Proceedings of 2017 4th International Conference on
Control, Decision and Information Technologies (CoDIT'17) / April 5-7, 2017, Barcelona, Spain

A branch-and-bound algorithm for the two-machine flow-shop problem with time delays

Mohamed Amine Mkadem, Aziz Moukrim and Mehdi Serairi
Sorbonne universités, Université de technologie de Compiègne, CNRS,
UMR 7253 Heudiasyc - CS 60 319 - 60 203 Compiègne cedex.
Email: {mohamed-amine.mkadem, aziz.moukrim, mehdi.serairi}@hds.utc.fr

***Abstract*—We address the flow-shop scheduling problem with two machines and time delays in order to minimize the makespan, i.e, the maximum completion time. We propose an exact algorithm based on a branch-and-bound enumeration scheme, for which we introduce a heuristic method based on a local search technique and three dominance rules. Finally, we present a computer simulation of the branch-and-bound algorithm, which was carried out on a set of 360 instances. The results show that our branch-and-bound method outperforms the state of the art exact method.**

Keywords: Two-machine flow-shop, time delays, makespan, exact method, branch-and-bound.

I. Introduction

In this paper, we tackle the flow-shop scheduling problem with two machines and time delays, denoted by $F2|l_j|C_{max}$. Let us consider an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$, where:

- $J = \{1, 2, ..., n\}$ is a set of n jobs.
- p_1 and p_2 are the vectors of processing times on the first and the second machine, respectively.
- l is the vector of time delays.

Each job j has two operations $O_{1,j}$ and $O_{2,j}$. The first operation $O_{1,j}$ must be executed during $p_{1,j}$ time units on the first machine M_1. The second operation $O_{2,j}$ has to be executed during $p_{2,j}$ time units on the second machine M_2. A feasible schedule is such that at most one operation is processed at a time on a given machine. In addition, the operations are executed without preemption, where interruption and switching of operations are not allowed. Finally, for each job j in J a time delay of at minimum l_j time units must separate the end of operation $O_{1,j}$ and the start of $O_{2,j}$. The objective consists in finding a feasible schedule that minimizes the makespan, usually denoted by C_{max}.

$F2|l_j|C_{max}$ is NP-hard in the strong sense even with unit-time operations [12]. Specific cases were discussed in the literature and were proved to be NP-hard in the strong sense. For example, we recall the following problems: $F2|p_{1,j} = p_{2,j}, l_j|C_{max}$ [4], $F2|p_{1,j} = p_{2,j} = 1, l_j|C_{max}$ [12] and $F2|p_{1,j} = p_{2,j}, l_j \in \{0, T\}|C_{max}$ [11]. Moreover, [3] tackled the preemptive flow-shop problem $F2|pmtn, l_j|C_{max}$ and showed that this problem is NP-complete. However, there exist well solvable cases: the well-known $F2||C_{max}$ problem where the time delays are equal to zero [5], $F2\pi|l_j|C_{max}$ [6] where feasible schedules have the same processing order on both machines and $F2|p_{i,j} = 1, l_j \in \{0, T\}|C_{max}$ for which a polynomial-time algorithm was proposed by [11].

As far as we know, the main contributions for $F2|l_j|C_{max}$ are those of [3] and [11], in which a set of lower bounds was proposed. Heuristic approaches were also investigated by [3]. Precisely, [3] introduced four constructive heuristics and a Tabu Search algorithm. It should be noted that [3] implemented an exact method based on the branch-and-bound method of [1], which was originally made for the job-shop problem. Indeed, each $F2|l_j|C_{max}$ instance can be modeled as a job-shop instance with n jobs and $n + 2$ machines, in which each job j in J must be executed on three machines. j must be assigned first to machine 1 during $p_{1,j}$ time units. Then, it is executed during l_j time units on machine $j + 1$. Finally, j is scheduled on the machine $n + 2$ during $p_{2,j}$ time units. The implemented version of [3] consists in applying all of his lower bounds and approximation procedures at the root node of the branch-and-bound method of [1].

The objective of this paper consists in proposing a branch-and-bound algorithm for $F2|l_j|C_{max}$. We introduce a set of lower bounds, an upper bound and three dominance rules.

The remainder of this paper is organized as follows. In Section 2, we provide a detailed description of the branch-and-bound algorithm. Computational experiments on a set of randomly generated instances are reported in Section 3. Finally, some concluding remarks are given in Section 4.

II. The Branch-and-Bound Algorithm

In what follows, the makespan value of a schedule S is denoted by $C_{max}(S)$ and the optimal makespan value of an instance I is given by $C^*_{max}(I)$. Furthermore, the starting time of $O_{k,j}$ in a schedule S is given by $t_{k,j}(S)$, j in J; k in $\{1, 2\}$ and J_σ represents the set of jobs that constitute a job sequence σ. Moreover, we denote by Ω_σ the set of all schedules where the job sequence σ is fixed first on M_1.

