RAIRO-Oper. Res. 48 (2014) 235–254
DOI: 10.1051/ro/2014004

RAIRO Operations Research
www.rairo-ro.org

A BRANCH AND BOUND ALGORITHM FOR THE TWO-MACHINE FLOWSHOP PROBLEM WITH UNIT-TIME OPERATIONS AND TIME DELAYS

Aziz Moukrim[1], Djamal Rebaine[2] and Mehdi Serairi[1]

Abstract. In this paper we consider the problem of scheduling, on a two-machine flowshop, a set of unit-time operations subject to time delays with respect to the makespan. This problem is known to be $\mathcal{NP}$-hard in the strong sense. We propose an algorithm based on a branch and bound enumeration scheme. This algorithm includes the implementation of new lower and upper bound procedures, and dominance rules. A computer simulation to measure the performance of the algorithm is provided for a wide range of test problems.

Keywords. Branch and bound, dominance rule, flowshop, lower and upper bounds, time delays.

Mathematics Subject Classification. 9008.

1. Introduction

The classical flowshop scheduling problem can be described as follows. Given are a set $J = \{1, \ldots, n\}$ of n jobs and a set $M = (M_1, \ldots, M_m)$ of m machines. Each job j has to be processed for a given time on each machine starting from M_1, then M_2 and so forth until M_m. We seek a schedule of the n jobs that minimizes the makespan.

Received September 11, 2013. Accepted November 28, 2013.

[1] Université de Technologie de Compiègne, Heudiasyc, CNRS UMR 7253 Rue Roger Couttolenc CS 60319 60203 Compiègne Cedex, France.
aziz.moukrim@hds.utc.fr; mehdi.serairi@hds.utc.fr

[2] Département d'Informatique et Mathématique, Université du Québec à Chicoutimi, Saguenay, Canada G7H 2B1. Djamal_Rebaine@uqac.ca

Article published by EDP Sciences

The flowshop problem is known to be solvable in $O(n \log n)$-time for $m = 2$ [5], and $\mathcal{NP}$-hard in the strong sense for fixed $m \geq 3$ [4]. It is also known that the permutation flowshop schedules are dominant for $m \leq 3$ (see for *e.g.* [3]).

In most of the classical shop scheduling models, it is assumed that once an operation is completed, the corresponding job instantaneously becomes available for further processing. In practice, however, there is often a time delay between two successive operations of the same job. This time delay may be attributed, for example, to the transfer of jobs through the machines or the cooling of jobs before they can be handled by the next machine. On the other hand, in some applications, the processing times might even be negligible compared to the time delays. We can then assume in this case that the processing times of the jobs are unitary, and therefore have a small influence on the makespan of the schedule. As an example, we might mention the application of painting a set of small items, each of which has to be painted several times. However, a minimum time delay must elapse between two successive paintings in order to allow for the previous color to dry. One can easily imagine that the drying process may take far more time than the painting process.

The present study is devoted to the minimization of the makespan in a two-machine flowshop with unit-time operations and time delay considerations. A schedule to be valid must thus be such that a time delay of at least $\tau_j \geq 0$ units of time must elapse between the completion of job j on machine M_1 and its start on machine M_2.

This problem is shown to be $\mathcal{NP}$-hard in the strong sense [9]. The investigation of enumerative and heuristic approaches is thus well justified. We consider in this paper the resolution of the above problem within the framework of the branch and bound scheme. Note that the heuristic approach has been also investigated for this problem (see for *e.g.* [7]. On the other hand, the permutation flowshops with time delays are not dominant anymore, even for $m = 2$. Indeed, it is shown in [8] that the best unit-time operation permutation schedule is worse than that of the best unit-time operation flowshop schedule by a factor of $(2 - 3/(n+2))$ and m respectively for $m = 2$ and $m \geq 3$. However, special cases exist where permutation schedules are still dominant (see for *e.g.* Yu [10]). Note that the permutation counterpart of the above problem can be solved respectively in $O(n)$, $O(n \log n)$, and $O(n^2)$ for $m = 2, m = 3$, and $m = 4$, and still open for $m \geq 5$ [6].

This paper is organized as follows. In Section 2, the branch and bound algorithm we are presenting is described in details. Section 3 presents the computational simulation we conducted to study the performance of this algorithm. Concluding remarks are given in Section 4.

2. The branch and bound algorithm

A branch and bound algorithm consists of breaking up the target problem into successively smaller sub-problems, computing bounds on the objective function associated with each sub-problem, and using them to discard certain of these

sub-problems from further consideration. The procedure ends up when each sub-problem has either produced a feasible solution or been shown to contain no better solution than the one already in hand. The best solution found at the end of the procedure is the global optimum. Applying a branch-and-bound algorithm requires specifying the ingredients described in the following subsections. It is worth mentioning that the effectiveness of branch and bound algorithms relies not only on the tightness of these ingredients but also on the running time to compute them (for more details, see for *e.g.* [1]).

2.1. Branching rule

In the search tree, generated by the branch and bound algorithm, a node at level k corresponds to a partial sequence in which the jobs in the first k positions have been already fixed (scheduled). A node at level $k \geq 0$ is then expanded into at most $(n-k)$ nodes according to the following rule: among the jobs left, we only consider those with distinct time delays. This rule is a consequence of the following observation.

Observation 2.1. *Let i and j be two jobs where $\tau_i = \tau_j$. If $i < j$ then there exists an optimal solution in which i precedes j.*

2.2. Lower bounds

The basic idea of the use of a lower bound, within the framework of a branch and bound algorithm, is to be able to discard nodes at early stages of the search tree. If this lower bound is greater than or equal to the current makespan, then this node and all the subtrees associated with that node can be discarded as they cannot lead to a better solution. Before proceeding further, we introduce the following notation for the remaining sections.

- Given a sequence σ, J_σ denotes the set of jobs in σ, and $|J_\sigma|$ or $|\sigma|$ its cardinality.
- The set of jobs that belong to set J but not to J_σ is denoted by $J - J_\sigma$.
- The starting time of job j on M_i in a given schedule is denoted by $t_i(j)$.

In the following we start by presenting known lower bounds that are relevant to the present work, and then we continue the discussion with new lower bounds.

