Proceedings of the Federated Conference on Computer Science and Information Systems pp. 627–635 DOI: 10.15439/2016F447 ACSIS, Vol. 8. ISSN 2300-5963

Minimizing Total Completion Time in Flowshop with Availability Constraint on the First Machine

Yumei Huo
Department of Computer Science
College of Staten Island, CUNY
Staten Island, New York 10314, USA,
Email: yumei.huo@csi.cuny.edu

Hairong Zhao
Department of Mathematics, Computer Science & Statistics
Purdue University Northwest
2200 169th Street, IN 46323, USA
Email: hairong@pnw.edu

***Abstract*—We study the problem of minimizing total completion time in 2-stage flowshop with availability constraint. This problem is NP-hard in the strong sense even if both machines are always available. With availability constraint, although a bulk of research papers have studied the makespan minimization problem, there is no research done on the total completion time minimization. This paper is the first attempt to tackle this problem. We focus on the case that there is a single unavailable interval on the first machine only. We show that several special cases can be solved optimally or approximated within a constant factor. For the general case, we develop some lower bounds and dominance rules. Then we design and implement a branch and bound algorithm. We investigate the effectiveness of different lower bounds and the dominance rules by computational experiments. We also study how the start time and the duration of the unavailable interval affects the efficiency of the branch and bound algorithm.**

I. Introduction

SCHEDULING with machine availability constraint has attracted more and more research effort since early nineties. Machine availability constraint is prevalent in all real industrial settings. A machine may be unavailable due to breakdown, preventive maintenance, or processing unfinished jobs from a previous scheduling horizon. With the machine availability constraint, many classical problems have to be reconsidered. While some of the problems can still be easily solved, many of them become more complicated and new optimal algorithms/heuristics need to be designed. Many papers have been devoted to various scheduling problems with machine availability constraint, see the survey papers by Lee, Lei and Pinedo ([18]), Sanlaville and Schmidt ([21]), Schmidt ([22]), Ma, Chu and Zuo ([20]), etc.

For flowshop scheduling with availability constraint, most research is done for two-stage flowshop problems([20]): there are two machines, machine 1 and machine 2. Each machine may have one or more unavailable intervals; There are n jobs, 1, 2, . . ., n. Each job j, $1 \leq j \leq n$, has two operations a_j and b_j which have to be processed on machine 1 and on machine 2, respectively. The operation b_j cannot start on machine 2 before a_j finishes on machine 1. We want to find a schedule of the jobs so that some objectives are optimized. The bulk of flow shop research in the last decades has been focused on the minimization of the maximum of the job completion time, i.e. the length or makespan of a schedule. However, Gupta and Dudek [8] pleaded that criteria in which the costs of each job are reflected have a better economic interpretation than the makespan objective has. This paper deals with the minimization of the sum of the job completion times in a two-machine flow shop.

Based on the effect of the availability constraint to the disrupted job's processing, researchers discuss three cases, namely resumable, nonresumable, and semiresumable. Resumable and nonresumable cases are defined by Lee in [16]. In the resumable case, preemption is allowed thus jobs can be resumed after being interrupted by the unavailable interval. In the nonresumable case, a job has to be restarted if interrupted by the interval. Semiresumable case is defined by Lee in [19] and is between the resumable case and nonresumble case. In this case, a job doesn't have to be restarted from scratch, instead, a fraction of the job needs to be reprocessed after the machine is available.

A. Literature Review

Almost all research papers about flow shop scheduling with availability constraint are with respect to makespan criterion ([20]). Lee ([17]) showed that if there is only one unavailable interval, either on the first machine or on the second machine,the problem is NP-hard in the ordinary sense. If there are an arbitrary number of unavailable intervals on the first machine but the second machine is always available, or there there are an arbitrary number of unavailable intervals on the second machine but the first machine is always available, the problem becomes strongly NP-hard ([15]) in either case. Furthermore, for the former case, Johnson's rule gives a 2-approximation regardless of the number of unavailable intervals; On the other hand, for the latter case, there is no polynomial time constant approximation for any constant even if there are only two unavailable intervals on the second machine and the first machine is always available. Many heuristics, meta-heuristics and exact methods are developed for the general problem, see the surveys [18],[21], [22], [20] and the references therein for more details.

For the problem of minimizing the total completion time with availability constraint, we are not aware of any research so far. On the other hand, when both machines are available, the problem has attracted a lot of attention because of its notorious intractability. A lot of research effort has been focused on finding the exact solution using branch and bound algorithm. In the following, we will review related literature.

The first research on this problem was done by Ignall and Schrage ([13]) in 1965. They gave two lower bounds and developed a branch-and-bound algorithm. For the experiments, they limited the number of jobs to be 10. In 1967, Conway et. al ([3]) showed that it is sufficient to study permutation schedules, i.e. a schedule in which both machines have the same job sequence with no unnecessary idle time between operations. By using local search and other heuristics to generate a good initial upper bound for the branch and bound algorithm, Kohler and Steiglitz ([14]) further improved the algorithm by Ignall and Schrage. They did the experiments on instances of size 10 to 50 jobs. For most instances of more than 15 jobs, only suboptimal solutions is obtained within preset time limit. In 1976, this problem was shown to be NP-Hard in the strong sense by Gary, John and Sethi([6]).

Since 1990s, the problem was studied again by a group of research papers, including [23], [10], [5], [4], [1], [12]. These papers try to improve the lower bounds by using Lagrangian relaxation ([23], [10]) or networking formulation ([1]), and/or propose some dominance rules ([5],[4]), consequently improve the performance of the branch-and-bound procedure and solve bigger problems. Della Croce et al.'s ([4]) branch-and-bound algorithm can solve up to 45 (30) job problems when processing times are uniformly distributed in the [1,10] ([1,100]) range. In 2004, using a lower bound scheme based on a network formulation, the branch and bound algorithm by Akkan and Karabati ([1]) can solve problems with up to 60 (45) jobs, where processing times are uniformly distributed in the [1,10] ([1,100]) range. At about the same time, Hoogeveen et al. ([12]) also used improved lower bounds by LP to solve instances of 40 jobs within reasonable time.

Very few papers studied the constant approximation algorithms or solvable cases. The first one is by Gonzales and Sahni ([7]) who showed that SPT (Shortest Processing Time first) rule gives an m-approximation for m stage flowshops. Thus for 2-stage flowshop, SPT is a 2-approximation. Later, Hoogeveen and Kawaguvhi ([9] refined this bound for 2-stage flowshop and they showed that the approximation ratio of SPT is $2\beta/(\alpha+\beta)$, where $\alpha = \min\{a_j, b_j\}$, $\beta = \max\{a_j, b_j\}$. They also studied some special cases. Specifically, they showed that

1) if $a_j = a$ for all jobs j, the problem remains NP-hard in the strong sense and SPT rule gives $4/3$- approximation schedule and the bound is tight.
2) if $b_j = b$ for all jobs j, then SPT rule generates an optimal schedule.
3) if $a_j \geq b_j$ for all jobs j, scheduling the jobs in non-decreasing order of a_j gives an optimal schedule.
4) if $a_j \leq b_j$ for all jobs j, the problem can be solved optimally in $O(n^3)$ time.

