with makespan less than or equal to C exists. For that purpose, a deadline $d_j = C - q_j$ is associated with each operation j.

The main purpose of adjustments is to reduce the time windows $[r_j, d_j]$. This kind of elimination rule has been widely studied over the last two decades, especially for solving jobshop scheduling problems. The importance of the adjustment rules is twofold. They are used in the branch-and-bound algorithm for discarding infeasible nodes, and they permit the adjustment of the release dates and deadlines, so that the lower bounds are tightened. The major breakthrough of adjustment procedures has been achieved by Carlier and Pinson [24] who solved the famous Muth and Thompson 10 × 10 jobshop instance for the first time.

A nonpermutation flowshop can be viewed as a particular case of the jobshop scheduling problem. Therefore, we can apply the same adjustment procedures of this last problem to the nonpermutation flowshop. Interestingly, if an adjustment of an operation is performed, then the sets of its predecessors (Pred) and successors (Succ) according to the conjunctive graph can be adjusted using the following global adjustment algorithm (Algorithm 1; $\overline{r_j}$ and $\overline{d_j}$ denote the adjusted values of r_j and d_j, resp.).

3.1. Adjustments from the Literature. In this section, we describe the most relevant adjustment procedures proposed in the literature.

3.1.1. Disjunction-Based Adjustment. Clearly, if two operations i and j are such that $r_j + p_j + p_i > d_i$ then i cannot be scheduled after j. In this case, a disjunctive arc (i, j) can be fixed, which means that i is processed before j in any feasible schedule [24]. Therefore, the release date of job j can be adjusted to $\overline{r}_j = \max(r_j, r_i + p_i)$. Similarly, the deadline of job i can be adjusted to $\overline{d}_i = \min(d_i, d_j - p_j)$. More generally, after determining all the disjunctive arcs using the above rule, the release date and the deadline of each job j can be adjusted to $\overline{r}_j = \max_{i \in \mathrm{pred}(j)}(r_j, r_i + p_i)$ and $\overline{d}_i = \min_{j \in \mathrm{succ}(i)}(d_i, d_j - p_j)$, where pred($j$) and succ($i$) denote the sets of predecessors of j and the sets of successors of i, respectively. Obviously, the instance is infeasible if $\overline{r}_j + p_j > \overline{d}_j$ or the obtained graph contains a cycle. Carlier and Pinson [40] proposed an algorithm allowing the determination of all immediate selections in a disjunctive graph in $O(n \log n)$ steps.

Example 1. Consider the 5-job instance defined by Table 2. We have $d_3 - p_3 = 29 < r_1 + p_1 = 30$. Therefore, a disjunctive arc (3, 1) should be added to the graph. Similarly, since $d_5 - p_5 = 11 < r_1 + p_1$, then job 5 should be processed before job 1 in any feasible schedule. By applying the same rule to each job pair, we obtain the graph depicted in Figure 4. Accordingly, the release dates can be adjusted as follows:

$$\begin{aligned}
\overline{r_1} &= \max\left(r_1, \overline{r_3} + p_3, \overline{r_5} + p_5\right) = 20,\\
\overline{r_2} &= \max\left(r_2, \overline{r_1} + p_1, \overline{r_3} + p_3, \overline{r_5} + p_5\right) = 35,\\
\overline{r_3} &= r_3 = 0,\\
\overline{r_4} &= \max\left(r_4, \overline{r_1} + p_1, \overline{r_3} + p_3, \overline{r_5} + p_5\right) = 35,\\
\overline{r_5} &= r_5 = 0.
\end{aligned} \tag{1}$$

Table 2: Data of the 5-job instance of Example 1.

Jobs	1	2	3	4	5
r_j	15	33	0	26	0
p_j	15	1	2	8	20
d_j	46	44	31	48	31

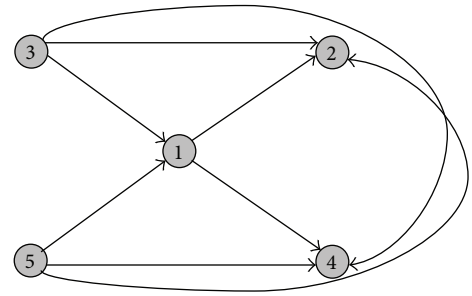

Figure 4: Disjunctive graph of the 5-job instance of Example 1.

The deadlines can be adjusted (decreased) in a similar way. The corresponding procedures are symmetric to those derived for the release dates. The procedure is reiterated until no adjustment can be performed.

Brucker et al. [41] proposed an extension of the procedure developed by Carlier and Pinson [24] in order to fix additional disjunctive arcs. This procedure referred to as "3-set condition" may be described as follows. Assume that all the arcs derived from Carlier and Pinson procedure are fixed. Suppose that $j \to i$. Let k be an operation such that $r_j + p_j + p_i + p_k > d_k$, $r_j + p_j + p_k + p_i > d_i$, and $r_k + p_k + p_j + p_i > d_i$; then k can be scheduled neither before j, nor between j and i, nor after i. Consequently, j cannot be processed before i in any feasible schedule, and a disjunctive arc (i, j) can therefore be fixed.

3.1.2. Preemption-Based Adjustment. Carlier and Pinson [42] presented an algorithm for adjusting release dates and deadlines in the jobshop problem. This algorithm is based on Jackson's preemptive schedule (JPS) for the one-machine problem. JPS provides the optimal solution for the $1|r_j, q_j, \mathrm{pmtn}|C_{\max}$ in $O(n \log n)$ time. It is constructed as follows. At the first moment t where the machine and at least one operation are available, the operation with the maximal tail is scheduled. This operation is processed either up to its completion or until a more urgent job (i.e., with larger tail) becomes available. Then, t is updated and the procedure is iterated until all operations are scheduled [43].

In the following, we describe the procedure of Carlier and Pinson [42] for adjusting the release date r_c of a given operation c. Let UB denote an upper bound of the optimal makespan and assume that Jackson's preemptive schedule has

(1) For each operation (i)
(2) Compute $\overline{r}_i$ and $\overline{d}_i$.
(3) If $(\overline{r}_i > r_i)$ then
(4) Let Succ$(i) = \{S_1, S_2, \ldots, S_k\}$ the set of successors of operation i.
(5) For $(h = 1, \ldots, k-1)$
(6) If $(\overline{r}_{S_h} + p_{S_h} > r_{S_{h+1}})$ then $\overline{r}_{S_{h+1}} = \overline{r}_{S_h} + p_{S_h}$.
(7) If $(\overline{r}_{S_h} + p_{S_h} > \overline{d}_{S_h})$ then the instance is infeasible.
(8) end (For)
(9) end (If)
(10) If $(\overline{d}_i < d_i)$ then
(11) Let Pred$(i) = \{R_1, R_2, \ldots, R_k\}$ the set of predecessors of operation i
(12) For $(h = k, \ldots, 2)$
(13) If $(\overline{d}_{R_h} - p_{R_h} < d_{R_{h-1}})$ then $\overline{d}_{R_{h-1}} = \overline{d}_{R_h} - p_{R_h}$.
(14) If $(\overline{r}_{R_h} + p_{R_h} > \overline{d}_{R_h})$ then the instance is infeasible.
(15) end (For)
(16) end (If)
(17) end (For)
(18) If for all operations we have $\overline{r}_i = r_i$ and $\overline{d}_i = d_i$ then STOP. Otherwise, set $r_i = \overline{r}_i$; $d_i = \overline{d}_i$ and go to line (1).

