MACHINE SCHEDULING WITH AN AVAILABILITY CONSTRAINT

Chung-Yee Lee
Department of Industrial and Systems Engineering
University of Florida
Gainesville, FL 32611

February 1995
Revised: July 1995

This research was supported in part by NSF grant DDM 9201627

ABSTRACT

Most literature in scheduling assumes that machines are available simultaneously at all times. However, this availability may not be true in real industry settings. In this paper, we assume that the machine may not always be available. This happens often in the industry due to a machine breakdown (stochastic) or preventive maintenance (deterministic) during the scheduling period. We study the scheduling problem under this general situation and for the deterministic case.

We discuss various performance measures and various machine environments. In each case, we either provide a polynomial optimal algorithm to solve the problem, or prove that the problem is NP-hard. In the latter case, we develop pseudo-polynomial dynamic programming models to solve the problem optimally and/or provide heuristics with an error bound analysis.

1. INTRODUCTION

Due to the popularity of Just-In-Time philosophy and Total Quality Management concept, on-time delivery has become one of the crucial factors for customer satisfaction. Scheduling plays an important role in achieving on-time delivery. In the last four decades, many papers have been published in the scheduling area, (see for example, survey papers by Graves(1981), Lawler, et al.(1993) and Herrmann, Lee and Snowdon(1993)). Recently this area has become even more popular. This popularity can be seen from the fertile publication of related books such as Baker (1993), Blaczwicz et al. (1993) Morton and Pentico (1993), Tanaev, Sotskov and Strusevich (1994) and Pinedo (1995).

Journal of Global Optimization 9: 395–416, 1996.

Most literature in scheduling assumes that machines are available simultaneously at all times. However, this availability may not be true in real industry settings. In this paper, we assume that the machine may not always be available. This happens often in the industry due to a machine breakdown (stochastic) or preventive maintenance (deterministic) during the scheduling period. We study the scheduling problem under this general situation and for *the deterministic case*.

We assume that machine j is unavailable during the period from s_j to t_j ($0 \le s_j \le t_j$). Namely, we assume that there is only one unavailability period during the scheduling horizon for machine j. Since we are studying the deterministic model, it is reasonable to assume that there is only one preventive maintenance during the scheduling horizon. Furthermore, we assume that for the multiple machines problem, one machine is always available. Note that the special case with $s_j = 0$ means that machine j is not available until t_j. This happens for instance in the case where the machine continues to process those unfinished jobs that were scheduled in the previous planning period.

Although this problem is important as it happens often in industry, there are only a few articles in the scheduling area that deal with this problem. Schmidt(1984) studies an n job m parallel machine scheduling problem where each job has a deadline and each machine has different availability intervals The purpose is to construct a feasible pre-emptive schedule. He presents an-O(nmlogn) time algorithm to find a feasible pre-emptive schedule whenever one exists. Adiri et al. (1989) consider the single machine scheduling problem with the objective of minimizing the total completion time where the machine is not available during some intervals. They studied both stochastic case where the location and duration of the unavailability periods are random, and the deterministic case where the machine unavailability is known in advance. For the deterministic case they show that the problem is NP-hard if there is only one unavailability during the scheduling period and if a job did not finish before the machine unavailability then the job needs to be restarted again. Lee (1991) studies the parallel identical machines scheduling problem of minimizing the makespan where machines may not be available at time zero; i.e. $s_j = 0$ and $t_j \ge 0$ for all j. He shows that the classical Longest Processing Time (LPT) algorithm will have a tight error bound 1/2,. He then provides a modified LPT algorithm with error bound equal to 1/3. Kaspi and Montreuil (1988) and Liman (1991) show that the Shortest Processing Time (SPT) algorithm is an optimal policy for multiple machines total completion time problem with non-simultaneous machine available times($s_j = 0$ for all j). Lee and Liman (1992) study the deterministic model of the problem of Adiri et al. (1989). They provide a simpler proof of NP-hardness and show a tight error bound for the SPT heuristic. Lee and Liman (1993) study the two-parallel-machine scheduling problem of minimizing the total completion time where one machine is not available from a particular instant, i.e. $s_1 = t_1 = 0$ and $s_2 > 0$, $t_2 = \infty$. They prove that the problem is NP-hard and use dynamic programming to solve it. Mosheiov(1994) studies

the same problem under the condition that machine j is available in the interval $[x_j, y_j]$ where $0 \leq x_j < y_j$. He shows that for the m-machine problem SPT is asymptotically optimal as the number of jobs increases.

A different but somehow related problem is the scheduling problem with time windows. In such problems, each job should be processed within certain window(see for example, Lei and Wong 1990 and Kraemer and Lee 1993). Namely, the availability constraint is imposed on jobs rather than machines.

In this paper, we study the problem for different performance measures (makespan, total weighted completion time, tardiness, and number of tardy jobs) and different machine situations (single machine and parallel machines). In each case, we either provide a polynomial optimal algorithm to solve the problem, or prove that the problem is NP-hard. In the later case, we develop pseudo-polynomial dynamic programming models to solve the problem optimally and/or provide heuristics with an error bound analysis. Please see Table 1 for a summary. This work is part of on-going research on the machine availability constraint problem. In the companion paper, Lee (1995) studies the two-machine flowshop scheduling problem with an availability constraint on one machine and the objective of minimizing makespan. He proves that the problem is NP-hard and proposes pseudo-polynomial dynamic programming to solve the problem optimally. He also provides two O(nlogn) heuristics with an error bound analysis. The first (second) heuristic is used to solve the problem with the availability constraint imposed on machine 1 (2), and has a worst case error bound of 1/2 (1/3 respectively).

Two cases, resumable and nonresumable, are discussed in this paper. We call a job *resumable* if it cannot finish before the unavailable period of a machine and can continue after the machine is available again. On the other hand, we call it *nonresumable* if it has to restart, rather than continue. In the next section, we first define the notation. We then deal with the resumable and the nonresumable cases in Section 3 and 4 respectively. Section 5 briefly describes future research topics.