Throughout this section we will be evaluating the complexity of the lower bounds on each internal node by assuming that the n jobs are already sorted at the root of the search tree. The running time of this procedure is $O(n \log n)$-time.

2.2.1. Previous lower bounds

Here we discuss some known results related to the problem we are considering.

Lemma 2.2 [9]. *Let us assume that the time delays are such that* $\tau_1 \geq \ldots \geq \tau_n$. *If* $C_{\max}(S_{\text{opt}})$ *denotes the makespan of an optimal schedule* S_{opt} *then*

$$C_{\max}(S_{\text{opt}}) \geq LB(J) = \max_{1 \leq k \leq n} \left\{ \left\lceil \sum_{j=1}^{k} \tau_j / k \right\rceil + k + 1 \right\}.$$

For the sake of completeness, we provide a proof of Lemma 2.2 along the following lines.

Proof. Let us first observe that a valid schedule with a makespan C may be easily transformed into a valid schedule of the same length such that the n jobs of set J are processed continuously on both machines. Therefore, finding a schedule of makespan C is the same as finding two permutations, say σ and π, on machine M_1 and M_2, respectively. Let denote by $\sigma^{-1}(j)$ and $\pi^{-1}(j)$ respectively the position of job j in σ and π. In terms of σ and π, a schedule with a makespan C is valid only if

$$\pi^{-1}(j) - \sigma^{-1}(j) + C - n \geq \tau_j + 1, j = 1, \ldots, n,$$

which may be rewritten as

$$\tau_j - C + n + 1 \leq \pi^{-1}(j) - \sigma^{-1}(j), j = 1, \ldots, n.$$

Adding together the above n equations, we get

$$\begin{aligned} \sum_{j=1}^{n} (\tau_j - C + n + 1) &\leq \sum_{j=1}^{n} \pi^{-1}(j) - \sum_{j=1}^{n} \sigma^{-1}(j). \\ &\leq 0. \end{aligned}$$

The result is then derived by developing the last above inequality and putting aside C:

$$C \geq \sum_{j=1}^{n} \tau_j / n + n + 1.$$

By observing that the makespan is integral, we obtain:

$$C \geq \left\lceil \sum_{j=1}^{n} \tau_j / n \right\rceil + n + 1.$$

Finally, by considering only the k first jobs, we have:

$$C \geq \left\lceil \sum_{j=1}^{k} \tau_j / k \right\rceil + k + 1.$$

□

Corollary 2.3 [10]. *Consider a schedule S with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Then it holds that*

$$C_{\max}(S) \geq LB_1(\sigma_1) = \max\{LB(J), |\sigma_1| + LB(J_{\sigma_2})\}.$$

Definition 2.4. If job j is completed at time k on M_1, then its arrival time on M_2 is $k + \tau_j$.

Definition 2.5 [10]. Let S be a schedule with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Let also $C(\sigma_1)$ be the minimal makespan obtained by scheduling jobs of J_{σ_1} according to their arrival time on $M2$, and t^* the minimal arrival time of jobs in σ_1 on M_2, such that M_2 is not idle in $[t^\star, C(\sigma_1)]$. In addition, let δ be the maximum cardinalty of matching between the positions on M_1 within $[|\sigma_1| + 1, n]$ and jobs in σ_2 with arrival times on M_2 less than t^*.

Let us mention that Yu [10] pointed out that δ corresponds to the number of time slots $i = |\sigma_1| + 1, \ldots, n$, which can be filled by jobs in σ_2, starting by those with the largest time delays and arrival times on M_2 less than t^*. This observation leads to the following lower bound, which can be implemented in $O(n)$.

Lemma 2.6 [10]. *Let S be a schedule with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Then it holds that*

$$C_{\max}(S) \geq LB_2(\sigma_1) = C(\sigma_1) + n - |\sigma_1| - \delta.$$

Let us observe that, besides the sorting procedure that is involved, the rest of the evaluation of LB, LB_1 and LB_2 takes $O(n)$-time.

2.2.2. New lower bounds

In the branch and bound algorithm that we are designing we are interested in building a solution, with a makespan less than or equal to a given value L. In this section we first present new lower bounds. In the second step, we continue with other results related to the existence of a schedule of a given length. We also discuss dominance properties. Before proceeding, we first make the following observation.

Observation 2.7. *Prior to a processing permutation on M_1, solutions in which the jobs are scheduled on M_2 according to their arrival times are dominant.*

Lemma 2.8. *Let S be a schedule with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Let the arrival times on M_2 of jobs in J_{σ_1} and J_{σ_2} be computed from their positions and $|\sigma_1| + 1$ on M_1, respectively. Denote by $LB_3(\sigma_1)$ the makespan generated by optimally scheduling the n jobs according to these arrival times on M_2. Then it holds that $C_{\max}(S) \geq LB_3(\sigma_1)$.*

Proof. First, observe that with respect to σ_1, the arrival times of the jobs in σ_2 are computed from time slot $|\sigma_1|+1$ their least starting time on M_1. On the other hand, M_2 cannot complete the processing of the jobs before the makespan produced by solving the one machine problem with release dates, which are nothing else than the arrival times of the jobs (Observation 2.7). Furthermore, we know that the latter problem can be solved optimally according to the nondecreasing order of the release dates. Therefore, the result is established. □

Theorem 2.9. *Let S be a schedule with a job sequence $\sigma = (\sigma_1, \sigma_2)$, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Then, it holds $LB_3(\sigma_1) \geq LB_2(\sigma_1)$. Furthermore, there exist instances for which the inequality is strict.*