Algorithm 1: Global_Adjustment_Algorithm.

been built until time $t = r_c$. Let p_k^+ denote the remaining processing times in the preemptive schedule. Then, r_c can be adjusted as follows.

(i) Compute $K_c = \{j \in J \mid q_j \geq q_c\}$ and $K_c^+ = \{j \in K_c \mid p_j^+ > 0\}$.

(ii) Take the operations of k_c^+ in increasing order of q_j and find the first j_1 such that $r_c + p_c + \sum_{\{j \in k_c^+ \mid q_j \geq q_{j_1}\}} p_j^+ + q_{j_1} > UB$ (if any exists).

(iii) Define $K_c^* = \{j \in k_c^+ \mid q_j \geq q_{j_1}\}$.

(iv) Set $\overline{r_c} = \max_{j \in K_c^*} C_j$ (C_j is the completion time of job j).

Carlier and Pinson [42] presented an $O(n^2)$ algorithm for adjusting all release dates. Brucker et al. [25] improved this complexity by proposing an $O(\max\{n \log n, f\})$ algorithm, where f is the number of new disjunctive arcs. The idea relies on the dual version of JPS which, starting from $d = \max_{j \in J} d_j$, calculates a schedule from right to left by applying the following dual rule. At each time t which is given by a deadline or a release date of an operation, schedule backwards an incompletely scheduled operation j with $d_j \geq t$ and $r_j = \max_{i \in J}\{r_i \mid d_i \geq t\}$. Such a schedule is referred to as backwards Jackson preemptive schedule (BJPS). The release date of a given operation c can be adjusted using the following procedure.

Earliest_Possible_Completion_Algorithm

Step 1. Calculate JPS up to r_c.

Step 2. Calculate BJPS without c in $[r_c, d]$ using the remaining processing times.

Step 3. Schedule operation c from left to right within the idle periods of $[r_c, d]$. Let s_c be the completion time of operation c and set $\overline{r_c} = s_c - p_c$.

Similarly, the deadline of a given operation c can be adjusted by applying the dual version of the above algorithm.

Latest_Possible_Completion_Algorithm

Step 1. Calculate BJPS up to d_c.

Step 2. Calculate JPS without c in $[\min_{j \in J} r_j, d_c]$ using the remaining processing times.

Step 3. Schedule operation c from right to left within the idle periods of $[\min_{j \in J} r_j, d_c]$. Let s_c' be the starting time of operation c and set $\overline{d_c} = s_c' + p_c$.

Example 1 (continued). Figures 5 and 6 depict the preemption-based adjustment of the release date and the deadline of job 1, respectively. We have $\overline{r_1} = s_1 - p_1 = 37 - 15 = 22$ and $\overline{d_1} = s_1' + p_1 = 24 + 15 = 39$.

3.2. New Adjustment Procedures. In this section, we introduce new adjustment procedures and show that they outperform those described in the previous section.

3.2.1. Improved Disjunction-Based Procedure. Recall that the disjunction-based adjustment rule consists in setting $\overline{r}_j = \max_{i \in \text{pred}(j)}(r_j, r_i + p_i)$ and $\overline{d}_j = \min_{i \in \text{succ}(j)}(d_j, d_i - p_i)$ for all $j \in J$. Interestingly, this adjustment rule can be improved in the following way.

(i) A job i is defined as an immediate predecessor of j if there exists an arc (i, j) in the disjunctive graph.

(ii) A job i is defined as a predecessor of j if there exists a path $(i, \ldots, j)$ in the disjunctive graph.

Let P_j denote the set of all the predecessors of j. Clearly, job j has to wait until all the jobs of P_j have been processed.

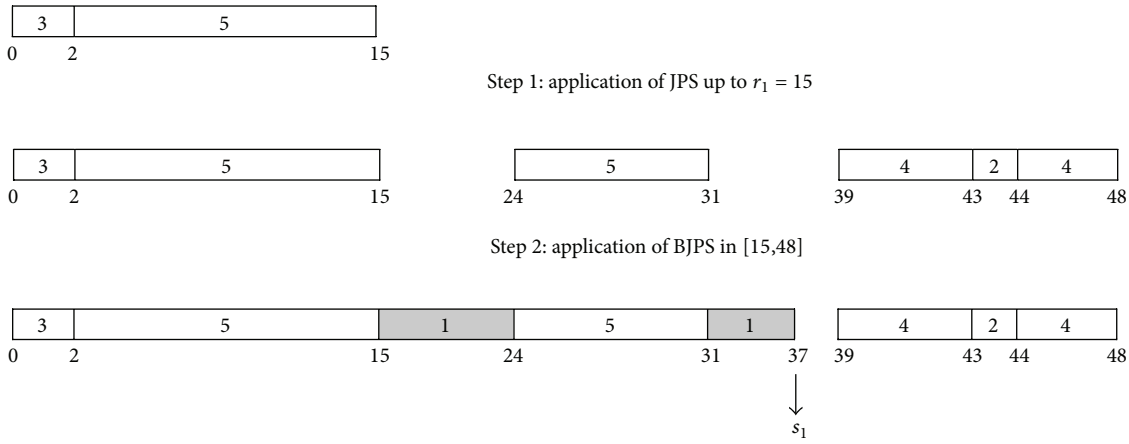

FIGURE 5: Preemption-based adjustment of r_1 (Example 1).

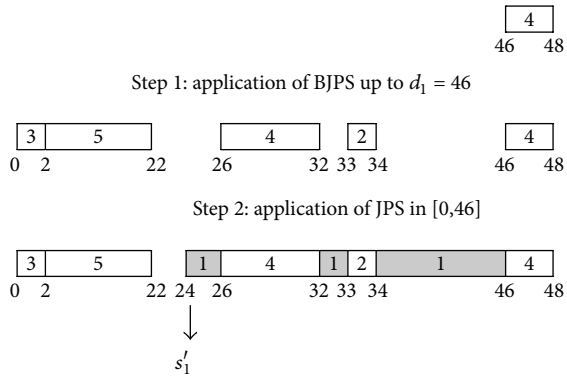

FIGURE 6: Preemption-based adjustment of d_1 (Example 1).

Let $C_{\max}(P_j)$ denote the minimum completion time of all jobs of P_j. Therefore, the minimum starting time of j is at least equal to $\max(r_j, C_{\max}(P_j))$. It is worth noting that $C_{\max}(P_j)$ corresponds to the optimal makespan of the $1|r_j|C_{\max}$ problem defined on P_j. The $1|r_j|C_{\max}$ is solvable in $O(n \log n)$ time by ranking jobs in nondecreasing order of their release dates. Clearly, since $C_{\max}(P_j) > \max_{i \in P_j}(r_i + p_i)$, then this adjustment rule dominates the classical one proposed by Carlier and Pinson [24].