Table 1
Summary of Complexity Classification

Problem	Complexity	Solution	Reference
$1/r\text{-}a/\Sigma C_i$	P	SPT	3.1.1.
$1/r\text{-}a/C_{max}$	P	Arbitrary Sequence	3.1.1.
$1/r\text{-}a/L_{max}$	P	EDD	3.1.1.
$1/r\text{-}a/\Sigma U_i$	P	Moore-Hodgson Algorithm	3.1.1.
$1/r\text{-}a/\Sigma w_i C_i$	NP even if $w_i=p_i$	Dynamic Programming, Heuristics and Error Bounds	3.1.2.
$Pm/r\text{-}a/C_{max}$	NP	Heuristics and Error Bounds	3.2.1.
$P2/r\text{-}a/\Sigma w_i C_i$	NP	Dynamic Programming	3.2.2.
$F2/r\text{-}a(M_1)/C_{max}$	NP	Dynamic Programming, Heuristics and Error Bounds	Lee (1995)
$F2/r\text{-}a(M_1)/C_{max}$	NP	Dynamic Programming, Heuristics and Error Bounds	Lee (1995)
$1/nr\text{-}a/C_{max}$	NP	Heuristics and Error Bounds	4.1.1.
$1/nr\text{-}a/L_{max}$	NP	Heuristics and Error Bounds	4.1.2.
$1/nr\text{-}a/\Sigma U_i$	NP	Heuristics and Error Bounds	4.1.2
$1/nr\text{-}a/\Sigma w_i C_i$	NP	Heuristics and Error Bounds	Adiri.et.al(1989) Lee and Liman(1991) 4.1.2.
$Pm/nr\text{-}a/C_{max}$	NP	Heuristics and Error Bounds	4.2.1.
$P2/nr\text{-}a/\Sigma w_i C_i$	NP	Dynamic Programming	4.2.2.
$F2/nr\text{-}a(M_1)/C_{max}$	NP	Dynamic Programming Heuristic and Error Bound	Lee (1995)
$F2/nr\text{-}a(M_2)/C_{max}$	NP	Heuristic and Error Bound	Lee (1995)

2. NOTATION

We are given m machines and n jobs with the following notation.

J_i : Job i, i = 1,...,n.

p_i : Processing time for J_i.

MS = $\sum_{i=1}^{n} p_i$, total processing time of n jobs,

A_i = $\sum_{k=1}^{i} p_k$, the sum of processing time of jobs in $\{J_1, J_2,...,J_i\}$,

d_i : Due date for J_i.

C_i : The completion time for J_i.

w_i : Weight for J_i. Hence $\Sigma w_i C_i$ is the total weighted completion time.

L_i : Lateness for J_i, $L_i = C_i - d_i$.

C_{max} : Makespan = $Max\{C_i, i = 1,...,n.\}$

L_{max} = $Max\{L_i, i = 1,...,n.\}$

U_i = 1, if $L_i > 0$, and 0 otherwise. Hence ΣU_i denotes total number of tardy jobs.

M_j : Machine j, j = 1,...,m.

s_j, t_j : M_j is unavailable from s_j to t_j for all j, where $0 \le s_j \le t_j$

$J_{[k]}$: The k-th job in a given sequence.

Whenever C_{max} is the performance measure, we use C^* to denote the optimal makespan and use C_H to denote the makespan of the problem if we apply heuristic H to it. To be concise, we follow the notation of Pinedo (1995), extended to include resource constra nts. This notation consists of three fields $\alpha/\beta/\gamma$, where α denotes the machine condition, β denotes problem characteristics and γrepresents the performance measure to be optimized. In particular, α = 1 and P denote single machine and parallel machines respectively. The second field can represent dynamic arrivals, special precedence constraints or special availability constraints, and the third field, γ, can be C_{max}, L_{max}, $\Sigma w_i C_i$ or ΣU_i. In this paper, we will use *r-a* in the second field to denote a resumable availability constraint, where M_j is not available from s_j to t_j for all *j* and a job is resumable if it cannot be finished before s_j. Similarly, β = *nr-a* denotes a nonresumable availability constraint. For example, $1/r\text{-}a/\Sigma w_i C_i$ denotes the problem of minimizing total weighted single machine with a resumable availability constraint.

3. THE RESUMABLE AVAILABILITY CONSTRAINT

In this section, we study the complexity, and provide algorithms to minimize different objective functions for different problems with a resumable availability constraint. In particular, we discuss the single machine and parallel machine problems.

3.1 Single Machine Problems

3.1.1. 1/r-a/C_{max}, 1/r-a/ΣC_i, 1/r-a/L_{max} and 1/r-a/ΣU_i

It can be checked by a simple job exchange scheme that for single machine problems, 1/r-a/C_{max}, 1/r-a/ΣC_i, and 1/r-a/L_{max} can be solved optimally by any sequence, the Shortest Processing Time (SPT) algorithm(*sequencing jobs in the nondecreasing order of* p_i), and the Earliest Due Date (EDD) rule(*sequencing jobs in the nondecreasing order of* d_i), respectively.

The 1//ΣU_i problem can be solved by Moore-Hodgson's algorithm (Baker 1993); restated as follows for convenience.

Moore-Hodgson's algorithm for 1//ΣU_i

Step 0: Set N = $\{J_1, J_2...J_n\}$.

Step 1: Sequence the jobs of N in EDD order. If there is no tardy job in N then stop. Otherwise, find the first tardy job, say $J_{[k]}$. Among the first k jobs find the job with largest processing time, delete it from N and assign it at the end of the sequence. Continue until no tardy jobs exist in set N.

For our 1/r-a/ΣU_i problem, we can apply Moore-Hodgson's algorithm to solve it optimally by noting that for those jobs finished after s_1, we need to add the unavailable time period (t_1- s_1) to their completion times.

3.1.2. 1/r-a/$\Sigma w_i C_i$

It is well known that 1//$\Sigma(w_i C_i)$ can be solved optimally by the WSPT(Weighted Shortest Processing Time) algorithm (*sequencing jobs in the nondecreasing order of* p_i/w_i)(see for example, Pinedo 1995). However, it is interesting to note that addition of the availability constraint will make this problem NP-hard. We will prove this by transforming the Partition Problem, a well-known NP-hard problem (Garey and Johnson 1979), into our problem.

Since we will use the Partition Problem for the proof of NP-hardness of several problems studied in this paper, we first state the Partition Problem.

Partition Problem

Instance: A finite set of positive integers $a_1, a_2, ..., a_n$, which total to 2A.

Question: Can this set be partitioned into two disjoint subsets such that the sum of each subset is equal to A?

For any schedule σ, let S_2 be the set of jobs that finished after t_1 and let $J_{[i]}$ be the i-th job in the schedule. Before we prove the NP-hardness of the problem, we state the following property that will be used later.

Property: For any schedule σ, the corresponding total weighted completion time is given by

$$F_w(\sigma) = \sum_{i=1}^{n} w_{[i]}p_{[i]} + \sum_{i>j} w_{[i]}p_{[j]} + \left(\sum_{k \in S_2} w_k\right)(t_1 - s_1). \quad (1)$$

Proof: Suppose that the first i jobs finished no later than t_1 and the remaining jobs finished after t_1. Namely, $J_{[i+1]}$ is the first job in S_2. Then we have

$$\begin{aligned} F_w(\sigma) &= w_{[1]}p_{[1]} + w_{[2]}(p_{[1]} + p_{[2]}) + \cdots + w_{[i]}(p_{[1]} + p_{[2]} + \cdots + p_{[i]}) \\ &\quad + w_{[i+1]}(p_{[1]} + p_{[2]} + \cdots + p_{[i+1]} + t_1 - s_1) + \cdots + w_{[n]}(p_{[1]} + p_{[2]} + \cdots + p_{[n]} + t_1 - s_1) \\ &= \sum_{i=1}^{n} w_{[i]}p_{[i]} + \sum_{i>j} w_{[i]}p_{[j]} + \left(\sum_{k=i+1}^{n} w_{[k]}\right)(t_1 - s_1) \\ &= \sum_{i=1}^{n} w_{[i]}p_{[i]} + \sum_{i>j} w_{[i]}p_{[j]} + \left(\sum_{k \in S_2} w_k\right)(t_1 - s_1). \end{aligned}$$

QED.