In this section, we present an exact method for $F2|l_j|C_{max}$ based on a branch-and-bound enumeration scheme. We adopt the same branching scheme as in [8] and [9]. To that aim, let us introduce the following observation.

Observation 1: Let σ be a fixed job sequence on M_1 that contains all jobs of J. The schedules of Ω_σ in which the jobs are executed on M_2 according to the nondecreasing order of their arrival times (i.e. $t_{1,j}(S) + p_{1,j} + l_j$, S in Ω_σ) are dominant.

According to this observation, our branch-and-bound enumerates job sequences on M_1 as follows. At a given node N_{σ_1} of the search tree of the branch-and-bound, a partial job sequence σ_1 of $L_{N_{\sigma_1}}$ jobs is fixed on M_1, where $L_{N_{\sigma_1}}$ is the level of the node N_{σ_1}. Note that the level of the root node is zero, one for the root sons, and so on. In order to reduce the number of explored nodes in the search tree, we invoke at each node N_{σ_1} the following features:

- A preprocessing procedure.
- Several lower bounds.
- An upper bound based on a local search technique applied on the current sub-sequence.
- Dominance rules.

If these features fail to prune the current node N_{σ_1}, then a *son node* is created from N_{σ_1} for each unscheduled job j where j is fixed after σ_1. Note that we adopt the depth-first node selection strategy.

A. *Preprocessing procedure*

We introduce here a preprocessing procedure based on [2] that aims to fix some precedence relationships between jobs on each machine. In fact, a quick increase of the fixed set of precedence relationships between jobs is important for the quality of the branch-and-bound method for many reasons. At first, we can limit the branching scheme by discarding nodes that contradict with the determined relationships. Moreover, the lower bounds could be enhanced by exploiting the additional information.

Interestingly, in each partial schedule S obtained at a given node N_{σ_1} of the search tree, we define for each operation of job j in J on M_k two sets of jobs $\varphi_{k,j}$ and $\psi_{k,j}$, k in $\{1,2\}$. Note that $\varphi_{k,j}$ (resp. $\psi_{k,j}$) contains the jobs of J that precede (resp. follow) j on M_k.

The preprocessing method is based on the following lemma:

Lemma 1: ([2]) Let us consider a partial schedule S of an instance I of $F2|l_j|C_{max}$ that is obtained at the node N_{σ_1} and UB an upper bound on I. For each pair of jobs (i,j) in $(J\backslash\{J_{\sigma_1}\})^2$, if it holds that:

$$\sum_{k\in\varphi_{1,j}} p_{1,k} + p_{1,j} + l_j + p_{2,j} + p_{2,i} + \sum_{k\in\psi_{2,i}} p_{2,k} \geq UB,$$

then i must precede j on M_2. Moreover, if it is true that:

$$\sum_{k\in\varphi_{1,i}} p_{1,k} + p_{1,i} + p_{1,j} + l_j + p_{2,j} + \sum_{k\in\psi_{2,j}} p_{2,k} \geq UB,$$

then j must be executed before i on M_1.

B. *Lower bounds*

We incorporate in our branch-and-bound the most competitive lower bounds of the literature.

We start by the $O(n)$ basic lower bound that was proposed by [11].

$$LB_1 = \max\Big(\sum_{j=1}^{n} p_{1,j} + \min_{1\le j\le n}(l_j + p_{2,j}), \sum_{j=1}^{n} p_{2,j} + \min_{1\le j\le n}(l_j + p_{1,j}), \max_{1\le j\le n}(p_{1,j} + l_j + p_{2,j})\Big). \quad (1)$$

The second lower bound was introduced by [3]. In this lower bound, it is supposed that all jobs are executed at $t=0$ on M_1. We obtain then a single-machine scheduling problem with release dates $r_j = p_{1,j} + l_j$ and processing times $p_j = p_{2,j}$, j in J. If we denote by I_r the obtained instance, $LB_2 = C^*_{max}(I_r)$ is a valid lower bound on the $F2|l_j|C_{max}$ original instance. This lower bound can be determined in $O(n\log n)$-time by scheduling the jobs in a nondecreasing order of r_j, j in J. Similarly, if we consider the reversed instance in which the operations are done in the reverse order on each machine, we yield a symmetric lower bound called LB_3.

Before proceeding further, let us consider the following proposition.

Proposition 1: ([3]) Given an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$, if it holds that:

$$l_j \le \min_{1\le i\le n}(p_{1,i} + l_i),\ \forall j \in J, \quad (2)$$

then there exists a permutation schedule that is optimal for I.

The author of [3] introduced a lower bound based on the above proposition. A new instance $\bar{I} = (J, p_1, \bar{l}, p_2)$ is derived from the original instance I, where the time delays are relaxed by setting $\bar{l}_j = \min(l_j, \min_{1\le i\le n}(l_i + p_{1,i}))$, j in J. Obviously, $\bar{I}$ verifies condition (2). Therefore, there exists a permutation schedule that is optimal for $\bar{I}$, which can be determined in $O(n\log n)$-time using Mitten algorithm [6]. As consequence, $LB_4 = C^*_{max}(\bar{I})$ is a valid lower bound on I.