Similarly, the deadline of job j can be adjusted by setting $\overline{d}_j = \min(d_j, \max_{i \in S_j} d_i - C_{\max}(S_j))$, where S_j denotes the set of all the successors of j, and $C_{\max}(S_j)$ is the optimal makespan of $1|r_j|C_{\max}$ problem defined on S_j by setting $r_j = \max_{i \in S_j} d_i - d_j$.

3.2.2. Semipreemption-Based Procedure. In this section, we develop a new adjustment rule which dominates the preemptive version while having the same complexity. The proposed procedure is similar, in spirit, to JPS in the sense that we apply the same algorithm with a small distinction. The idea of semipreemption has been introduced in Haouari and Gharbi [44].

TABLE 3: Data of the 5-job instance of Example 2.

Jobs	1	2	3	4	5
r_j	5	2	4	14	1
p_j	11	2	11	6	14
d_j	50	49	41	46	27

First, by remarking that the latest starting time of any job $j \in J$ is $d_j - p_j$, and its earliest finishing time is $r_j + p_j$, we can state the following observation.

Observation 1. Assume that there exists a job j such that $r_j + p_j > d_j - p_j$. Then, in any nonpreemptive schedule, there is a part of job j which must be processed during the interval $[d_j - p_j, r_j + p_j]$.

According to the above observation, each job j satisfying $r_j + p_j > d_j - p_j$ is composed of a fixed and a free part. Its fixed part is the amount of time $2p_j - (d_j - r_j)$ which must be processed in $[d_j - p_j, r_j + p_j]$, and its free part is the amount of time $p_j' = d_j - (r_j + p_j)$ which has to be processed in $[r_j, d_j - p_j] \cup [r_j + p_j, d_j]$. The other jobs are composed only of a free processing part $p_j' = p_j$ which has to be processed in $[r_j, d_j]$. Let Jackson's semipreemptive schedule (JSPS) denote Jackson's preemptive schedule applied on the modified instance where each job is replaced by two jobs which designate its free part and fixed part, respectively. Clearly, the preemption-based adjustment is improved if JPS is replaced by JSPS.

Example 2. Consider the 5-job instance defined by Table 3.

We remark that $r_5 + p_5 > d_5 - p_5$. Then, job 5 has a fixed part $(5')$ in $[13, 15]$ with processing time equal to 2 and a free part in $[1, 27]$ with processing time equal to 12. The modified instance is depicted in Table 4.

Figure 7 displays the semipreemption-based adjustment of the release date of job 1. We have $\overline{r}_1 = s_1 - p_1 = 18 - 11 = 7$.

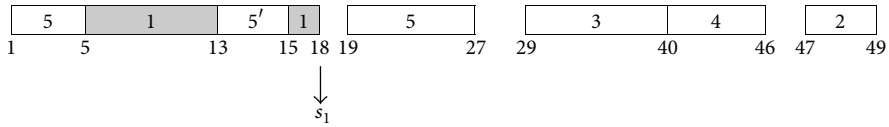

Figure 7: Semipreemption-based adjustment of r_1 (Example 2).

Table 4: Modified instance of Example 2.

Jobs	1	2	3	4	5	5′
r_j	5	2	4	14	1	13
p_j	11	2	11	6	12	2
d_j	50	49	41	46	27	15

Table 5: Semipreemptive-based adjusted data of Example 2.

Jobs	1	2	3	4	5
$\overline{r}_j$	**15**	2	**15**	**15**	1
p_j	11	2	11	6	14
$\overline{d}_j$	50	49	41	46	**22**

The procedure is reiterated until no adjustment can be performed. The obtained adjusted release dates and deadlines are depicted in Table 5.

It is worth noting that no adjustment can be made on this instance using the classical preemptive-based procedure.

3.2.3. Two-Machine-Based Adjustment Procedure. First, we observe that there exists an optimal $F||C_{\max}$ schedule such that there is no idle time on the first machine. That is, no job finishes processing on M_1 later than $\sum p_{1j}$. Therefore, the deadlines on M_1 can be adjusted by setting $\overline{d}_{1j} = \min(d_{1j}, \sum p_{1j})$ for all $j \in J$. In a similar way, using the symmetry of the $F||C_{\max}$ the release dates on the last machine can be adjusted by setting $\overline{r}_{mj} = \max(r_{mj}, C - \sum p_{mj})$.

Now, we introduce a new approach for adjusting the deadlines on the second machine. Recall that there exists an optimal schedule such that the sequences on M_1 and M_2 are the same. Consider a particular job j^* and let σ^* denote the sequence (or permutation) of $J \setminus \{j^*\}$ which *maximizes* the makespan on M_2. Let C_{j^*} denote the completion time of j^* on M_2 and let $\overline{C}_{\max}(\sigma^*)$ denote an upper bound on the completion time of σ^* on M_2. Clearly, we have $C_{j^*} \leq \max\{\sum p_{1j}, \overline{C}_{\max}(\sigma^*)\} + p_{2j^*}$. Consequently, the deadline of j^* on machine M_2 can be adjusted to $\overline{d}_{2j^*} = \min\{d_{2j^*}, \max\{\sum p_{1j}, \overline{C}_{\max}(\sigma^*)\} + p_{2j^*}\}$.

In what follows, we show how to compute $\overline{C}_{\max}(\sigma^*)$. Let σ_{j_0} denote a permutation such that j_0 is the job that starts processing on M_2 after its last idle period (see Figure 8). Denote by t_{ij} and C_{ij} the starting time and the completion time of job j on machine M_i, respectively. Clearly, we have $C_{1j_0} = t_{2j_0}$.

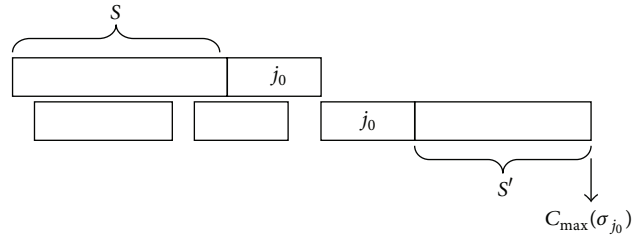

Figure 8: Illustration of the permutation σ_{j_0}.

Let S and S' denote the set of jobs that are scheduled before and after j_0 in σ_{j_0}, respectively. Denote by x_j the variable that equals 1 if $j \in S$ and 0 otherwise. We have

$$\begin{aligned} C_{\max}\left(\sigma_{j_0}\right) &= \sum_{j\in J\setminus\{j_0\}} p_{1j}x_j + p_{1j_0} + p_{2j_0} + \sum_{j\in J\setminus\{j_0\}} p_{2j}\left(1 - x_j\right) \\ &= \sum_{j\in J\setminus\{j_0\}} \left(p_{1j} - p_{2j}\right)x_j + p_{1j_0} + p_{2j_0} + \sum_{j\in J\setminus\{j_0\}} p_{2j}. \end{aligned} \tag{2}$$

Clearly $C_{\max}(\sigma_{j_0})$ is maximized by setting $x_j = 1$ for all j satisfying $p_{1j} > p_{2j}$ and $x_j = 0$ otherwise. Let $C_{\max}(\sigma^*_{j_0})$ denote the corresponding makespan and $J' = \{j \in J : p_{1j} > p_{2j}\}$. We have $C_{\max}(\sigma^*_{j_0}) = \sum_{j\in J'\setminus\{j_0\}}(p_{1j} - p_{2j}) + p_{1j_0} + p_{2j_0} + \sum_{j\in J\setminus\{j_0\}} p_{2j}$.