B. New Contributions

Our paper is the first attempt to tackle the total completion time minimization problem in the 2-stage flowshop with availability constraint. Given the complexity of the problem, we focus on the case that there is a single unavailable interval on the first machine. We consider the resumable case only. We first show that SPT still provides a 2-approximation if the single unavailable interval starts early so that no a_j can finish before it. We also show that some other special cases can be solved or approximated within a constant factor.

For the general case, we give some lower bounds and dominance rules and develop a branch and bound algorithm. We investigate the effectiveness of different lower bounds and the dominance rules by computational experiments. We also study how the start time and the duration of the unavailable interval affects the efficiency of the branch and bound algorithm.

C. Organization

The paper is organized as follows. We first give some preliminary results in Section II. In Section III, we give a mathematical formulation of the problem and develop five lower bounds of the optimal solution. In Section IV, we develop some dominance rules and a branch and bound algorithm. Then we give analysis of the experimental results. Finally, we conclude in section V.

II. Preliminary results

Let us first introduce some notations. To indicate the machine unavailability constraint, most literature extends the $\alpha \mid \beta \mid \gamma$ notation, by adding two new components. In the α field, h_{ik} represents the problem with k unavailable intervals on the ith machine. So h_{11} and h_{21} represents the problem with one unavailable interval on the first machine and on the second machine respectively. The second component is in the β field, we use $r-a$, $nr-a$ and $sr-a$ to denote resumable, nonresumable and semiresumable availability constraints, respectively. Thus, our problem, minimizing the total completion time with a single unavailable interval on machine 1, can be denoted as $F2, h_{11} \mid r-a \mid \sum C_j$.

Like the classical model, one can show that there is an optimal schedule that is a permutation schedule. So we only consider permutation schedules. Let S be a permutation schedule of the jobs, for job j, we use $C_{a,j}(S)$, $C_{b,j}(S)$ to denote the completion time of the first operation a_j and the second operation b_j in S, respectively. The completion time of job j in S, is denoted by $C_j(S)$ which is equal to $C_{b,j}(S)$. We use $[j]$ to denote the job scheduled in position j on both machines, and use $C_{a,[j]}(S)$ and $C_{b,[j]}(S)$ to represent the completion of $a_{[j]}$ and $b_{[j]}$ in S, respectively. If it is clear from the context, S may be omitted.

We frequently sort the jobs, for convenience, we use SPT_a, SPT_b, SPT to denote the rule that schedules the jobs in non-decreasing order with respect to a_j, b_j, and $(a_j + b_j)$ respectively.

From [7], we know that SPT is a 2-approximation algorithm when both machines are available. When there is a single unavailable interval, we show in the following that the performance of SPT depends on the start time of the interval.

Lemma 2.1: Let $[s,t]$ be the unavailable interval for $F2, h_{11} \mid r-a \mid \sum C_j$, if $s \geq \sum_{j=1}^n a_j$ or $s < \min\{a_j\}$, SPT is a 2-approximation.

Proof: First, if $s \geq \sum_{j=1}^n a_j$, then all a_j-s can finish before the unavailable interval, thus the interval has no effect on the jobs at all. By [7], SPT is a 2-approximation.

Now, let us consider the case $s < \min\{a_j\}$. Suppose the jobs are indexed so that $(a_j + b_j) \leq (a_{j+1} + b_{j+1})$ for $1 \leq j \leq n-1$. We denote $P1$ to be the problem of scheduling of n jobs with processing time $(a_j + b_j)$ on a single machine with the unavailable interval $[s,t]$. Then one can easily show that the optimal schedule S_1 for $P1$ is obtained by SPT rule, and the completion time of j-th job will be $C_{[j]}(S_1) = t + \sum_{k=1}^j (a_k + b_k)$.

Let S be the schedule generated by SPT rule for $F2, h_{11} \mid r-a \mid \sum C_j$. Apparently, we have

$$C_j(S) \leq C_{[j]}(S_1) = t + \sum_{k=1}^{j} (a_k + b_k).$$

Now, consider the relaxed flowshop problem $P2$ where a_j and b_j can be concurrently executed and the completion time is the time when both a_j and b_j finish. Let S_2 be the optimal schedule for this relaxed flowshop problem. We must have that

$$C_{[j]}(S_2) \geq \tfrac{1}{2}(t + \sum_{k=1}^{j} (a_{[k]} + b_{[k]})) \geq \tfrac{1}{2}(t + \sum_{k=1}^{j} (a_k + b_k)).$$

Let S^* be the optimal schedule for $F2, h_{11} \mid r-a \mid \sum C_j$. Then we have

$$C_{[j]}(S^*) \geq C_{[j]}(S_2) \geq \frac{1}{2}(t + \sum_{k=1}^{j} (a_k + b_k)) \geq \frac{1}{2} C_{[j]}(S).$$

Thus, $\sum_{j=1}^n C_j(S) \leq \sum_{j=1}^n 2C_{[j]}(S^*)$. This completes the proof. ■

Unfortunately, if s is arbitrary, SPT may generate a schedule that has a very large approximation ratio.

Lemma 2.2: For $F2, h_{11} \mid r-a \mid \sum C_j$ with a single unavailable interval $[s,t]$, if $s \geq a_1$ where $a_1 + b_1 = \min\{a_j + b_j\}$, SPT can generate a schedule whose approximation ratio is n in the worst case.

Proof: Given an instance I of the problem with a single unavailable interval $[s,t]$. Let I' be the corresponding instance of $F2 \mid\mid \sum C_j$ which is obtained from I by removing the unavailable interval. Let C_{opt} and C'_{opt} be the minimum total completion time for I and I', respectively. It is obvious that $C'_{opt} \leq C_{opt}$.