Proof. Let S_2 be a one machine optimal schedule of the n jobs on M_2 according to their arrival times as defined in Lemma 2.8. Let also $t_2(j)$ be the starting time of job j in S_2, and δ and t^* be as in Definition 2.5. Let us now denote by $\xi_1, \ldots, \xi_u$ the jobs of J_{σ_2} processed in that order in S_2 between time $|\sigma_1|+1$ and t^*-1. In order to prove the first statement of the theorem, we first need to prove $u \leq \delta$. In fact, we need to prove that jobs $\xi_1, \ldots, \xi_u$ can take positions on M_1 within $[|\sigma_1|+1, n]$ and their arrival times are less than t^*. We proceed as follows. Note first that if $u = 0$, we are done. Otherwise, by definition of u, we have

$$t_2(\xi_u) \leq t^* - 1.$$

Therefore,

$$|\sigma_1| + 1 + \tau_{\xi_u} \leq t^* - 1.$$

Since jobs ξ_k and ξ_{k+1}, $k \in \{1, \ldots, u-1\}$, are processed in that order then we have

$$t_2(\xi_k) + 1 \leq t_2(\xi_{k+1}).$$

Summing up the above inequality from k to $u-1$ we get

$$t_2(\xi_k) + (u-k) \leq t_2(\xi_u), \ k = 1, \ldots, u-1.$$

From the structure of S we have

$$|\sigma_1| + 1 + \tau_{\xi_k} \leq t_2(\xi_k), \text{ for } k = 1, \ldots, u-1.$$

It then follows

$$\begin{aligned} |\sigma_1| + 1 + \tau_{\xi_k} + (u-k) &\leq t_2(\xi_k) + (u-k) \\ &\leq t_2(\xi_u) \\ &\leq t^* - 1. \end{aligned}$$

Therefore, we deduce $u \leq \delta$. From the definition of u, the number of jobs scheduled on M_2 after $C(\sigma_1)$ in S is equal to $n - |\sigma_1| - u$ and $LB_3(\sigma_1) \geq C(\sigma_1) + n - |\sigma_1| - u$. So, since $\delta \geq u$, we derive that $LB_2(\sigma_1) \leq LB_3(\sigma_1)$.

With regard to the strict inequality (the second statement of the theorem), consider the instance with $n = 3$ jobs, $\tau_1 = 7$, $\tau_2 = 2$, and $\tau_3 = 0$. Let us now assume that $\sigma_1 = (2, 3)$. It follows that $LB_2(\sigma_1) = 5$, whereas $LB_3(\sigma_1) = 11$. Thus, the result of the theorem follows. □

Following is another lower bound similar to that of Lemma 2.2. However, the time delays of the jobs already scheduled are updated here to express the real times of processing the jobs on both machines.

Proposition 2.10. *Let S be a schedule with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Let $\gamma_j = t_2(j) - t_1(j) - 1$, $j \in J_{\sigma_1}$, be the new time delay of job j, where $t_i(j)$ denotes the starting time of j on M_i. The time delays of jobs of J_{σ_2} do not change, and we set $\gamma_j = \tau_j$, $j \in J_{\sigma_2}$. If the time delays are such that $\gamma_{j_1} \geq \ldots \geq \gamma_{j_n}$, then it holds that*

$$C_{\max}(S) \geq LB_4(\sigma_1) = \max_{1 \leq k \leq n} \left\{ \left\lceil \sum_{i=1}^{k} \gamma_{j_i}/k \right\rceil + k + 1 \right\}.$$

Proof. Let I_1 and I_2 be two instances with time delays τ_j and γ_j, respectively. It is clear that any feasible solution, starting with σ_1 for I_1, is also a feasible solution starting with σ_1 for I_2, with no change on the makespan. Thus, the result follows immediately. □

Note that as the sorting procedure is involved in the evaluation of LB_3 and LB_4, their running time is $O(n \log n)$.

The remaining results of this section are based on the following. To eliminate partial solutions from further expansion at earlier stages, we incorporated in the implementation of the branch and bound algorithm a procedure to detect whether a partial solution is feasible by looking at the existence of a schedule with a makespan $\leq L$ for a given value L. Let us start with the following obvious observation.

Observation 2.11. *A schedule of makespan L may be assumed to be such that the jobs are sequenced continuously on both machines without an idle time, from time zero and $(L - n)$ on M_1 and M_2, respectively.*

It is worth noting from Observation 2.11 that LB_4 may be computed by tightening the time delays as follows without increasing the makespan and still leaving a valid schedule: $\gamma_j = \max\{L - n, t_2(j) - t_1(j) - 1\}$ for $j \in J_{\sigma_1}$ and $\gamma_j = \tau_j$ for $j \in J_{\sigma_2}$.

Proposition 2.12. *Let L be an integer, such that there is no feasible schedule with a makespan strictly less than L and let S be a schedule with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Let the arrival times of job j be $\max\{L - n, s(j) + 1 + \tau_j\}$ where $s(j) = t_1(j)$ if j is in σ_1 and $s(j) = |\sigma_1|$ if j is in J_{σ_2}. Denote by $val_3(\sigma_1)$ the makespan generated*

by optimally scheduling the n jobs according to the nondecreasing order of these arrival times on M_2. If $val_3(\sigma_1) > L$ then there is no schedule with a makespan $\leq L$. Furthermore, $val_3(\sigma_1) \geq LB_3(\sigma_1)$.

Proof. From Observation 2.11, we may assume the arrival time of jobs of J_{σ_1} in Lemma 2.8 can be tightened to $\max(L-n, |\sigma_1| + \tau_j + 1)$. Therefore, the result follows immediately. □

Let us examine the computation time of val_3. First, we recall that the arrival times of jobs in σ are in $I = [L-n, L]$. Using the counting sorting procedure [2], we get a sorted list of jobs in a nondecreasing order of their arrival times in $O(n)$-time. These jobs can be optimally scheduled on M_2 in $O(n)$-time, which represents the overall time complexity to evaluate this lower bound.

In what follows we discuss another result based on checking the feasibility of several schedules starting with a known sequence. We first make the following obvious observation.

Observation 2.13. *Let I be an arbitrary instance of jobs and $I' \subset I$. If there is no feasible solution for I' with a makespan $\leq L$, then there is no feasible solution for I with a makespan $\leq L$.*

Let us now consider S a schedule of an instance I with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Let I_p $(p > |\sigma_1|)$ be the instance that is composed of jobs from σ_1 and $(p - |\sigma_1|)$ jobs from σ_2 with the largest time delays. Let us make the following computations.

1. For $k = 1$ to $|\sigma_1|$ do
   (a) Let j be the job processed at time $k-1$ on M_1 $(j \in J_{\sigma_1})$.
   (b) Process job j on $M2$ as soon as possible from time $\theta(j) = \max\{t_1(j) + \tau_j + 1, L - p\}$.
   (c) Set $\widetilde{\tau}_j = \theta(j) - t_1(j) - 1$.
2. For $j \in I_p - J_{\sigma_1}$, we set $\widetilde{\tau}_j = \tau_j$.

