Computers & Industrial Engineering 114 (2017) 22–30

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

# Minimizing total tardiness in a two-machine flowshop scheduling problem with availability constraint on the first machine

Ju-Yong Lee[a,*], Yeong-Dae Kim[b]

[a] System Technology Team, Semiconductor Business, Samsung Electronics Co., Ltd., Yongin-Si, Gyeonggi-Do 17113, Republic of Korea
[b] Department of Industrial Engineering, Korea Advanced Institute of Science and Technology, Yuseong-gu, Daejeon 34141, Republic of Korea

ARTICLE INFO

*Keywords:*
Scheduling
Flowshop
Machine availability constraint
Branch and bound algorithm
Total tardiness

ABSTRACT

This paper deals with a two-machine flowshop problem in which the machine at the first stage requires preventive maintenance activities that have to be started within a given cumulative working time limit after the previous maintenance. That is, a maintenance activity can be started at any time unless the cumulative working time after the end of the previous maintenance exceeds the given limit. For the problem with the objective of minimizing total tardiness, we develop dominance properties and lower bounds for this scheduling problem as well as a heuristic algorithm, and suggest a branch and bound algorithm in which these properties, lower bounds, and heuristic algorithm are used. Computational experiments are performed to evaluate the algorithm and the results are reported.

## 1. Introduction

Over the past decades, a large number of researchers have studied flowshop scheduling problems. In a typical flowshop problem, $m$ machines are arranged in a series and $n$ jobs visit the machines in the same order, that is, each job consists of $m$ operations and the $k$th operation of all jobs are processed on machine $k$ for $k = 1, \dots, m$. In most research on operations scheduling problems including flowshop problems, it is assumed that machines are continuously available at all times. However, this assumption oversimplifies the scheduling problems in real manufacturing systems since in reality there should be preventive maintenance, which has a significant impact on a variety of performance aspects such as productivity, reliability, and profitability. Note that delayed maintenance may lead to machine failure or deterioration in the quality of outputs. For these reasons, preventive maintenance tasks such as inspection, repair, replacement, cleaning, lubrication, adjustment and alignment are required to be performed for keeping machines in good condition and decrease the probability of machines' failures (Cui & Lu, 2017). Therefore, operations scheduling with maintenance is very important in operating manufacturing systems.

This study investigates a scheduling problem of minimizing total tardiness of jobs in a two-machine flowshop. In this problem, the machine in the first stage requires preventive maintenance within a given cumulative working time limit after the previous maintenance (since any delay of the maintenance activity increases the risk of machine failure significantly). That is, a maintenance activity can be (or should be) started at any time before the cumulative working time after the end of the previous maintenance exceeds a given time limit. This type of maintenance is called *flexible maintenance* since the start times of maintenance activities are not fixed but flexible. This scheduling problem can be denoted by $F2/fm/T$ according to the nomenclature of Graham, Lawler, Lenstra, and Rinnooy-Kan (1979), where $F2$, $fm$, and $T$ represent the two-machine flowshop, flexible maintenance, and total tardiness, respectively. In the literature, constraints on machines due to maintenance activities are treated as a machine availability constraint.

For scheduling problems with preventive and flexible maintenance tasks, a large number of studies have been performed on a single-machine problem with various objectives. Among others, Qi, Chen, and Tu (1999), Yang, Ma, Xu, and Yang (2011), and Mashkani and Moslehi (2016) present branch and bound (BnB) algorithms, and Akturk, Ghosh, and Gunes (2003) and Gurel and Akturk (2008) propose heuristic algorithms for minimizing total completion time, while Mosheiov and Sarig (2009) develop a dynamic programming (DP) algorithm and a heuristic algorithm for minimizing total weighted completion times. Also, Chen (2006) gives mixed binary-integer programming models and develops a heuristic algorithm for minimizing mean flow time, Chen (2008a, 2008b) presents mixed binary-integer programming models for minimizing the total tardiness and for minimizing makespan, respectively, while Sbihi and Varnier (2008) suggest a heuristic and a BnB algorithm for minimizing maximum tardiness. Recently, Luo, Cheng, and Ji (2015), Zhu, Li, and Zhou (2015) and Ying, Ju, and Chen (2016) consider problems with flexible maintenance and various scheduling

* Corresponding author.
*E-mail address:* jy7433.lee@samsung.com (J.-Y. Lee).

http://dx.doi.org/10.1016/j.cie.2017.10.004
Received 4 March 2017; Received in revised form 2 October 2017; Accepted 3 October 2017
Available online 05 October 2017
0360-8352/ 

measures.

For flowshop scheduling problems with machine availability constraints, most studies have focused on the makespan criterion (Ying, Chu, & Zuo, 2010). Also, the problems can be classified into two cases according to the characteristics of jobs: those of resumable and non-resumable jobs. In the resumable-job case, it is assumed that if a job is not completed before starting a maintenance activity, processing of the job can be continued after the maintenance. Unlike the resumable-job case, in the non-resumable job case, processing of a job that is preempted by a maintenance activity should be restarted from the beginning after the maintenance activity. In the following, we briefly review literature on two-machine flowshop problems of the three cases with the makespan measure.

Under the assumption of the resumable-job case, Lee (1997) proves that the problem is NP-hard when a maintenance activity is performed on only one machine. Also, Lee (1997) develops a pseudo-polynomial dynamic programming algorithm and two heuristic algorithms. One heuristic, which is for a case in which a maintenance activity is performed on machine 1, has a worst case error bound of 1/2 and the other heuristic, for a case with a maintenance activity performed on machine 2, has a worst case error bound of 1/3. Later, for the first case, Cheng and Wang (2000) propose a heuristic algorithm with a worst case error bound of 1/3, while Breit (2004) presents an improved heuristic with a worst case error bound of 1/4 for the second case. On the other hand, for problems in which multiple maintenance activities are performed, Błażewicz, Breit, Formanowicz, Kubiak, and Schmidt (2001) develop heuristic algorithms and Kubiak, Błażewicz, Formanowicz, Breit, and Schmidt (2002) propose a branch and bound (BAB) algorithm. In addition, Kubzin, Potts, and Strusevich (2009) present a 3/2-approximation algorithm for a problem with multiple maintenance activities on the first machine.

Under the assumption of the non-resumable job case, Allaoui, Artiba, Elmaghraby, and Riane (2006) consider a problem in which a maintenance activity is required on machine 1 only and develop a dynamic programming algorithm and a heuristic algorithm, Allaoui, Lamouri, Artiba, and Aghezzaf (2008) consider that one of the two machines should be maintained once before a given maximum allowed continuously working time and propose some optimal solution properties, while Yang, Hsu, and Kuo (2008) propose a heuristic algorithm to solve a problem in which a maintenance activity is required on each of the two machines before completing a given number of jobs. Also, Hnaien, Yalaoui, and Mhadhbi (2015) investigates two mixed-integer programming models and a BAB algorithm for the problem in which a maintenance activity is performed at a given start time. On the other hand, Espinouse, Formanowlcz, and Penz (1999, 2001), Wang and Cheng (2001) and Cheng and Liu (2003) consider the non-resumable job case in a special flowshop, *no-wait flowshop*, in which jobs have to be processed on machine 2 immediately after they are completed on machine 1 without waiting.

In this paper, we present an optimal-solution algorithm minimizing total tardiness in a two-machine flowshop scheduling problem in which the machine in the first stage requires preventive maintenance within a given cumulative working time limit after the previous maintenance. Here, it is assumed that: (1) the cumulative working time between two successive maintenance activities should not exceed a given time limit; (2) the maximum processing time of all jobs is shorter than the cumulative working time limit; (3) the time required to perform maintenance activities are the same and given; (4) all jobs are ready at the beginning of the scheduling horizon; and (5) all jobs are not resumable and preemption of jobs is not allowed. Note that since preemption is not permitted in the problem, one and only one schedule can be obtained from a permutation of jobs, or a sequence of jobs. Therefore, only $n!$ sequences of jobs need to be considered to find an optimal solution of the problem.

We can prove that the problem considered in this study, *F2/fm/T*, is strongly NP-hard, since the ordinary two-machine flowshop total tardiness problem, *F2//T*, is NP-hard (Koulamas, 1994). Note that *F2/fm/T* is the same as *F2//T* if the cumulative working time limit is extremely large or the time needed to carry out maintenance activities are zero.

This paper is organized as follows. In the next section, we give a mixed integer-linear programming formulation for the problem considered in this study. In Section 3, we develop dominance properties for the problem. In Section 4, we suggest a branch and bound (BAB) algorithm, with lower bounds on the total tardiness of partial schedules used in the BAB algorithm, and a heuristic algorithm to obtain an initial upper bound for the BAB algorithm. In Section 5, computational experiments are performed to evaluate the performance of the suggested algorithms. In Section 6, we conclude this study with a short summary.