Let S be the schedule generated by SPT rule for instance I and S' be the SPT schedule for I'. For convenience, suppose the jobs are indexed in non-decreasing order of $(a_j + b_j)$, $1 \leq j \leq n$. Let i be the last job such that $C_{a,i}(S) \leq s$. Since $a_1 \leq s$, $i \geq 1$. It is clear that for $j \leq i$, we have $C_j(S) = C_j(S')$; for all $j > i$, we have $C_{a,j}(S) = C_{a,j}(S') + (t-s)$, so $C_j(S) = C_{b,j}(S) \leq C_{b,j}(S') + (t-s)$. Thus,

$$\begin{aligned}\sum_{j=1}^n C_j(S) &\leq \sum_{j=1}^i C_j(S') + \sum_{j=i+1}^n (C_j(S') + (t-s)) \\ &\leq \left(\sum_{j=1}^n C_j(S')\right) + (n-i)(t-s).\end{aligned}$$

On the other hand, since $\sum_{j=1}^n a_j > s$, at least one job must finish after t in any schedule. Thus the completion time of the last job in the optimal schedule is

$$C_{[n]}(S^*) \geq ((t-s) + \sum_{j=1}^n (a_j + b_j))/2 \geq \tfrac{(t-s)}{2} + \tfrac{C_n(S')}{2}.$$

For j-th job in S^*, $j \neq n$, we have that $C_{[j]}(S^*) \geq C_j(S')/2$. Thus we have $\sum_{j=1}^n C_j(S^*) \geq \sum_{j=1}^n C_j(S')/2 + (t-s)/2$ and

$$\begin{aligned}\sum_{j=1}^n C_j(S) &\leq \left(\sum_{j=1}^n C_j(S')\right) + (n-i)(t-s) \\ &\leq (2\left(\sum_{j=1}^n C_j(S^*) - (t-s)\right) + (n-i)(t-s) \\ &\leq 2\sum_{j=1}^n C_j(S^*) + (n-i-1)(t-s) \\ &\leq (n-i+1)\sum_{j=1}^n C_j(S^*).\end{aligned}$$

Since we assume $i \geq 1$, $\sum_{j=1}^n C_j(S) \leq nC_{opt}$. ■

Next, we show that for some special cases in terms of a_j-s and/or b_j-s, we can use SPT_a or SPT_b to solve the problem optimally or get a good approximation.

Lemma 2.3: SPT_a generates a schedule whose total completion time is minimum for

(a) $F2, h_{11} \mid r-a, b_j = b \mid \sum C_j$; and
(b) $F2, h_{11} \mid r-a, a_j \geq b_j \mid \sum C_j$

Proof: First consider case (a): $F2, h_{11} \mid r-a, b_j = b \mid \sum C_j$. Let S be an arbitrary schedule and suppose that the jobs are scheduled in the order of 1, 2, $\cdots$. Define $C_0 = 0$, then the completion time of job i in S is

$$C_i(S) = \max(C_{i-1}, C_{a,i}) + b,$$

where

$$C_{a,i} = \begin{cases} \sum_{j=1}^i a_j, & \text{if } \sum_{j=1}^i a_j \leq s \\ \sum_{j=1}^i a_j + (t-s), & \text{if } \sum_{j=1}^i a_j > s. \end{cases}$$

Apparently scheduling the jobs in SPT_a minimizes the total completion time.

Now we consider case (b): $F2, h_{11} \mid r-a, a_j \geq b_j \mid \sum C_j$. Suppose that $a_1 \leq a_2 \leq \cdots \leq a_n$. Since $a_j \geq a_{j-1} \geq b_{j-1}$, in the schedule generated by SPT_a, b_j can be scheduled immediately after a_j completes, thus minimizing the total completion time is the same as minimizing the total completion time of the a_j-s which is obtained by SPT_a rule. ∎

Lemma 2.4: SPT_b generates a schedule for $F2, h_{11} \mid r-a, a_j = a \mid \sum C_j$ whose total completion time is at most $7/3$ times that of the optimal schedule.

Proof: Given an instance I of problem $F2, h_{11} \mid r-a, a_j = a \mid \sum C_j$ with the unavailable interval $[s,t]$, let C_{opt} be its minimum total completion time for I. Let S be the schedule generated by the SPT rule. Let I' be the instance of $F2 \mid a_j = a \mid \sum C_j$ obtained from I by removing the unavailable interval. Let C'_{opt} be the minimum total completion time for I'. Apparently, we have $C'_{opt} \leq C_{opt}$. Let S' be the SPT schedule for I'. From [9], we know that $\sum_{j=1}^{n} C_j(S') \leq \frac{4}{3}C'_{opt} \leq \frac{4}{3}C_{opt}$. Let $i = \lfloor \frac{s}{a} \rfloor$. Then only i jobs can finish before t on the first machine in any schedule which implies that $C_{opt} > (n-i)t$. Thus we have $C_j(S) = C_j(S')$ for $1 \leq j \leq i$. For $j > i$, its completion time in S is increased by at most the length of the interval compared with that in S', thus, $C_j(S) \leq C_j(S') + (t-s)$. So we have

$$\begin{aligned}\sum_{j=1}^{n} C_j(S) &\leq \left(\sum_{j=1}^{n} C_j(S')\right) + (n-i)(t-s)\\ &\leq \tfrac{4}{3}C'_{opt} + C_{opt}\\ &\leq \tfrac{4}{3}C_{opt} + C_{opt}\\ &\leq \tfrac{7}{3}C_{opt},\end{aligned}$$

and this completes the proof. ∎

III. Mathematical Formulation

Our problem can can be formulated as follows:
$\min \sum_{j=1}^{n} C_{b,j}$ subject to

$$C_{a,j} \geq a_j \quad \forall j \tag{1}$$

$$C_{b,j} \geq C_{a,j} + b_j \quad \forall j \tag{2}$$

$$C_{a,i} \geq C_{a,j} + a_i \bigvee C_{a,j} \geq C_{a,i} + a_j \quad \forall i,j,\ i \neq j \tag{3}$$

$$C_{b,i} \geq C_{b,j} + b_i \bigvee C_{b,j} \geq C_{b,i} + b_j \quad \forall i,j,\ i \neq j \tag{4}$$

$$C_{a,j} \leq s \bigvee C_{a,j} > t \quad \forall j \tag{5}$$

$$\text{The schedule is a permutation schedule.} \tag{6}$$

The first four constraints are the same as the classical model. Constraint (1) says that the processing of any job on first machine cannot start before time 0; and the constraint (2) specifies the second operation of any job can't start before the first operation finishes; and the constraints (3) and (4) show that a machine can not process more than one job at a time. The constraint (5) is unique to our model, it says that the completion time of the first operation is either before or after the breakdown. An additional redundant constraint is (7) or (8), which will be used when we develop lower bounds.

$$C_{b,j} \geq \left(\min_{1\leq k\leq n} a_k\right) + b_j \quad \forall j \tag{7}$$

$$C_{b,j} \geq \left(\min_{1\leq k\leq n} a_k\right) + (t-s) + b_j \quad \text{if } \min_{1\leq k\leq n} a_k > s \ \forall j \tag{8}$$

A. Lower Bounds

Let C_{opt} denote the minimum total completion time of the jobs. Based on the formulation, we can develop several lower bounds for C_{opt} by relaxing some of the constraints. Quite a few lower bounds were developed in previous literature for the classical flowshop model, see [5] and the references therein. With breakdown, those lower bounds either need to be modified or not work at all. In the following, we will examine those lower bounds.