In the following, we recall two lower bounds of [7] for $F2|l_j|C_{max}$. These latter are based on the next proposition.

Proposition 2: ([11]) Given an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$, if it holds that:

$$l_j \le \min_{1\le i\le n}(l_i + \max(p_{1,i}, p_{2,i})),\ \forall j \in J, \quad (3)$$

then there exists a permutation schedule that is optimal for I.

From an instance I of $F2|l_j|C_{max}$, we derive a new instance $\tilde{I}(J, p_1, \tilde{l}, p_2)$, where $\tilde{l}_j = \min(l_j, \min_{1\le i\le n}(l_i + \max(p_{1,i}, p_{2,i})))$. Since $\tilde{I}$ verifies condition (3), $LB_5 = C^*_{max}(\tilde{I})$ is a valid lower bound on I. This lower bound can be computed in $O(n\log n)$-time using Mitten algorithm [6].

Interestingly, the last lower bound can be improved by considering the following observation.

Observation 2: Let $I = (J, p_1, l, p_2)$ and $I' = (J', p'_1, l', p'_2)$ be two $F2|l_j|C_{max}$ instances, where $J' \subset J$ and for all j in J', $p'_{1,j} \le p_{1,j}$, $p'_{2,j} \le p_{2,j}$ and $l'_j \le l_j$. It holds that $C^*_{max}(I) \ge C^*_{max}(I')$. Thus, a lower bound on I' is a valid lower bound on instance I.

Proof: We observe that from an optimal schedule of instance I, we draw a feasible schedule S' of instance I' such that $C_{max}(S') \le C^*_{max}(I)$. ■

Given an instance I of $F2|l_j|C_{max}$, we define a lower bound called LB_6 of [7] that consists in invoking LB_5 on n sub-instances of I. Starting by I, the next sub-instance is made after removing the job with the minimum value of $l_j + \max(p_{1,j}, p_{2,j})$, j in J, from the current instance.

C. Time delays adjustments

In this section, we focus on the partial schedule obtained at a given node N_{σ_1} of the search tree. First, we show how the time delays of the scheduled jobs are updated. Then, we describe how the previous lower bounds are invoked in the internal nodes of the branch-and-bound search tree. To that aim, let $\sigma = \sigma_1 \oplus \sigma_2$ be a job sequence of all jobs on M_1 where σ_1 is a fixed sub-sequence and σ_2 is an arbitrary one of $J \setminus J_{\sigma_1}$. Let β_j be the starting time on M_2 of j in J_{σ_1} if the jobs of J_{σ_1} are scheduled on M_2 with respect to their arrival times.

Proposition 3: At a given node N_{σ_1} and for all S in Ω_{σ_1}, the time delay of each job j in J can be updated as follows:

$$l_j^{\sigma_1} = \begin{cases} \beta_j - t_{1,j}(S) - p_{1,j}, & \text{if } j \in J_{\sigma_1} \\ l_j, \text{ otherwise.} \end{cases} \quad (4)$$

Proof: Let us consider a node N_{σ_1} and a schedule S of Ω_{σ_1}. We show that $t_{2,j}(S) \geq \beta_j$, for all j in J_{σ_1}. Therefore, the time delay that is observed by job j is at least $\beta_j - t_{1,j}(S) - p_{1,j}$. ■

In order to use the above described lower bounds inside the search tree, we consider the following two instances.

- $I^u_{\sigma_1} = (J, p_1, l^{\sigma_1}, p_2)$.
- $I^r_{\sigma_1} = (J_{\sigma_2}, p_1, l, p_2)$.

From Observation 1 and Proposition 3, the following corollary holds:

Corollary 1: Given a valid lower bound LB on $F2|l_j|C_{max}$ instances and a node N_{σ_1} of the search tree, it yields that for each schedule S of Ω_{σ_1}:

$$C_{max}(S) \geq \max(LB(I^u_{\sigma_1}), \sum_{j \in J_{\sigma_1}} p_{1,j} + LB(I^r_{\sigma_1})). \quad (5)$$

D. Upper bounds

In the following, we describe a new heuristic algorithm, which is intended to be used on each node of the branch-and-bound search tree. This heuristic is based on a local search exploration of the partial sub-sequences. At a given node N_{σ_1} of the search tree, we complete the fixed sub-sequence σ_1 to obtain a complete job sequence σ by invoking the constructive heuristics of [3]. From σ, we schedule on M_2 the jobs according to the nondecreasing order of their arrival times as in Observation 1. Then by considering the obtained job sequence on M_2, σ is updated according to the arrival times of the jobs. This process is iterated until no amelioration is detected on the makespan between two successive iterations. A pseudo-code of the procedure is described in Algorithm 1.