The resulting instance we get from I_p is denoted by $\widetilde{I}_p$. We suppose that the jobs of $\widetilde{I}_p$ are sorted in nonincreasing order of $\widetilde{\tau}_j$. The second result is stated as follows.

Proposition 2.14. *Let L be an integer, such that there is no feasible schedule with a makespan strictly less than L and S be a schedule of an instance I with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Let $val_4(\sigma_1) = \max_{|\sigma_1| < p \leq n} LB(\widetilde{I}_p)$. If $val_4(\sigma_1) > L$, then the makespan of S is strictly greater than L.*

Proof. We first deduce from Observation 2.11 if there exists a schedule for I_p with a makespan less than or equal to L, then there exists a schedule for $\widetilde{I}_p$ with a makespan less than or equal to L. From Observation 2.13 we derive that if $LB(\widetilde{I}_p) > L$, for a given p, then there is no feasible schedule for I with a makespan $\leq L$. Therefore, the result follows. □

Let us now examine the time complexity associated with the evaluation of $val_4(\sigma_1)$. First, note that the new time delays $\widetilde{\tau}_j$ should be in nonincreasing order to compute $LB(\widetilde{I}_p)$. Therefore, the evaluation of $LB(\widetilde{I}_p)$ takes $O(|\widetilde{I}_p| \log |\widetilde{I}_p|)$-time. Since p varies from $(|\sigma_1| + 1)$ to n, a running time of $O(n^2 \log n)$ follows in order to complete the computation of val_4.

Observation 2.15. *$LB(\widetilde{I}_n)$ and $LB_4(\sigma_1)$ are evaluated similarly on the same instance by modifying the time delays. Thus, $val_4(\sigma_1) \geq LB_4(\sigma_1)$.*

2.3. Dominance rules

A dominance rule is a testing procedure which discards a partial schedule from further expansion as there exists another better solution starting with another partial sequence. The existence of a dominance rule, when used, may speed up the computations of a branch and bound algorithm. In this section, two dominance rules are presented. Clearly, at each node, the evaluation of these dominance rules can be done in $O(n)$-time.

Lemma 2.16. *Let S be a schedule with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Let $\beta(j)$ be the time at which job j in σ_2 is processed on M_2 if it is processed at time $|\sigma_1|$ on M_1. If there exists a job j' in σ_2 such that $\tau_{j'} > \tau_j$ and $\beta(j) - |\sigma_1| - 1 \geq \tau_{j'}$, then the partial solution starting with σ_1 followed by job j can be fathomed.*

Proof. The above lemma can be established through a simple interchange argument between job j and j'. □

Example 2.17. Consider the instance with $n = 5$ jobs, $\tau_1 = \tau_2 = \tau_3 = 2$, $\tau_4 = 1$, and $\tau_5 = 0$. Let us assume that $\sigma_1 = (1, 2)$. From Lemma 2.16, solutions starting with σ_1 followed by either job 4 or job 5 may be ignored.

Let us observe that we may have instances for which Lemma 2.16 is of no help as illustrated by Example 2.19. Lemma 2.18 may thus be useful to ignore additional partial solutions from further exploration.

Lemma 2.18. *Let S be a schedule with a job sequence $\sigma = (\sigma_1, \sigma_2)$ on M_1, where σ_1 is fixed and σ_2 an arbitrary subsequence of $J - J_{\sigma_1}$. Let j be the last job scheduled in J_{σ_1}. If $t_2(j) - (t_1(j) + 1) > \tau_j$ and there exists a job j' in J_{σ_2} such that $\tau_{j'} > \tau_j$, then the partial solution starting with σ_1 followed by j' can be fathomed.*

Proof. Let j be the last job scheduled in J_{σ_1} and assume that $t_2(j) - (t_1(j) + 1) > \tau_j$. Also, assume there exists a job j' in J_{σ_2} such that $\tau_{j'} > \tau_j$. It is then clear that there is no point to process job j' at time $|\sigma_1|$ on M_1 since these two jobs can be interchanged on M_1 without increasing the overall makespan. □

Example 2.19. Consider the instance with $n = 6$ jobs, $\tau_1 = 7, \tau_2 = 5, \tau_3 = \tau_4 = 4, \tau_5 = 2$, and $\tau_6 = 0$. Let $\sigma_1 = (1, 3, 2, 5)$. It is clear that Lemma 2.16 cannot be applied. However, when it comes to expand this node, it is not difficult to see that we should ignore schedule starting with σ_1 followed by job 4 since schedule starting with $(1, 3, 2, 5, 4)$ will not produce a better makespan than $(1, 3, 2, 4, 5)$.

2.4. Upper bounds

The following heuristic algorithms are intended to be used as an upper bound on the makespan on each node of the search tree of the branch and bound algorithm.

We discuss three different ways of building a priority list, and two different strategies to schedule a given priority list. Observe that six heuristic algorithms can be built out of this approach.

Step 1. The priority list is constructed any of the following rules:
1. Nonincreasing order of the time delays.
2. Given the nonincreasing order of the time delays, we first consider the set of jobs with time delays of a difference of at least two, followed by the set of jobs with time delays of a difference of at least two and so on until there is no job to consider anymore.
3. Given the nonincreasing order of the time delays, we consider first the set of jobs with time delays of a difference of at least two or the same time delays, followed by the set of jobs with time delays of difference at least two or the same time delays, and so on until there is no job to consider anymore.

Step 2. The priority list is then scheduled with either of the following strategies according to the ordering generated in Step 1.
1. On M_1 the jobs are processed by examining the priority list from left to right. On M_2, the jobs are scheduled according to the nondecreasing order of their arrival times as in Observation 2.7.
2. First, place the next job of the priority list on the first available position on M_1, and as soon as possible on M_2. Then, we replace this job on M_1 to make it processed now as close as possible to its processing on M_2 according to its time delay.

In what follows, we denote by H_{ij} the corresponding heuristic algorithm in which priority rule i of Step 1 is applied to strategy j of Step 2.