## 2. Mathematical formulation

We give a mathematical formulation of the problem considered in this study. In the formulation and throughout this paper, the following notation is used.

Indices and parameters

| | |
|---|---|
| $J$ | set of jobs, $J = \{1, \dots, n\}$ |
| $i, j$ | indices for jobs |
| $[r]$ | index for the $r$th job in a sequence, $r = 1, \dots, n$ |
| $k$ | index for machines, $k = 1, 2$ |
| $p_{ik}$ | processing time of job $i$ on machine $k$ |
| $d_i$ | due date of job $i$ |
| $W$ | cumulative working time limit |
| $h$ | time required to perform a maintenance activity |
| $L$ | a large number |

Decision variables

| | |
|---|---|
| $s_{[r]k}$ | nonnegative real variable that is equal to the starting time of the $r$th job on machine $k$ |
| $c_{[r]}$ | nonnegative real variable that is equal to the completion time of the $r$th job |
| $T_{[r]}$ | nonnegative real variable that is equal to the tardiness of the $r$th job |
| $e_{[r]}$ | nonnegative real variable that is equal to the cumulative working time when the $r$th job is completed on machine 1 |
| $x_{ir}$ | binary variable that is equal to 1 if job $i$ is the $r$th job in a sequence, 0 otherwise |
| $y_{[r]}$ | binary variable that is equal to 1 if a maintenance task is performed immediately after the $r$th job on machine 1, 0 otherwise |

Now, a mixed integer-linear programming formulation is given.

$$[\mathrm{P}]\ \text{Minimize} \quad \sum_r T_{[r]}$$

$$\text{subject to} \quad \sum_r x_{ir} = 1 \quad \forall\, i \in J \tag{1}$$

$$\sum_i x_{ir} = 1 \quad \forall\, r \tag{2}$$

$$\sum_i p_{i1} x_{i1} = e_{[1]} \tag{3}$$

$$e_{[r-1]} + \sum_i p_{i1} x_{ir} - L y_{[r-1]} \leqslant e_{[r]} \quad r = 2,\dots,n \tag{4}$$

$$\sum_i p_{i1} x_{ir} - L(1 - y_{[r-1]}) \leqslant e_{[r]} \quad r = 2,\dots,n \tag{5}$$

$$e_{[r]} \leqslant W \quad \forall\, r \tag{6}$$

$$s_{[r]1} + \sum_i p_{i1} x_{ir} + h y_{[r]} \leqslant s_{[r+1]1} \quad r = 1,\dots,n1 \tag{7}$$

$$s_{[r]1} + \sum_i p_{i1}x_{ir} \leqslant s_{[r]2} \quad \forall\, r \tag{8}$$

$$s_{[r]2} + \sum_i p_{i2}x_{ir} \leqslant s_{[r+1]2} \quad r = 1,\ldots,n-1 \tag{9}$$

$$s_{[r]2} + \sum_i p_{i2}x_{ir} \leqslant c_{[r]} \quad \forall\, r \tag{10}$$

$$c_{[r]} - \sum_i d_i x_{ir} \leqslant T_{[r]} \quad \forall\, r \tag{11}$$

$$s_{[r]k}, c_{[r]}, T_{[r]}, e_{[r]} \geqslant 0 \quad \forall\, r,k \tag{12}$$

$$x_{ir}, y_{[r]} \in \{0,1\} \quad \forall\, i,r \tag{13}$$

The objective function, to be minimized, represents the total tardiness of jobs. Constraints (1) and (2) ensure that each job is placed at only one position in a sequence and only one job can be placed at each position, respectively. Constraints (3)–(6) specify the cumulative working time of machine 1 when each job is completed, and constraints (7)–(10) specify the starting times of the jobs in the flowshop. Constraint (11) is used to compute the tardiness of each job. Finally, constraints (12) and (13) define the domain of the decision variables.

## 3. Dominance properties

This section presents dominance properties that can be used to identify partial schedules dominated by others in the BAB algorithm. In the BAB algorithm, a partial schedule of $i$ jobs corresponds to a node at the $i$th level of the BAB tree, and the partial schedule is placed at the front part of a complete schedule. By using the dominance properties, we can identify dominated partial schedules and prune the nodes (in the BAB tree) associated with the dominated partial schedules. Note that a partial schedule $\boldsymbol{\sigma}$ is dominated by another partial schedule $\boldsymbol{\sigma'}$ if a complete schedule resulting from $\boldsymbol{\sigma'}$ is better than the best complete schedule among those resulting from $\boldsymbol{\sigma}$.

Here, we define a *processing period* as the time period between two successive maintenance activities. In addition, we state the properties and describe the algorithms with the following notation as well as that given in the previous section.

| | |
|---|---|
| $\boldsymbol{\sigma}$ | a (partial) schedule |
| $\boldsymbol{\sigma i \sigma' j}$ | partial schedule obtained with $\boldsymbol{\sigma}$ followed by job $i$, jobs in $\boldsymbol{\sigma'}$, and job $j$ in this order |
| $\pi_x$ | partial sequence of jobs processed in the $x$th processing period in $\boldsymbol{\sigma}$ |
| $\pi_1\, \pi_2 \ldots \pi_x$ | partial schedule with $\pi_1, \pi_2, \ldots, \pi_x$ in this order |
| $S$ | set of jobs already included in $\boldsymbol{\sigma}$ |
| $n(\boldsymbol{\sigma})$ | number of jobs included in $\boldsymbol{\sigma}$ |
| $C_i(\boldsymbol{\sigma})$ | completion time of job $i$ on the second machine in $\boldsymbol{\sigma}$ |
| $T_i(\boldsymbol{\sigma})$ | tardiness of job $i$ in $\boldsymbol{\sigma}$ |
| $T^{\Sigma}(\boldsymbol{\sigma})$ | $\Sigma_{i\in S}\, T_i(\boldsymbol{\sigma})$ |
| $\alpha(\boldsymbol{\sigma})$ | completion time of the last scheduled job on the first machine in $\boldsymbol{\sigma}$ |
| $\beta(\boldsymbol{\sigma})$ | completion time of the last scheduled job on the second machine in $\boldsymbol{\sigma}$ |
| $w(\boldsymbol{\sigma})$ | cumulative working time after the last, i.e., most recently, performed maintenance activity in $\boldsymbol{\sigma}$ |
| $r(\boldsymbol{\sigma})$ | remaining working time before the next maintenance activity to be performed after $\boldsymbol{\sigma}$, i.e., $r(\boldsymbol{\sigma}) = W - w(\boldsymbol{\sigma})$ |

The dominance properties to be presented in this study may be considered as extended versions of those for two-machine flowshop tardiness problems, which are suggested by Schaller (2005). In such extended versions, flexible maintenance is to be considered in the dominance conditions. First, the following three propositions present dominance properties related to two adjacent jobs that are to be placed last in a partial schedule.

**Proposition 1.** *Given a partial schedule, $\boldsymbol{\sigma}ij$, if (a) $C_i(\boldsymbol{\sigma}ji) \leq d_i$, (b) $C_i(\boldsymbol{\sigma}ji) \leq C_j(\boldsymbol{\sigma}ij)$, and (c) $w(\boldsymbol{\sigma}ij) \geq w(\boldsymbol{\sigma}ji)$, then $\boldsymbol{\sigma}ij$ is dominated by $\boldsymbol{\sigma}ji$.*

**Proof.** Since condition (a) is satisfied, job $i$ is not tardy in $\boldsymbol{\sigma}ji$, i.e., $T_i(\boldsymbol{\sigma}ji) = 0$. In addition, since job $j$ in $\boldsymbol{\sigma}ji$ is completed earlier than in $\boldsymbol{\sigma}ij$, $T_j(\boldsymbol{\sigma}ij) \geq T_j(\boldsymbol{\sigma}ji)$ and hence $T_i(\boldsymbol{\sigma}ij) + T_j(\boldsymbol{\sigma}ij) \geq T_i(\boldsymbol{\sigma}ji) + T_j(\boldsymbol{\sigma}ji)$. Also, the start time of the next job after job $i$ in a complete schedule resulting from $\boldsymbol{\sigma}ji$ is not delayed because of condition (b). On the other hand, since $w(\boldsymbol{\sigma}ij) \geq w(\boldsymbol{\sigma}ji)$, the cumulative working time on the first machine is decreased by the interchange of job $i$ and job $j$, and hence the next job to be scheduled after $\boldsymbol{\sigma}ji$ can be scheduled in the same period as or an earlier processing period than the period the job is scheduled if it is scheduled after $\boldsymbol{\sigma}ij$. Therefore, $\boldsymbol{\sigma}ij$ is dominated by $\boldsymbol{\sigma}ji$. □