Two cases have to be considered.

(i) If $j_0 \in J'$ then

$$\begin{aligned} C_{\max}\left(\sigma^*_{j_0}\right) &= \sum_{j\in J'} p_{1j} - p_{1j_0} - \left(\sum_{j\in J'} p_{2j} - p_{2j_0}\right) \\ &\quad + p_{1j_0} + p_{2j_0} + \sum_{j\in J} p_{2j} - p_{2j_0} \\ &= \sum_{j\in J'} p_{1j} + \sum_{j\in J} p_{2j} - \sum_{j\in J'} p_{2j} + p_{2j_0} \\ &\leq \sum_{j\in J'} \left(p_{1j} - p_{2j}\right) + \sum_{j\in J} p_{2j} + \max_{j_0\in J'}\left\{p_{2j_0}\right\}. \end{aligned} \tag{3}$$

(ii) If $j_0 \notin J'$ then

$$\begin{aligned} C_{\max}\left(\sigma^*_{j_0}\right) &= \sum_{j\in J'} p_{1j} - \sum_{j\in J'} p_{2j} + p_{1j_0} + p_{2j_0} + \sum_{j\in J} p_{2j} - p_{2j_0} \\ &\leq \sum_{j\in J'} \left(p_{1j} - p_{2j}\right) + \sum_{j\in J} p_{2j} + \max_{j_0\notin J'}\left\{p_{1j_0}\right\}. \end{aligned} \tag{4}$$

Table 6: Data of 5-job-4-machine instance of Example 3.

	1	2	3	4	5
M_1	6	20	2	9	20
M_2	9	13	5	11	10
M_3	15	1	8	6	4
M_4	2	4	4	8	3

Table 7: Initial values of release dates of Example 3.

Jobs	1	2	3	4	5
M_1	0	0	0	0	0
M_2	6	20	2	9	20
M_3	15	33	7	20	30
M_4	30	34	15	26	34

Table 8: Initial values of deadlines of Example 3.

Jobs	1	2	3	4	5
M_1	59	67	68	60	62
M_2	68	80	73	71	78
M_3	83	81	81	77	82
M_4	85	85	85	85	85

Consequently, an upper bound $\overline{C}_{\max}(\sigma^*_{j_0})$ on $C_{\max}(\sigma^*_{j_0})$ is obtained by

$$\overline{C}_{\max}\left(\sigma^*_{j_0}\right) = \sum_{j\in J'}\left(p_{1j} - p_{2j}\right) + \sum_{j\in J} p_{2j} + \max\left\{\max_{j_0\notin J'}\left\{p_{1j_0}\right\}, \max_{j_0\in J'}\left\{p_{2j_0}\right\}\right\}. \tag{5}$$

Finally, $\overline{C}_{\max}(\sigma^*) = \max_{j_0\in J}\overline{C}_{\max}(\sigma^*_{j_0})$ is an upper bound on the completion time of all possible permutations on M_2. Using appropriate data structure, all the deadlines on M_2 can be adjusted in $O(n)$ time.

Thanks to the symmetry of the problem, the release dates on M_{m-1} can be adjusted by setting $\overline{r}_{m-1,j} = \max(r_{m-1,j}; C - \overline{C}_{\max}(\sigma^*))$, where $\overline{C}_{\max}(\sigma^*)$ is computed on the two-machine flowshop problem defined on M_m and M_{m-1}.

Example 3. Consider the 5-job-4-machine instance depicted in Table 6 and let $C = 85$. The values of the releases dates and the deadlines before performing any adjustment are depicted in Tables 7 and 8, respectively.

First, we have

$$\overline{d}_{1j} = \min\left(d_{1j}, \sum p_{1j}\right) = 57,$$
$$\overline{r}_{4j} = \max\left(r_{4j}, C - \sum p_{4j}\right) = 64 \quad \forall j \in J. \tag{6}$$

Now, we have $J' = \{2, 5\}$ and $\overline{C}_{\max}(\sigma^*) = 55$. Therefore, the deadline of job 2 on M_2 can be adjusted to

$$\overline{d}_{22} = \min\left\{d_{22}, \max\left\{\sum p_{1j}, \overline{C}_{\max}\left(\sigma^*\right)\right\} + p_{22}\right\} = \min\{80, \max\{57, 55\} + 13\} = 70. \tag{7}$$

Similarly, the deadline of job 5 on M_2 is adjusted to $\overline{d}_{25} = 68$ (instead of 78). The adjusted release dates on M_3 obtained by the two-machine-based procedure are the following:

$$\overline{r}_{31} = 40; \quad \overline{r}_{32} = 43; \quad \overline{r}_{33} = 40; \quad \overline{r}_{34} = 44, \quad \overline{r}_{35} = 40. \tag{8}$$

4. Bounding Procedures

In this section, we introduce lower and upper bounding procedures for the nonpermutation flowshop problem. These procedures are derived from the adjustments described in Section 3.

4.1. Lower Bounds. A simple and efficient way of deriving a good lower bound for the $F||C_{\max}$ consists in relaxing the capacities of all the machines but one denoted by M_k. The obtained problem is a one machine problem with release dates and delivery times denoted by $1|r_j, q_j|C_{\max}$, where $r_j = \sum_{i<k} p_{ij}$ and $q_j = \sum_{i>k} p_{ij}$.

Although the latter problem is strongly $\mathcal{NP}$-hard, it is efficiently solved using the branch-and-bound algorithm developed by Carlier [45]. Let $C^*_{\max}(M_k)$ denote the optimal makespan of the $1|r_j, q_j|C_{\max}$ problem defined on M_k. A valid lower bound for the $F||C_{\max}$ is

$$LB_0 = \max_{1\le k\le m} C^*_{\max}\left(M_k\right). \tag{9}$$

It is worth noting that LB_0 is considered as a good lower bound in the context of jobshop scheduling problem. Moreover, it constitutes a fundamental component in several effective heuristics, such as the well-known shifting bottleneck procedure [27]. In the sequel, we introduce an interesting way of deriving a stronger lower bound using the developed adjustment procedures.

Let UB denote an upper bound on the optimal makespan and let $C \in [LB_0, UB-1]$ denote a trial value of the makespan. With each operation O_{ij} is associated a release date r_{ij}, a delivery time q_{ij}, and a deadline $d_{ij} = C - q_{ij}$. Clearly, if after performing the adjustment procedure the instance is identified to be infeasible, then $C + 1$ is a valid lower bound on the optimal makespan. Consequently, a bisection search on $[LB_0, UB - 1]$ combined with the proposed adjustment procedure provides a lower bound denoted by LB_1 which dominates LB_0. The computation of LB_1 can be described as follows.

Compute LB_1

Step 1. Set $C^- = LB_0$, $C^+ = UB - 1$.

Step 2. Let $C = \lfloor (C^- + C^+)/2 \rfloor$.