Algorithm 1 Local search exploration

Require: Sub-sequence σ_1
Ensure: UB

```
Using the heuristics of [3], complete the job sequence σ1 in order to have a schedule S of Ω_σ1. Let σ be the obtained job sequence on M1.
Complete S in such a way that the second operations of all jobs are scheduled according to Observation 1. Let π be the obtained job sequence on M2.
UB ← +∞ ;
while UB > C_max(S) do
    UB ← C_max(S);
    Starting from π, determine the job sequence σ on M1 by scheduling the jobs according to Observation 1.
    Starting from σ, determine the job sequence π on M2 by scheduling the jobs according to Observation 1.
    Update the feasible schedule S.
end while
```

E. Dominance rules

We introduce here three new dominance rules that allow to discard nodes from additional expansions in order to reduce the computational burden of the proposed branch-and-bound algorithm. The three dominance rules are given by the following three lemmas.

Lemma 2: Consider an instance I of $F2|l_j|C_{max}$ and two jobs i and j where $p_{1,j}+l_j \leq p_{1,i}+l_i$, $l_i \leq l_j+p_{2,j}$ and $p_{1,j} \leq p_{2,j}$. If there exists a schedule where i and j are adjacent on M_1, then j must be executed first.

Proof: The aim of this proof is to construct a schedule $S^{'}$ of $\Omega_{(\sigma_1 \oplus j \oplus i)}$ from an arbitrary schedule S of $\Omega_{(\sigma_1 \oplus i \oplus j)}$ in a way that $C_{max}(S) = C_{max}(S^{'})$. We identify two cases:

Case 1: j precedes i on M_2
$S^{'}$ is derived from S by simply interchanging i and j on M_1 and by keeping the same execution order on M_2, where $t_{2,k}(S^{'}) = t_{2,k}(S)$, for all k in J. Therefore, it is true that $t_{1,i}(S) = t_{1,j}(S^{'})$, $t_{1,i}(S^{'}) + p_{1,i} = t_{1,j}(S) + p_{1,j}$ and $t_{1,j}(S) + p_{1,j} + l_j \leq t_{2,j}(S)$. Since $t_{1,k}(S^{'}) = t_{1,k}(S)$ for all k in $J\backslash\{i,j\}$, each job of $J\backslash\{i,j\}$ is available on M_2 before its starting time. Moreover, it holds that:

$$\begin{aligned} t_{1,j}(S^{'}) + p_{1,j} + l_j &\leq t_{1,j}(S) + p_{1,j} + l_j \\ &\leq t_{2,j}(S) = t_{2,j}(S^{'}) \end{aligned}$$

and

$$t_{1,i}(S^{'}) + p_{1,i} + l_i = t_{1,j}(S) + p_{1,j} + l_i.$$

Since $l_i \leq l_j + p_{2,j}$, we get:

$$\begin{aligned} t_{1,i}(S^{'}) + p_{1,i} + l_i &\leq t_{1,j}(S) + p_{1,j} + l_j + p_{2,j} \\ &\leq t_{2,j}(S) + p_{2,j} \\ &\leq t_{2,i}(S^{'}). \end{aligned}$$

From the above remarks, we conclude that all jobs are available for processing on M_2 before their starting times in $S^{'}$.

Case 2: i precedes j on M_2
$S^{'}$ is derived from S by simply interchanging i and j on M_1

and by shifting left j on M_2 to locally precede i in a way that $t_{2,j}(S^{'}) = t_{2,i}(S)$. Therefore, it is true that $t_{1,i}(S) = t_{1,j}(S^{'})$, $t_{1,i}(S^{'})+p_{1,i} = t_{1,j}(S)+p_{1,j}$ and $t_{1,i}(S)+p_{1,i}+l_i \leq t_{2,i}(S)$. Since $t_{1,k}(S^{'}) = t_{1,k}(S)$ and $t_{2,k}(S^{'}) \geq t_{2,k}(S)$ for all k in $J\backslash\{i,j\}$, each job of $J\backslash\{i,j\}$ is available on M_2 before its starting time. Since $p_{1,j}+l_j \leq p_{1,i}+l_i$, it holds that:

$$\begin{aligned} t_{1,j}(S^{'})+p_{1,j}+l_j &= t_{1,i}(S)+p_{1,j}+l_j \\ &\leq t_{1,i}(S)+p_{1,i}+l_i \\ &\leq t_{2,i}(S) = t_{2,j}(S^{'}). \end{aligned}$$

Moreover, we have that:

$$\begin{aligned} t_{1,i}(S^{'})+p_{1,i}+l_i &= t_{1,j}(S^{'})+p_{1,j}+p_{1,i}+l_i \\ &= t_{1,i}(S)+p_{1,j}+p_{1,i}+l_i \\ &\leq t_{2,i}(S)+p_{1,j}. \end{aligned}$$

Since $p_{1,j} \leq p_{2,j}$, we get:

$$\begin{aligned} t_{1,i}(S^{'})+p_{1,i}+l_i &\leq t_{2,i}(S)+p_{2,j} = t_{2,j}(S^{'})+p_{2,j} \\ &\leq t_{2,i}(S^{'}). \end{aligned}$$

From the above remarks, we conclude that all jobs are available for processing on M_2 before their starting times in $S^{'}$. ■

Lemma 3: Consider a valid job sequence σ on M_1 that is composed from a fixed sub-sequence σ_1 and followed by an arbitrary one called σ_2. If there are two jobs i, j in J_{σ_2} where $p_{1,j} \leq p_{1,i}$, $p_{2,i} \leq p_{2,j}$, $p_{1,j}+l_j \leq p_{1,i}+l_i$ and $p_{2,j}+l_j \geq p_{2,i}+l_i$, then j must precede i on M_1.