Example 2.20. Let us consider the following instance I with $n = 8$ jobs, and $\tau_1 = \tau_2 = 9, \tau_3 = \tau_4 = 8, \tau_5 = \tau_6 = 7$ and $\tau_7 = \tau_8 = 6$. We first apply priority rules of Step 1 on this instance. We derive the following lists: $(1, 2, 3, 4, 5, 6, 7, 8)$, $(1, 5, 2, 6, 3, 7, 4, 8)$, and $(1, 2, 5, 6, 3, 4, 7, 8)$ corresponding respectively to priority rule 1, 2, and 3. If we now apply the two strategies of Step 2 on the above lists, then we get the schedules shown in Table 1. Each entry in this table corresponds to the sequence generated by one of the heuristic algorithms and processed on M_1 and M_2, respectively.

TABLE 1. Schedules generated by the six heuristic algorithms on instance I.

H_{ij} \ Time slot	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
H_{11}	1	2	3	4	5	6	7	8			1	2	3	4	5	6	7	8
H_{12}	1	2	7	3	4	8	5	6		7	1	2	3	4	5	6	8	
H_{21}	1	5	2	6	3	7	4	8		5	1	6	2	3	7	4	8	
H_{22}	1	5	2	6	3	8	4	7		5	1	6	2	3	7	4	8	
H_{31}	1	2	5	6	3	4	7	8			1	2	5	6	3	4	7	8
H_{32}	1	2	7	3	5	6	4	8		7	1	2	5	6	3	4	8	

Algorithm 1 BB($\sigma_1, \mathbf{L}$)

Let σ_2^k be the k-th job in σ_2;
Let LB_{σ_1} be the lower bound associated with σ_1;
if $LB_{\sigma_1} > L$ **then**
  Prune node starting with σ_1;
end if
Let UB_{σ_1} be the makespan produced by heuristic H_{32} when σ_1 is processed first on M_1;
if $UB_{\sigma_1} = L$ **then**
  Stop the process and a feasible solution with a makespan of value L is found.
end if
if $UB_{\sigma_1} < UB$ **then**
  $UB = UB_{\sigma_1}$;
end if
$k = 1$;
while $k \leq |\sigma_2|$ **do**
  Consider σ_2^k;
  if Not(Dominance rules(σ_2^k)) **then**
    $\sigma_1 = \sigma_1 \cup \sigma_2^k$;
    $\sigma_2 = \sigma_2 \backslash \sigma_2^k$;
    call $BB(\sigma_1, L)$;
  end if
end while

With regard to the implementation of the above heuristic algorithms in the branch and bound scheme, we proceeded as follows. At the root of the search tree, the six heuristic algorithms are run, and the solution generated with the smallest makespan is retained as the best upper bound to start with. Within the internal nodes, we only invoke heuristic H_{32}. This choice is justified by the good

performance of this heuristic that emerged through the computational experiment discussed in Section 3.1.

Let us now examine the time complexity of the above heuristic algorithms. We start with the priority rules in Step 1. The first is clearly in $O(n \log n)$-time. To be processed, each job in the second and third rule has to find its right position within the jobs already processed. This step takes $O(n)$-time. As this process is done for every job, an $O(n^2)$-time follows for the corresponding rule. With regard to Step 2, in the first strategy, once we compute the arrival times of the jobs on M_2, which can be done in $O(n)$, we process the jobs on M_2 according to the nondecreasing order of the arrival times. This step takes $O(n \log n)$-time. In the second strategy, each job scans the list of the scheduled jobs before finding its time of processing. This task takes $O(n)$-time. Since every job is concerned with this process, this strategy takes $O(n^2)$-time. To conclude, the time complexity of the above six heuristic algorithms varies from $O(n \log n)$ to $O(n^2)$.

2.5. Algorithm scheme

The branch and bound algorithm we implemented is mainly a series of feasibility test problems. Each time we check whether there exists a solution with a given makespan of value L along the lines of Algorithm 1. If it is the case, then we stop the process. Otherwise, we increment the value of L, and repeat the same process until eventually reaching $(UB-1)$, where UB denotes the best upper bound found so far. Initially, L is set to a lower bound.

We discuss in the next section the empirical performance of the heuristic algorithms, lower bounds, dominance rules within the framework of the implementation of the branch and bound scheme we are designing.

3. Experimental results

We conducted a computational experiment where the heuristic algorithms and the branch and bound algorithm are coded in C language, and executed on an Intel CORE Duo 2×2.4 GHz PC with 3 GB RAM memory. The main parameters characterizing an instance of the problem we are investigating are the number and the time delays of jobs. The problem sizes tested are $n = 20, 40, 60, 80, 100, 120, 150, 160, 180, 200, 250$, and 300. The time delays are drawn from a uniform distribution in $[0, n]$ in Sections 3.1 and 3.2, and in $\left[0, \left\lceil \frac{n}{r} \right\rceil\right]$ in Section 3.3, where r is a parameter defined in that section. For each value of n, 10 instances were randomly generated. Let us note that, in the instances we generated, we assumed that the smallest time delay is always 0. If it is not the case, then it is straightforward to derive an equivalent instance in which the new time delays are respectively the old time delays minus the smallest time delay.

This section is organized as follows. First, we study and compare the performance of the heuristic algorithms. Then, we perform an experimental protocol to determine the best configuration for the branch and bound scheme. Finally, we

discuss the impact of the time delay distribution on the performance of the selected branch and bound configuration.

3.1. Performance of the heuristic algorithms

We report in this section the performance of the heuristic algorithms proposed in Section 2.4. For each heuristic algorithm, the following points are provided in Table 2.

- Opt: number of times a given heuristic algorithm achieves the optimal makespan.
- Best: number of times a given heuristic algorithm achieves the best solution among the rest of the heuristic algorithms.
- Gap_O: average gap related to the best lower bound.
- Gap_B: average gap related to the best generated solution among the proposed heuristic algorithms.

From Table 2, we observed the two best heuristic algorithms are H_{32} and H_{22}. Actually, H_{22} exhibits a good performance on small size instances. However, this good performance degrades as the size instances get larger. On the other hand, H_{32} outperforms the rest of the heuristic algorithms, and has a stable performance. Moreover, it generates a better performance on larger size instances.