Step 3. Set $d_j = C - q_j$ for all $j \in J$.

Step 4. Apply an adjustment procedure to the instance obtained in step 3. If the instance is infeasible, then set $C^- = C + 1$. Otherwise, set $C^+ = C - 1$.

Step 5. If $C^- = C^+$, then stop and set $LB_1 = C$. Else, go to step 2.

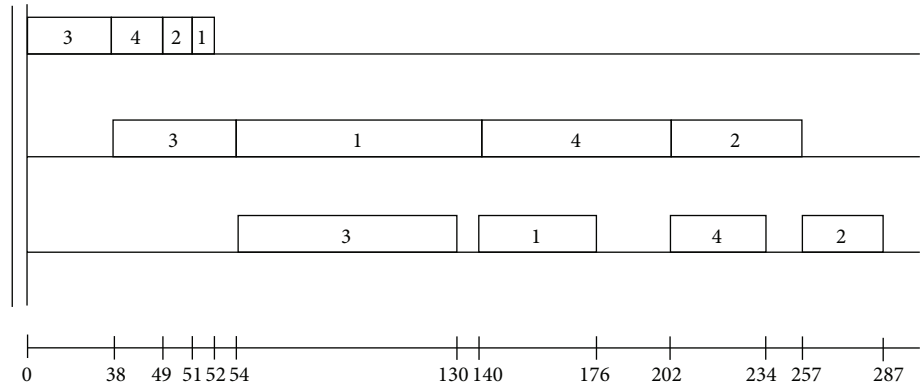

Figure 9: Schedule of Example 4 according to LPT rule.

In the computation of LB_1, we implemented the new proposed adjustment procedures. The value of UB is equal to the makespan of the schedule obtained by applying the longest processing time (LPT) dispatching rule. It consists in scheduling in the first available machine, the available operation with longest processing time.

4.2. Heuristics. In this section we describe how we can use the adjustment procedures in order to construct an upper bound for the nonpermutation flowshop scheduling problem.

4.2.1. Heuristic H1. We are interested in building a nonpreemptive schedule with makespan less than or equal to a trial value $C \in [LB, UB - 1]$. First, we set $d_{ij} = C - q_{ij}$ and we adjust the release dates and the deadlines using an adjustment procedure. An operation such that $d_{ij} = r_{ij} + p_{ij}$ is referred to as a fixed operation and is considered as already scheduled. Let L denote the list of the free (nonfixed) first operations of each job, sorted according to the nondecreasing order of their release dates. The ties are settled according to the nondecreasing order of deadlines, then by the nondecreasing order of processing times. At each iteration, we use an adjustment procedure to check whether the first operation O_{ij_0} can be scheduled at its release date. In this case, we set $d_{ij_0} = r_{ij_0} + p_{ij_0}$. The list L is then updated by the adjustment procedure.

Now, assume that the adjustment procedure yields an infeasibility. That is, scheduling j_0 at this position is not the right decision. Therefore, we have to skip operation O_{ij_0} and move to the next operation in the list. Note that in this case the minimum starting time of O_{ij_0} is $r_{ij_0} + 1$. Obviously, there may be no possible operation to be scheduled at the current iteration. In this case, finding a schedule with makespan less than or equal to the trial value C is assumed as impossible. So we have to move on to $C + 1$ and so on. The algorithm stops when a feasible schedule is constructed.

Example 4. Consider the 4-job-3-machine instance whose data are depicted in Table 9.

Assume that $LB = 252$ and $UB = 287$ (UB is a makespan obtained by LPT dispatching rule and it is depicted in Figure 9).

Table 9: Data of 4-job-3-machine instance of Example 4.

	M_1	M_2	M_3
1	1	86	36
2	2	55	30
3	38	16	76
4	11	62	32

Assume that we are interested in constructing a schedule with makespan equal to 254. The list L contains only the first free operations in each machine. Thus, $L = \{O_{11}, O_{12}, O_{14}, O_{13}\}$. At the first iteration, operation O_{11} is scheduled at time $r_{11} = 0$. That is, the deadline is set to $d_{11} = 1$. The current data is updated by applying the adjustment procedures. Thus, we have $L = \{O_{12}, O_{14}, O_{13}, O_{21}\}$. In the second iteration, the operation O_{12} is scheduled to finish at $d_{12} = 3$. Next, we apply the adjustment procedures. Thus, we have $L = \{O_{21}, O_{14}, O_{13}, O_{22}\}$. Assume that operation O_{21} is scheduled at $r_{21} = 1$ at the third iteration. That is, its deadline is set to $d_{21} = 87$.

Applying the adjustment procedures to the obtained instance yields an infeasibility. That is, the minimum starting time of O_{21} is equal to 2, and scheduling operation O_{21} at the third iteration is not the right choice. Consequently, we move to the next operation, and we have $L = \{O_{14}, O_{13}, O_{21}\}$ and so on. The obtained schedule is depicted in Figure 10. Its makespan is equal to 254.

4.2.2. Heuristic H2. Using the symmetry of the problem, a second heuristic can be obtained by applying $H1$ to the symmetric $F||C_{\max}$ instance (backward).

Example 4 (continued). Assume that $LB = 252$ and $UB = 254$. We are interested in constructing a schedule with makespan equal to 253. The list of the free jobs is $L = \{O_{11}, O_{12}, O_{14}, O_{13}\}$. At the first iteration, operation O_{11} is scheduled at time $r_{11} = 0$. That is, the deadline is set to $d_{11} = 36$. Applying the adjustment procedures to the obtained instance yields an infeasibility. That is, the minimum starting time of O_{11} is equal to 1. Consequently, scheduling

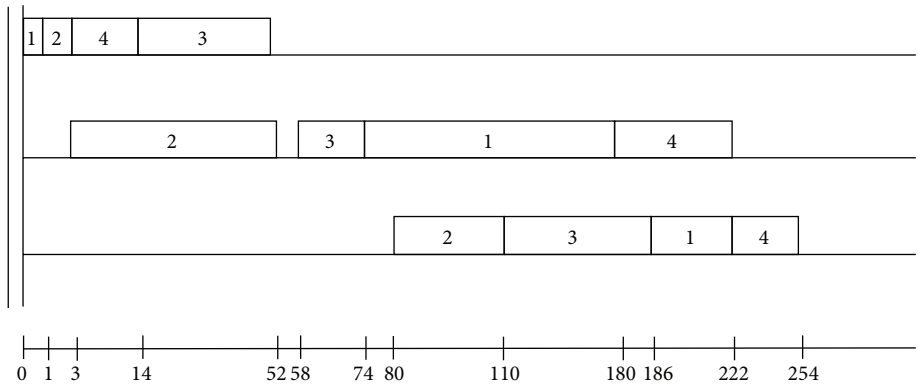

Figure 10: Schedule provided by $H1$ for Example 4.