1) LB1: This lower bound corresponds to the first lower bound of Ignall and Schrage [13]. The first lower bound from [13] is obtained by relaxing the constraint (4) so that the second machine can simultaneously process as many jobs as needed. Thus, the jobs should be scheduled in SPT_a order on the first machine, and b_j-s can be scheduled immediately without any delay after a_j finishes. With breakdown, the same idea still works, except that we have to count the number of a_j-s that are scheduled after the breakdown.

Suppose $a_1 \leq a_2 \leq \cdots \leq a_n$ and k is the smallest integer such that $\sum_{j=1}^{k+1} a_k > s$ where s is the start time of the unavailable interval. It is easy to see that there are at least $(n-k)$ jobs finish after the breakdown on the first machine in any schedule. Thus

$$LB1 = \left(\sum_{j=1}^{n}\sum_{i=1}^{j} a_i\right) + (n-k)(t-s) + \sum_{j=1}^{n} b_j.$$

2) $LB2$: This corresponds to the second lower bound of Ignall and Schrage [13], which is obtained by relaxing the constraints (1) and (3). Thus, the jobs should be scheduled in SPT_b order subject to constraint (7). Again, with breakdown, we need to do a modification. If $\min\{a_i\} \leq s$, then the lower bound is not affected by the breakdown; otherwise, constraint (8) should be satisfied. Assume $b_{i_1} \leq b_{i_2} \leq \cdots \leq b_{i_n}$, then we have that

$$LB2 = \begin{cases} n\min\{a_i\} + \sum_{j=1}^{n}\sum_{p=1}^{j} b_{i_p}, & \text{if } \min a_i \leq s \\ n(\min\{a_i\} + (t-s)) + \sum_{j=1}^{n}\sum_{p=1}^{j} b_{i_p}, & \text{otherwise.}\end{cases}$$

3) $LB3$: this corresponds to the lower bound in [23], which is obtained by applying Lagrangian relaxation to constraint (2) to the classical model. The bound from [23] is based on the assumption that $C_{a,[j]} = \sum_{i=1}^{j} a[i]$ which is true for classical model but not true with breakdown. However, we can still use it to get lower bounds for partial schedules whose completion time on the first machine passes the unavailable interval. For the details on how to calculate/estimate the lower bound, see [23].

From the analysis, we can see LB4 givens a stronger bound than both LB1 and LB2.

4) $LB4$: This corresponds to LB_{DNT2} in [5] which dominates the first two lower bounds in classical model. We can adapt this bound for our model.

First note that with breakdown, it is still the case that for any schedule S and the j-th job in S, $C_{a,[j]}(SPT_a) \leq C_{a,[j]}(S)$, where SPT_a represents the schedule generated by SPT_a rule. If we relax the constraint (3), then the problem is equivalent to a single machine problem with each job j has a release time a_j or $a_j + (t-s)$ if $a_j > s$, and processing time b_j. A lower bound of the new problem is given by schedule the jobs using $SRPT$ (Shortest Remaining Processing Time First). As with the classical model, it is still true that $C_{b,[j]}(SRPT) \leq C_{b,[j]}(S)$ for S and j.

Use similar argument as in [5], we can show that for any schedule S,

$$\sum_{j=1}^{n} C_j(S) \geq \sum_{j=1}^{n} \max\left(C_{a,[j]}(SPT_a) + b_{[j]}(S), C_{b,j}(SRPT)\right) = \left(\sum_{j=1}^{n} C_{b,j}(SRPT)\right) + \Delta,$$

where

$$\Delta = \sum_{j=1}^{n} \max(0, b_j(S) - (C_{b,[j]}(SRPT) - C_{a,[j]}(SPT_a))),$$

and Δ is minimized by sorting both b_j-s of S and $(C_{b,[j]}(SRPT) - C_{a,[j]}(SPT_a))$ in non-decreasing order. By the analysis, it is easy to see that LB4 generates a bound that is better than both LB1 and LB2.

5) LB5-Lower Bound by Linear Programming: If we know that the number of first operations that finish before breakdown in the optimal schedule is K, then we can modify the formulation from [12] as follows. Let w_j be the time that $b_{[j]}$ has to wait after $a_{[j]}$ finishes and $w_1 = 0$. Thus we have

$$C_{b,[j]} = C_{a,[j]} + b_{[j]} + w_j \text{ for each position } j = 1, \ldots, n.$$

Then the problem is to minimize

$$\sum_{j=1}^{n} (C_{a,[j]} + w_j + b_{[j]})$$

subject to

$$a_{[k]} + w_k \geq b_{[k-1]} + w_{k-1}\ \forall k = 2, \ldots, n, \text{ and } k \neq K+1 \quad (9)$$

$$a_{[K+1]} + w_{[K+1]} + (t-s) \geq b_{[K]} + w_K \quad (10)$$

$$\sum_{i=1}^{K} a_{[i]} \leq s \quad (11)$$

Let x_{ij} be the binary variable such that $x_{ij} = 1$ means job i is the j-th job in the schedule and $x_{ij} = 0$ otherwise. Then the problem can be formulated as to

minimize

$$\sum_{j=1}^{n} (n-j+1) \sum_{i=1}^{n} x_{ij} a_i + \sum_{j=1}^{n} w_j + \sum_{j=1}^{n} b_j + (n-K)(t-s)$$

subject to

$$\sum_{j=1}^{n} x_{ij} \geq 1, \forall i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} \leq 1, \forall j = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ik} a_i + w_k - \left(\sum_{i=1}^{n} x_{ik-1} b_i + w_{k-1}\right) \geq 0, \forall k \neq K+1$$

$$\sum_{i=1}^{n} x_{ik} a_i + w_k + (t-s) - \left(\sum_{i=1}^{n} x_{ik-1} b_i + w_{k-1}\right) \geq 0, \ k = K+1$$

$$\sum_{j=1}^{K} \sum_{i=1}^{n} x_{ij} a_i \leq s$$

$$x_{ij} \in \{0,1\} \text{ for } i = 1, \ldots n; j = 1, \ldots, n$$

$$w_j \geq 0 \text{ for } j = 2, \ldots, n$$

The problem is that we don't know the magic number K. However, we can find its minimum possible value of $K_{\min}$ and its maximum possible value $K_{\max}$ by sorting the first operations in SPT_a and LPT_a (longest processing time first by a_js) respectively. Let $LP(k)$ be the lower bound for our problem with a specific $K = k$. Then the lower bound for our problem is $\min_{k=K_{\min}}^{K_{\max}} LP(k)$.

