JOB SCHEDULING WITH UNFIXED AVAILABILITY CONSTRAINTS

Hoong Chuin LAU
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
Phone: +65-68744589
Email: lauhc@comp.nus.edu.sg

Chao ZHANG
BuildFolio Technologies PTE LTD
83 Science Park Dr #03-04
Singapore 118258
Phone: +65-67796366
Email:czhang@buildfolio.com

ABSTRACT

Standard scheduling theory assumes that all machines are continuously available throughout the planning horizon. In many manufacturing and service management situations however, machines need to be maintained periodically to prevent malfunctions. During the maintenance period, a machine is not available for processing jobs. Hence, a more realistic scheduling model should take into account machine maintenance activities. In this paper, we study the problem of job scheduling with ***unfixed*** *availability constraints on a single machine. We first propose a preliminary classification for the scheduling problem with unfixed availability constraints based on maintenance constraints, job characteristics and objective function. We divide our analysis into six cases. For each case, we present complexity analysis and exact or approximation algorithms. We also present results of tabu search on NP-hard problem instances.*

Keywords: *Maintenance, Availability, Scheduling.*

1. INTRODUCTION

Standard machine scheduling literature assumes that machines are available at all times. However, in many manufacturing or service management situations, machines are maintained periodically to prevent malfunctions. During the maintenance period, a machine is not available for processing jobs. A more realistic scheduling model should take into account machine maintenance activities.

The schedule of maintenance activities can be determined either before the scheduling of jobs, or jointly with the scheduling of jobs. In the first case, the maintenance periods are known and fixed at the time when jobs are to be scheduled. The problem of scheduling jobs with this type of maintenance reduces to the problem often referred in the literature as *scheduling with machine availability constraints* because during maintenance periods the machine is not available for processing jobs. Lee and Liman [7] study a single machine problem with a machine availability constraint. Lee [6] studies a parallel-machine problem where machines are available starting from different times. Lee and Liman [8] consider a two-parallel-machine problem where one machine has an availability constraint. Mosheiov [12] studies the same problem by assuming that each machine is available during an interval. Note that all of these results assume that the machine unavailability is known and fixed in advance. Schmidt [15, 16] studies a parallel machine problem where each machine has different availability intervals. In [9], dynamic programming procedures are given for two-machine flow shops with one non-availability period and non-preemptive operations. Espinouse & al. [4] have also considered non-preemptive tasks for a no-wait flow shop problem and proposed heuristics with performance guarantee. In [10], Lee handles the preemptive flow shop problem with two machines and one unavailability period first imposed on machine 1 and then on machine 2 with the makespan objective. Lee proved that both problems are NP-hard in the ordinary sense, and proposed heuristics with error bounding analysis. [11] extends the complexity results found in [10] for a generalized model where an operation is completely (nonresumable) or partially (resumable) restarted if its execution is preempted by a maintenance task. He also developed heuristics with an error bound analysis. Blazewicz & al [1] investigated the 2-machine problem with arbitrary number of unavailability periods on one machine, and proved that the problem is strongly NP-hard. Also, they proposed a branch and bound algorithm for the problem with arbitrary number of unavailability periods on both machines. Cheng and Wang [2] investigated the problem studied in [10], in which an availability constraint is imposed only on the first machine. They proposed a heuristic that improves the worst-case error bound provided by Lee [10]. Lee [9] considers scheduling problems with availability constraints under different machine configurations and performance measures. According to the case, he has provided a polynomial algorithm to solve the problem, or proved that the problem was NP-hard and developed pseudo-polynomial algorithms or heuristics with an error bound analysis. Besides, Lee introduces a way of giving the error bound for non-resumable case. Instead of dealing with non-resume case directly, he considers resumable case first, and uses it to obtain an error bound for non-resumable case.