The following theorem shows that even the special case is NP-hard, hence our problem is NP-hard.

Theorem 1: $1/r\text{-}a/\Sigma w_i C_i$ is NP-hard even if $w_i = p_i$ for all i.

Proof: We will prove the theorem by transforming the Partition Problem into our problem. Given the instance of the Partition Problem, generate an instance with n jobs for our problem.

$p_i = a_i$, i =1,..., n.
$w_i = a_i$, i = 1,...,n.
s_1= A and t_1 = 2A.

Does there exist a schedule with total weighted completion time equal to $B = \sum_{i=1}^{n} a_i^2 + \sum_{i>j} (a_i a_j)$ $+ A^2$?

$\Rightarrow$ If there exists a partition S_1 and S_2, then schedule the jobs corresponding to S_1 and then S_2. Since $a_i = p_i$ for all i, any sequence is a WSPT sequence. Since $\sum_{k \in S_1} p_k = \sum_{k \in S_1} a_k = A$, the last job

in S_1 finishes at $s_1 = A$, and the total weighted completion time, by Equation (1), is equal to $\sum_{i=1}^{n} a_i^2 + \sum_{i>j} a_i a_j + \left(\sum_{k \in S_2} a_k\right)(t_1 - s_1) = \sum_{i=1}^{n} a_i^2 + \sum_{i>j} a_i a_j + AA = B$.

$\Leftarrow$ Suppose that there exists a schedule with total weighted completion time $\sum_{i=1}^{n} \Sigma a_i^2 + \sum_{i>j} a_i a_j + A^2$. Consider any schedule, let S_1 be the set of jobs finished no later than t_1 and let S_2 be the set of the remaining jobs. Again, by Equation (1), the total weighted completion time is equal to $\sum_{i=1}^{n} a_i^2 + \sum_{i>j} a_i a_j + \left(\sum_{k \in S_2} a_k\right)(t_1 - s_1)$ which also equals $\sum_{i=1}^{n} a_i^2 + \sum_{i>j} (a_i a_j) + \left(\sum_{k \in S_2} a_k\right)A$. By the hypothesis, there exists one schedule with total weighted completion time equal to $B = \sum_{i=1}^{n} a_i^2 + \sum_{i>j} (a_i a_j) + AA$, and hence for such a schedule we must have $\sum_{k \in S_2} a_k = A$. Thus we have a partition. QED.

Since the problem is NP-hard, it is justifiable to try a heuristic method. The most intuitive heuristic is the WSPT algorithm; hence, we will analyze its worst case error bound. Let F_w(WSPT) and F_w(WSPT, $t_1 = s_1$) denote the total weighted completion time by using the WSPT algorithm to this problem *with and without availability constraint respectively*, and let F_w^* denote the optimal total weighted completion time.

Lemma 1: If there exists a job J_j that started before s_1 and finished after t_1 in the WSPT algorithm then F_w(WSPT) - $F_w^* \le w_j(t_1 - s_1)$.

Proof: First note that the first two terms of Equation (1) are the total weighted completion time of the problem without the availability constraint. Hence these two terms can be minimized by the WSPT algorithm. Namely, F_w(WSPT, $t_1 = s_1$) is the minimum possible value of the first two terms of Equation (1). Note also that in the WSPT algorithm, we sequence the jobs in nondecreasing order of p_i/w_i, or equivalently nonincreasing order of w_i/p_i. We can consider w_i/p_i as the weight per unit processing time. Namely, the highest weighted job is assigned first in WSPT. Hence the total weight of jobs finished after t_1 in "any" schedule can not be less than $\left[\sum_{k \in S_2(\text{WSPT})} w_k - w_j\right]$, where S_2(WSPT) is the corresponding set S_2 of WSPT. Therefore, we have

$$F_w^* \geq F_w(\text{WSPT}, t_1 = s_1) + \left[\sum_{k \in S_2(\text{WSPT})} w_k - w_j \right](t_1 - s_1)$$

$$= F_w(\text{WSPT}) - w_j(t_1 - s_1).$$

Namely $F_w(\text{WSPT}) - F_w^* \leq w_j(t_1 - s_1)$. QED.

Note that if in the WSPT schedule there exists no job that started before s_1 and finished after t_1, namely, there exists one job J_i with $C_i = s_1$, then $\left[\sum_{k \in S_2(\text{WSPT})} w_k \right]$ is the smallest possible total weight we can get in S_2, and hence the WSPT schedule is optimal.

Corollary 1: If we apply the WSPT algorithm to the $1/r\text{-}a/\Sigma w_i C_i$ problem and have one job J_i with $C_i = s_1$, then the WSPT schedule is optimal.

Lemma 2: $F_w(\text{WSPT})/F_w^*$ can be arbitrarily large even if $w_i = p_i$ for all i.

Proof: Let $p_1 = w_1 = 1$, $p_2 = w_2 = n$, $s_1 = n$, $t_1 = n + n^2$. Using the WSPT algorithm we may have J_1 followed by J_2 with total weighted completion time, $F_w(\text{WSPT}) = 1 \cdot 1 + (n^2 + n + 1) \cdot n = n^3 + n^2 + n + 1$. However, in the optimal solution there should be J_2 followed by J_1 with total weighted completion time, $F_w^* = n \cdot n + (n^2 + n + 1) \cdot 1 = 2n^2 + n + 1$. It can be seen that $F_w(\text{WSPT})/F_w^*$ can be arbitrarily large as n approaches infinity. QED.

It is interesting to observe, by (1), that in order to improve the error bound, we may want to "squeeze" as much weight as possible into $(0,s_1)$, which in turn is similar to the Knapsack Problem. Hence we may adopt those simple greedy heuristics used in the Knapsack Problem (see for example, Sahni 1975). The following is the combine-algorithm for the greedy heuristics.

Combine-Algorithm:

(i) Use the WSPT algorithm, and let the corresponding objective value be $F_w(\text{WSPT})$.

(ii) Reindex the jobs in the WSPT order. Then assign jobs, as many as possible, to be processed in $(0,s_1)$. Assign the remaining jobs in the WSPT order. Let the corresponding objective value be F_{w1}.

(iii) Reindex the jobs in a Largest Weight order, that is, $w_1 \geq w_2 \geq \ldots, \geq w_n$, and then assign jobs, as many as possible, to be processed in $(0,s_1)$. Reschedule those jobs in the WSPT order and also assign the remaining jobs in the WSPT order. Let the corresponding objective value be F_{w2}.

(iv) Let $F_w(\text{COMB}) = \min \{ F_w(\text{WSPT}), F_{w1}, F_{w2} \}$.

Theorem 2: $F_w(COMB)/F_w^*$ can be arbitrarily large.

Proof: Consider a problem with the following instance.