**Proposition 2.** *Given a partial schedule, $\boldsymbol{\sigma}ij$, if (a) $C_j(\boldsymbol{\sigma}ji) \geq d_j$, (b) $C_i(\boldsymbol{\sigma}ij) \geq C_j(\boldsymbol{\sigma}ji)$, (c) $min\{p_{i1}, p_{j2}\} \geq p_{j1}$, and (d) $w(\boldsymbol{\sigma}ij) \geq w(\boldsymbol{\sigma}ji)$, then $\boldsymbol{\sigma}ij$ is dominated by $\boldsymbol{\sigma}ji$.*

**Proof.** Sen, Dileepan, and Gupta (1989) show that $C_j(\boldsymbol{\sigma}ij) \geq C_i(\boldsymbol{\sigma}ji)$ if condition (c) is satisfied. Thus, from conditions (b) and (c), we have $C_i(\boldsymbol{\sigma}ij) + C_j(\boldsymbol{\sigma}ij) \geq C_i(\boldsymbol{\sigma}ji) + C_j(\boldsymbol{\sigma}ji)$. Then, this inequality and condition (a) result in $T_i(\boldsymbol{\sigma}ij) + T_j(\boldsymbol{\sigma}ij) \geq T_i(\boldsymbol{\sigma}ji) + T_j(\boldsymbol{\sigma}ji)$. On the other hand, from condition (d), the next job to be scheduled after $\boldsymbol{\sigma}ji$ can be scheduled in the same period as or an earlier processing period than the period the job is scheduled if it is scheduled after $\boldsymbol{\sigma}ij$. Therefore, $\boldsymbol{\sigma}ij$ is dominated by $\boldsymbol{\sigma}ji$. □

**Proposition 3.** *Given a partial schedule, $\boldsymbol{\sigma}ij$, if (a) $T_i(\boldsymbol{\sigma}ij) + T_j(\boldsymbol{\sigma}ij) \geq T_i(\boldsymbol{\sigma}ji) + T_j(\boldsymbol{\sigma}ji)$, (b) $T_i(\boldsymbol{\sigma}ij) + T_j(\boldsymbol{\sigma}ij) \geq T_i(\boldsymbol{\sigma}ji) + T_j(\boldsymbol{\sigma}ji) + (n - n(\boldsymbol{\sigma}))\cdot(C_i(\boldsymbol{\sigma}ji) - L_C)$, where $L_C = max\{\alpha(\boldsymbol{\sigma}) + h\cdot\lceil(w(\boldsymbol{\sigma}) + min_{l\in U}\{p_{l1}\})/I^M - 1\rceil, \beta(\boldsymbol{\sigma})\}$, and (c) $w(\boldsymbol{\sigma}ij) \geq w(\boldsymbol{\sigma}ji)$, then $\boldsymbol{\sigma}ij$ is dominated by $\boldsymbol{\sigma}ji$.*

**Proof.** From condition (a), the tardiness of jobs $i$ and $j$ in $\boldsymbol{\sigma}ji$ is no greater than the tardiness of these jobs in $\boldsymbol{\sigma}ij$. However, since $\boldsymbol{\sigma}ji$ is obtained from $\boldsymbol{\sigma}ij$ by interchanging jobs $i$ and $j$, the completion time of each job to be scheduled after $\boldsymbol{\sigma}ji$ may be increased by $C_i(\boldsymbol{\sigma}ji) - L_C$ at most from that of $\boldsymbol{\sigma}ij$. Thus, $(n - n(\boldsymbol{\sigma}))\cdot(C_i(\boldsymbol{\sigma}ji) - L_C)$ is an upper bound on the increase in tardiness of the jobs scheduled after job $i$ in $\boldsymbol{\sigma}ji$, and hence if condition (b) is satisfied, the increase of total tardiness due to this interchange is less than or equal to 0. On the other hand, from condition (c), the next job to be scheduled after $\boldsymbol{\sigma}ji$ can be scheduled in the same period as or an earlier processing period than the period the job is scheduled if it is scheduled after $\boldsymbol{\sigma}ij$. Therefore, $\boldsymbol{\sigma}ij$ is dominated by $\boldsymbol{\sigma}ji$. □

Schaller (2005) also develops a dominance condition related to any two jobs in a partial schedule. In this study, we use the dominance condition for two jobs that are scheduled in the same processing period in a partial schedule.

**Proposition 4.** *Given a partial schedule, $\boldsymbol{\sigma i \sigma' j}$, in which two jobs i and j are scheduled in the same processing period, if (a) $C_j(\boldsymbol{\sigma j \sigma' i}) \geq d_j$, (b) $C_i(\boldsymbol{\sigma i \sigma' j}) \geq C_j(\boldsymbol{\sigma j \sigma' i})$, (c) $p_{i1} \geq p_{j1}$ and (d) $p_{i2} \leq p_{j2}$, then $\boldsymbol{\sigma i \sigma' j}$ is dominated by $\boldsymbol{\sigma j \sigma' i}$.*

The above proposition is extended to the following proposition, which is to be applied to both cases, in which two jobs are processed in different processing periods and in the same period.

**Proposition 5.** *Given a partial schedule, $\boldsymbol{\sigma i \sigma' j}$, if (a) $T^{\Sigma}(\boldsymbol{\sigma i \sigma' j}) \geq T^{\Sigma}(\boldsymbol{\sigma j \sigma' i})$, (b) $\alpha(\boldsymbol{\sigma i \sigma' j}) \geq \alpha(\boldsymbol{\sigma j \sigma' i})$, (c) $\beta(\boldsymbol{\sigma i \sigma' j}) \geq \beta(\boldsymbol{\sigma j \sigma' i})$, and (d) $w(\boldsymbol{\sigma i \sigma' j}) \geq w(\boldsymbol{\sigma j \sigma' i})$, then $\boldsymbol{\sigma i \sigma' j}$ is dominated by $\boldsymbol{\sigma j \sigma' i}$.*

**Proof.** Since $\boldsymbol{\sigma j \sigma' i}$ satisfies $\alpha(\boldsymbol{\sigma i \sigma' j}) \geq \alpha(\boldsymbol{\sigma j \sigma' i})$ and $\beta(\boldsymbol{\sigma i \sigma' j}) \geq \beta(\boldsymbol{\sigma j \sigma' i})$, the start times of jobs to be scheduled after $\boldsymbol{\sigma j \sigma' i}$ are not greater than those after $\boldsymbol{\sigma i \sigma' j}$. Also, since $w(\boldsymbol{\sigma i \sigma' j}) \geq w(\boldsymbol{\sigma j \sigma' i})$, the cumulative working time on the first machine is decreased by this interchange and hence the next job to be scheduled after $\boldsymbol{\sigma j \sigma' i}$ can be scheduled in the same period as or an earlier processing period than the period the job is scheduled if it is scheduled after $\boldsymbol{\sigma i \sigma' j}$. In addition, since $T^{\Sigma}(\boldsymbol{\sigma i \sigma' j}) \geq T^{\Sigma}(\boldsymbol{\sigma j \sigma' i})$, the total tardiness resulting from this

interchange can be decreased. Therefore, $\sigma i \sigma' j$ is dominated by $\sigma j \sigma' i$. □

Proposition 6 is associated with a dominance property related to jobs scheduled after the first processing period. Let $\upsilon_i(\sigma)$ be the set of jobs scheduled after job $i$ in $\sigma$ and let $\omega_i(\sigma)$ be the cumulative working time of the processing period where job $i$ is scheduled.

**Proposition 6.** *Consider a partial schedule, $\sigma$, in which job i is scheduled last and another partial schedule, $\sigma'$, which is constructed from $\sigma$ by moving job i to the last position in an earlier processing period than the period in which job i is processed. If $\omega_i(\sigma') \le W$ and $\sum_{j\in\upsilon_i(\sigma')} T_j(\sigma') - \sum_{j\in\upsilon_i(\sigma')} T_j(\sigma) \le T_i(\sigma) - T_i(\sigma')$, $\sigma$ is dominated by $\sigma'$.*

**Proof.** Since $\sigma'$ is constructed from $\sigma$ by moving job $i$ only, the tardiness of jobs in $\upsilon_i(\sigma')$ may be increased. The increase of total tardiness resulting from the movement is at most $\sum_{j\in\upsilon_i(\sigma')} T_j(\sigma') - \sum_{j\in\upsilon_i(\sigma')} T_j(\sigma) + T_i(\sigma') - T_i(\sigma) \le 0$. Therefore, $\sigma$ is dominated by $\sigma'$. □

We also suggest a dominance property based on the Johnson's rule (Johnson, 1954), which is used to solve a two-machine flowshop makespan problem to optimality. Here, let $\pi_x^J$ denote a partial sequence obtained by applying Johnson's rule to the jobs in the $x$th processing period in a partial schedule.