IV. Branch and Bound Algorithm

Our branch-and-bound procedure builds the search tree in a depth-first-search fashion. It starts with the root node at level 0. Each node at level k of the tree corresponds to an initial partial schedule in which k jobs have been put in the first k positions. For this node, at most $(n-k)$ child nodes will be created, one for each unscheduled job. The size of the search tree can be reduced by applying lower bounding technique and dominance rules at each node.

A. Upper Bound

To get an initial upper bound, we first generate some random schedules. We also generate the neighbors of these random schedules which are obtained by local interchanges. Then, we pick the best among these schedules and the schedules obtained by applying SPT, SPT_a, SPT_b. The upper bound is updated any time a leaf of the search tree results in a better schedule. At each internal node, we also calculate the upper bound using SPT, SPT_a, SPT_b based on the partial schedule, the upper bound is updated if a better schedule is obtained.

B. Lower Bound of a Partial Schedule

The LB1, LB2, LB3, LB4 and LB5 mentioned in previous section assume that no jobs have been scheduled yet. All of them can be adjusted if some jobs have been scheduled. However, it will be too expensive to calculate LB5 at each node. So we only calculate LB5 at the root node, and enhance the lower bound based on the partial schedule using the technique from [9]. All other lower bounds will be calculated at each node. If the maximum lower bound is greater than the current upper bound, the node will be cut from the search tree.

C. Dominance Rules

For the aim of improving the performance of the branch and bound algorithm, dominance rules can be used to reduce the size of the search space.

Given a (partial) schedule S, we use $C_A(S)$, $C_B(S)$ to represent the completion time of the last operation on machine 1 and 2 in S, respectively. Let $TC(S)$ be the total completion time of the jobs in S. The earliest available time for the remaining jobs on the second machine, denoted by $r(S)$, is $\max(C_B(S), C_A(S) + \min_{i\notin S} a_i)$ or $\max(C_B(S), C_A(S) + \min_{i\in S} a_i + (t-s))$ if $s < C_A(S) + \min_{i\notin S} a_i \leq t$.

We have the following dominance rules that can be applied to cut the branches of the search tree.

Lemma 4.1: Dominance rule 1. Given two partial schedule S_1 and S_2 of the same set of jobs. If $TC(S_1) \leq TC(S_2)$ and $C_B(S_1) \leq r(S_2)$, then to find the optimal schedule of all jobs, there is no need to consider the schedules based on S_2.

It is easy to see that the above lemma is true. Given a schedule S, a particular schedule that can be checked to see if it dominates S is the schedule resulting from Johnson's rule. If $C_A(S) \leq s$, we check the schedule generated by Johnson's rule for the same set of jobs. Otherwise, $C_A(S) > t$, we check the schedule that schedules the jobs in the same way as S before t, and schedule the remaining jobs after t using Johnson's rule. In both cases, the schedules being checked are guaranteed to have a makespan not greater than that of S.

Lemma 4.2: Dominance rule 2 ([4]). Given two partial schedules of the same set of jobs, S_1 and S_2, then S_2 can be pruned if both of the following inequalities hold:

$$TC(S_1) \leq TC(S_2)$$

and

$$(C_B(S_1) - C_B(S_2))q \leq TC(S_2) - TC(S_1),$$

where q is the number of unscheduled jobs.

The Lemma is obviously true, actually it holds no matter how many breakdowns on the first machine, as long as there is no breakdown on the second machine. The rule can be applied to a schedule S by checking any adjacent schedules of S. Adjacent schedule can be obtained by swapping any two jobs i and j or by inserting j before or after i, etc.

Lemma 4.3: Dominance rule 3. Given a partial schedule S and an unscheduled job i such that $a_i \leq b_i$. If for any other unscheduled job j , we have $a_i \leq a_j$ and $b_i \leq b_j$, and $C_A(S) + a_i + a_j \leq s$ or $C_A(S) + \min_{k\notin S} a_k > s$ then there exists an optimal schedule that job i is the first among all the unscheduled jobs.

Proof: A similar rule for the classical model without availability constraint is given by Croce et. al ([4]) (Dominance rule D2). One can prove the lemma by adapting the proof from [4]. ■

It should be noted that this dominance rule holds only if $C_A(S) + a_i + a_j \leq s$ or $C_A(S) + \min_{k\notin S} a_k > s$. For example, consider $a_1 = a_2 = a_3 = 2$, $b_1 = 2$, $b_2 = 3$ and $b_3 = 4$. Suppose the unavailable interval is $[5, 11]$. At the beginning, S is empty, the conditions are satisfied for $i = 1$, thus by Lemma 4.3, we know there must exist an optimal schedule starting with job 1. So we don't need to consider the schedules that start with job 2 or 3. On the other hand, if the unavailable interval is $[3, 9]$, neither $C_A(S) + a_i + a_j \leq s$ nor $C_A(S) + \min_{k\notin S} a_k > s$ holds for $i = 1$. So it is not clear that there exists an optimal schedule starting with job 1. It turns out the the optimal schedule starts with the last job 3, not job 1.

Lemma 4.4: Dominance rule 4. Given a partial schedule π and two unscheduled jobs i and j, such that $a_i \leq a_j$, $b_i \geq b_j$. Let $L = s - C_A(\pi)$, and π_1 and π_2 be sequences of unscheduled jobs. Let $S_1 = \pi i \pi_1 j \pi_2$ and $S_2 = \pi j \pi_1 i \pi_2$. Then S_1 dominates S_2 if one of the following cases is true,

(a) $0 < L \leq a_i \leq a_j$, and

$$\begin{aligned}&\max(C_A(\pi) + a_i + (t-s), C_B(\pi)) + b_i\\ &\leq \max(C_A(\pi) + a_j + (t-s), C_B(\pi)) + b_j;\end{aligned}$$

(b) $L \leq 0$ or $L > a_i$, and

$$\begin{aligned}&\max(C_A(\pi) + a_i, C_B(\pi)) + b_i\\ &\leq \max(C_A(\pi) + a_j, C_B(\pi)) + b_j.\end{aligned}$$

Proof: If we could show (1) $C_{b,i}(S_1) + C_{b,j}(S_1) \leq C_{b,i}(S_2) + C_{b,j}(S_2)$, and (2) $C_{b,k}(S_1) \leq C_{b,k}(S_2)$ for any $k \neq i, j$, then both the makespan and the total completion time of S_1 are less than or equal to that of S_2, thus the lemma is true.

We consider case (a) first. Apparently, for any $k \in \pi$, we have $C_{a,k}(S_1) = C_{a,k}(S_2)$ and $C_{b,k}(S_1) = C_{b,k}(S_2)$. For job i in S_1 and job j in S_2, the condition $0 < L \leq a_i \leq a_j$ means that neither a_i not a_j can finish before the unavailable interval given the partial schedule π. So for S_1, we have $C_{a,i}(S_1) = C_A(\pi) + a_i + (t-s)$, $C_{b,i}(S_1) = \max(C_{a,i}(S_1), C_B(\pi)) + b_i$.