$p_1 = p_2 = \ldots = p_n = 1$, $p_{n+1} = n$, $p_{n+2} = n^2$,

$w_1 = w_2 = \ldots = w_n = n$, $w_{n+1} = n$, $w_{n+2} = n^2$,

$s_1 = n^2 + n$ and $t_1 = n^3 + n^2 + n$.

The optimal solution will sequence the jobs in $J_1,\ldots,J_n$, J_{n+2} and then J_{n+1} with total weighted completion time equal to $(n(n+1)/2)n + (n^2 + n)n^2 + (n^3 + n^2 + n+n)n = 2n^4 + (5/2)n^3 + (5/2)n^2$ (See Figure 1a). If we use Step (i) or Step (ii) of the Combine-Algorithm we may get, in both cases, the sequence $J_1,\ldots,J_n$, J_{n+1} and then J_{n+2} with a total weighted completion time equal to $(n(n+1)/2)n + (2n)n + (n^3 + n^2 + n + n)n^2 = n^5 + n^4 + (5/2)n^3 + (5/2)n^2$ (See Figure 1b). If we use Step (iii) of the Combine-Algorithm, we may get the sequence J_{n+2}, J_{n+1} and then $J_1,J_2,\ldots,J_n$, with a total weighted completion time equal to $(n^2)(n^2) + (n^2 +n)n + (n^3 + n^2 + n +1)n + (n^3 + n^2 + n +2)n + \ldots + (n^3 + n^2 + n +n)n = (n^4 + n^3 + n^2) + (n^3 + n^2 +n)n^2 + (n(n+1)/2)n = n^5 + 2n^4 + (5/2)n^3 + (3/2)n^2$ (see Figure 1c). In all three cases, we can see that $F_w(COMB)/F_w^*$ can be arbitrarily large as n approaches infinity. QED.

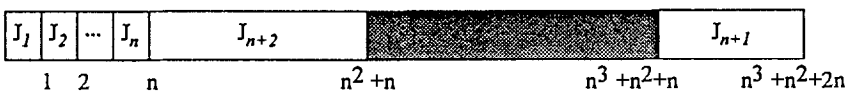

Figure 1a: Optimal Solution

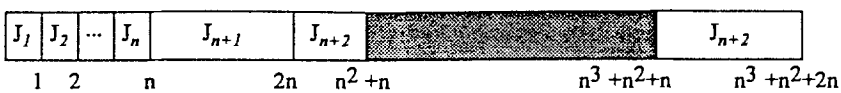

Figure 1b: Schedule of Steps (i) and (ii)

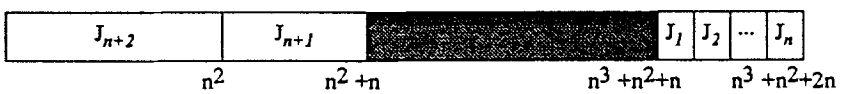

Figure 1c: Schedule of Step (iii)

Theorem 2 implies that if we want to improve the error bound, a more complicated heuristic is required. Recall that Theorem 2 shows that our problem is NP-hard even if $p_i = w_i$ for all i. The following lemma shows that if $p_i = w_i$ for all i then the error bound for using Step (iii) of the Combine-Algorithm is only 1.

Lemma 3: If $p_i = w_i$ for all i, and we use Step (iii) of the Combine-Algorithm, we will have $(F_{w2} - F_w^*)/ F_w^* \leq 1$ and the bound is tight.

Proof: Since $p_i/w_i = 1$ for all i, any job sequence is in the WSPT order. Hence Lemma 1 is true for any sequence. Use Step (iii) of the Combine-Algorithm, and let J_j be the first job that cannot fit in before s_1. The maximum error is not greater than $p_j(t_1 - s_1)$. However, since we assign the jobs in a Largest Weight order, all the jobs assigned before J_j have weights no less than w_j. Hence in the optimal solution there must exist at least one job, J_k, with $w_k \geq w_j$ and $C_k > t_1$. Namely, $F_w^* \geq w_k t_1 \geq w_j(t_1 - s_1) \geq F_{w2} - F_w^*$. To show that the bound is tight, consider a example with the following instance;

$p_1 = w_1 = n+1$,

$p_2 = w_2 = n$,

$p_3 = w_3 = n$,

$s_1 = 2n$ and $t_1 = 2n + n^2$.

The optimal solution will sequence the jobs in J_2, J_3, and then J_1 with total weighted completion time equal to $F_w^* = n^2 + 2n^2 + (n^2+2n+n+1)(n+1) = n^3 + 7n^2 + 4n + 1$. If we use Step (iii) of the Combine-Algorithm we may get the sequence J_1,J_2,J_3 with a total weighted completion time equal to $F_{w2} = (n+1)(n+1) + (n^2 + 2n+1)n + (n^2 + 2n+1+n)n = 2n^3 + 6n^2 + 4n + 1$. Hence $(F_{w2} - F_w^*)/ F_w^*$ approaches 1 as n approaches infinity. QED.

Dynamic Programming for 1/r-a/ $\Sigma w_i C_i$

Now we find an optimal solution by using dynamic programming. We first establish an optimality property.

Lemma 4: There exists an optimal solution such that those jobs finished no later than s_1 follow the WSPT order, and those jobs finished after s_1 also follow WSPT order.

Proof: This lemma can be proved by an exchange scheme. We omit the detailed proof here by noting that it is possible that in the exchange procedure, we may move one job finished after t_1 to end up finished before s_1. In such a case, we can reorganize the order for those jobs finished before s_1 by the WSPT order without increasing the objective function, and Lemma 4 is true.

QED.

Remark: Lemma 4 implies that we can first sort the jobs in the WSPT order and then apply dynamic programming to solve it, as we will do in the following.

Dynamic Programming Algorithm

Reindex the jobs into WSPT order, and recall that $A_i = \sum_{j=1}^{i} p_j$.

Define $f^s(i,t)$ as the minimum total weighted completion time if we have scheduled jobs from J_1 up to J_i, the total processing time finished before s_1 is t, and the first job finished after t_1 starts at time s.

$$f^s(i,t) = \begin{cases} \min\{f^s(i-1,t-p_i) + w_i t,\ f^s(i-1,t) + w_i[s+(A_i-t)+(t_1-s_1)]\}, & \text{if } s+(A_i-t) > s_1 \\ f^s(i-1,t-p_i) + w_i t & \text{otherwise.} \end{cases}$$

where $s = s_1 - p_{max},\ldots,s_1$, $p_{max} = \max\{p_i: i=1,\ldots,n\}$,
$i = 1,\ldots,n$, and
$t = 0,\ldots,s_1$.

Initial Condition: For each s,

$$f^s(1,t) = \begin{cases} w_1 p_1 & \text{if } t = p_1 \\ w_1[s+p_1+(t_1-s_1)] & \text{if } t = 0 \\ \infty & \text{otherwise} \end{cases}$$

Optimal solution: Min $f^s(n,t)$ over all $s = s_1 - p_{max},\ldots,s_1$, and $t = 0,\ldots,s$

Justification: We can either assign job J_i to be finished before s_1 or after s_1. The former case will result in the objective function $f^s(i-1, t-p_i) + w_i t$ and the other case will result in the objective function $f^s(i-1, t) + w_i(s+A_i - t + t_1 - s_1)$. Note that we have tried all possible values of s and hence have covered all possible cases.