In our experience with industry applications, especially in semiconductor manufacturing, it is often observed that a machine is idle while waiting for maintenance personnel to perform preventive maintenance, even though jobs are waiting and the machine has not broken down. This is due to lack of coordination between production planning and maintenance operations. Obviously, a careful coordination between maintenance and job processing would result in a better overall schedule. Hence the scheduling problem to be studied in this paper is to determine simultaneously when to perform each maintenance activity and when to process each job so that a certain performance measure is optimized. Little research has been done in the literature for the model of jointly scheduling maintenance activities and jobs. Qi, Chen, and Tu [14] conducted a study on scheduling the maintenance on a single machine. In their model, there is a constraint on the maximum allowed continuously working time *T* of the machine. Graves and Lee [5] considered a special case where the sum of processing time is less than *2T* of this model. These studies addressed the problem of maintenance, but only in a limited way. They considered constraints only on the maximum length of an available interval, and the jobs scheduled are without release times and deadlines constraints, which is inadequate for solving real problems. One can imagine for instance that it is unreasonable to schedule a maintenance immediately after another one. Hence, we need to consider constraints on the minimum length of an available interval.

In this paper, we will study the single machine problem where there are constraints either on the length of available intervals or the number of jobs completed before starting a new maintenance, and we need to schedule both jobs and maintenance activities. We term this the *Job Scheduling with Unfixed Availability Constraints problem* (JSUA). We will provide a classification designed to emulate the classical Job Scheduling Problem, where the three key parameters are maintenance properties, job characteristics and objective function.

2. PRELIMINARIES

Following scheduling terminology, each problem is defined on three fields **α/β/μ,** where **α** denotes the maintenance constraint, **β** denotes the job characteristic, and **u** represents the performance measure to be optimized. Each field has the following possible cases:

For **α**,

Case 1: Constraints on the length between two adjacent maintenances. Here, we are given two positive values *A* and *B*, and the length between two adjacent maintenances must be between *A* and *B*. Notation: *A-B/ _ / _*

Case 2: Constraints on the number of jobs completed between two adjacent maintenances. Here, we are given a positive integer *C*, and we must complete *C* jobs within an available interval. Notation: *C/ _ / _*

For **β**,

Case 1: jobs do not have release time or deadline. Notation: _ / _ / _

Case 2: jobs have release time. Notation: _ / *r* / _

Case 3: jobs have deadline. Notation: _ / *d* /_

Case 4: jobs have both release time and deadline. Notation: _ /*r- d* /_

For **μ**,

Case 1: length of the schedule. Notation: _/_ / *MakeSpan*

Case 2: number of jobs that can be completed in time in the available intervals. Notation: _/_ / *Fulfilment*

Case 3: total tardiness of all the jobs. Notation: _/_/*Total-tardiness*

The notations used in this paper are:

s_i	: starting time of *i*th maintenance
t_i	: end time of *i*th maintenance
u_i	: job *i, i=1, …,n*
p_i	: processing time of u_i
r_i	: release time of u_i
d_i	: deadline of u_i
c_i	: completion time of u_i
L	: length of schedule i.e. makespan
L^*	: optimal length of schedule
f	: number of jobs completed in time (i.e. fulfilled)
f^*	:optimal number of jobs completed in time
Q	: maintenance duration

In this paper, we assume that the set of jobs are independent with deterministic processing times to be scheduled on a single machine. All jobs are available at the beginning. Unless we mention that job is preemptive, job preemption is not permitted. No two jobs can be scheduled at the same time on the single machine. Note that machine idle times are allowed, hence the assumptions are exactly the ones defined in Baker (Baker (1974, p.10) - assumptions C1, C2, C3, C5).

3. THEORETICAL ANALYSIS

In this section, we present theoretical results for the 6 JSUA problem cases.

3.1 *A-B/_ /MakeSpan*

Input: *n* jobs with processing time $p_i \left(1 \le i \le n\right)$, maintenance duration *Q*,, two positive real value *A, B.*

Output: A schedule of minimum length such that the length of available intervals between adjacent maintenance is between *A* and *B.*

Complexity Analysis

The Bin Packing Problem is defined as follows: Given a finite set *U* of *N'* items; for each item *u* in *U* a positive integer size *s(u)*; positive integer *C* (called the *bin capacity*), partition *U* into disjoint sets such that for each set, the total sum of the sizes of the items in the set does not exceed *C* and the number of sets is minimized. Since Bin-Packing problem is a well-known NP-hard problem and a special case of *A-B/_ /MakeSpan* where *A=B*, *A-B/_ /MakeSpan* is also NP-hard problem.