Similarly for S_2, we have $C_{a,j}(S_2) = C_A(\pi) + a_j + (t-s)$, $C_{b,j}(S_2) = \max(C_{a,j}(S_2), C_B(\pi)) + b_j$.

Since $a_i \leq a_j$, we have $C_{a,i}(S_1) \leq C_{a,j}(S_2)$. The condition $\max(C_A(\pi) + a_i + (t-s), C_B(\pi)) + b_i \leq \max(C_A(\pi) + a_j + (t-s), C_B(\pi)) + b_j$ implies

$$C_{b,i}(S_1) \leq C_{b,j}(S_2).$$

Consequently, for any $k \in \pi_1$, it must be true that $C_{a,k}(S_1) \leq C_{a,k}(S_2)$ and $C_{b,k}(S_1) \leq C_{b,k}(S_2)$. Therefore $C_B(\pi i \pi_1) \leq C_B(\pi j \pi_1)$.

Clearly, $C_{a,j}(S_1) = C_{a,i}(S_2)$. Since $b_i \geq b_j$, we have $C_{b,j}(S_1) = \max(C_{a,j}(S_1), C_B(\pi i \pi_1) + b_j \leq C_{b,i}(S_2) = \max(C_{a,i}(S_1), C_B(\pi j \pi_1)) + b_i$ Thus, for any $k \in \pi_2$, $C_{a,k}(S_1) = C_{a,k}(S_2)$ and $C_{b,k}(S_1) \leq C_{b,k}(S_2)$.

For case (b), given the partial schedule π, if $L \leq 0$, then all the remaining jobs including i and j have to be scheduled after the unavailable interval. If $L \geq a_i$, then a_i can finish before the unavailable interval, but a_j may or may not finish before the unavailable interval. Either way, the inequality

$$\max(C_A(\pi)+a_i, C_B(\pi))+b_i \leq \max(C_A(\pi)+a_j, C_B(\pi))+b_j$$

guarantees that $C_{b,i}(S_1) \leq C_{b,j}(S_2)$. We can use similar argument as case (a) to prove that $C_{b,k}(S_1) \leq C_{b,k}(S_2)$ for any $k \neq i, j$, thus the lemma is true. ■

D. Experimental Results

The instances are generated as follows. For processing times, we use three distributions, $[1, 10]$, $[1, 50]$, $[1, 100]$. It is known that the distribution $[1, 100]$ generates the most difficult type of problem instances. The $[1, 10]$ distribution is the easiest and also the most practical distribution. The number of jobs n takes on values 10, 15, 20, 25, 30, 35. The start time s of the unavailable intervals are generated from four uniform distributions, $[0, \frac{1}{4}A]$, $[\frac{1}{4}A, \frac{1}{2}A]$, $[\frac{1}{2}A, \frac{3}{4}A]$, $[\frac{3}{4}A, A]$, where $A = \sum_{j=1}^{n} a_j$. The duration of the unavailable intervals, $(t-s)$, takes the values of 1% 20%, 40%, 60%, 80%, 100% of A. For each combination of n and a processing time distribution, we generate 25 sets of n jobs; then for each set of n jobs, we generate 16 instances by combining different distribution of s and $(t-s)$.

We pre-process the jobs in SPT, SPT_a, SPT_b and Johnson's rule and keep this order so that they can be accessed at each node of the tree. We set the termination criteria of the branch and bound algorithm by limiting the number of the nodes explored to be 5000000.

1) Comparison of Lower Bounds At the Root Node: From previous discussion, when $min_{1 \leq i \leq n} a_i < s$, the LB2 bound doesn't put much consideration of the unavailable interval on the first machine, so its is expected to be the worst lower bound. $LB3$ can only be applied for partial schedules passing the unavailable interval. We also know that LB4 dominates LB1 and LB2. LB5 is generated by linear programming.

Experiments show that at the root node, LB5 always gives the best lower bound, and LB4 is very close to LB5. On average, LB1 is not that bad. The performance of LB2 can vary a lot from instance to instance. Table I shows the ratios of the LB5 with LB1, LB2, and LB4 for the instances with $n = 35$. For each distribution of the processing times, we list the minimum, maximum, average ratios and the standard deviation.

We also notice that the gap of LB5 at the root and the optimal solutions is very small. Table II shows the value of LB5 at the root compared with the optimal solution (or the best solution found if optimal not found) for the instances with $n = 35$. We can see on average, LB5 is within 1% of the optimal (best) solution.

2) Lower Bounds at the Internal Node: Although LB5 is very good and close to the optimal solution, but its computation takes a lot of time, thus we cannot afford to compute at other nodes. So we use the method from [12] to enhance LB5 based on the partial schedule at each node. For each instance, we record the number of nodes it explores before it finds the optimal solution or reach the maximum node limit, 5000000. For each of the lower bound LB1, . . ., LB5, We also record the number of nodes such that the lower bound is maximum. We observed that LB4 provides the best lower bound at internal nodes for almost all instances. Fig.1 shows 125 instances of 35 jobs. For each instance, among all the nodes explored, the figure shows the number of times when each of LB3, LB4 and LB5 equals to $\max(LB3, LB4, LB5)$. We can clearly see that $LB4$ provides the best lower bounds at the majority of the nodes in the search tree, and $LB3$ provides good lower bound only for a very small portion of the nodes in the tree.

3) Dominance Rule: To find out the effectiveness of the dominance rules, we run experiment with and without applying dominance rules. For each instance, we compare the number of nodes explored by the algorithm in both cases. We compute the ratio of the number of nodes with dominance rules and the number of nodes without dominance rules. Table III lists minimum, maximum, and average ratio for different processing time distribution when $n = 20$. We can see on average, the number of the nodes with dominance rules is only 10% of the number of nodes with no dominance rules, i.e 10 times speed up of the algorithm.

4) Unavailable Interval: Fig. 2 shows how the starting time of the unavailable interval affects the size of the search tree. In the figure, we show the number of nodes explored for 120 instances of $n = 20$. The instances are divided into 4 groups of 30 instances whose unavailable interval starting times are drawn from the distribution $[0, \frac{1}{4}A]$, $[\frac{1}{4}A, \frac{1}{2}A]$, $[\frac{1}{2}A, \frac{3}{4}A]$, $[\frac{3}{4}A, A]$, respectively. The figure suggests that the search tree size is getting bigger as the unavailable interval starts later. We can also see that the LB3 plays a bigger role when the unavailable intervals starts before $\frac{1}{4}A$.

Fig. 3 shows how the length of the unavailable interval affects the size of the search tree. The effect is not that obvious.