**Proposition 7.** *Consider a partial schedule, $\pi_1 \pi_2 \dots \pi_x$, and another partial schedule, $\pi_1 \pi_2 \dots \pi_x^J$, which is identical to $\pi_1 \pi_2 \dots \pi_x$ except that the sequence of jobs in the xth processing period is different. If $T^\Sigma(\pi_x) \ge T^\Sigma(\pi_x^J)$, then $\pi_1 \pi_2 \dots \pi_x$ is dominated by $\pi_1 \pi_2 \dots \pi_x^J$.*

**Proof.** Since $\pi_1 \pi_2 \dots \pi_x^J$ is obtained by changing only the sequence of jobs in the xth processing period while leaving sequences of all other jobs in periods 1 through x, the tardiness of jobs in $\pi_1 \pi_2 \dots \pi_{x-1}$ is the same for both partial schedules. Also, by Lemma 2 given in Lee (1997), the completion time of the last scheduled job in $\pi_x^J$ is not greater than the completion time of the last job in $\pi_x$ since $\pi_x^J$ is obtained by Johnson's rule. Thus, the start times of jobs to be scheduled after $\pi_1 \pi_2 \dots \pi_x^J$ will not be greater than those after $\pi_1 \pi_2 \dots \pi_x$. Therefore, if $T^\Sigma(\pi_x) \ge T^\Sigma(\pi_x^J)$, $\pi_1 \pi_2 \dots \pi_x$ is dominated by $\pi_1 \pi_2 \dots \pi_x^J$. □

In our BAB algorithm, we use the following two properties that are suggested in other studies on flowshop scheduling problem without availability constraints. Proposition 8 gives a dominance property related to two adjacent jobs that are processed in the same processing period, based on that of Sen et al. (1989), while Proposition 9 gives a dominance property related to the first job in an optimal schedule presented in Pan, Chen, and Chao (2002).

**Proposition 8.** *Given a partial schedule, $\sigma ij$, in which two adjacent jobs i and j are processed in the same processing period, if $d_j \le d_i$, $p_{j2} - d_j \le p_{i2} - d_i$, and $p_{j1} \le \min\{p_{j2}, p_{i1}\}$, then $\sigma ij$ is dominated by $\sigma ji$.*

**Proposition 9.** *For any job i, if there is a job, say job j, such that $p_{i2} \le p_{j2}$, $p_{i1} + p_{i2} \ge p_{j2} + p_{j2}$, and $p_{i2} - d_i \le p_{j2} - d_j$, then there exists an optimal schedule in which job i is not the first job in the schedule.*

## 4. Branch and bound algorithm

In this section, a branch and bound (BAB) algorithm is developed for the problem, *F2/fm/T*, considered in this paper. As stated above in this paper, each node in the BAB tree corresponds to a partial schedule, and a node at the $i$th level in the tree represents a partial schedule for the first $i$ jobs in complete schedules. For construction of a BAB tree, we use the depth-first rule. That is, a node at the highest level is selected and branched to generate its children nodes. If there are two or more nodes at the highest level, we select a node with the minimum lower bound among these. For each child node, we check whether or not the partial schedule corresponding to the child node is dominated by using the dominance conditions given in Section 3. If the partial schedule is dominated, the node associated with this partial schedule is deleted from further consideration. Otherwise, a lower bound is computed. If the current best solution is better than the lower bound, the child node is also deleted.

### 4.1. Lower bound

In this subsection, we suggest two methods to compute a lower bound on the total tardiness of complete schedules resulting from a partial schedule. This lower bound is computed for the partial schedule that is not dominated by the dominance conditions presented above. As stated earlier, the partial schedule is placed at the front part of a complete schedule. A lower bound for a partial schedule is obtained as the sum of the total tardiness of the jobs in the partial schedule and the lower bound on the tardiness of jobs not included in the partial schedule.

To improve the lower bound, we add a maintenance activity at the end of the last job on the first machine if the remaining working time is less than the minimum processing time of the jobs that are not included in partial schedule $\sigma$, i.e., $r(\sigma) < \min_{i\notin S(\sigma)}\{p_i\}$, before computing the lower bound. Then, the completion time of the first machine is increased by $h$, and $w(\sigma)$ and $r(\sigma)$ are set to 0 and $W$, respectively. For computing lower bounds, we assume that maintenance activities are performed periodically with a time interval $W$, and that jobs are preempted by maintenance activities and resumed. That is, when computing lower bounds, it is assumed that the jobs not included in $\sigma$ can be preempted for the maintenance activities and resumed after the maintenance.

In this study, based on the following proposition of Kim (1993), a lower bound on the total tardiness of jobs is obtained. Here, $d_{[j]}$ denotes the $j$th earliest due date among the jobs that are not included in the partial schedule associated with the current node, and $(x)^+ = \max(x, 0)$.

**Proposition 10** (*Kim, 1993*). *For nonnegative real numbers $c_j$ and $d_j$ corresponding to job j, if $c_1 \le c_2 \le \dots \le c_n$, then the following inequality holds.*

$$\sum_{j=1}^{n} (c_j - d_{[j]})^+ \le \sum_{j=1}^{n} (c_j - d_j)^+$$

The following notation as well as the notation given in the previous section is used in the description of the lower bounds presented in this section.

| | |
|---|---|
| $\alpha$ | ready time of the first machine |
| $\beta$ | ready time of the second machine |
| $p_{[i]k}$ | the $i$th shortest processing time among all jobs on machine $k$ |
| $P_{ik}$ | of the $i$ shortest processing times of jobs on machine $k$ |
| $U$ | set of jobs not included in $\sigma$ |
| $\sigma^*$ | an arbitrary complete schedule resulting from $\sigma$ |

The following proposition provides a lower bound, LB1, which is obtained by modifying a lower bound suggested by Pan et al. (2002) for the two-machine flowshop total tardiness problem without maintenance activities.

**Proposition 11.** *For a given partial schedule $\sigma$, the total tardiness of jobs not included in $\sigma$ is no less than $\Sigma_{j\in U}(\max\{\max\{\alpha + P_{j1} + h\lceil (w(\sigma) + P_{j1})/W - 1\rceil, \beta\} + p_{[i]2}, \max\{\alpha + p_{[1]1}, \beta\} + P_{j2}\} - d_{[j]})^+$ in any complete schedule resulting from a given partial schedule $\sigma$, where i is initially set to 1 and increased by one if $\max\{\max\{\alpha + P_{j1} + h\lceil (w(\sigma) + P_{j1})/I^M - 1\rceil, \beta\} + p_{[i]2}, \max\{\alpha + p_{[1]1}, \beta\} + P_{j2}\} - d_{[j]} > 0$.*

In another lower bound, LB2, we do not use the due dates of jobs but modified due dates which are introduced by Schaller (2009) for obtaining a lower bound on the total tardiness of jobs on identical-parallel machines. In this study, the modified due date of jobs not included in a partial schedule is defined as

$$d'_j = \begin{cases} \max\{d_j, \max\{\alpha + h + p_{j1}, \beta\} + p_{j2}\}, & \text{if } w(\sigma) + p_{j1} > W, \\ \max\{d_j, \max\{\alpha + p_{j1}, \beta\} + p_{j2}\}, & \text{otherwise.} \end{cases}$$

Then, LB2 is computed as

$$\begin{aligned} LB2 = & \sum_{j \in U} [\max\{\alpha + h \cdot \lceil (w(\sigma) + P_{j1})/W - 1 \rceil + p_{j1}, \beta\} + p_{j2} - d_j]^+ \\ & + \sum_{j \in U} [\max\{\max\{\alpha + P_{j1} + h \cdot \lceil (w(\sigma) + P_{j1})/W - 1 \rceil, \beta\}, \\ & + p_{[i]2}, \max\{\alpha + p_{[1]1}, \beta\} + P_{j2}\} - d'_{[j]}]^+ \end{aligned}$$

where $i$ is initially set to 1 and increased by one if $\max\{\max\{\alpha + P_{j1} + h\lceil (w(\sigma) + P_{j1})/I^M - 1 \rceil, \beta\} + p_{[i]2}, \max\{\alpha + p_{[1]1}, \beta\} + P_{j2}\} - d'_{[j]} > 0$.

**Proposition 12.** *Given a partial schedule, σ, the total tardiness of jobs not included in σ is no less than LB2 in any complete schedule resulting from σ.*

**Proof.** We have $\Sigma_{j \in U} T_j(\sigma^*) = \Sigma_{j \in U} \{C_j(\sigma^*) - d_j\}^+ = \Sigma_{j \in U} \{C_j(\sigma^*) - d'_j + d'_j - d_j\}^+$.