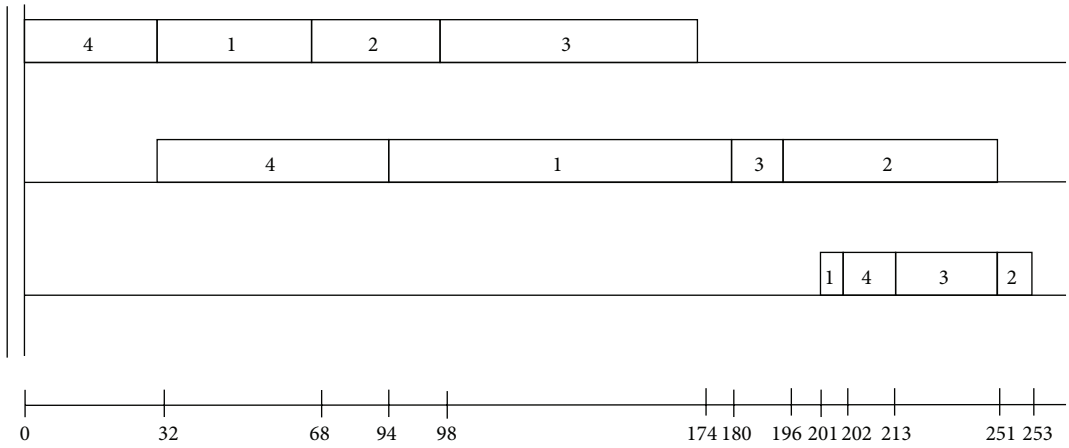

Figure 11: Schedule provided by $H2$ for Example 4.

operation O_{11} at the first iteration is not the right choice and we move to the next operation, and so on. The obtained schedule is depicted in Figure 11. Its makespan is equal to 253.

4.2.3. Heuristic H3. For each value of C, we first consider the forward instance and try to construct a feasible schedule. In case of failure, the backward instance is considered. We move on to $C+1$ only if a failure has been obtained for both forward and backward instance. Note that the obtained value of the makespan is equal to the minimum between those obtained by $H1$ and $H2$. However, our experimental results show that the required computational effort is substantially reduced.

4.2.4. Heuristic H4. In order to reduce the computational effort, a bisection search on the trial value C is embedded within $H3$. The value of C lies in the interval $[LB_1, UB]$ where UB denotes the makespan obtained by LPT dispatching rule. It is worth noting that if a failure is obtained for a given value C then all the values less than C will not be considered. However, it may be possible to obtain a feasible schedule if a smaller value of C is considered. Therefore, it is possible that $H4$ yields a solution which is worse than that obtained by $H3$. Our experimental results show that $H4$ often exhibits a good trade-off between the decrease of the computation time and the decrease of the solution's quality.

4.2.5. Heuristic H5. A randomization component is included in $H4$ by selecting the operation that has to be scheduled randomly between the two first operations in the list L. In case of failure for a given value of C, the randomized procedure is performed until a feasible schedule is obtained or the maximum number of iterations is reached. In our experiments, we fixed the maximum number of iterations to 40. Also, a CPU time limit of 600 seconds has been fixed for $H5$.

5. Computational Results

In this section, we present an empirical analysis of the performance of the proposed lower and upper bounds that are derived from our adjustment procedures. The algorithms were coded in C and compiled with Visual C++ 6.0. All the computational experiments were carried out on a Pentium IV 3.2 GHz Personal Computer with 1 GB RAM.

5.1. Test Generation. We carried out a series of experiments on 240 test problems that were randomly generated in the following way. The processing times are drawn from the

Table 10: Performance of the lower bounds for $m = 4$.

n	Opt_0	Imp	Red_{avg}	Red_{max}	Opt_1	Time
10	30	85.71	76.32	100	50	0.04
20	50	40	71.15	92.31	0	0.06
30	50	60	73.22	100	66.67	0.17
40	70	66.67	100	100	100	0.28
50	50	40	100	100	100	0.59
60	50	60	100	100	100	1.05
70	40	50	69.89	100	66.67	1.09
80	60	50	72.5	100	50	1.49

Table 11: Performance of the lower bounds for $m = 5$.

n	Opt_0	Imp	Red_{avg}	Red_{max}	Opt_1	Time
10	20	100	48.92	100	28.57	0.02
20	20	100	36.24	100	12.5	0.08
30	10	88.89	21.91	100	12.5	0.20
40	10	77.78	47.77	100	14.27	0.53
50	10	100	46.56	100	22.22	0.67
60	30	71.43	26.02	100	20	1.46
70	30	71.43	40.59	77.78	0	2.03
80	50	80	61.17	100	25	2.79

discrete uniform distribution on $[1, 100]$. The number of jobs n is taken equal to 10, 20, 30, 40, 50, 60, 70, and 80. The number of machines m is taken equal to 4, 5, and 6. We combined these problem characteristics to obtain 24 classes of instances. For each class, 10 instances are generated.

5.2. Performance of the Lower Bounds. In order to assess the impact of the proposed adjustment procedures in the computation of a lower bound, we performed a thorough comparison between the one-machine-based lower bound LB_0 and the adjustment-based lower bound LB_1. The objective of this analysis is to determine how many times LB_1 improved LB_0 and how important is this improvement. For that purpose, we first computed, for each class of instances, the percentage of times (Opt_0) where LB_0 provides the optimal makespan, that is, $LB_0 = UB$, where UB denotes the best value of the makespan provided by our five heuristics. Clearly, LB_1 is not computed for these instances since there is no room for improving LB_0. Then, we computed for the remaining instances the percentage of times (Imp) where LB_1 outperformed LB_0. The importance of this improvement is emphasized in the following way. For the instances where an improvement of LB_0 occurred, we compared the reduced gap $UB-LB_1$ with respect to the initial gap $UB-LB_0$ by computing the relative gap reduction defined as $100((LB_1 - LB_0)/(UB - LB_0))$. We reported the average gap reduction (Red_{avg}), the maximum gap reduction (Red_{max}), and the percentage of times (Opt_1) where the reduced gap reached 100%; that is, the optimal makespan is provided by LB_1. Finally, the average CPU time (in seconds) required by LB_1 is computed. At this point, it is worth noting that the CPU time of LB_0 is always less than 0.001 sec and has not been reported.

The results of our analysis are reported in Tables 10, 11, and 12. These tables provide strong evidence of the dominance of our adjustment-based lower bound. Indeed, we observe that, in all problem classes, LB_1 was able to improve LB_0. Moreover, this improvement often occurred in more than 70% of the cases (14 out of 24 classes) and reached 100% in some few cases. In addition, the average gap reduction is in most cases larger than 30% and the maximal gap reduction exceeds 70% in 87.5% of the problem classes (21 out of 24). We observe that this gap reduction is more important for smaller values of m. Indeed, for $m = 4$, the average gap reduction is always more than about 70% and the maximal gap reduction is equal to 100% in all problem classes except one. Furthermore, except for $m = 6$, the bound LB_1 was often able to provide the optimal makespan. Finally, we observe that the adjustment-based lower bound is very fast, since the average CPU time never exceeds 3.5 sec.

Table 12: Performance of the lower bounds for $m = 6$.

n	Opt_0	Imp	Red_{avg}	Red_{max}	Opt_1	Time
10	10	77.78	21.51	46.42	0	0.04
20	0	70	34.49	78.12	0	0.19
30	10	88.89	8.40	14.28	0	0.43
40	10	44.45	11.52	25.32	0	0.95
50	0	80	27.92	85.71	0	1.35
60	20	50	36.87	85	0	2.09
70	30	57.14	23.79	85.71	0	2.88
80	30	85.71	20.19	100	16.68	3.44

5.3. Performance of the Heuristics. The results of a comparison of our five heuristics are depicted in Tables 13, 14, and 15. For each heuristic, we provide the following.