5) Problems Size Solved: Our algorithm is implemented in C++, and Gurobi is used for linear programming. Within the maximum number of nodes 5000000, we were able to solve the instances of 30 jobs. When $n = 35$, we can solve the majority of the instance if the processing time distribution is from $[1, 10]$. For the other two distributions, $[1, 50]$ and $[1, 100]$, we can only solve some of the instances. For example, for the 125 instances we used from Table I and Table II, the number of solved instances are 88, 28, 19, respectively. On the other hand, from the Table II, we know that the best solutions found by the branch and bound algorithm are actually very close to the optimal solution.

V. Conclusion

In this paper, we studied the problem of minimizing total completion time subject to the constraint that there is a

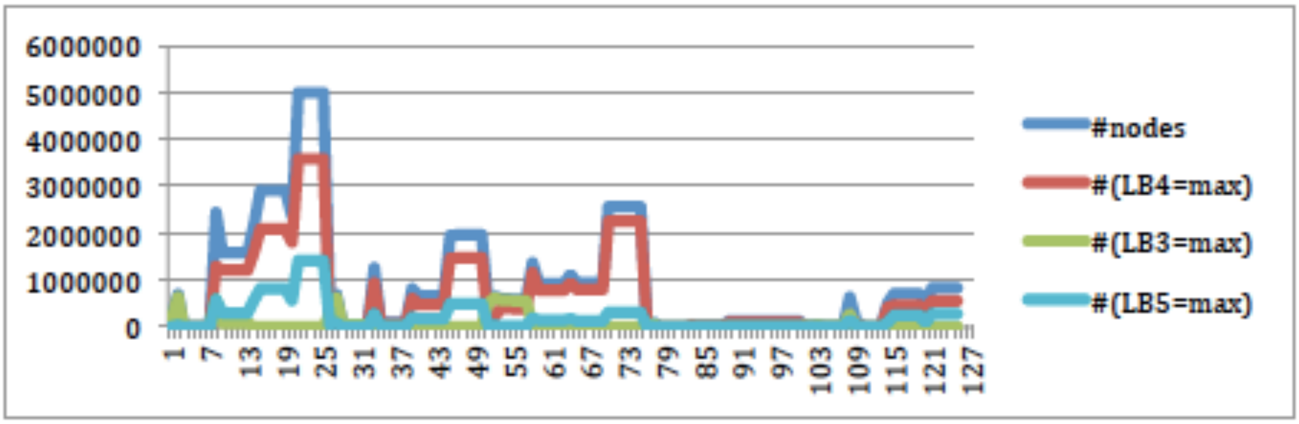

Fig. 1. Comparison of the lower Bounds at Internal Nodes

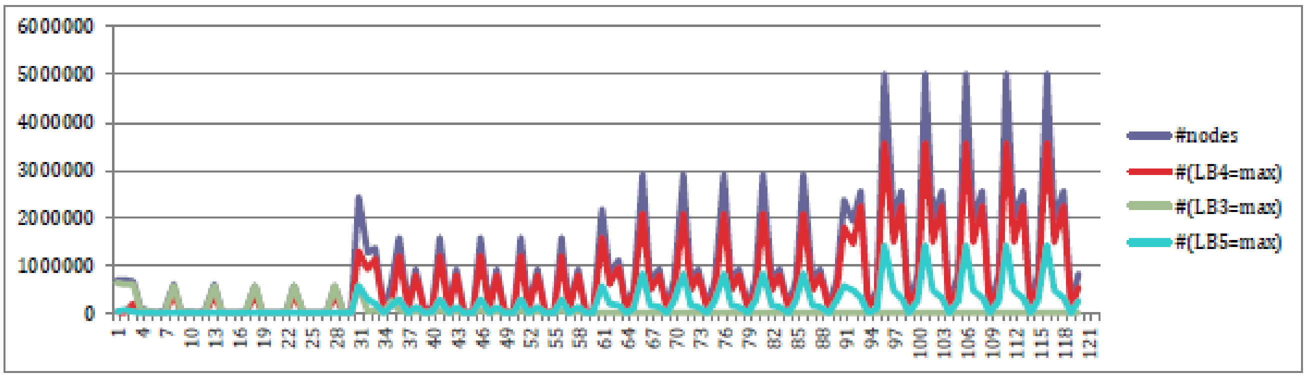

Fig. 2. Number of nodes explored by starting time of the unavailable interval, each of $[0, \frac{1}{4}A]$, $[\frac{1}{4}A, \frac{1}{2}A]$, $[\frac{1}{2}A, \frac{3}{4}A]$, $[\frac{3}{4}A, A]$ has 30 instances

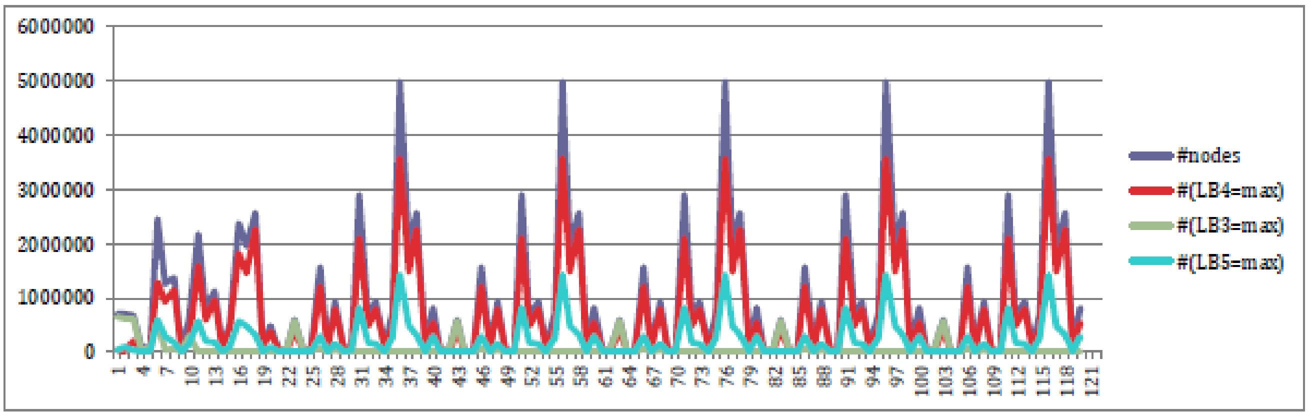

Fig. 3. Number of nodes explored by length of the unavailable interval, each of $1\%A$, $20\%A$, $40\%A$,$60\%A$,$80\%A$ and $100\%A$ has 20 instances

TABLE I
LOWER BOUNDS COMPARISON AT THE ROOT NODE

	[1,10] distribution			[1,50] distribution			[1,100] distribution		
	LB5/LB1	LB5 /LB2	LB5 / LB4	LB5/LB1	LB5 /LB2	LB5 / LB4	LB5/LB1	LB5 /LB2	LB5 / LB4
min	1.018	1.089	1.018	1.005	1.072	1.005	1.005	1.054	1.005
max	1.174	3.410	1.119	1.301	3.454	1.151	1.253	3.351	1.160
average	1.080	1.601	1.069	1.102	1.692	1.067	1.089	1.598	1.064
standard deviation	0.042	0.485	0.027	0.084	0.552	0.038	0.069	0.486	0.043