Approximation Ratio

Heuristic 1 (Best-Fit), defined as follows:
Step 1: Set the length of available interval as *B*.
Step 2: Sort all the jobs in decreasing order of processing time. For each job in the job list, place it into the available interval that will leave the least amount of room left over after the item is placed in the bin. If it will not fit into current any available interval, create a new bin for it.
Step 3: After all the jobs have been assigned, scan the available intervals. If there is space left in an available interval, we shift the schedule, while ensuring that the length of any available interval is at least *A*.

Observe that Heuristic 1 takes O(*n* log *n*) time in the worst case.

Theorem 1: *A-B/_ /MakeSpan* can be approximated within a factor of 2 in O(*n* log *n*) time.

Proof: First we obtain a schedule λ_1 using the Best-Fit heuristic. Then in this schedule, if there is any available interval, the length of which is smaller than *B*, shift the latter jobs to the left, as though all the jobs are preemptive. After this step, we get a new schedule λ_2 in which all the lengths of the available intervals are *B* (no space wasted), and the length *L* of this schedule λ_2 is smaller than or equal to L^* where L^* is the optimal solution of the non-preemptive case. On the other hand, we can obtain a schedule λ_1 from λ_2**.** But in this case, instead of shifting the job to the right, we can get this job out and set a new available interval for it at the end of the schedule. We denote the schedule as λ_3**.** Clearly, λ_3 is no better than $\lambda_{1.}$ The number of jobs put at the end is exactly the number of maintenances in $\lambda_{2,}$ and the length of the available interval set at the end for each of this kind of tardy jobs is $\max(p_i, A)$. Hence, the length of the part added at the end of λ_2 is less than that of $\lambda_{2.}$ Hence, λ_3 is no worse than $2 * L^*$, since λ_2 is the optimal solution of preemptive case and it is better than or equal to the optimal solution L^* of non-preemptive case. After moving those jobs, when we get λ_3**,** we can also shift the schedule to the left, because of the space around the maintenances where the jobs are slipped. But this schedule is still no better than $\lambda_{1.}$ Hence, we can see that, the solution of the Best-Fit heuristic is much better than $2 * L^*$. (QED)

It turns out that the above result can be further tightened if we consider the following two special cases:

Case *A*=*B:* Since this problem is exactly the Bin Packing problem which can be approximated within 1.5 [17], it can also be approximated likewise.
Case |*B*-*A*| $\geq p_{\max}$: If we use Best-Fit to solve *A-B/_ /MakeSpan*, we see that for each available interval between any two adjacent maintenances, the length is between value *A* and value *B*, since we can always assign one more job or continue processing the current job after the point of *A*. Thus there is no empty space before a new maintenance started. And for the schedule achieved from the Best-Fit heuristic the total sum of the lengths of available intervals is always fixed because there is no wasted space.

Corollary 1: When |*B*-*A*| $\geq p_{\max}$, *A-B/_ /MakeSpan* can be approximated within $1 + 1/_{1+k}$, where *k* is the ratio of $\sum_{i=0}^{n} p_i$ and the sum of maintenance time in the optimal solution.

Proof: Again, we use the Best-Fit heuristic to solve *A-B/_ /MakeSpan*. But for $\lambda_{1,}$ the sum of the lengths of all the available intervals is exactly $\sum_{i=0}^{n} p_i$, since there is no empty space in each available interval as mentioned above. From migrating from λ_2 to $\lambda_{3,}$ we can easily see that the number of maintenances is at most two times of that of the optimal solution. Then we know the solution *L* achieved from Best-Fit, where

$L \leq \sum_{i=0}^{n} p_i + 2/_k \cdot \sum_{i=0}^{n} p_i$. Since $L^* \geq \left(1 + 1/_k\right)\sum_{i=0}^{n} p_i$, we get $L/_{L^*} \leq \left(1 + 2/_k\right) \cdot \sum_{i=0}^{n} p_i \Big/ \left(1 + 1/_k\right)\sum_{i=0}^{n} p_i = 1 + 1/_{1+k}$

Note that in a typical manufacturing environment, the processing time of a machine is much larger than its maintenance time. Hence, the approximation ratio is very close to 1 (i.e. optimal).

3.2 *A-B/ r /MakeSpan*

Input: *n* jobs with processing time p_i and release time r_i, maintenance duration *Q*,, two positive values *A, B*
Output: A schedule of minimum length such that the length of available intervals must be between *A* and *B*.