If $d_j \geq C_j(\sigma^*)$, then $d'_j = d_j$ and $(C_j(\sigma^*) - d'_j + d'_j - d_j)^+ = 0$ since $C_j(\sigma^*) \geq \alpha + h\lceil (w(\sigma) + P_{j1})/W - 1 \rceil + p_{j1}$.

If $d_j < C_j(\sigma^*)$, then $(C_j(\sigma^*) - d'_j + d'_j - d_j)^+ = (C_j(\sigma^*) - d'_j)^+ + (d'_j - d_j) > 0$.

$$\begin{aligned} \text{Note that } d'_j - d_j &= \max\{d_j, \max\{\alpha + h \cdot \lceil (w(\sigma) + P_{j1})/W - 1 \rceil + p_{j1}, \beta\} \\ &\quad + p_{j2}\} - d_j \\ &= \max\{0, \max\{\alpha + h \cdot \lceil (w(\sigma) + P_{j1})/W - 1 \rceil + p_{j1}, \beta\} + p_{j2} \\ &\quad - d_j\} \\ &= [\max\{\alpha + h \cdot \lceil (w(\sigma) + P_{j1})/W - 1 \rceil + p_{j1}, \beta\} + p_{j2} - d_j]^+. \end{aligned}$$

Note also that this is in the first term of LB2.

Based on Proposition 10, $C_j(\sigma^*) \geq \max\{\max\{\alpha + P_{j1} + h\lceil (w(\sigma) + P_{j1})/W - 1 \rceil, \beta\} + p_{[i]2}, \max\{\alpha + p_{[1]1}, \beta\} + P_{j2}\}$.

Also, from Proposition 11, we have $\Sigma_{j \in U}(C_j(\sigma^*) - d'_{[j]})^+ \leq \Sigma_{j \in U}(C_j(\sigma^*) - d'_j)^+$.

Therefore, $LB2 \leq \Sigma_{j \in U} T_j(\sigma^*)$. □

In our BAB algorithm, the larger value between LB1 and LB2 is used for the lower bound on the tardiness of jobs that are not included in the partial schedule.

### 4.2. Upper bound

This subsection proposes a simple heuristic algorithm to obtain an initial upper bound at the root node of the BAB tree. Since we only need to consider permutation schedules in this study, only a sequence of jobs (and maintenance activities) on the first machine is obtained in the heuristic algorithm. Once a sequence of jobs is obtained, according to the sequence, the jobs are scheduled one by one to be included in the current processing period being considered, starting from the first period. If a job being considered cannot be included in the current period, a maintenance activity is added and then the job is scheduled after the maintenance.

In this study, we use a constructive algorithm, called NEH, developed by Nawaz, Enscore, and Han (1983). NEH is known to work well in flowshop scheduling problems and it has been modified by many researchers for various scheduling problems. In the original NEH, jobs are sorted in a non-increasing order of the sum of processing times on the machines. Then, the sorted jobs are added into a partial schedule one by one in that order, and each job is inserted into the best position in the current partial schedule until a complete schedule is obtained.

In this study, to find a seed sequence we use a method different from the one of NEH. First, jobs are sequenced in a non-decreasing order of due dates, i.e., with the earliest due date (EDD) rule. If there is a tie, a job with a shorter processing time is sequenced earlier, that is, the shortest processing time (SPT) rule is applied. Next, the resulting sequence is improved by interchanging a pair of jobs. All pairs of jobs are considered for interchanges at each attempt for improvement. This procedure is terminated when the current solution cannot be improved any more, and the result is a seed sequence of the NEH algorithm.

## 5. Computational experiments

We performed a series of computational experiments to evaluate the performance of the algorithms presented in this study. For the experiments, we generated problem instances randomly. In the instances, the processing times and due dates were generated following the method by Kim (1993). The processing times of jobs were generated from the discrete uniform (DU) distribution with range [1, 100], and due dates were generated from the DU distribution with range $[X(1 - T - R/2), X(1 - T + R/2)]$, where $X$ is a lower bound on the makespan, and $T$ and $R$ are parameters, called the tardiness factor and relative range of due dates, respectively, which have an impact on the tightness of due dates. Here, $X = \Sigma_{j \in J}(p_{j1} + p_{j2})/2$, and three levels (0.25, 0.5, and 0.75) for each of $T$ and $R$ were used to generate instances, resulting in nine combinations for the values of $(T, R)$. Note that if the resulting due dates were less than 0, they were reset to 0. In addition, the cumulative working time limit, $W$, was set to $\lambda_1 P^*$ with two levels (0.4 and 0.5) for $\lambda_1$, where $P^* = \Sigma_{j \in J} p_{j1}$, and the time required to perform a maintenance activity, $h$, was set to $W/\lambda_2$ with two levels (5 and 10) for $\lambda_2$.

For the computational experiments, 2520 instances were generated randomly, 10 instances for each of 252 combinations of three levels for $T$, three levels for $R$, two levels for $\lambda_1$, two levels for $\lambda_2$, and seven levels (13, 15, 17, 19, 21, 23 and 25) for the number of jobs, $n$. The tests were executed on a PC with a Quad-Core processor operating at 3.2-GHz clock speed. To avoid excessive computation time for the tests, we set the limit on the CPU time for each instance to 3600 s. All algorithms are coded in Java.

First, we compare the overall performance of a method using a commercial integer and linear program solver for the mixed integer-linear programming (MILP) formulation, the BAB algorithm, and the heuristic algorithm (HEU) which is used to find an initial incumbent solution for the BAB algorithm. Note that a commercial linear/integer programming solver, CPLEX 12.5, was used to solve the MILP. The results of the tests are summarized in Tables 1 and 2. As can be seen from Table 1, the BAB algorithm found optimal solutions for the problem instances with up to 23 jobs, whereas the MILP showed better performance in terms of the average computation time required to achieve an optimal solution. To show the difference between the performance of the MILP and the BAB algorithm, we compared both methods for the instances with 21, 23 and 25 jobs in detail. As shown in Table 2, the MILP solved problems with a small tardiness factor ($T$) in a

**Table 1**
Performance of the proposed methods.

| $n$ | MILP | | BAB | | HEU | |
|---|---|---|---|---|---|---|
| | ACPUT[a] | NS[b] | ACPUT | NS | NO[c] | APE[d] |
| 13 | 0.61 | 0 | 0.02 | 0 | 108 | 4.03 |
| 15 | 1.49 | 0 | 0.09 | 0 | 109 | 2.79 |
| 17 | 3.59 | 0 | 0.55 | 0 | 70 | 3.41 |
| 19 | 7.09 | 0 | 2.17 | 0 | 70 | 4.44 |
| 21 | 31.43 | 1 | 16.57 | 0 | 34 | 4.32 |
| 23 | 136.08 | 5 | 81.65 | 0 | 44 | 4.18 |
| 25 | 326.23 | 24 | 431.37 | 12 | 49 | 4.72 |

[a] Average CPU time (in seconds) required to solve an instance (3600 s were used for computing the average values when an instance was not solved within 3600 s).
[b] Number of instances (among 360 instances) that were not solved in 3600 s.
[c] Number of instances (among 360 instances) for which the heuristic algorithm found optimal solutions.
[d] Average percentage error of the solution values from solutions obtained by the BAB algorithm.