TABLE II
LB5 AT THE ROOT COMPARED WITH OPTIMAL SOLUTION/BEST SOLUTION FOUND

	[1,10]	[1,50]	[1,100]
min	0.001	0.002	0.002
max	0.013	0.046	0.028
average	0.001	0.014	0.012
standard deviation	0.003	0.009	0.007

TABLE III
THE RATIO OF NUMBERS OF NODES IN SEARCH TREES USING DOMINANCE RULES WITH THAT NOT USING DOMINANCE RULES

	[1,10]	[1,50]	[1,100]
min	0.002	0.011	0.004
max	0.149	0.714	0.244
average	0.027	0.117	0.069

single unavailable interval on the first machine. We show that some special cases can be solved optimally or within a constant factor; and for the general case, SPT rule gives an n-approximation. We performed computational experiments to investigate the effectiveness of different lower bounds and the dominance rules. It is observed that at the root node, LB5 provides the best lower bound with the expense of time, LB4 can provide almost good lower bounds but more efficiently at the root node and gives the best lower bounds at most internal nodes. The dominance rules speeds up the algorithm dramatically. We also observed that the earlier the the unavailable interval starts, the more efficient the algorithm is; on the other hand, the duration of the unavailable interval does not affect the efficiency of the branch and bound algorithm very much.

REFERENCES

[1] Akkan, C. and S. Karabati, The two-machine flowshop total completion time problem: Improved lower bounds and a branch-and-bound algorithm, European Journal of Operational Research, 159, 420-429, 2004, http://dx.doi.org/10.1016/S0377-2217(03)00415-6.

[2] Cadambi, B. W. and Y. S. Sathe, Two-machine flowshop scheduling to minimise mean flow time, Opsearch, 30, 35-41,1993.

[3] Conway, R. W., W. L. Maxwell, and L. W. Miller, Theory of Scheduling. Addison-Wesley, Reading, MA, 1967.

[4] Della Croce, F., M. Ghirardi, and R. Tadei, An improved branch-and-bound algorithm for the two machine total completion time flow shop problem, European Journal of Operational Research, 139, 293-301, 2002, http://dx.doi.org/10.1016/S0377-2217(01)00374-5.

[5] Della Croce, F., V. Narayan, and R. Tadei, The two-machine total completion time flow shop problem, European Journal of Operational Research, 90, 227-237, 1996, http://dx.doi.org/10.1016/0377-2217(95)00351-7.

[6] Garey, M. R., D. S. Johnson, and R. Sethi, The complexity of flowshop and jobshop scheduling, Mathematics of Operations Research, 13, 330-348, 1976, http://dx.doi.org/10.1287/moor.1.2.117.

[7] T. Gonzalez and S. Sahni, Flowshop and jobshop schedules: Complexity and approximation. Operations Research 26, 36-52, 1978,http://dx.doi.org/10.1287/opre.26.1.36.

[8] J.N.D. Gupta and R.A.Dudek, Optimality criteria for flowshop schedules, *AIIE Trans*, 3, 199-205, 1971, http://dx.doi.org/10.1080/05695557108974807.

[9] H. Hoogeveen and T. Kawaguchi, Minimizing total completion time in a two-machine flowshop: Analysis of special cases, Mathematics of Operations Research, Vol. 24 Issue 4, 887-910, 1999, http://dx.doi.org/10.1287/moor.24.4.887.

[10] Hoogeveen, J. A. and S. L. Van de Velde, Stronger Lagrangian bounds by use of slack variables: applications to machine scheduling problems, Mathematical Programming, 70, 173-190, 1995, http://dx.doi.org/10.1007/BF01585935.

[11] Hoogeveen, J. A. and S. L. Van de Velde, Scheduling by positional completion times: Analysis of a two-stage flow shop problem with a batching machine, Mathematical Programming, 82, 273-289, 1998, http://dx.doi.org/10.1007/BF01585876.

[12] Hoogeveen, J. A., L. van Norden and S. L. Van de Velde, Lower bounds for minimizing total completion time in a two-machine flow shop, Journal of Scheduling, 9: 559-568, 2006, http://dx.doi.org/10.1007/s10951-006-8789-x.

[13] Ignall, E., and L. E. Scharge, Application of the branch and bound technique to some flow-shop problems, Operations Research, 13, 400-412, 1965, http://dx.doi.org/10.1287/opre.13.3.400.

[14] Kohler, W. H. and K. Steiglitz, Exact, approximate and guaranteed accuracy algorithms for the flowshop problem n/2/F/F, Journal ACM, 22, 106 -114, 1975, http://dx.doi.org/10.1145/321864.321872.

[15] W. Kubiak, J.Blazewicz, P. Formanowicz, J. Breit and G. Schmidt, Two-machine flow shops with limited machine availability, European Journal of Operational Research Volume 136, Issue 3, pp. 528-540, 2002, http://dx.doi.org/10.1016/S0377-2217(01)00083-2.

[16] C.-Y. Lee, Machine scheduling with an availability constraints, Journal of Global Optimization, 9, pp. 363-382, 1996, http://dx.doi.org/10.1007/BF00121681.

[17] C.-Y. Lee, Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint, Operations Research Letters, 20, pp. 129-139, 1997, http://dx.doi.org/10.1016/S0167-6377(96)00041-7

[18] C. Y. Lee, L. Lei and M. Pinedo, Current trends in deterministic scheduling, Annals of Operations Research,70(0) : 1-41, 1997.

[19] C. Y. Lee, Two-machine flowshop scheduling with availability constraints, European Journal of Operational Research, 114, pp. 420-429, 1999, http://dx.doi.org/10.1016/S0377-2217(97)00452-9

[20] Ma, Y., C. Chu and C. Zuo, A survey of scheduling with deterministic machine availability constraints, *Computers & Industrial Engineering*, 58(2), pp. 199-211, 2010, http://dx.doi.org/10.1016/j.cie.2009.04.014

[21] E. Sanlaville and G. Schmidt, Machine scheduling with availability constraints, Acta Informatica, 35, pp. 795-811, 1998.

[22] G. Schmidt. Scheduling with limited machine availability, *European Journal of Operational Research*, 121(1), pp. 1-15, 2000, http://dx.doi.org/10.1016/S0377-2217(98)00367-1

[23] Van de Velde, S. L, Minimizing the sum of the job completion times in the two-machine flow shop by Lagrangian relaxation, Annals of Operations Research, 26, 257 - 268, 1990.