Proof: Let S be an arbitrary schedule of $\Omega_{\sigma_1\oplus(i)}$ where job i is executed directly after σ_1 on M_1. From S, we can derive a schedule $S^{'}$ of $\Omega_{\sigma_1\oplus(j)}$ without increasing the makespan. We identify two cases:

Case 1: j precedes i on M_2
In this case, $S^{'}$ is derived from S by simply interchanging i and j on M_1 and by keeping the same execution order on M_2, where $t_{2,k}(S^{'}) = t_{2,k}(S)$, for all k in J. Then, it is obvious that $t_{1,i}(S) = t_{1,j}(S^{'})$ and $t_{1,i}(S^{'})+p_{1,i} = t_{1,j}(S)+p_{1,j}$. Since $p_{1,j} \leq p_{1,i}$, it is true that $t_{1,k}(S^{'}) \leq t_{1,k}(S)$, for all k in $J\backslash\{i\}$. Therefore, each job of $J\backslash\{i\}$ is available on M_2 before its starting time. Moreover, it holds that:

$$t_{1,i}(S^{'})+p_{1,i}+l_i = t_{1,j}(S)+p_{1,j}+l_i.$$

Since $l_i \leq l_j+p_{2,j}$, we get:

$$\begin{aligned} t_{1,i}(S^{'})+p_{1,i}+l_i &\leq t_{1,j}(S)+p_{1,j}+l_j+p_{2,j} \\ &\leq t_{2,j}(S)+p_{2,j} \\ &\leq t_{2,i}(S^{'}). \end{aligned}$$

From the above remarks, we conclude that all jobs are available for processing on M_2 before their starting times in $S^{'}$.

Case 2: i precedes j on M_2
In this case, $S^{'}$ is derived from S by simply interchanging i and j on M_1 and on M_2, where the same jobs are executed between i and j as in S. Then, it holds that $t_{1,i}(S) = t_{1,j}(S^{'})$, $t_{1,i}(S^{'})+p_{1,i} = t_{1,j}(S)+p_{1,j}$, $t_{2,i}(S) = t_{2,j}(S^{'})$ and $t_{2,i}(S^{'})+p_{2,i} = t_{2,j}(S)+p_{2,j}$. For each job k in $J\backslash\{i,j\}$, we observe that $t_{1,k}(S^{'}) \leq t_{1,k}(S)$ since $p_{1,j} \leq p_{1,i}$ and $t_{2,k}(S^{'}) \geq t_{2,k}(S)$ since $p_{2,j} \geq p_{2,i}$. Therefore, each job of $J\backslash\{i,j\}$ is available on M_2 before its starting time. Since $p_{1,j}+l_j \leq p_{1,i}+l_i$, it holds that:

$$\begin{aligned} t_{1,j}(S^{'})+p_{1,j}+l_j &= t_{1,i}(S)+p_{1,j}+l_j \\ &\leq t_{1,i}(S)+p_{1,i}+l_i \\ &\leq t_{2,i}(S) \\ &\leq t_{2,j}(S^{'}). \end{aligned}$$

Moreover, since $l_i \leq l_j+p_{2,j}-p_{2,i}$, we have that:

$$\begin{aligned} t_{1,i}(S^{'})+p_{1,i}+l_i &= t_{1,j}(S)+p_{1,j}+l_i \\ &\leq t_{1,j}(S)+p_{1,j}+l_j+p_{2,j}-p_{2,i} \\ &\leq t_{2,j}(S)+p_{2,j}-p_{2,i} = t_{2,i}(S^{'}). \end{aligned}$$

From the above remarks, we conclude that all jobs are available for processing on M_2 before their starting times in $S^{'}$. ■

Let us consider the following notation. We denote by $I(S^{'},S,t,C)$ the difference between the total idle time value observed on M_2 from time t to the instant C in $S^{'}$ and that in S.

Lemma 4: Let us consider two partial schedules S and $S^{'}$ that are obtained at two nodes N_{σ_1} and $N_{\sigma_1^{'}}$ of the search tree, respectively, such that $J_{\sigma_1} = J_{\sigma_1^{'}}$. Moreover, let C be the maximum makespan value observed on all schedules of Ω_{σ_1} and $\Omega_{\sigma_1^{'}}$. If it holds that:

$$I(S^{'},S,t,C) \geq 0,\ \forall t \in \{0,...,C\}, \qquad (6)$$

then the node N_{σ_1} can be fathomed. In this case, we say that $\sigma_1^{'}$ dominates σ_1.