**Table 2**
Comparison the BAB algorithm with MILP.

| $(T, R)$ | $(\lambda_1, \lambda_2)$ | MILP | | | BAB | | |
|---|---|---|---|---|---|---|---|
| | | $n = 21$ | 23 | 25 | 21 | 23 | 25 |
| (0.25, 0.25) | (0.4, 5) | 1.67[†] | 2.71 | 2.42 | 67.76 | 404.20 | 2165.75(3)[‡] |
| | (0.4, 10) | 1.77 | 3.33 | 3.01 | 95.71 | 654.19 | 2775.48(6) |
| | (0.5, 5) | 2.96 | 1.89 | 2.87 | 26.65 | 126.79 | 767.52 |
| | (0.5, 10) | 1.90 | 1.68 | 3.26 | 33.97 | 180.57 | 1164.39 |
| (0.25, 0.50) | (0.4, 5) | 2.80 | 2.34 | 6.79 | 58.94 | 146.37 | 1272.84 |
| | (0.4, 10) | 3.22 | 4.76 | 8.32 | 81.66 | 199.22 | 1502.51(1) |
| | (0.5, 5) | 3.46 | 3.03 | 11.31 | 22.57 | 76.33 | 446.53 |
| | (0.5, 10) | 4.21 | 2.84 | 5.75 | 26.07 | 83.41 | 465.32 |
| (0.25, 0.75) | (0.4, 5) | 376.26(1) | 18.57 | 391.07(1) | 26.50 | 89.70 | 744.59(1) |
| | (0.4, 10) | 34.21 | 13.30 | 383.25(1) | 20.85 | 117.28 | 823.75 |
| | (0.5, 5) | 55.92 | 19.91 | 413.67(1) | 14.46 | 69.32 | 555.92(1) |
| | (0.5, 10) | 27.98 | 19.47 | 395.20(1) | 10.96 | 67.74 | 345.22 |
| (0.50, 0.25) | (0.4, 5) | 2.91 | 4.77 | 3.23 | 5.54 | 24.38 | 88.52 |
| | (0.4, 10) | 2.43 | 19.96 | 6.94 | 4.80 | 27.02 | 98.43 |
| | (0.5, 5) | 1.72 | 2.66 | 3.72 | 6.68 | 29.50 | 123.92 |
| | (0.5, 10) | 2.40 | 18.56 | 6.90 | 5.51 | 27.94 | 119.67 |
| (0.50, 0.50) | (0.4, 5) | 4.18 | 11.22 | 12.41 | 3.59 | 27.35 | 100.63 |
| | (0.4, 10) | 42.52 | 34.80 | 55.50 | 5.41 | 30.50 | 142.70 |
| | (0.5, 5) | 17.49 | 7.23 | 30.27 | 5.71 | 38.81 | 155.57 |
| | (0.5, 10) | 37.19 | 42.63 | 74.94 | 6.65 | 37.51 | 157.69 |
| (0.50, 0.75) | (0.4, 5) | 14.76 | 451.48 | 845.55(2) | 4.92 | 43.17 | 127.19 |
| | (0.4, 10) | 74.02 | 1000.29(2) | 1231.32(3) | 6.25 | 66.83 | 166.03 |
| | (0.5, 5) | 17.63 | 477.33 | 656.65(1) | 5.82 | 49.88 | 157.03 |
| | (0.5, 10) | 67.35 | 979.91(2) | 1195.68(3) | 6.86 | 53.89 | 156.24 |
| (0.75, 0.25) | (0.4, 5) | 5.29 | 12.71 | 38.98 | 4.69 | 18.91 | 20.40 |
| | (0.4, 10) | 15.50 | 31.36 | 75.08 | 7.28 | 44.43 | 41.19 |
| | (0.5, 5) | 12.38 | 42.95 | 38.87 | 7.26 | 31.62 | 32.22 |
| | (0.5, 10) | 15.59 | 37.54 | 76.76 | 10.13 | 61.62 | 48.43 |
| (0.75, 0.50) | (0.4, 5) | 11.33 | 23.51 | 178.51 | 1.25 | 5.73 | 29.78 |
| | (0.4, 10) | 44.66 | 67.71 | 641.3(1) | 1.66 | 8.15 | 44.56 |
| | (0.5, 5) | 37.13 | 26.18 | 70.98 | 2.54 | 7.35 | 27.58 |
| | (0.5, 10) | 31.71 | 39.19 | 371.96 | 2.33 | 7.94 | 46.92 |
| (0.75, 0.75) | (0.4, 5) | 21.06 | 106.29 | 1060.48(2) | 1.03 | 11.20 | 99.50 |
| | (0.4, 10) | 36.23 | 611.00(1) | 1277.75(3) | 1.18 | 22.25 | 146.89 |
| | (0.5, 5) | 47.24 | 361.78 | 965.48(2) | 1.74 | 21.42 | 181.86 |
| | (0.5, 10) | 52.32 | 393.94 | 1197.49(3) | 1.61 | 26.87 | 186.63 |
| Average (sum)[#] | | 31.43(1) | 136.08(5) | 326.23(24) | 16.57 | 81.65 | 431.37(12) |

[†] Average CPU time (in seconds) required to solve an instance (3600 s were used for computing the average values when an instance was not solved within 3600 s).
[‡] Number of instances (among 360 instances) that were not solved in 3600 s.
[#] Average CPU time in seconds/sum of the numbers of instances that were not solved.

relatively short computation time, while the BAB algorithm showed better performance than the MILP when the tardiness factor is large. That is, when the due dates of jobs are tighter, the BAB algorithm can find optimal solutions in a relatively short time. On the other hand, the error percentage of the solutions of the heuristic algorithm from optimal solutions was 3.98% on average. Note that the computation time required for the heuristic to solve an instance is less than 0.01 s regardless of the instance sizes and parameter values.

Note that the computation time to solve problem instances may be affected by maintenance activities and due dates of jobs. Thus, we show the performance of BAB for different levels of parameters related to the maintenance activities and due dates in Tables 3 and 4, respectively. Table 3 shows the results for different values of $\lambda_1$ and $\lambda_2$ related to the maintenance. Note that a larger value of $\lambda_1$ results in a longer cumulative working time limit, and a larger value of $\lambda_2$ results in a shorter maintenance activity time. As can be seen in Table 3, the CPU time of the BAB algorithm is dependent on the values of both $\lambda_1$ and $\lambda_2$. The CPU time of the BAB algorithm tended to decrease as the value of $\lambda_1$ increased. This may be because if the cumulative working time limit is longer, more jobs can be scheduled within the cumulative working time limit, and hence the dominance properties can be applied more often. On the other hand, the instances with longer maintenance activity times required shorter computation time.

Table 4 shows the results for the different levels of tightness of due dates. The performance of the algorithm was also affected by the tightness of due dates considerably. For instance, when $T$ is large and $R$ is small, the algorithm solved problems to optimality in a relatively short time. This may be because due dates of jobs are tighter, and hence most jobs become tardy in this case. Then, the dominance properties can be applied more often and the lower bounds used in the BAB algorithm are tighter. On the other hand, the performance of the algorithm was not good when both $T$ and $R$ are small.

In addition, to see the effectiveness of dominance properties and lower bounds presented in this study, additional tests were performed. For the tests, 180 instances were generated randomly in a way similar to the one explained earlier. That is, one instance was generated for each of all combinations of five levels for the number of jobs (15, 17, 19, 21, and 23), four pairs of values for the parameters used to generate maintenance activities, and nine pairs of values for the parameters related to due dates of jobs.

For a check on the effectiveness of the dominance properties used in the BAB algorithm, we tested ten BAB algorithms: BB, in which all

**Table 3**
Performance of the BAB algorithm (affected by the maintenance related parameters).

| $\lambda_1$ | $n$ | $\lambda_2$ | | | | Average/sum[#] | |
|---|---|---|---|---|---|---|---|
| | | 5 | | 10 | | | |
| | | ACPUT[†] | NS[‡] | ACPUT | NS | ACPUT | NS |
| 0.4 | 13 | 0.02 | 0 | 0.02 | 0 | 0.02 | 0 |
| | 15 | 0.08 | 0 | 0.10 | 0 | 0.09 | 0 |
| | 17 | 0.64 | 0 | 0.66 | 0 | 0.65 | 0 |
| | 19 | 2.20 | 0 | 3.07 | 0 | 2.64 | 0 |
| | 21 | 19.36 | 0 | 24.98 | 0 | 22.17 | 0 |
| | 23 | 85.67 | 0 | 129.99 | 0 | 107.83 | 0 |
| | 25 | 516.58 | 4 | 637.95 | 7 | 577.27 | 11 |
| | Average/sum | 89.22 | 4 | 113.82 | 7 | 101.52 | 11 |
| 0.5 | 13 | 0.02 | 0 | 0.02 | 0 | 0.02 | 0 |
| | 15 | 0.08 | 0 | 0.08 | 0 | 0.08 | 0 |
| | 17 | 0.46 | 0 | 0.45 | 0 | 0.46 | 0 |
| | 19 | 1.62 | 0 | 1.79 | 0 | 1.71 | 0 |
| | 21 | 10.38 | 0 | 11.56 | 0 | 10.97 | 0 |
| | 23 | 50.11 | 0 | 60.83 | 0 | 55.47 | 0 |
| | 25 | 272.02 | 1 | 298.94 | 0 | 285.48 | 1 |
| | Average/sum | 47.81 | 1 | 53.38 | 0 | 50.60 | 1 |
| Overall | | 68.52 | 5 | 83.60 | 7 | 76.06 | 12 |

[†] Average CPU time in seconds (if an instance was not solved within 3600 s, the CPU time for the instance was assumed to be 3600 s when computing the average CPU time).
[‡] Number of instances out of 90 instances (out of 2520 instances for overall) that were not solved within 3600 s.
[#] Average CPU time in seconds/sum of the numbers of instances that were not solved.

dominance properties are used; $BB_{D-x}$, for $x = 1, 2, \ldots, 9$, in which all dominance properties except the property related to Proposition $x$ are used. Table 5 shows the results of this test. From the table, dominance properties related to Propositions 1, 5, and 7 reduced the computation time, and properties given in Propositions 6 and 9 also helped to reduce the number of nodes generated in the algorithm. However, the values of the average ratio of the numbers of nodes generated in the branch and bound procedures to the number of nodes that would have been generated without the dominance conditions (ARNG) related to Propositions 2, 4 and 8 are 1, meaning these properties could not fathom many nodes. Thus, we tested an additional BAB algorithm without Propositions 2, 4 and 8, denoted by BB*, and the results of this test are also given in Table 6. Since the dominance properties related to