Complexity: Since there are p_{max} possible s, n possible i, and s_1 possible t in the functional equation, the complexity is $O(n \cdot p_{max} \cdot s_1)$.

3.2 Parallel Machine Problems

3.2.1 Pm/r-a/C_{max}

Note that the problem Pm//C_{max}, where there is no availability constraint, is a classical parallel machine scheduling problem and is NP-hard. Since our problem is more general than

$Pm//C_{max}$, it is obviously NP-hard. A special case, where $s_j = 0$ for all j and t_j may not be zero, has been studied by Lee (1991). He shows that the classical LPT algorithm will have a tight error bound of 1/2. He then provides a Modified LPT (MLPT) algorithm with error bound equal to 1/3. The MPLT treats each t_i as a special job, and then applies the LPT algorithm to all jobs. Whenever, a machine may be assigned more than one such special job then an exchange rule is applied to ensure that there is at most one special job in each machine. Finally, all special jobs are sequenced first in each machine.

Among those heuristics for the $Pm//C_{max}$ problem, LPT is the most popular one. The LPT algorithm can be described as "*Sorting the jobs in the nonincreasing order of their processing times and then assigning the jobs one by one to the minimum loaded machine.*" We first analyze the error bound of LPT for $Pm/r\text{-}a/C_{max}$. Recall that C^* denotes the optimal makespan and C_H denotes the makespan corresponding to heuristic H.

Lemma 5: If $s_j > 0$ for all j, then C_{LPT}/C^* can be arbitrarily large even for the two-machine problem.

Proof: Consider a problem with the following instance: $p_1 = p_2 = 3$, $p_3 = p_4 = p_5 = 2$, $s_1 = s_2 = 6$, $t_1 = t_2 = n$, and $m = 2$. The optimal solution is $C^* = 6$. However, $C_{LPT} = n+1$, and hence C_{LPT}/C^* can be arbitrarily large as n approaches infinity. QED.

Therefore, as mentioned at the beginning of the paper, we assume that *there is at least one machine always available. Namely, there exists at least one j such that* $s_j = t_j = 0$.

Note that in the LPT algorithm, assigning jobs one by one to the *minimum loaded machine* is aiming for *the job to be finished as early as possible*. In the traditional $Pm//C_{max}$ problem, these two goals are equivalent. However, in our $Pm/r\text{-}a/C_{max}$ problem, we may have different results. In the remainder of this subsection, we will call the traditional LPT(assigning job to the minimum loaded machine) ***LPT1***. We call it ***LPT2*** if we assign a job on the top of the list to a machine such that the finishing time of that job is minimized.

Lemma 6: C_{LPT1}/C^* can be arbitrarily large, even for $m = 2$.

Proof: Consider a problem with the following instance: $m = 2$, $s_1 = t_1 = 0$, $s_2 = 1$, and $t_2 = n$ Also let $p_1 = 2$ and $p_2 = 1$. The optimal solution is $C^* = 2$ with J_1 assigned to M_1 and J_2 assigned to M_2. The LPT1 algorithm can result in $C_{LPT1} = n+1$ with J_1 assigned to M_2 and J_2 assigned to M_1. Hence C_{LPT}/C^* can be arbitrarily large if n approaches infinity. QED.

Remark: If we apply LPT2 to the instance in the proof of Lemma 6, we will assign J_1 to M_1 and J_2 to M_2 with $C_{LPT2} = C^*$.

Theorem 3: For the Pm/r-a/C_{max} problem , $(C_{LPT2} - C^*)/C^* \le 1/2(1 - 1/m)$, and the bound is tight.

Proof: Let J_k be the job with $C_k = C_{LPT2}$. If $p_k > (1/2)C^*$, then in the optimal solution, there is at most one job from $\{J_1,...J_k\}$ on each machine. Also in LPT2, let M_j be the machine that contains J_k, then M_j contains only J_k. Hence we have $C_{LPT2} = C^*$. Now suppose that $p_k \le (1/2)C^*$. Note that there is no idle time before $C_k - p_k$. Furthermore, since we assume that there is at least one machine that is always available, there must be at least one machine which is busy up to $C_k - p_k$.

$$C_k - p_k \le \{(\sum_{i=1}^{n} p_i) - p_k + \sum_{j=1}^{m} [(C_k - p_k - s_j)^+ - (C_k - p_k - t_j)^+]\}/m,$$

where $(C_k - p_k - s_j)^+ - (C_k - p_k - t_j)^+$ is the unavailable time up to $C_k - p_k$ for M_j.
Therefore,

$$C_k - (1-1/m)p_k \le \{(\sum_{i=1}^{n} p_i) + \sum_{j=1}^{m} [(C_k - p_k - s_j)^+ - (C_k - p_k - t_j)^+]\}/m \le C^*.$$

We have $C_{LPT2} = C_k$
$\le C^* + (1-1/m)p_k$
$\le C^* + (1 - 1/m)(1/2)C^*$.

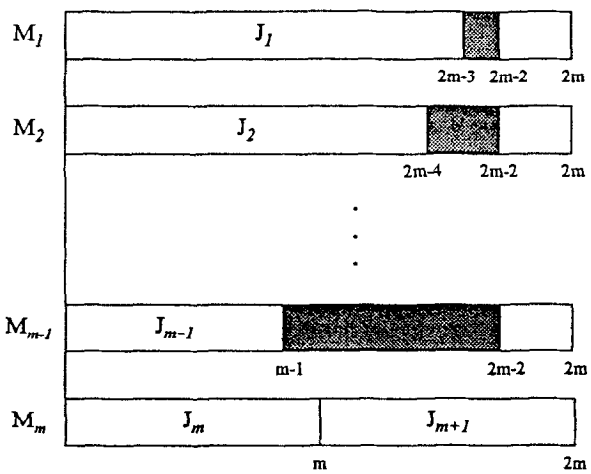

Figure 2a. Optimal Solution

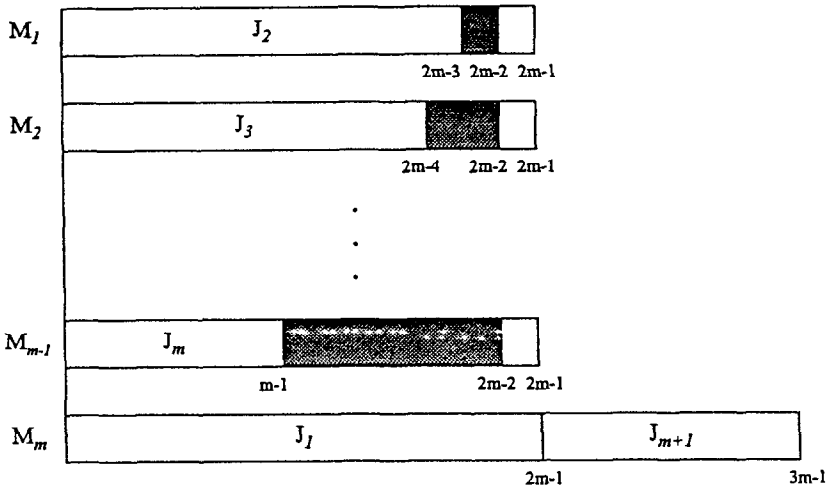

Figure 2b. LPT Solution

To show that the bound is tight, consider a problem with the following instance: there are n jobs to be assigned to m machines with n = m+1 and
t_j = 2m-2 and $s_j = t_j$ - j for j = 1,...,m-1, $s_m = t_m = 0$,
p_i = 2m-i for i = 1,...,m-1,
$p_m = p_{m+1}$ = m.