Proof: It should be noted that if (6) holds then $C_{max}(S^{'}) \leq C_{max}(S)$. Otherwise, it contradicts with the assumption that $I(S^{'},S,C_{max}(S),C) \geq 0$. In this proof, we proceed as follows. We schedule iteratively the jobs of $J\backslash J_{\sigma_1}$ in S and in $S^{'}$, then we prove that (6) is true for the obtained schedules. Let S_j (resp. $S_j^{'}$) be the partial schedule obtained from S (resp. $S^{'}$) after scheduling the job j of $J\backslash J_{\sigma_1}$ at the position $|\sigma_1|+1$ on M_1. We denote by $\gamma_j = t_{1,j}(S_j)+p_{1,j}+l_j = t_{1,j}(S_j^{'})+p_{1,j}+l_j$ the arrival time of job j on M_2 and τ (resp. $\tau^{'}$) represents the minimal time such that the total idle time observed between γ_j and τ (resp. $\tau^{'}$) on M_2 in S (resp. $S^{'}$) is equal to $p_{2,j}$. The job sequences on M_2 of S_j and $S_j^{'}$ are obtained as follows. We start by removing all jobs that are scheduled between γ_j and τ (resp. $\tau^{'}$) on M_2 in S (resp. $S^{'}$). Then, we process j followed by the removed jobs from S (resp. $S^{'}$) continuously such that they end their processing at time τ (resp. $\tau^{'}$) on M_2 in S_j (resp. $S_j^{'}$). Since j is processed after γ_j on M_2, we observe that the idle time value of S_j (resp. $S_j^{'}$) is equal to the idle time value of S (resp. $S^{'}$) minus $p_{2,j}$ time units. Therefore, S_j and $S_j^{'}$ verify that:

$$I(S_j^{'},S_j,t,C) = I(S^{'},S,t,C) \geq 0,\ \forall t \in \{0,...,\gamma_j\}.$$

Moreover, we notice that the schedule is the same in S_j and in S (resp. in $S_j^{'}$ and in $S^{'}$) after the instant $\max(\tau,\tau^{'})$. Therefore, it holds that:

$$I(S_j^{'},S_j,t,C) \geq 0,\ \forall t \in \{\max(\tau,\tau^{'}),...,C\}.$$

Interestingly, if the second machine for one of the two schedules $S_j^{'}$ and S_j is not idle during an interval $[t_1, t_2]$, then $I(S_j^{'}, S_j, t, C)$ consists in determining the difference between a constant and a non-increasing value for all t in $[t_1, t_2]$. Therefore, $I(S_j^{'}, S_j, t, C)$ is monotonic in the interval $[t_1, t_2]$. Consequently, we prove that $I(S_j^{'}, S_j, t, C)$ is monotonic in the interval $[\gamma_j, \max(\tau, \tau^{'})]$. In addition, since $I(S_j^{'}, S_j, \gamma_j, C) \geq 0$ and $I(S_j^{'}, S_j, \max(\tau, \tau^{'}), C) \geq 0$, it holds that:

$$I(S_j^{'}, S_j, t, C) \geq 0, \ \forall t \in \{\gamma_j, ..., \max(\tau, \tau^{'})\}.$$

This process is reiterated for each job of $J \backslash J_{\sigma_1}$. ■

In order to apply the above dominance rule in an efficient way, we implement the no-good list technique of [10]. The no-good list consists in recording the most dominant job sequence for each set of jobs. At first, the no-good list is empty. Then, the dominant job sequences are stored while exploring internal nodes of the search tree. For each dominant job sequence σ, we add the following data: the set of jobs J_σ, the job sequence σ and the idle time periods on the second machine.

Now suppose that a new node N_σ is developed by our branch-and-bound method. At first, we verify the existence of a job sequence π in the no-good list such that $J_\pi = J_\sigma$. If no job sequence is found, then σ is stored in the no-good list and the node N_σ is developed. Otherwise, we verify the existence of a dominance relationship between the two job sequences π and σ where three cases are possible:

- π does not dominate σ and then the node N_σ is developed.
- π dominates σ and then N_σ is pruned.
- σ dominates π and then the no-good list is updated and N_σ is developed.

For a better convergence of the branch-and-bound method, we apply a neighborhood function that aims to improve the job sequence σ. It consists in scheduling the jobs of J_σ on M_2 according to the nondecreasing order of their arrival times. Then by considering the obtained job sequence on M_2, σ is updated according to the arrival times of the jobs. This process is iterated until no amelioration is detected on the makespan between two successive iterations. If the obtained job sequence dominates the older one, then the node N_σ can be pruned.