Complexity Analysis

This problem is NP-hard, since *A-B/_ /MakeSpan* can be treated as a special case of *A-B/r/MakeSpan* where the release times of all the jobs are zero.

Approximation Ratio

Heuristic 2: We can solve the corresponding problem for *A-B/_ /MakeSpan* by Heuristic 1 first, and obtain the final solution by shifting jobs to the respective release time.

Although Heuristic 2 does not seem very efficient (since there is wasted slot before its release time), it is interesting to obtain the following approximation result:

Theorem 2:
A-B/r/MakeSpan can be approximated within a factor of 1+*k*, for some $k \le 2$ in O(*n* log *n*) time.

Proof: Let L_1^* denote the optimal solution for the corresponding problem *A-B/_ /MakeSpan.* Let L_1 denote the solution achieved by Heuristic 1 for *A-B/_ /MakeSpan.* Let L denote the solution achieved by Heuristic 2. Clearly, $L^* \ge \max(r_{\max,} L_1^*)$

Since $L = r_{\max} + L_1$, we get $L/L^* = (r_{\max} + L_1)/L^* \le (r_{\max} + L_1)/\max(r_{\max}, L_1^*)$

Suppose $L_1 \le k * L_1^*$ for some *k*.

Suppose $r_{\max} \le L_1^*$, then $L/L^* \le (r_{\max} + L_1)/L_1^* \le (L_1^* + L_1)/L_1^* \le 1 + k$

Suppose $r_{\max} \ge L_1^*$, then $L/L^* \le (r_{\max} + L_1)/r_{\max} \le (r_{\max} + k * L_1^*)/r_{\max} \le 1 + k$

Hence, $L/L^* \le 1 + k$. By **Theorem 1**, we know that $k \le 2$, and hence $L/L^* \le 1 + k$, for some $k \le 2$.

3.3 *A-B/ d /Fulfilment*

Input: *n* jobs with processing time p_i and deadline d_i, maintenance duration *Q*, two positive real values *A*, *B*.

Output: A schedule such that the length of available intervals must be between *A* and *B*, and the number of jobs completed in time is maximized.

Complexity Analysis
Like *A-B/_/MakeSpan*, based on the reduction from bin-packing, *A-B/ d /Fulfilment* is also NP-hard.

Approximation ratio

From Moore-Hodgson's algorithm [13], we derive a new algorithm: Algorithm 3

Step 0: Set the length of each available interval as *B*.
Step 1: Sort the jobs in EDD order, and place the jobs into the schedule as though all the jobs are preemptive. If there exists a tardy job, among the first *k* jobs find the job with largest processing time, delete it from the job list and assign it at end of the sequence. From the above, we obtain a schedule $\lambda_{1.}$
Step 2: From λ_1, delete the jobs started before s_i, and completed after t_i of the *i*th maintenance. Then shift the schedule to the left if there is gap before any maintenance, but ensure that the length of any available interval is at least *A*. We obtain a new schedule λ_2. We also use the jobs removed to form a new schedule λ_3.
Step 3: Compare schedule λ_2 and λ_3, and choose the one with more jobs in schedule.

Theorem 3:
A-B/d/Fulfillment can be approximated within $\max\left(1/2, k\text{-}1/k\right)$, where $k = \left\lfloor B/p_{\max} \right\rfloor$, in O(*n* log *n*) time.

Proof: Let *f* denote the solution achieved from algorithm 3, f^* denote the optimal solution of non-preemptive case, and f^*_r denote the optimal solution of preemptive case. From [9], we know that the solution of λ_1 is f^*_r.

Note that, $f^*{}_r \geq f^*$ since we can shift those jobs completed after t_i of the non-preemptive problem to the left to fill in the gap. Thus $f^*{}_r$ must have at least the same number of jobs as f^*. We can see that the complete time of the jobs in either schedule λ_2 or schedule λ_3 is smaller than the corresponding complete time in schedule λ_1, then all the jobs in schedule λ_2 and λ_3 can be completed in time. Since $f^*{}_r \geq f^*$ and $2 * f \geq f^*{}_r$, we can say $f/f^* \geq 1/2$. Furthermore, let $k = \left\lfloor B/\max(p_i) \right\rfloor$, we can say that for each available interval of schedule λ_1, there are at least k jobs. And after we remove the jobs slipped by maintenances, we can still have k-1 jobs in each available interval of λ_2. Then, we can get $f/f^* \geq k\text{-}1/k$. Hence, the approximation ratio of Heuristic 3 is max ($k\text{ -}1/k$, $1/2$).