In the optimal solution we will assign J_i to M_i for i = 1,...,m and then J_{m+1} to M_m with makespan C^* = 2m(see Figure 2a). However, If we apply the LPT2 algorithm we may assign J_1 to M_m, J_{i+1} to M_i for i = 1,...,m-1, and then J_{m+1} to M_m with makespan C_{LPT2} = 3m-1(see Figure 2b). Hence $(C_{LPT2} - C^*)/C^* = 1/2\ (1 - 1/m)$. QED.

3.2.2. P2/r-a/($\Sigma w_i C_i$)

The problem P2/r-a/($\Sigma w_i C_i$) is NP-hard, because 1/r-a/($\Sigma w_i C_i$) was shown in Theorem 1 to be NP-hard. For the special case where $s_j = 0$ for all j and $w_i = 1$ for all i, Kaspi and Montreuil (1988) and Liman (1991) showed that SPT is an optimal policy.

Dynamic Programming for P2/r-a/$\Sigma w_i C_i$

Note that Lemma 4 is true for each machine in this case. Hence we can solve the problem in pseudo-polynomial time by dynamic programming. Without loss of generality, assume that M_1 *is available all the time*.

Reindex the jobs into WSPT order.

Define $f^s(i, v_1, v_2)$ as the minimum total weighted flow time if we have scheduled jobs from J_1 up to J_i, with the total processing time finished on M_1 as v_1, the total processing time on M_2 before s_2 as v_2, and the first job on M_2 that finishes after t_2 starts at time s.

$$f^s(i,v_1,v_2)=\begin{cases}\min\{f^s(i,v_1-p_i,v_2)+w_iv_1, f^s(i,v_1,v_2-p_i)+w_iv_2, f^s(i,v_1,v_2)+w_i(s+A_i-v_1-v_2+t_2-s_2)\} & \text{if } s+A_i-v_1-v_2>s_2\\ \min\{f^s(i,v_1-p_i,v_2)+w_iv_1, f^s(i,v_1,v_2-p_i)+w_iv_2\} & \text{otherwise.}\end{cases}$$

where $s = s_2 - p_{max},\ldots,s_2$, $p_{max} = \max\{p_i: i=1,\ldots,n\}$,
$i = 1,\ldots,n$, and
$v_1 = 0,\ldots,MS$, $v_2 = 0,\ldots, \min\{MS-v_1,s\}$.

Initial Condition: For each s,

$$f^s(1,v_1,v_2)=\begin{cases}w_1p_1 & \text{if } v_1=p_1 \text{ and } v_2=0\\ w_1p_1 & \text{if } v_1=0 \text{ and } v_2=p_1\\ w_1[s+p_1+(t_2-s_2)] & \text{if } v_1=0, v_2=0 \text{ and } s+p_1>s_2\\ \infty & \text{otherwise.}\end{cases}$$

Optimal solution: Min $f^s(n,v_1,v_2)$ over all $s = s_2 - p_{max},\ldots,s_2$, $v_1 = 0,..$ MS, and $v_2 = 0,\ldots,$ $\min\{MS-v_1,s\}$.

Justification: For job J_i we can either assign to M_1 or to M_2 and finished before s_2, or to M_2 and finished after s_2 as shown in the three terms of the equations.
Complexity: $O(n \cdot MS \cdot s_2 \cdot p_{max})$

4. THE NONRESUMABLE AVAILABILITY CONSTRAINT

4.1 Single Machine Problems

4.1.1. 1/nr-a/C_{max}

It can be shown easily that 1/nr-a/C_{max} is NP-hard by transforming the Partition Problem to this problem. Note also that, as mentioned earlier in the paper, we have only allowed a single disruption for each machine. However, if we allow multiple periods of maintenance, then the problem becomes NP-hard in the strong sense. This can be proved by transforming the 3-Partition problem. Following is a brief sketch of the proof. Given a 3-Partition problem with 3n jobs and B

as the size of each partition. We generate an instance for our scheduling problem, where there are 3n jobs, each having a size corresponding to 3-partition problem, and the maintenance periods are [(2i-1)B, 2iB], i =1,...,n-1. It can be seen easily that there exists a solution to the 3-Partition problem if and only if there exists a schedule for our problem with makespan equal to (2n-1)B.

Theorem 4: If we order the jobs in the LPT order, assigning to the machine as many jobs as possible before s_1, and then assigning the remaining jobs after t_1 in arbitrary order, then $(C_{LPT} - C^*)/C^* \leq 1/3$, and the bound is tight.

Proof: We consider two cases:

Case (i): $p_1 > s_1$.

(ia): If $p_2 > s_1$ then $C^* \geq p_1 + p_2 + t_1 > 3s_1$. Since $C_{LPT} - C^* \leq s_1$, we have $(C_{LPT} - C^*)/C^* \leq 1/3$.

(ib): If $p_2 \leq s_1$ and n = 2, then $C_{LPT} = t_1 + p_1 = C^*$. If $p_2 \leq s_1$ and n > 2, then $C_{LPT} - C^* \leq p_3$ and $C^* \geq p_1 + p_2 + p_3 \geq 3p_3$, and hence $(C_{LPT} - C^*)/C^* \leq 1/3$.

Case (ii): $p_1 \leq s_1$.
If n = 2, then it is obvious that $C_{LPT} = C^*$. Hence we discuss the case with n > 2.

(iia): If $p_1 + p_2 \leq s_1$, then $C_{LPT} - C^* \leq p_3$ and $C^* \geq p_1 + p_2 + p_3 \geq 3p_3$, and hence $(C_{LPT} - C^*)/C^* \leq 1/3$.

(iib): If $p_1 + p_2 > s_1$, and $p_1 + p_3 \leq s_1$, then $C_{LPT} - C^* \leq p_3$ and $C^* \geq p_1 + p_2 + p_3 \geq 3p_3$, and hence $(C_{LPT} - C^*)/C^* \leq 1/3$. If $p_1 + p_2 > s_1$, and $p_1 + p_3 > s_1$, then $(C_{LPT} - C^*) \leq s_1 - p_1 < p_3$. Since $C^* \geq p_1 + p_2 + p_3 \geq 3p_3$, we $(C_{LPT} - C^*)/C^* \leq 1/3$.