(i) Gap: the average gap with respect to the lower bound LB_1, where the gap of the heuristic H_i ($i = 1, \ldots, 5$) with makespan UB_i is defined as $100((UB_i - LB_1)/LB_1)$.

(ii) Time: the average CPU time

(iii) Opt: the percentage of times where the provided solution is proven optimal (i.e., the provided makespan is equal to LB_1)

We observe that $H1$ and $H2$ exhibit a similar performance but are outperformed by $H3$. We note that the obtained value of the makespan provided by $H3$ is equal to the minimum between those obtained by $H1$ and $H2$. However, our experimental results show that the required computational effort is substantially reduced.

The comparison of $H3$ with $H4$ shows that the bisection search performed by $H4$ allows a substantial decrease of the CPU time while it slightly deteriorates the quality of the obtained solution. $H4$ often exhibits a good trade-off between the decrease of the computation time and the decrease of the solution's quality. In addition, $H5$ outperforms all the other heuristics for small value of n ($n \leq 50$) by providing a small average gap. However it requires much more CPU time.

For all the heuristics, we observe that the average gap increases when the number of machines increases (especially for $m = 6$) and decreases when the number of jobs increases.

Table 13: Performance of the heuristics for $m = 4$.

n	10	20	30	40	50	60	70	80
$H1$								
Gap	3.57	2.15	1.29	0.50	0.72	0.36	0.31	0.09
Time	0.76	6.65	24.78	39.09	119.40	115.96	208.47	229.45
Opt	50	10	30	60	40	60	40	50
$H2$								
Gap	3.88	3.08	1.00	0.49	0.80	0.77	0.80	0.56
Time	0.71	9.03	15.80	23.74	236.01	218.44	379.63	507.28
Opt	40	20	50	50	40	50	40	50
$H3$								
Gap	2.69	2.04	0.84	0.22	0.51	0.33	0.20	0.07
Time	0.92	10.56	29.25	34.45	256.20	140.97	271.56	260.30
Opt	60	30	60	70	60	80	50	70
$H4$								
Gap	3.50	2.99	2.03	0.52	0.97	0.47	0.55	0.44
Time	0.33	3.25	13.91	35.38	95.81	185.55	360.32	628.47
Opt	50	30	20	60	60	70	50	50
$H5$								
Gap	1.62	0.95	0.66	0.25	0.46	0.29	0.48	0.22
Time	2.72	26.90	159.39	149.75	290.63	314.27	482.18	599.55
Opt	60	50	60	90	60	80	50	30

Table 14: Performance of the heuristics for $m = 5$.

n	10	20	30	40	50	60	70	80
$H1$								
Gap	5.79	7.18	5.73	3.08	1.77	1.68	1.34	0.84
Time	1.66	31.35	134.53	222.54	300.07	681.81	1391.61	881.23
Opt	20	10	10	0	10	40	20	30
$H2$								
Gap	4.94	5.29	6.35	3.44	1.27	1.36	1.82	0.94
Time	1.62	25.63	170.30	213.26	319.76	606.98	1228.84	1652.91
Opt	20	10	10	0	10	20	10	50
$H3$								
Gap	3.86	4.43	4.93	2.36	0.73	0.91	1.23	0.55
Time	2.10	35.78	233.55	285.89	291.89	851.41	1882.48	1485.87
Opt	30	20	10	0	20	40	20	60
$H4$								
Gap	5.15	6.21	5.35	3.34	1.38	1.26	1.59	0.90
Time	0.53	6.38	24.17	59.67	152.74	281.72	537.82	907.04
Opt	20	20	10	0	10	40	10	40
$H5$								
Gap	2.90	3.17	3.99	2.40	0.96	1.32	1.40	1.12
Time	9.58	108.53	376.25	529.69	560	498.20	590.75	600
Opt	40	30	20	20	20	40	20	0

Furthermore, for the case where $m = 4$, we remark that the average gap is in most cases less than 1%. The maximum number of occurrences where the average gap is less than 1% is reached for $H5$ which is often able to provide the optimal solution.

6. Conclusion

In this paper, we present new adjustment and bounding procedures for the nonpermutation flowshop scheduling problem. We improve the main proposed adjustment procedures of the literature and develop new ones. The resulting adjustments have been efficiently used to derive lower and upper bounds for the problem. Our experimental results show that the developed bounds provide good results especially for small values of the number of machines. In particular, a heuristic based on randomization and bisection search seems to exhibit promising performance.

This research can be extended by incorporating all the new adjustment procedures and the developed lower and upper bounds in a branch-and-bound algorithm. Another avenue of future investigation is to apply the adjustment

Table 15: Performance of the heuristics for $m = 6$.

n	10	20	30	40	50	60	70	80
$H1$								
Gap	9.47	7.67	6.91	5.14	3.80	1.62	2.76	2.68
Time	3.78	53.16	253.38	728.58	1063.13	1228.46	3719.93	6279.36
Opt	0	0	0	0	0	20	20	20
$H2$								
Gap	9.61	7.43	7.31	5.12	4.57	2.04	2.56	2.72
Time	4.28	49.41	242.64	628.18	1192.89	1173.98	2274.25	5679.16
Opt	0	0	0	10	0	0	20	40
$H3$								
Gap	7.99	6.49	6.23	4.48	3.33	1.27	2.32	2.18
Time	5.85	83.94	412.25	1180.35	1563.44	1476.98	4525.30	7984.64
Opt	0	0	0	10	0	20	30	40
$H4$								
Gap	9.75	8.29	6.90	5.45	4.72	2.55	2.81	2.78
Time	0.87	10.34	40.37	106.24	217.00	453.18	922.69	1491.32
Opt	0	0	0	10	0	10	20	20
$H5$								
Gap	6.39	4.84	5.61	5.05	5.01	3.43	3.31	5.04
Time	16.85	223.17	554.18	553.06	594.42	576.12	600	600
Opt	10	0	10	10	0	10	0	0

procedures to the permutation flowshop and the jobshop scheduling problems.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The authors would like to extend their sincere appreciation to the Deanship of Scientific Research at King Saud University for its funding of this research through the Research Group Project no. RGP-VPP-296.

References

[1] S. Pugazhendhi, S. Thiagarajan, C. Rajendran, and N. Anantharaman, "Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems," *Computers & Industrial Engineering*, vol. 44, no. 1, pp. 133–157, 2003.

[2] R. Swaminathan, M. E. Pfund, J. W. Fowler, S. J. Mason, and A. Keha, "Impact of permutation enforcement when minimizing total weighted tardiness in dynamic flowshops with uncertain processing times," *Computers & Operations Research*, vol. 34, no. 10, pp. 3055–3068, 2007.

[3] C. N. Potts, D. B. Shmoys, and D. P. Williamson, "Permutation vs. non-permutation flow shop schedules," *Operations Research Letters*, vol. 10, no. 5, pp. 281–284, 1991.