**Table 5**
Effectiveness of the dominance properties.

| | | $n = 15$ | 17 | 19 | 21 | 23 |
|---|---|---|---|---|---|---|
| ACPUT[†] | BB | 0.09 | 0.48 | 2.97 | 12.23 | 74.07 |
| | $BB_{D-1}$ | 0.09 | 0.59 | 3.85 | 17.37 | 106.04 |
| | $BB_{D-2}$ | 0.09 | 0.48 | 2.93 | 12.14 | 74.07 |
| | $BB_{D-3}$ | 0.09 | 0.49 | 2.96 | 12.39 | 75.16 |
| | $BB_{D-4}$ | 0.08 | 0.48 | 2.93 | 12.15 | 74.04 |
| | $BB_{D-5}$ | 0.12 | 0.95 | 8.40 | 37.25 | 269.56 |
| | $BB_{D-6}$ | 0.08 | 0.48 | 2.96 | 12.30 | 77.59 |
| | $BB_{D-7}$ | 0.09 | 0.50 | 3.10 | 13.33 | 73.52 |
| | $BB_{D-8}$ | 0.08 | 0.48 | 2.96 | 12.14 | 73.94 |
| | $BB_{D-9}$ | 0.09 | 0.49 | 3.04 | 12.68 | 78.28 |
| | BB* | 0.09 | 0.48 | 2.96 | 12.19 | 73.69 |
| ARNG[‡] | $BB/BB_{D-1}$ | 0.969 | 0.910 | 0.883 | 0.827 | 0.826 |
| | $BB/BB_{D-2}$ | 1 | 1 | 1 | 1 | 1 |
| | $BB/BB_{D-3}$ | 0.994 | 0.996 | 0.989 | 0.993 | 0.997 |
| | $BB/BB_{D-4}$ | 1 | 0.999 | 0.993 | 1 | 1 |
| | $BB/BB_{D-5}$ | 0.742 | 0.717 | 0.594 | 0.344 | 0.423 |
| | $BB/BB_{D-6}$ | 0.985 | 0.980 | 0.929 | 0.944 | 0.934 |
| | $BB/BB_{D-7}$ | 0.956 | 0.962 | 0.932 | 0.894 | 0.966 |
| | $BB/BB_{D-8}$ | 1 | 0.990 | 1 | 1 | 1 |
| | $BB/BB_{D-9}$ | 0.961 | 0.976 | 0.983 | 0.960 | 0.959 |

[†] Average CPU time.
[‡] Average ratio of the numbers of nodes generated in the branch and bound procedures.

**Table 4**
Performance of the BAB algorithm (affected by the due date related parameters).

| $T$ | $n$ | $R$ | | | | | | Average/sum[#] | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0.25 | | 0.50 | | 0.75 | | | |
| | | ACPUT[†] | NS[‡] | ACPUT | NS | ACPUT | NS | ACPUT | NS |
| 0.25 | 13 | 0.04 | 0 | 0.04 | 0 | 0.03 | 0 | 0.04 | 0 |
| | 15 | 0.22 | 0 | 0.17 | 0 | 0.09 | 0 | 0.16 | 0 |
| | 17 | 1.44 | 0 | 1.39 | 0 | 0.89 | 0 | 1.24 | 0 |
| | 19 | 7.06 | 0 | 5.29 | 0 | 2.86 | 0 | 5.07 | 0 |
| | 21 | 56.02 | 0 | 47.31 | 0 | 18.19 | 0 | 40.51 | 0 |
| | 23 | 341.44 | 0 | 126.33 | 0 | 86.01 | 0 | 184.59 | 0 |
| | 25 | 1718.28 | 9 | 921.80 | 1 | 617.37 | 2 | 1085.82 | 12 |
| | Average/sum | 303.50 | 9 | 157.48 | 1 | 103.63 | 2 | 188.20 | 12 |
| 0.50 | 13 | 0.01 | 0 | 0.01 | 0 | 0.01 | 0 | 0.01 | 0 |
| | 15 | 0.06 | 0 | 0.05 | 0 | 0.11 | 0 | 0.07 | 0 |
| | 17 | 0.19 | 0 | 0.24 | 0 | 0.40 | 0 | 0.28 | 0 |
| | 19 | 1.31 | 0 | 1.01 | 0 | 1.16 | 0 | 1.16 | 0 |
| | 21 | 5.63 | 0 | 5.34 | 0 | 5.96 | 0 | 5.64 | 0 |
| | 23 | 27.21 | 0 | 33.54 | 0 | 53.44 | 0 | 38.06 | 0 |
| | 25 | 107.63 | 0 | 139.15 | 0 | 151.62 | 0 | 132.80 | 4 |
| | Average/sum | 20.29 | 0 | 25.62 | 0 | 30.39 | 0 | 25.43 | 0 |
| 0.75 | 13 | < 0.01 | 0 | 0.01 | 0 | 0.01 | 0 | 0.01 | 0 |
| | 15 | 0.02 | 0 | 0.03 | 0 | 0.02 | 0 | 0.02 | 0 |
| | 17 | 0.13 | 0 | 0.14 | 0 | 0.14 | 0 | 0.14 | 0 |
| | 19 | 0.18 | 0 | 0.28 | 0 | 0.40 | 0 | 0.29 | 0 |
| | 21 | 7.34 | 0 | 1.95 | 0 | 1.39 | 0 | 3.56 | 0 |
| | 23 | 39.14 | 0 | 7.29 | 0 | 20.44 | 0 | 22.29 | 0 |
| | 25 | 35.56 | 0 | 37.21 | 0 | 153.72 | 0 | 75.50 | 0 |
| | Average/sum | 11.77 | 0 | 6.70 | 0 | 25.16 | 0 | 14.54 | 0 |
| Overall | | 111.85 | 9 | 63.27 | 1 | 53.06 | 2 | 76.06 | 12 |

†, ‡, # See the footnotes in Table 3.

**Table 6**
Effectiveness of the lower bounds.

| $n$ | ACPUT[†] | | | | ARNG[‡] | |
|---|---|---|---|---|---|---|
| | $BB_N$ | $BB_{LB1}$ | $BB_{LB2}$ | $BB^*$ | $BB^*/BB_{LB1}$ | $BB^*/BB_{LB2}$ |
| 15 | 68.82 | 0.10 | 0.10 | 0.09 | 0.863 | 0.683 |
| 17 | 1125.78 (6) | 0.55 | 0.60 | 0.48 | 0.929 | 0.663 |
| 19 | 2451.86 (18) | 3.33 | 3.45 | 2.96 | 0.913 | 0.679 |
| 21 | 3262.60 (28) | 13.86 | 16.93 | 12.19 | 0.937 | 0.840 |
| 23 | 3600.00 (36) | 90.63 | 99.81 | 73.69 | 0.874 | 0.697 |

[†] Average CPU time in seconds.
[‡] Average ratio of the numbers of nodes generated in the branch and bound procedures.

**Table 7**
Performance of the heuristic algorithm on large-size instances.

| $n$ | NI[†] | | | | | APG[‡] (%) |
|---|---|---|---|---|---|---|
| | $0 \le PG < 5$ | $5 \le PG < 10$ | $10 \le PG < 15$ | $15 \le PG < 20$ | $20 \le PG$ | |
| 50 | 111 | 109 | 68 | 24 | 48 | 12.2 |
| 60 | 107 | 130 | 39 | 30 | 54 | 11.4 |
| 70 | 124 | 89 | 68 | 20 | 59 | 12.1 |
| 80 | 132 | 96 | 56 | 31 | 45 | 11.2 |
| 90 | 124 | 118 | 39 | 21 | 58 | 11.2 |
| 100 | 149 | 92 | 58 | 19 | 42 | 10.1 |
| 150 | 160 | 83 | 55 | 26 | 36 | 9.4 |
| Sum/average | 907 | 717 | 383 | 171 | 342 | 11.2 |

[†] Number of instances of which the percentage gap (PG) is in the specified range.
[‡] Average percentage gap.

these propositions were not checked and the time for this check is saved, $BB^*$ worked better than the other BAB algorithms including BB.