The following example shows that the bound is tight. Consider an instance with $p_1 = n+1$, $p_2 = p_3 = n$, and $s_1 = 2n$, $t_1 = 2n+1$. The LPT algorithm will schedule the jobs in J_1, J_2 and J_3 sequence with makespan equal to 4n+1 The optimal solution will sequence the jobs as J_2, J_3 and then J_1 with makespan equal to 3n+2. Hence $(C_{LPT} - C^*)/C^*$ approaches 1/3 as n approaches infinity.

4.1.2. 1/nr-a/L_{max}, 1/nr-a/ΣU_i, and 1/nr-a/$\Sigma w_i C_i$

Since 1/nr-a/C_{max} is NP-hard, hence 1/nr-a/L_{max}, and 1/nr-a/ΣU_i are also NP-hard. However, the following two lemmas show that the classical approach can be applied here with small error.

Lemma 7: If we apply Moore and Hodgson's algorithm to solve 1/nr-a/ΣU_i problem, then P(NR) $\leq P^*(NR) + 1$, where P(NR) denotes the corresponding number of tardy jobs and $P^*(NR)$ is the optimal number of tardy jobs.

Proof: Let $P^*(R)$ denote the optimal number of tardy jobs for the *resumable* case. Note that $P^*(R) \leq P^*(NR)$ since we can shift those jobs finished after t_J of the nonresumable problem to the left to fill the gap and have a feasible solution for the resumable problem with at least the same number of non-tardy jobs.

If we apply Moore and Hodgson's algorithm to solve the resumable problem and delete the job that started before s_J and finished after t_J and shift all the later jobs to start at t_J, this is a feasible solution to the nonresumable problem. Let the corresponding number of tardy jobs be P. Hence $P \leq P^*(R) + 1 \leq P^*(NR) + 1$. If we apply Moore and Hodgson's algorithm to solve 1/nr-a/ΣU_i problem, we will get a result no worse than P. Hence $P(NR) \leq P \leq P^*(NR) + 1$. QED.

Lemma 8: If we apply EDD algorithm to solve 1/nr-a/L_{max} problem then $L_{max} - L_{max}^* \leq p_{max}$, where $p_{max} = \{p_i: i=1,\ldots,n\}$.

Proof: This can be proved by noting that the maximum idle time before s_J cannot be greater than p_{max} and the fact that EDD is optimal for the resumable case. QED.

Adiri et al.(1989) and Lee and Liman (1991) show that 1/nr-a/ΣC_i is NP-hard. Lee and Liman prove that the error bound of applying SPT is 2/7. Now we consider our problem 1/nr-a/Σw_iC_i. This problem is obviously NP-hard. Furthermore, the example provided in the proof of Lemma 2 for the resumable case can be modified and applied here to show that the error bound of applying WSPT algorithm can be arbitrarily large.

Lemma 9: For the 1/nr-a/Σw_iC_i problem, $(F_W(WSPT) / F_w^*)$ can be arbitrarily large even if $w_i = p_i$ for all i.

4.2 Parallel machine Problems

4.2.1. Pm/nr-a/C_{max}

Since the problem is NP-hard we analyze two most popular heuristics; List Scheduling and LPT.

List Scheduling(LS): *Given an arbitrary order of jobs, assign the job to the machine such that the job can be finished as early as possible.*

Theorem 5: $C_{LS}/C^* \leq m$, where m is the number of machines.

Proof: Recall we assume that at least one machine is always available. Hence $C_{LS} \leq \Sigma(P_i) \leq m((\Sigma P_i)/m) \leq mC^*$. To show that the bound is tight, consider an example with the following instance: $p_1 = p_2 = ... = p_m = 1$, $p_{m+1} = p_{m+2} = ... = p_{2m-1} = p_{2m} = n$, $s_1 = t_1 = 0$, and $s_i = n$, $t_i = n^2$ for i = 2,...m. Using list scheduling, we may assign J_i to M_i for i = 1,2,...,m, and the remaining jobs are all assigned to M_1, with makespan equal to mn+1. However, in the optimal solution we should assign J_{m+i} to M_i for i = 1,...,m, and assign the remaining $J_1,...J_m$ to M_1 with equal to n+m. C_{LS}/C^* approaches m as n approaches infinity. QED.

We now apply the LPT algorithm to our problem. Note that the sequence on any machine may not be in the LPT order. The reason is that we may not be able to assign a large job to be finished before s_j, yet a small job which was assigned later may be able to finish before s_j.

Theorem 6: $C_{LPT}/C^* \leq (m+1)/2$ and the bound is tight.

Proof: Let J_k be the job such that $C_k = C_{LPT}$. First note that deleting all jobs that are assigned after J_k will not affect the worst case error bound. Note also that if $s_j \leq C_k$, then the idle time on M_j before s_j is not greater than p_k. Furthermore, if s_j is smaller than p_k, then this machine can be deleted without affecting the worst case error bound. Hence we assume that each machine has at least one job before s_j.

If $p_k > (1/2)C^*$, then similar to the proof of Theorem 3, it can be shown that $C_{LPT} = C^*$. Suppose that $p_k \leq (1/2)C^*$. If $C_k > s_j$ then the idle time of M_j before s_j is at most p_k, and if $C_k \leq s_j$ then the idle time of M_j before C_k is at most p_k. Let t be the time instant such that the sum of the available time for all machines up to t is equal to Σp_i. Namely

$$(\sum_{i=1}^{n} p_i) = mt - \sum_{j=1}^{m} [(t - s_j)^+ - (t - t_j)^+],$$

where $(t - s_j)^+ - (t - t_j)^+$ is the unavailable time up to t for M_j.

Therefore, $C^* \geq t$. Note that at least one machine is always available. It can be checked that $C_k \leq t + (m - 1)p_k$. This is true because in LPT schedule, after t we have at most $(m - 1)p_k$ to be processed and we at least can put all of them on M_1 and end up with a makespan not greater than $t + (m - 1)p_k$. Hence $C \leq C^* + (m - 1)(1/2)C^* = [(m + 1)/2]C^*$.

To show that the bound is tight, consider a problem with the following instance.

$p_i = n + m - i$ for i = 1, 2, .., m - 1,

$p_i = n$ for i = m, m + 1,..., 2m.

Also $s_j = 2n + m - j - 1$ and $t_j = s_j + n^2$ for j = 1, 2, ..., m - 1. $s_m = t_m = 0$.

Assume that $n > m^2$. If we apply the LPT algorithm, we may assign J_i to M_i for i = 1, .., m - 1 and the remaining jobs to M_m with makespan $C_{LPT} = (m + 1)n$. However, in the optimal solution, we will assign J_{i+1} and J_{m+i} to M_i for i = 1,..., m - 1, and J_1 and J_{2m} to M_m with a makespan $C^* = 2n + m - 1$. Hence C_{LPT}/C^* approaches (m + 1)/2 as n approaches infinity. Hence the bound is tight. QED.