III. Computational results

We present in this section the computational results of the branch-and-bound algorithm. We performed the tests on a set of six classes of instances that was proposed by [3]. For each class, the processing times on M_1 and M_2 and the time delays were randomly generated between $[1..\alpha]$, $[1..\beta]$ and $[1..\gamma]$, respectively. The details of these classes are provided in Table I.

TABLE I. Classes generation

	[3]					
	A	B	C	D	E	F
α	100	100	100	200	100	200
β	100	100	100	200	200	100
γ	100	200	500	100	100	100

The number of jobs tested for each class was taken to be equal to $10, 30, 50, 100, 150$ and 200. For each pair of class and number of jobs, 10 instances were randomly generated. All algorithms were coded in C++ and compiled under CentOS 6.6. Moreover, the implementation of the no-good list was done using a hash table. We used the code of Bernstein (see http://burtleburte.net/bob/hash/doobs.html) since his methods are efficient in term of computational time. The experiments were conducted on an Intel(R) Xeon(R) @ 2.60GHz processor except for the literature exact method that was done on a PC486 with a clock at 33 Mhertz running under MS_DOS.

In the following, we evaluate the performance of the different components of the branch-and-bound algorithm. Hereafter, we denote by LB_{best} the maximum lower bound value of LB_1, LB_2, LB_3 and LB_6. We implemented three versions of the branch-and-bound algorithm:

- $B\&B_{v1}$: Only LB_{best} and the preprocessing procedure are applied.
- $B\&B_{v2}$: The heuristic method, LB_{best} and the preprocessing procedure are invoked in each node.
- $B\&B_{v3}$: All the proposed components are activated in each node.

We compare the performance of these versions to the branch-and-bound algorithm of [3] denoted $B\&B_{Dell}$. Note that we set a time limit of 2000 seconds for each instance. We summarize in Table II for each version the number of unsolved instances within the given time limit (USI), the average number of the explored nodes for the solved instances (Nodes) and the average computational time in seconds for the solved instances (Time).

From Table II, we made the following observations:

- $B\&B_{v3}$ provides a better performance than $B\&B_{v1}$ and $B\&B_{v2}$ with only 2 unsolved instances.
- $B\&B_{v2}$ outperforms $B\&B_{v1}$. The use of the heuristic method allows us to solve 5 additional instances for Class C.
- The same remark is observed when we incorporate the dominance rules within the branch-and-bound. The number of unsolved instances drops by 3 instances comparing to $B\&B_{v2}$.

Interestingly, $B\&B_{v3}$ outperforms $B\&B_{Dell}$ with only 2 unsolved instances compared to 4 for $B\&B_{Dell}$. These two methods exhibit the same performance on all classes except for Class C where $B\&B_{v3}$ solves two more instances.

IV. Conclusion

In this paper, we presented an exact method based on a branch-and-bound scheme for $F2|l_j|C_{max}$. First, we proposed a heuristic method and developed three dominance rules. Also, an experimental study of the exact algorithm was given. In particular, our branch-and-bound outperforms the state of the art exact method and solves 358 instances among 360 possible ones.

Future research needs to be focused on improving the performance of the proposed branch-and-bound by investigating