3.4 *A-B/ r-d /Total-tardiness*

Input: *n* jobs with processing time p_i, release time r_i, and deadline d_i, maintenance duration *Q*,, two positive real values *A*, *B*.
Output: A schedule such that the length of available intervals must be between *A* and *B*, and the total tardiness is minimized.

Complexity Analysis
In [3], the problem of minimizing total tardiness on one machine has been proved to be NP-hard, and it can be treated as a special case of *A-B/ r-d /Total-tardiness* where the values of *A* and *B* is infinitely large and the release time of all the jobs are at the zero point. Thus, *A-B/ r-d /Total-tardiness* is also NP-hard.

3.5 *C/ d /Fulfilment*

Input: *n* jobs with processing processing time p_i and deadline d_i, maintenance duration *Q*, a positive value *C*
Output: A schedule such that we have to process *C* jobs, and the number of tardy jobs is minimized.

Heuristic 4:
Step 0: sort the jobs in EDD order.
Step 1: place the jobs into the schedule. If there is a tardy job u_k, among the first *k* jobs find the job with largest processing time, delete it from the job list and assign it at end of the sequence. When there are *C* jobs in the current available interval, start a new maintenance.
Step 2: Repeat Step 1.

Theorem 4:
C/ d /Fulfilment can be solved optimally in in O(*n* log *n*) time.

Proof: Since we index the jobs in EDD order, then when the job u_j is found tardy, we can say before the current completion time *t* of u_j there exists at least one tardy job no matter how to schedule the jobs. Suppose there is a schedule λ_1 with no tardy job before *t*, hence the deadlines of $u_1, u_2 \cdots u_j$ are smaller than *t*, then the jobs $u_1, u_2 \cdots u_j$ should be all finished before *t* in schedule λ_1**.** However, the sum of processing time of $u_1, u_2 \cdots u_j$ equals *t*., and surely the last job in the schedule is a tardy job. This is a contradiction with the assumption. So we can say there exists at least one tardy job before *t*. Hence, after the step that the job with the largest processing time is taken out, the schedule λ_2 achieved is optimal currently, and it leaves more time for the following jobs compared with the schedules that any other job is taken out. We can perform the above process for the following jobs, and after we schedule all the jobs, the schedule λ_2 achieved is optimal. In this algorithm, the time complexity for sorting in EDD order is O(*n* log *n*), the scheduling algorithm itself is linear. Hence, the time complexity is O(*n* log *n*).

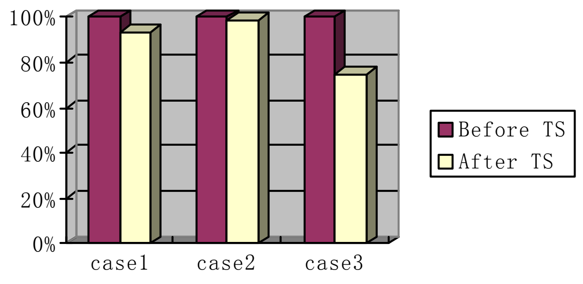
3.6 *C/ r/Makespan*

Input: *n* jobs with processing time p_i and release time r_i, maintenance duration *Q*,, a positive value *C*.
Output: A schedule such that *C* jobs have been processed, and the length of schedule is minimized.

Analysis
The number of available intervals is $\left\lceil n/c \right\rceil$, implying that the number of maintenances is $\left\lceil n/c \right\rceil$-1. Whenever a job is released, we insert it into a queue. When the machine is available, fetch a job from the queue. If the queue is empty, we have to wait. Hence in each available interval, no time is wasted. The total length of available intervals is always optimal. And the number of maintenances is always the same. Hence the sum of available intervals and maintenance is optimal.