4.2.2. P2/nr-a/Σw_iC_i

Since the single machine problem, 1/nr-a/Σw_iC_i, is NP-hard, our problem is also NP-hard. The special case with $w_i = 1$ for all i, has been studied under different availability constraints environments. Lee and Liman (1993) study the P2/nr-a/(ΣC_i) problem with t_2 = infinity. They prove that the problem is NP-hard and solved it with a dynamic programming algorithm. They also provide a SPT-based heuristic and showed that the error bound is 1/2. Mosheiov(1994) studies Pm/r-a/(ΣC_i) under the condition that M_j is available in the interval $[x_i, y_i]$ where $0 \le x_i < y_i$, and shows that SPT is asymptotically optimal as the number of jobs approaches infinity.

Assume that M_1 is available all the time. Similar to that for P2/r-a/ΣwiCi ,the following dynamic programming can solve our problem optimally. Note that the complexity is smaller than that for P2/r-a/Σw_iC_i.

Reindex the jobs into WSPT order.

Define f(i, v_1, v_2) as the minimal total weighted completion time if we have assigned jobs J_1,..., J_i, total processing time of M_1 as v_1 and the total processing time of M_2 before s_2 as v_2.

$$f(i, v_1, v_2) = \min \begin{cases} f(i-1, v_1 - p_i, v_2) + w_i v_1 \\ f(i-1, v_1, v_2 - p_i) + w_i v_2 \\ f(i-1, v_1, v_2) + w_i (t_2 + A_i - v_1 - v_2) \end{cases}$$

where i = 1,...,n, v_1 = 0,...,MS, v_2 = 0,...,min{MS - v_1, s_2}.

Initial Condition:

$$f(1, v_1, v_2) = \begin{cases} w_1 p_1 & \text{if } v_1 = p_1 \text{ and } v_2 = 0 \\ w_1 p_1 & \text{if } v_1 = 0 \text{ and } v_2 = p_1 \\ w_1 (p_1 + t_2) & \text{if } v_1 = 0 \text{ and } v_2 = 0 \\ \infty & \text{otherwise.} \end{cases}$$

Complexity: $O(n \cdot MS \cdot s_2)$.

5. CONCLUSION

Most literature in scheduling assumes that machines are available simultaneously at all times. However, this availability may not be true in real industry settings. In this paper, we assume that the machine may not always be available. This happens often in the industry due to a machine breakdown (stochastic) or preventive maintenance (deterministic) during the scheduling period. We study the scheduling problem under this general situation and for *the deterministic case*.

This paper is part of an on-going research on the machine availability constraint problems. In this paper, we contentrate on the extension of basic theoretical results in the scheduling literature. We have provided an extensive study for the machine scheduling problem with an availability constraint under different performance measures (makespan, total weighted completion time, tardiness, and number of tardy jobs) and different machine situations (single machine and parallel machines). Most problems are NP-hard excepted for several single machine resumable problems. In most NP-hard problems, we develop pseudo polynomial dynamic programming to solve them optimally. Furthermore, heuristic methods, which are important for practical applications, are provided and their worst case error bounds are also analyzed.

Several extensions are in progress, which include (i) the semi-resumable case where some extra setup time may be required when a job is resumed, and (ii) more complicated job shop and open shop scheduling, (iii) parallel machine scheduling environments with partially overlapping preventive maintenance periods on different machines, (iv) the problem with more than one maintenance intervals, (v) combining job sequencing with maintenance scheduling.

References:

Adiri, I., Bruno J., Frostig E., and A.H.G. Rinnooy Kan, "Single Machine Flow-Time Scheduling with a Single Breakdown," *Acta Informatica*, 26, (1989), pp. 679-696.

Baker, K., *Elements of Sequencing and Scheduling*, (1993), unpublished manuscript.

Blazewicz, J., K. Ecker, G. Schmidt, and J. Weglarz, *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, 1993, New York.

Garey M. R., and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979, New York.

Graves, S. C, "A Review on Production Scheduling," *Operations Research*, 29, (1981), pp. 646-676.

Herrmann, J., C.-Y. Lee, and J. Snowdon, "A Classification of Static Scheduling Problems," in *Complexity in Numerical Optimization*, P. M. Pardalos (ed.), (1993), pp. 203-253, World Scientific.

Kaspi, M. and B. Montreuil, "On the Scheduling of Identical Parallel Processes with Arbitrary Initial Processor Available Times," Research Report 88-12, School of Industrial Engineering, Purdue University, 1988.

Kraemer, F., ad C.-Y. Lee, "Common Due-Window Scheduling," *Production and Operations Management*, 2, (1993), pp.262-275.

Lawler, E.L., J. K.,Lenstra, A.H.G. Rinnooy Kan, and D. Shmoys, "Sequencing and Scheduling: Algorithms and Complexity," in *Handbook in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, S.S. Graves, A.H.G. Rinnooy Kan, and P. Zipkin (eds.), pp.445-522, North-Holland, New York, 1993.

Lee, C.-Y., "Parallel Machines Scheduling with Non-Simultaneous Machine Available Time," *Discrete Applied Mathematics*, 30,(1991), pp. 53-61.

Lee, C.-Y., "Minimizing the Makespan in the Two-Machine Flowshop Scheduling Problem with an Availability Constraint," (1995), submitted for publication.

Lee, C.-Y., and S. D. Liman, "Single Machine Flow-Time Scheduling With Scheduled Maintenance," *Acta Informatica*, 29, (1992), pp. 375-382.

Lee, C.-Y., and S. D. Liman, "Capacitated two-parallel machines scheduling to minimize sum of job completion times," *Discrete Applied Mathematics*, 41, (1993), pp. 211-222.

Lei, L., and T.-J. Wong, "The Minimum Common-Cycle Algorithm for Cyclic Scheduling of Two Material Handling Hoists with Time Window Constraints," *Management Science*, 37, (1991), pp.1629-1639.

Liman, S., Scheduling with Capacities and Due-Dates, Ph.D. Dissertation, Industrial and Systems Engineering Department, University of Florida, 1991.

Morton, T. E., and D. W. Prentico, *Heuristic Scheduling Systems*, John Wiley & Sons, Inc. New York, 1993.

Mosheiov G., "Minimizing the Sum of Job Completion Times on Capacitated Parallel Machines," *Mathl. Comput. Modelling*, 20, 1994, pp.91-99.

Pinedo, M., *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, 1995, Englewood Cliffs, New Jersey, 1995.

Sahni, S., "Approximation Algorithms for the 0/1 Kanpsack Problem," *Journal of the Association for Computing Machinery*, 20, 1975, pp. 115-124.

Schmidt G., "Scheduling Independent Tasks with Deadlines on Semi-identical Processors," *Journal of Operational Research Society*, 39, 1984, 271-277.

Tanaev, V. S., Y. N. Sotskov, and V. A. Strusevich, *Scheduling Theory. Multi-Stage Systems*, Kluwer Academic Publishers, 1994, The Netherlands.