3.2. Configuration of the branch and bound algorithm

We started by incorporating into the branch and bound scheme the lower bounds LB_1 and LB_4, the computing procedure of Proposition 2.12 namely val_3, and the dominance rules of Lemmas 2.16 and 2.18. This choice is mainly justified by their time complexity of at most $O(n \log n)$. Let us also mention that the depth first search method is the search strategy utilized to explore the search tree of the branch and bound algorithm. In what follows, we discus the empirical performance of different versions of the algorithm by varying the strategies according to the following observations:

- Which time delay ordering has an impact on the branching strategy.
- What is the impact of val_4 on the performance of the branch and bound algorithm.
- What is the impact of the heuristic algorithms when they are invoked in the internal nodes of the search tree.

The results of the simulation are summarized in Tables 3–5. For each instance, we set a time limit to 300 s. So, if the branch and bound algorithm is still running after 300 s, then it is aborted. The following points are computed in each of these tables.

- USI: the number of unsolved instances after 300 s.
- Nodes: the average number of nodes generated for the solved instances.
- Time: the average CPU times required for the solved instances.

TABLE 2. Performance of the heuristic algorithms.

n	H_{11}				H_{12}				H_{21}			
	Opt	Best	Gap_O	Gap_B	Opt	Best	Gap_O	Gap_B	Opt	Best	Gap_O	Gap_B
20	0	0	20.32	18.43	0	1	4.71	3.06	0	2	8.03	6.33
40	0	0	25.04	21.70	0	1	6.89	4.00	0	0	15.25	12.14
60	0	0	26.73	22.50	0	1	7.19	3.60	0	0	15.20	11.34
80	0	0	28.59	24.56	0	0	8.10	4.71	0	0	21.91	18.05
100	0	0	28.83	25.08	0	0	7.46	4.35	0	0	16.21	12.85
120	0	0	28.79	24.76	0	0	7.55	4.17	0	0	19.59	15.84
150	0	0	29.61	26.15	0	0	7.65	4.78	0	0	23.73	20.42
160	0	0	30.14	26.27	0	0	7.51	4.31	0	0	22.71	19.07
180	0	0	29.81	26.83	0	0	7.55	5.08	0	0	23.16	20.33
200	0	0	29.93	26.13	0	0	7.73	4.57	0	0	26.49	22.79
250	0	0	29.75	27.06	0	0	7.59	5.35	0	0	24.60	21.99
300	0	0	30.54	26.72	0	0	8.10	4.94	0	0	25.10	21.43
n	H_{22}				H_{31}				H_{32}			
	Opt	Best	Gap_O	Gap_B	Opt	Best	Gap_O	Gap_B	Opt	Best	Gap_O	Gap_B
20	5	10	1.61	0.00	0	1	6.96	5.29	3	7	3.21	1.60
40	1	4	7.29	4.37	0	0	10.35	7.38	1	8	3.96	1.14
60	0	3	8.07	4.45	0	0	9.03	5.37	0	9	3.58	0.10
80	0	0	14.19	10.58	0	0	9.59	6.14	0	10	3.26	0.00
100	0	0	10.57	7.37	0	0	9.14	5.95	0	10	3.01	0.00
120	0	0	12.44	8.91	0	0	8.94	5.49	0	10	3.26	0.00
150	0	0	16.68	13.56	0	0	8.19	5.30	0	10	2.75	0.00
160	0	0	15.75	12.33	0	0	8.97	5.71	0	10	3.08	0.00
180	0	0	16.55	13.87	0	0	7.38	4.89	0	10	2.36	0.00
200	0	0	18.87	15.38	0	0	8.04	4.85	0	10	3.03	0.00
250	0	0	17.78	15.32	0	0	7.68	5.43	0	10	2.13	0.00
300	0	0	18.96	15.48	0	0	8.84	5.65	0	10	3.02	0.00

3.2.1. Impact of the branching strategy

Let us first recall from Lemma 2.1 that we are only branching on the time delay values that are distinct. Throughout our experiments we suspected that if we adopt a branching rule that considers the jobs in nondecreasing order of time delays, the dominance rules of Lemmas 2.16 and 2.18 are started off more often at the higher levels of the search tree. As a consequence, nodes are pruned at the early stages

TABLE 3. Impact of the time delay ordering on the branching rule in the search tree.

n	*BB1*			*BB2*			*BB3*		
	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
20	0	51.4	0	0	326.7	0	0	36	0
40	0	716	0	2	907 637.5	1.72	0	92 264.2	0.15
60	0	24 172.5	0.04	8	221 593	0.51	0	6 797 069.9	14.04
80	0	4 330 328.1	7.82	10	***	***	2	319 630.5	0.81

of the algorithm. In order to check the validity of this assumption, we focused our tests on three different branching strategies in which jobs are considered in:

– nondecreasing order of the time delays *(BB1)*,
– nonincreasing order of the time delays *(BB2)*, and
– arbitrary order of the time delays *(BB3)*.

We first reported in Table 3 the performance of *BB1*, *BB2* and *BB3*, and then observed the following points:

– For the 40 instances *BB2* fails to solve 20 instances and the average CPU times for the solved instances is greater than those of *BB1* and *BB3*. Compared to the two other strategies, the branching order based on the nonincreasing order of the time delays seems to be irrelevant.
– *BB1* outperforms *BB3*. Indeed, the average CPU times is less important for *BB1*, and for $n = 80$ *BB3* fails to solve two instances. However the 10 instances are solved to optimality by *BB1*.
– It is interesting to note that, for $n = 80$, *BB3* seems to be less time consuming than *BB1*. This is due to the fact that the average CPU times is calculated over the solved instances, and for *BB1* there are 2 more solved instances that *BB3* failed to solve.

We stopped the simulation for $n = 80$ as we are confident on the validity of our assumption. Therefore, we assumed in our implementation that the jobs are in nondecreasing order of the time delays.

3.2.2. Impact of Proposition 2.14

So far we have not tested yet the efficiency of val_4. We did so because of the high cost in time to evaluate it. In this section, we intend to study its impact on the performance of the branch and bound algorithm to see whether it is worth to incorporate it into the final implementation. To do so we performed some experiments with a variant of *BB1*, denoted *BB4*, which includes the computing procedure of Proposition 2.14, namely val_4. Table 4 summarizes the results we obtained.