4. TABU SEARCH

In this section, we present results of a tabu search scheme for solving *A-B/_/MakeSpan*, *A-B/r/MakeSpan* and *A-B/r-d total*-tardiness. The basic ideas for all three problems are similar, except their initial solution and objective function. Since there is no benchmark available for the above problems, we compare the results with the standard greedy approach. We illustrate our results with problem instances that are generalized from the OR Library Bin-Packing problem instances by randomly generating the range (from *A* to *B*) based on the bin sizes given by Bin-Packing problem instances. By applying our tabu search scheme**,** we improve the results achieved by the greedy algorithms discussed above. The following chart summarizes the results obtained:



case1: *A-B/_/MakeSpan*
case2: *A-B/r /MakeSpan*
case3: *A-B/r-d/Total-tardiness*

From the above results, we observe that the greedy results of *A-B/r-d/Total-tardines* can be improved by 20% using tabu search. For *A-B/_/MakeSpan*, the results can be improved by around 10%. However, *A-B/r/MakeSpan* is not improved significantly by using tabu search. In fact, for each instance of *A-B/r/MakeSpan* and the corresponding instance of *A-B/_/MakeSpan*, the difference is that *A-B/r/MakeSpan* has one more constraint: the release time. This constraint apparently restricts the potential of tabu search of making local improvement significantly.

5. CONCLUSION

In this paper, we proposed a taxonomy that categorizes the JSUA problem according to 3 driving factors we believe to be important. As this is the first such taxonomy, we anticipate much future research work to refine the results presented here.

References

[1] Blazewicz J., Formanowicz P., Kubiak W., Przysuch M. and Schmidt G., "Parallel branch and bound algorithms for the two-machine flow shop problem with limited machine availability", *Bulletin of the Polish Academy of Science*s, (2000).
[2] Cheng T. C. E. and Wang G., "An improved heuristic for the two-machine flowshop scheduling with an availability constraint", *Oper Res Lett*, 26,223-229, (2000).
[3] Du, J. and Leung, J. Y. T., Minimizing Total Tardinesson One Machine is NP-hard. *Math. Oper. Res,* 15, pp. 483-495 (1990).
[4] Espinouse M.-L., Formanowicz P. and Penz B., "Minimizing the makespan in the two-machine no-wait flow-shop", *Computers & Industrial Engineerin*g, 37, 497-500, (1999).
[5] Graves G. H. and Lee C.-Y., “Scheduling maintenance and semi-resumable jobs on a single machine”, *Naval Research Logistics*, 46, pp. 845-863, (1999).
[6] Lee C.-Y., “Parallel machines scheduling with non-simultaneous machine available time”, *Discrete Appl Math,* 30, 53-61, (1991).
[7] Lee C.-Y. and Liman S.D., “Single machine flow-time scheduling with scheduled maintenance”, *Acta Inf,* 29, 375-382, (1992).
[8] Lee C.-Y. and Liman S.D., “Capacitated two-parallel machines scheduling to minimize sum of job completion times”, *Disc Appl Math* 41, 211-222, (1993).
[9] Lee C.-Y., "Machine scheduling with an availability constrain*t", J. Global Optimizatio*n, 9, 395-416, (1995).
[10] Lee C.-Y., “Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint”, *Oper Res Lett* 20, 129-139, (1997).
[11] Lee C.-Y., "Two-machine flowshop scheduling with availability constraints", *European J. Operational Researc*h, 114, 420-429, (1999).
[12] Mosheiov G., “Minimizing the sum of job completion times on capacitated parallel machines”, *Math Comput Model,* 20, 91-99, (1994).
[13] Moore J. M., “An n job, one machine sequencing algorithm for minimizing the number of late jobs”, *Management Science*, 15:102-109, (1968).
[14] Qi X., Chen T.,and Tu F., “Scheduling the maintenance on a single machine”, *J Oper Res Soc*, 50, 1071-1078. (1999).
[15] Schmidt G., “Scheduling on semi-identical processors”, *Z Oper Res* 28, 153-162, (1984).
[16] Schmidt G., “Scheduling independent tasks with deadlines on semi-identical processors”, *J Oper Res Soc*, 39, 271-277, (1988).
[17] Simchi-Levi D., “New worst-case results for the bin-packing problem”, *Naval Res. Logistics*, 41, 579-585 (1994).