To see the effectiveness of the lower bounds on the performance of the BAB algorithm, we compared four BAB algorithms: BB, which is the BAB algorithm with both lower bounds, i.e., BB algorithm in which both lower bounds are used; $BB_N$, with neither of the lower bounds; $BB_{LB1}$, with LB1 only; and $BB_{LB2}$, with LB2 only. All the dominance properties presented in this paper were used in these BAB algorithms. Results are summarized in Table 6, which shows the average CPU time (ACPUT) for an instance and the average ratio of the number of nodes generated (ARNG) in the BAB algorithm to the number of nodes that would have been generated without the lower bounds. As can be seen in the table, using the two lower bounds helped to reduce computation time significantly, which shows the effectiveness of the lower bounds. Actually, when the number of jobs were 17, 19, 21 and 23, $BB_N$ did not solve (to optimality) 6, 18, 28 and 36 instances, respectively, among 36 instances within the CPU time limit, 3600 s. Between the two lower bounds, LB1 was more effective than LB2.

Finally, we tested the heuristic algorithm for large-size instances with the number of jobs (50, 60, 70, 80, 90, 100, and 150). For this test, 2520 instances were generated randomly in the same way as the one explained earlier. To show the performance of the heuristic algorithm, we compared the heuristic solutions with lower bounds obtained by applying CPLEX to a mixed integer-linear programming problem [RP] that is obtained by relaxing the integrality constraint of $x_{ir}$ of [P] given in Section 2. That is, the binary variable $x_{ir}$ is relaxed to a continuous variable with range from 0 to 1. Then, the solution value of [RP] is a lower bound on the solution value of [P]. Results are summarized in Table 7. Overall, the average percentage gap of the heuristic solutions from the solutions obtained by CPLEX was 11.2%, and the value decreases gradually as the number of jobs increases.

## 6. Concluding remarks

This study investigated a scheduling problem of minimizing total tardiness of jobs in a two-machine flowshop in which the first machine requires maintenance activities. We assumed that maintenance activities have to be started before the cumulative working time after the previous maintenance exceeds a given limit. We developed dominance properties and lower bounds, and suggested a branch and bound (BAB) algorithm by using those. Computational experiments were performed to evaluate the BAB algorithm and results showed that the suggested BAB algorithm could find optimal solutions for problems with up to 24 jobs in a reasonable amount of computation time.

Further research may be needed for extended versions of this research. For example, it may be necessary to develop effective dominance properties and lower bounds for the flowshop problems composed of multiple machines. Also, one interesting problem is a more general case such as those in which maintenance activities are required not only on the first machine but also on the other machine(s). In addition, one may need to consider flexible maintenance in terms of the number of processed jobs. Also, future works are required to devise heuristic algorithms, which give good or near-optimal solutions in a reasonably short computation time.

## References


Akturk, M. S., Ghosh, J. B., & Gunes, E. D. (2003). Scheduling with tool changes to minimize total completion time: A study of heuristics and their performance. *Naval Research Logistics, 50*, 15–30.

Allaoui, H., Artiba, A., Elmaghraby, S. E., & Riane, F. (2006). Scheduling of a two-machine flowshop with availability constraints on the first machine. *International Journal of Production Economics, 99*, 16–27.

Allaoui, H., Lamouri, S., Artiba, A., & Aghezzaf, E. (2008). Simultaneously scheduling *n* jobs and the preventive maintenance on the two-machine flow shop to minimize the makespan. *International Journal of Production Economics, 112*, 161–167.

Błażewicz, J., Breit, J., Formanowicz, P., Kubiak, W., & Schmidt, G. (2001). Heuristic algorithms for the two-machine flowshop with limited machine availability. *Omega, 29*, 599–608.

Breit, J. (2004). An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint. *Information Processing Letters, 90*, 273–278.

Chen, J. S. (2006). Single-machine scheduling with flexible and periodic maintenance. *Journal of the Operational Research Society, 57*, 703–710.

Chen, J. S. (2008a). Optimization models for the tool change scheduling problem. *Omega-International Journal of Management Science, 36*, 888–894.

Chen, J. S. (2008b). Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan. *European Journal of Operational Research, 190*, 90–102.

Cheng, T. C. E., & Liu, Z. (2003). 3/2-approximation for two-machine no-wait flowshop scheduling with availability constraints. *Information Processing Letters, 88*, 161–165.

Cheng, T. C. E., & Wang, G. (2000). An improved heuristic for two-machine flowshop scheduling with an availability constraint. *Operations Research Letters, 26*, 223–229.

Cui, W. W., & Lu, Z. (2017). Minimizing the makespan on a single machine with flexible maintenances and jobs' release dates. *Computers & Operations Research, 80*, 11–22.

Espinouse, M. L., Formanowlcz, P., & Penz, B. (1999). Minimizing the makespan in the two-machine no-wait flow-shop with limited machine availability. *Computers & Industrial Engineering, 37*, 497–500.

Espinouse, M. L., Formanowlcz, P., & Penz, B. (2001). Complexity results and approximation algorithms for the two-machine no-wait flow-shop with limited machine availability. *Journal of the Operational Research Society, 52*, 116–121.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy-Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Gurel, S., & Akturk, M. S. (2008). Scheduling preventive maintenance on a single CNC machine. *International Journal of Production Research, 46*, 6797–6821.

Hnaien, F., Yalaoui, F., & Mhadhbi, A. (2015). Makespan minimization on a two-machine flowshop with an availability constraint on the first machine. *International Journal of Production Economics, 164*, 95–104.

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

Kim, Y.-D. (1993). A new branch and bound algorithm for minimizing mean total tardiness in two-machine flowshops. *Computers & Operations Research, 20*, 391–401.

Koulamas, C. (1994). The total tardiness problem: Review and extensions. *Operations Research, 42*, 1025–1041.

Kubiak, W., Błażewicz, J., Formanowicz, P., Breit, J., & Schmidt, G. (2002). Two-machine flow shops with limited machine availability. *European Journal of Operational Research, 136*, 528–540.

Kubzin, M. A., Potts, C. N., & Strusevich, V. A. (2009). Approximation results for flowshop scheduling problems with machine availability constraints. *Computers & Operations Research, 36*, 379–390.

Lee, C. Y. (1997). Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint. *Operations Research Letters, 20*, 129–139.
Luo, W., Cheng, T. C. E., & Ji, M. (2015). Single-machine scheduling with a variable maintenance activity. *Computers & Industrial Engineering, 79*, 168–174.
Mashkani, O., & Moslehi, G. (2016). Minimizing the total completion time in a single machine scheduling problem under bimodal flexible periodic availability constraints. *International Journal of Computer Integrated Manufacturing, 29*, 323–341.
Mosheiov, G., & Sarig, A. (2009). Scheduling a maintenance activity to minimize total weighted completion-time. *Computers & Mathematics with Applications, 57*, 619–623.
Nawaz, M., Enscore, E. E., & Han, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*, 91–95.
Pan, J. C. H., Chen, J. S., & Chao, C. H. (2002). Minimizing tardiness in a two-machine flow-shop. *Computers & Operations Research, 29*, 869–885.
Qi, X., Chen, T., & Tu, F. (1999). Scheduling the maintenance on a single machine. *Journal of the Operational Research Society, 50*, 1071–1078.
Sbihi, M., & Varnier, C. (2008). Single-machine scheduling with periodic and flexible periodic maintenance to minimize maximum tardiness. *Computers & Industrial Engineering, 55*, 830–840.
Schaller, J. (2005). Note on minimizing total tardiness in a two-machine flowshop. *Computers & Operations Research, 32*, 3273–3281.
Schaller, J. (2009). Note on Shim and Kim's lower bounds for scheduling on identical parallel machines to minimize total tardiness. *European Journal of Operational Research, 197*, 422–426.
Sen, T., Dileepan, P., & Gupta, J. N. D. (1989). The two-machine flowshop scheduling problem with total tardiness. *Computers & Operations Research, 16*, 333–340.
Wang, G., & Cheng, T. C. E. (2001). Heuristics for two-machine no-wait flowshop scheduling with an availability constraint. *Information Processing Letters, 80*, 305–309.
Yang, D.-L., Hsu, C.-J., & Kuo, W.-H. (2008). A two-machine flowshop scheduling problem with a separated maintenance constraint. *Computers & Operations Research, 35*, 876–883.
Yang, S. L., Ma, Y., Xu, D. L., & Yang, J. B. (2011). Minimizing total completion time on a single machine with a flexible maintenance activity. *Computers & Operations Research, 38*, 755–770.
Ying, M., Chu, C., & Zuo, C. (2010). A survey of scheduling with deterministic machine availability constraint. *Computers & Industrial Engineering, 58*, 199–211.
Ying, K.-C., Ju, C.-C., & Chen, J.-C. (2016). Exact algorithms for single-machine scheduling problems with a variable maintenance. *Computers & Industrial Engineering, 98*, 427–433.
Zhu, H., Li, M., & Zhou, Z. (2015). Machine scheduling with deteriorating and resource-dependent maintenance activity. *Computers & Industrial Engineering, 88*, 479–486.