TABLE 4. Impact of val_4.

n	*BB1*			*BB4*		
	USI	Nodes	Time	USI	Nodes	Time
20	0	51.4	0	0	50.2	0
40	0	716	0	0	346.9	0
60	0	24 172.5	0.04	0	738.6	0
80	0	4 330 328.1	7.82	0	497.6	0
100	4	2 263 952.83	5.26	0	2 008 048.1	10.18

From Table 4, we made the following observations:

- *BB4* exhibits a good performance, and outperforms *BB1*. Indeed, all the 50 instances generated in this experiment are solved to optimality, and *BB4* exhibits smaller CPU times. Moreover, the average number of visited nodes are smaller than those of *BB1*. Therefore, the computing procedure val_4 is efficient as it prunes additional instances.
- For $n = 100$, *BB1* seems to be less time consuming than *BB4*. This is due to the fact that *BB4* solves the 10 instances. However, *BB1* failed to solve 4 instances.

To conclude, the incorporation of val_4 makes the branch and bound algorithm more efficient.

3.2.3. Impact of the heuristic algorithms

The third and final point we are looking at in our quest for the best configuration for the branch and bound algorithm that we are designing is the impact of the heuristic algorithms when run on each internal nodes of the search tree. We focused on *BB4* version. So far, this method has the best configuration. We considered a modified version of *BB4* in which we invoke in each internal node of the search tree heuristic H_{32} that outperforms the rest of the implemented heuristic algorithms as discussed above. We denote this version by *BB5*. Table 5 summarizes the performance of *BB4* and *BB5*. From Table 5, we make the following observations:

- *BB5* exhibits a good performance and outperforms *BB4*. Actually, *BB5* fails to solve only 10 instances out of 100. In addition, *BB5* exhibits shorter CPU times.
- The average number of visited nodes in *BB5* is about 1000 times less than that of *BB4*.
- For $n = 200$, *BB4* fails to solve all the 10 instances. However, when *BB5* is used, only 2 instances are not solved.

TABLE 5. Impact of the heuristic algorithms.

n	*BB4*			*BB5*		
	USI	Nodes	Time	USI	Nodes	Time
20	0	50.2	0	0	9.7	0
40	0	346.9	0	0	269.9	0
60	0	738.6	0	0	250	0
80	0	497.6	0.01	0	392.9	0
100	0	2 008 048.1	10.19	0	1282.2	0.01
120	1	3 595 143.44	13.53	1	787.33	0.01
150	5	97 716.2	0.48	4	1174.17	0.02
160	4	1 763 743	10.31	2	1490.13	0.04
180	5	3 995 767.2	19.5	1	3571.22	0.06
200	10	***	***	2	21 117.13	0.21

3.3. IMPACT OF THE TIME DELAY DISTRIBUTION

In this section, we study the impact of the time delay distribution on *BB5*. To do so, we introduced new instance classes to see the impact of the interval size from which the time delays are drawn on the performance of the branch and bound algorithm we are designing. A class is characterized by a rational number r to measure somehow the sparsity of the time delays: the smaller is r, the bigger is the sparsity of the time delays over the interval they are drawn.

For each class, the time delays are drawn from a uniform distribution of values in the range $\left[0, \left\lceil \frac{n}{r} \right\rceil\right]$, where n is $20, 40, 60, 80, 100, 120, 150, 160, 180, 200, 250$, and 300. For each value of n, 10 instances are randomly generated.

Tables 6 and 7 summarize the results obtained for the values of $r \geq 1$ and $r \leq 1$. For each class, the numbers of unsolved instances and the average CPU times for the solved instances are provided.

From Table 6, we observe that for the values of r in $(2, 4)$, the corresponding class of instances are more difficult to solve. Especially, for $r = 2$, we note that the number of unsolved instances for $n = 120, 150, 160, 180,$
$200, 250$, and 300 is $7, 7, 6, 7, 10, 10$, and 10, respectively. However, when $r > 4$, the class instances seem to be more easily solved.

On the other hand, from Table 7, which summarizes the results generated for $r \leq 1$, we observe that, when r decreases, *BB5* solves more efficiently these class instances.

From the previous tables, we observed that the difficult class instances are obtained for the values of r in $[2, 4]$. In what follows we present a computational study in order to detect the values of r that slow down the running time of the branch and bound algorithm. The values of r that we tested are $1.5, 2, 2.5, 3, 3.5$, and 4. Table 8 summarizes the generated results. We observed that the class instances for

TABLE 6. Performance of *BB5* for class instances in which $r \geq 1$.

r	16		8		4		2		1	
n	USI	Time	USI	Time	USI	Time	USI	Time	USI	Time
20	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0.02	0	0
80	0	0.01	0	0.01	0	7.6	0	15.18	0	0
100	0	0.01	0	0.01	0	0.2	3	11.62	0	0.01
120	0	0.02	0	0.01	2	0.02	7	0.01	1	0.01
150	0	0.04	0	0.04	3	0.03	7	1.29	4	0.02
160	0	0.04	0	19.58	4	0.03	6	0.3	2	0.04
180	0	0.22	0	6.25	7	0.05	7	4.08	1	0.06
200	0	0.08	1	0.08	9	0.07	10	0	2	0.21
250	0	0.15	2	0.47	6	0.14	10	0	4	4.47
300	0	0.65	4	0.29	10	0	10	0	3	12.76

TABLE 7. Performance of *BB5* for class instances in which $r \leq 1$.

r	1		$\frac{2}{3}$		$\frac{1}{2}$		$\frac{2}{5}$		$\frac{1}{3}$	
n	USI	Time	USI	Time	USI	Time	USI	Time	USI	Time
20	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0.19	0	0	0	0	0	0
80	0	0	0	0.17	1	0	0	0	0	0
100	0	0.01	1	0.03	1	0.01	0	0	0	0
120	1	0.01	0	0.19	0	0.01	0	0.01	0	0
150	4	0.02	1	0.09	0	0.23	0	0.01	0	0.01
160	2	0.04	0	0.06	0	11.85	0	0.02	0	0.01
180	1	0.06	1	3.48	0	0.05	0	0.03	0	0.01
200	2	0.21	4	0.54	0	0.06	0	0.04	0	0.02
250	4	4.47	2	0.74	0	1.45	0	0.06	0	0.04
300	3	12.76	3	33.41	2	0.66	0	0.13	0	0.08

which the values of r are in $(2, 2.5, 3, 3.5, 4)$ are more difficult to solve. To complete the study we visualized in Figure 1 the total number of unsolved instances as a function of the considered values of r.