[4] M. I. Sviridenko, "A note on permutation flow shop problem," *Annals of Operations Research*, vol. 129, pp. 247–252, 2004.

[5] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[6] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 61–68, 1954.

[7] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117–129, 1976.

[8] E. Ignall and L. Schrage, "Application of the branch and bound technique to some flow-shop scheduling problems," *Operations Research*, vol. 13, no. 3, pp. 400–412, 1965.

[9] L. Lomnicki, "A branch and bound algorithm for the exact solution of the three-machine scheduling problem," *Operational Research Quarterly*, vol. 16, no. 1, pp. 89–100, 1965.

[10] J. Grabowski, "On two-machine scheduling with release dates to minimize maximum lateness," *Operations Research*, vol. 17, pp. 133–154, 1980.

[11] J. Carlier and I. Rebaï, "Two branch and bound algorithms for the permutation flow shop problem," *European Journal of Operational Research*, vol. 90, no. 2, pp. 238–251, 1996.

[12] J. Cheng, H. Kise, and H. Matsumoto, "A branch-and-bound algorithm with fuzzy inference for a permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 96, no. 3, pp. 578–590, 1997.

[13] M. Haouari and T. Ladhari, "A computational study of the permutation flow shop problem based on a tight lower bound," *Computers and Operations Research*, vol. 32, no. 7, pp. 1831–1847, 2005.

[14] R. Ruiz and C. Maroto, "A comprehensive review and evaluation of permutation flowshop heuristics," *European Journal of Operational Research*, vol. 165, no. 2, pp. 479–494, 2005.

[15] M. Nawaz, E. E. Enscore Jr., and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[16] M. Haouari and T. Ladhari, "A branch-and-bound-based local search method for the flow shop problem," *Journal of the Operational Research Society*, vol. 54, no. 10, pp. 1076–1084, 2003.

[17] I. H. Osman and C. N. Potts, "Simulated annealing for permutation flow-shop scheduling," *Omega*, vol. 17, no. 6, pp. 551–557, 1989.

[18] C. R. Reeves, "Improving the efficiency of tabu search for machine sequencing problems," *Journal of the Operational Research Society*, vol. 44, no. 4, pp. 375–382, 1993.

[19] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job shop problem," *Management Science*, vol. 42, no. 6, pp. 797–813, 1996.

[20] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," in *Industrial Scheduling*, J. F. Muth and G. L. Thompson, Eds., pp. 225–251, Prentice Hall, Englewood Cliffs, NJ, USA, 1963.

[21] P. Brucker, S. A. Kravchenko, and Y. N. Sotskov, "On the complexity of two machine job-shop scheduling with regular objective functions," *Operations-Research-Spektrum*, vol. 19, no. 1, pp. 5–10, 1997.

[22] P. Brucker, S. A. Kravchenko, and Y. N. Sotskov, "Preemptive job-shop scheduling problems with a fixed number of jobs," *Mathematical Methods of Operations Research*, vol. 49, no. 1, pp. 41–76, 1999.

[23] P. Brucker, Y. N. Sotskov, and F. Werner, "Complexity of shop-scheduling problems with fixed number of jobs: a survey," *Mathematical Methods of Operations Research*, vol. 65, no. 3, pp. 461–481, 2007.

[24] J. Carlier and E. Pinson, "An algorithm for solving the job-shop problem," *Management Science*, vol. 35, no. 2, pp. 164–176, 1989.

[25] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete Applied Mathematics*, vol. 49, no. 1–3, pp. 107–127, 1994.

[26] W. Brinkkötter and P. Brucker, "Solving open benchmark instances for the job-shop problem by parallel head–tail adjustments," *Journal of Scheduling*, vol. 4, no. 1, pp. 53–64, 2001.

[27] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, vol. 34, no. 3, pp. 391–401, 1988.

[28] S. Dauzere-Peres and J.-B. Lasserre, "A modified shifting bottleneck procedure for job-shop scheduling," *International Journal of Production Research*, vol. 31, no. 4, pp. 923–932, 1993.

[29] E. Balas, J. K. Lenstra, and A. Vazacopoulos, "The one-machine problem with delayed precedence constraints and its use in job shop scheduling," *Management Science*, vol. 41, no. 1, pp. 94–109, 1995.

[30] H. Wenqi and Y. Aihua, "An improved shifting bottleneck procedure for the job shop scheduling problem," *Computers & Operations Research*, vol. 31, no. 12, pp. 2093–2110, 2004.

[31] V. A. Armentano and C. R. Scrich, "Tabu search for minimizing total tardiness in a job shop," *International Journal of Production Economics*, vol. 63, no. 2, pp. 131–140, 2000.

[32] E. Nowicki and C. Smutnicki, "New algorithm for the jobshop problem," Tech. Rep., Institute of Engineering Cybernetics, Wroclaw University of Technology, Wroclaw, Poland, 2003.

[33] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling*, Addison-Wesley, Reading, Mass, USA, 1967.

[34] C. L. Monma and A. H. G. Rinnooy Kan, "A concise survey of efficiently solvable special cases of the permutation flow-shop problem," *RAIRO Recherche Opérationnelle*, vol. 17, no. 2, pp. 105–119, 1983.

[35] A. H. G. R. Kan, *Machine Scheduling Problems: Classification, Complexity and Computations*, Nijhoff, The Hague, The Netherlands, 1976.

[36] J. K. Lenstra, *Sequencing by Enumerative Methods*, vol. 69 of *Mathematical Center Tracts*, Mathematisch Centrum, Amsterdam, The Netherlands, 1977.

[37] M. Dell'Amico, "Shop problems with two machines and time Lags," *Operations Research*, vol. 44, no. 5, pp. 777–787, 1996.

[38] D. Rebaine, "Flow shop vs. permutation shop with time delays," *Computers and Industrial Engineering*, vol. 48, no. 2, pp. 357–362, 2005.

[39] B. Benson and M. Dror, "Linear time procedure for the two job flowshop," *Foundations of Computing and Decision Sciences*, vol. 23, no. 3, pp. 179–186, 1998.

[40] J. Carlier and E. Pinson, "Adjustment of heads and tails for the job-shop problem," *European Journal of Operational Research*, vol. 78, no. 2, pp. 146–161, 1994.

[41] P. Brucker, B. Jurisch, and A. Kramer, "The job-shop problem and immediate selection," *Annals of Operations Research*, vol. 50, pp. 73–114, 1994.

[42] J. Carlier and E. Pinson, "A practical use of Jackson's Preemptive Schedule for solving job-shop problem," *Annals of Operations Research*, vol. 26, no. 1–4, pp. 269–287, 1990.

[43] J. R. Jackson, "Scheduling a production line to minimize maximum tardiness," Management Science Research Project, University of California, Los Angeles, Calif, USA, 1955.

[44] M. Haouari and A. Gharbi, "An improved max-flow-based lower bound for minimizing maximum lateness on identical parallel machines," *Operations Research Letters*, vol. 31, no. 1, pp. 49–52, 2003.

[45] J. Carlier, "The one-machine sequencing problem," *European Journal of Operational Research*, vol. 11, no. 1, pp. 42–47, 1982.