TABLE II. THE BRANCH-AND-BOUND ALGORITHM PERFORMANCE

		$B\&B_{Dell}$			$B\&B_{v1}$			$B\&B_{v2}$			$B\&B_{v3}$		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
Class A	10	0	0.00	0.08	0	0.00	0.04	0	0.00	0.01	0	0.00	0.01
	30	0	0.00	0.33	1	0.00	0.03	1	0.00	0.01	0	3128.00	8.98
	50	0	1.00	25.60	0	0.00	0.03	0	0.00	0.02	0	0.00	0.01
	100	0	1.00	54.13	0	0.00	0.02	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	4.51	0	0.00	0.03	0	0.00	0.04	0	0.00	0.03
	200	0	0.00	11.74	0	0.00	0.05	0	0.00	0.06	0	0.00	0.05
	AVG	0	0.33	16.06	1	0.00	0.03	1	0.00	0.02	0	421.33	1.51
Class B	10	0	8.40	13.62	0	73.30	0.02	0	66.70	0.03	0	14.10	0.01
	30	0	1.00	3.81	0	102.30	0.27	0	2.40	0.02	0	2.30	0.01
	50	0	1.00	13.79	0	7.60	0.02	0	0.10	0.01	0	0.10	0.01
	100	0	1.00	46.52	0	7.40	0.08	0	0.00	0.02	0	0.00	0.02
	150	0	1.00	330.00	0	0.00	0.04	0	0.00	0.04	0	0.00	0.04
	200	0	1.00	695.61	0	0.00	0.07	0	0.00	0.07	0	0.00	0.07
	AVG	0	2.23	183.90	0	31.77	0.08	0	11.53	0.03	0	2.75	0.03
Class C	10	0	10.20	7.20	0	70.10	0.03	0	52.30	0.02	0	30.00	0.02
	30	2	23.60	19.5	3	23429.71	97.22	2	837.62	3.46	0	7232.10	44.97
	50	1	1.00	8.57	2	9483.37	137.39	1	16.66	0.08	1	14.66	0.08
	100	0	3.10	140.72	1	333.44	10.64	0	0.60	0.04	0	0.20	0.02
	150	0	1.00	285.24	2	574.00	35.91	0	0.00	0.05	0	0.00	0.04
	200	1	1.00	915.59	1	315.55	22.44	1	0.00	0.01	1	0.00	0.08
	AVG	4	6.65	229.47	9	5701.03	50.60	4	151.19	0.61	2	1213.99	7.54
Class D	10	0	0.00	0.08	0	0.10	0.01	0	0.00	0.01	0	0.00	0.01
	30	0	0.00	0.15	0	0.00	0.01	0	0.00	0.02	0	0.00	0.01
	50	0	0.00	0.23	0	0.00	0.02	0	0.00	0.02	0	0.00	0.01
	100	0	0.00	1.18	0	0.20	0.02	0	0.20	0.03	0	0.10	0.02
	150	0	0.00	2.87	0	0.00	0.02	0	0.00	0.03	0	0.00	0.02
	200	0	1.00	650.65	0	0.00	0.03	0	0.00	0.03	0	0.00	0.03
	AVG	0	0.17	109.19	0	0.05	0.02	0	0.03	0.02	0	0.01	0.02
Class E	10	0	0.00	0.56	0	0.00	0.02	0	0.00	0.04	0	0.00	0.03
	30	0	1.00	3.12	0	0.00	0.01	0	0.00	0.04	0	0.00	0.03
	50	0	0.00	0.29	0	0.20	0.02	0	0.10	0.02	0	0.00	0.04
	100	0	0.00	1.49	0	0.00	0.02	0	0.00	0.02	0	0.00	0.04
	150	0	0.00	3.31	0	0.00	0.03	0	0.00	0.03	0	0.00	0.05
	200	0	1.00	563.20	0	0.00	0.04	0	0.00	0.04	0	0.00	0.04
	AVG	0	0.33	95.32	0	0.03	0.02	0	0.01	0.03	0	0.00	0.04
Class F	10	0	0.00	0.06	0	2.60	0.02	0	2.00	0.04	0	0.90	0.02
	30	0	1.00	3.36	0	0.00	0.02	0	0.00	0.01	0	0.00	0.02
	50	0	0.00	0.21	0	0.00	0.01	0	0.00	0.02	0	0.00	0.03
	100	0	0.00	1.11	0	0.00	0.02	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	3.35	0	0.00	0.02	0	0.00	0.03	0	0.00	0.03
	200	0	0.00	29.35	0	0.00	0.04	0	0.00	0.04	0	0.00	0.06
	AVG	0	0.17	6.24	0	0.43	0.02	0	0.33	0.02	0	0.15	0.03

new dominance properties and developing lower bound methods. Moreover, new classes of instances should be investigated where the time delays are very large compared to processing times. Indeed, we noticed that the lower bounds perform badly in this case.

ACKNOWLEDGEMENTS

This work is carried out in the framework of the Labex MS2T, which was funded by the French Government, through the program Investments for the future managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02). It is also partially supported by the ATHENA project (ANR-13-BS02-0006-02).

REFERENCES

[1] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1):107 – 127, 1994.

[2] J. Carlier and E. Pinson. A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26(1-4):269 – 287, 1990.

[3] M. Dell'Amico. Shop problems with two machines and time lags. *Operations Research*, 44(5):777–787, 1996.

[4] M. Dell'Amico and R. J. M. Vaessens. In *Flow and open shop scheduling on two machines with transportation times and machine-independant processing times is NP-hard*. Dipartimento di Economia politica, Università di Modena, 1996.

[5] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[6] L. G. Mitten. Sequencing n jobs on two machines with arbitrary time lags. *Management Science*, 5(3):293–298, 1959.

[7] M. A. Mkadem, A. Moukrim, and M. Serairi. Lower bounds for the two-machine flow shop problem with time delays. In A. Fink, A. Fügenschuh, and M. J. Geiger, editors, *Operations Research Proceedings 2016*. Springer International Publishing, 2017.

[8] A. Moukrim, D. Rebaine, and M. Serairi. A branch and bound algorithm for the two-machine flowshop problem with unit-time operations and time delays. *RAIRO - Operations Research*, 48(2):235–254, 2014.

[9] M. K. Msakni, W. Khallouli, M. Al-Salem, and T. Ladhari. Minimizing the total completion time in a two-machine flowshop problem with time delays. *Engineering Optimization*, 48(7):1164–1181, 2016.

[10] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. In *Proceedings of 1993 IEEE Conference on Tools with Al (TAI-93)*, pages 48–55, Nov 1993.

[11] W. Yu. *The two-machine flow shop problem and the one-machine total tardiness problem*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1996.

[12] W. Yu, H. Hoogeveen, and J.K. Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard. *Journal of Scheduling*, 7(5):333 – 348, 2004.