From Tables 6–8, and Figure 1, we observed that the branch and bound algorithm produces efficiently an optimal solution for instances of size less than or equal to 100. For larger instances, the performance depends on the value of r and thus on the size of the interval where the time delays are drawn. This leads to make the distinction between three groups of instances:

– The sparse instances obtained when $r \leq 1$: since the size of the interval of the time delays is large, the probability to have more distinct values is high. The fact the probability that the difference between the time delays of two successive

TABLE 8. Performance of *BB5* for instance classes ($1.5 \leq r \leq 4$).

r	4		3.5		3		2.5		2		1.5	
n	USI	Time	USI	Time	USI	Time	USI	Time	USI	Time	USI	Time
20	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00
40	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0	0.00
60	0	0.00	0	0.00	0	0.01	0	1.16	0	0.02	0	0.01
80	0	7.60	1	0.01	0	0.04	1	1.64	0	15.18	0	0.00
100	0	0.20	2	0.01	1	3.18	4	16.56	3	11.62	0	1.10
120	2	0.02	6	0.01	1	0.01	5	0.01	7	0.01	3	0.01
150	3	0.03	4	0.04	4	0.03	7	3.17	7	1.29	7	0.03
160	4	0.03	6	0.04	6	27.56	7	0.28	6	0.30	4	0.71
180	7	0.05	7	0.06	6	0.10	8	80.41	7	4.08	4	10.10
200	9	0.07	4	0.14	8	0.16	7	0.52	10	0.00	7	0.06
250	6	0.14	8	0.15	10	0.00	10	0.00	10	0.00	5	2.04
300	10	0.00	10	0.00	10	0.00	10	0.00	10	0.00	6	16.36

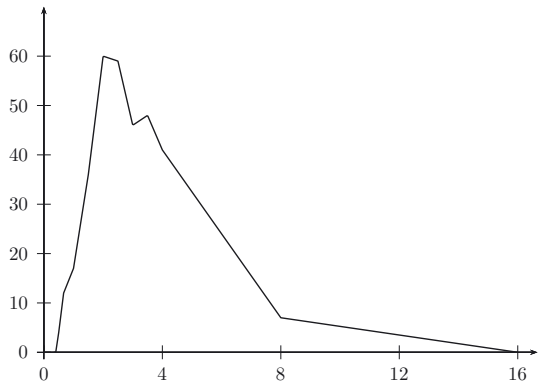

FIGURE 1. Total unsolved instances as a function of r.

jobs is greater than 2 is higher makes these classes easier to solve. For these class instances, the average number of visited nodes is very small. Any subsequence not starting with the biggest time delay job is rapidly ignored in the search tree. This is the main reason why the number of nodes explored is relatively small for these instances.

– The dense instances obtained when $r > 4$: The larger is the value of r, the larger is the number of jobs with the same time delays. Thus, the statement of Lemma 2.1 is frequently invoked in the course of the execution of the branch and bound algorithm. This means that the number of internal nodes in the search tree might be significantly reduced since we are only branching on distinct values of the time delays. As a consequence, the size of the search tree is reduced and the branch and bound algorithm converges to the optimal solution more quickly.

– Finally, we are left with the third group composed with instances that are not sparse neither dense ($2 \leq r \leq 4$). These instance classes seem to be the most difficult to solve since they generate a huge number of nodes to explore.

4. Conclusion

In this paper, we designed a branch and bound algorithm for the two-machine flowshop unit-time operations problem with time delay considerations in order to minimize the makespan. We included in this algorithm several new lower bounds and upper bounds, dominance rules, and test procedures that detect whether a partial solution is feasible. The experimental study we performed shows that our algorithm generates efficiently the optimal solution for instances of the problem up to $n = 100$ jobs. It is interesting to note the positive impact of the different ingredients we developed in this paper to speed up the branch and bound algorithm. Indeed, without them, we noticed through the different simulation we performed that the instances that can be solved can hardly go beyond $n = 20$ jobs.

For further research, it would be interesting to design better lower and upper bounds, and dominance rules in order to increase the size of instances that can be solved within a reasonable amount of time.

Acknowledgements. The authors would like to thank anonymous referee for helpful comments which improved the presentation of this paper. This research is funded for Djamal Rebaine by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] E. Balas and P. Toth, Branch and bound methods, chapter 10, in *The traveling salesman problems: a guided tour of Combinatorial Opt.*, edited by E.L. Lawler. John Wiley (1985).
[2] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms*. McGraw-Hill (1990).
[3] S. French, Sequencing and scheduling: an introduction to the mathematics of the job-shop. Series: *Math. and its Appl.* Ellis Horwood Ltd. (1982).
[4] T. Gonzalez and S. Sahni, Flow shop and job shop schedules: complexity and approximations. *Oper. Res.* **26** (1978) 36–52.
[5] S. Johnson, Optimal two- and three-stage production with set-up times included. *Nav. Res. Log. Quart.* **1** (1954) 61–68.
[6] A. Munier-Kordon and D. Rebaine, Polynomial time algorithms for the UET permutation flowshop problem with time delays. *Comput. Oper. Res.* **35** (2008) 525–537.
[7] V.J. Rayward-Smith and D. Rebaine, Analysis of heuristics for the UET two-machine flow shop. *Comput. Oper. Res.* **35** (2008) 3298-3310.
[8] D. Rebaine, Permutation shop vs. non permutation flow shop with delays. *J. Comput. Industrial Engrg.* **48** (2005) 357–362.
[9] W. Yu, H. Hoogeveen and J.K. Lenstra, Minimizing makespan in a two-machine flow with delays and unit-time operations is NP-hard. *J. Schedul.* **7** (2004) 333–348.
[10] W. Yu, *The two-machine flow shop problem and the one-machine total tardiness problem*. Ph.D. thesis, Eindhoven University of Technology, The Netherlands (1996).