Preemptive Scheduling of Uniform Machines Subject to Release Dates

J. Labetoulle

Centre National d'Etudes des Télécommunications
Issy les Moulineaux, France

E.L. Lawler

University of California
Berkeley, U.S.A.

J.K. Lenstra

Centre for Mathematics and Computer Science
Amsterdam, The Netherlands

A.H.G. Rinnooy Kan

Erasmus University
Rotterdam, The Netherlands

We shall be concerned with finding optimal preemptive schedules on parallel machines, subject to release dates for the jobs. Two polynomial-time algorithms are presented. The first algorithm minimizes maximum completion time on an arbitrary number of uniform machines. The second algorithm minimizes maximum lateness with respect to due dates for the jobs on an arbitrary number of identical machines or on two uniform machines. A third algorithm for minimizing maximum lateness on an arbitrary number of uniform machines is briefly discussed. *NP*-hardness is established for the problem of minimizing total weighted completion time on a single machine.

1. Introduction

We consider scheduling problems in which n independent jobs $J_1, \ldots, J_n$ have to be processed on m parallel machines $M_1, \ldots, M_m$. Each machine can handle at most one job at a time and each job can be executed on at most one machine at a time. Each job J_j becomes available for processing at its release date r_j. It has an execution requirement p_j and possibly also a due date or deadline d_j and a

ISBN 0-12-566780-9

weight w_j. Unlimited preemption is allowed: the processing of any job may arbitrarily often be interrupted and resumed at the same time on a different machine or at a later time on any machine. The machines are assumed to be uniform, i.e., each machine M_i has a speed s_i, and complete execution of J_j on M_i would require p_j/s_i time units. If all speeds are equal, the machines are identical; if $m = 1$, we have a single machine. We assume that all numerical data r_j, p_j, d_j, w_j, s_i are integers.

A feasible schedule defines a completion time C_j and a lateness $L_j = C_j - d_j$ for each J_j. We may choose to minimize the maximum completion time $C_{\max} = \max_{1 \le j \le n}\{C_j\}$, the maximum lateness $L_{\max} = \max_{1 \le j \le n}\{L_j\}$, the total completion time $\sum C_j = \sum_{j=1}^{n} C_j$, or the total weighted completion time $\sum w_j C_j = \sum_{j=1}^{n} w_j C_j$.

When scheduling jobs subject to release dates, one can distinguish between three types of algorithms. An algorithm is on-line if at any time only information about the available jobs is required. It is nearly on-line if in addition the next release date has to be known. It is off-line if all information is available in advance.

In Section 2 we consider the minimization of $C_{\max}$ on m uniform machines. For the case that all release dates are equal, Horvath, Lam and Sethi [8] derived a closed form expression for the optimum value of $C_{\max}$. Gonzalez and Sahni [6] proposed an $O(m \log m + n)$ algorithm which produces a schedule meeting this value and containing at most $2(m-1)$ preemptions. For the case that the release dates are arbitrary, Sahni and Cho [18] gave an $O(n \log n + mn)$ off-line algorithm to determine if there exists a schedule in which no job is completed after a common deadline. We will present an $O(n^2)$ nearly on-line algorithm to minimize $C_{\max}$; Sahni and Cho [17] independently developed an $O(mn \log n + m^2 n)$ nearly on-line algorithm that is very similar to ours. We will indicate how to obtain an $O(n \log n + mn)$ off-line implementation of our algorithm. These methods can also be used to minimize $L_{\max}$ in the case of equal release dates.

In Section 3 we consider the minimization of $L_{\max}$. For the case of equal release dates, Horn [7] proposed an $O(n^2)$ algorithm to minimize $L_{\max}$ on m identical machines. For the case of arbitrary release dates, he gave an off-line algorithm, based on a network flow computation, to determine if there exists a schedule in which no job is completed after its deadline. Bruno and Gonzalez [3] adapted this feasibility test to the case of two uniform machines. We will extend both methods by presenting polynomial-time algorithms to minimize $L_{\max}$. Martel [13,14] recently proposed a feasibility test for the case of m uniform machines, based on a polymatroidal network flow model, and used it to obtain a polynomial-time algorithm to minimize $L_{\max}$. We will discuss these results as well.

In Section 4 we consider the minimization of $\sum C_j$ and $\sum w_jC_j$. For the case of equal release dates, Bruno and Gonzalez [5] proposed an $O(n \log n + mn)$ algorithm to minimize $\sum C_j$ on m uniform machines. It is well known that in the case of identical machines allowing preemptions will not decrease the optimal value of $\sum w_jC_j$ [15]. It follows that $\sum w_jC_j$ is minimized on a single machine by scheduling the jobs in order of nonincreasing ratios w_j/p_j [19], and that the problem on two identical machines is already *NP*-hard [2,12]. For the case of arbitrary release dates, $\sum C_j$ is minimized on a single machine by an obvious on-line extension of the above ordering rule [1]; we will establish *NP*-hardness for the problem of minimizing $\sum w_jC_j$.

In Section 5 we conclude by indicating a major open problem and some important recent developments in the area of preemptive scheduling.

2. Maximum Completion Time

We first consider the problem of minimizing the *maximum completion time* $C_{\max}$ on m uniform machines. The jobs and the machines are assumed to be ordered in such a way that $r_1 \le \cdots \le r_n$ and $s_1 \ge \cdots \ge s_m$.

We will describe a nearly on-line algorithm that considers the time intervals $R_k = [r_k, r_{k+1}]$ in order of increasing k. For each successive interval R_k $(k = 1,\ldots,n-1)$, denote the remaining execution requirement of J_j at r_k by $p_j^{(k)}$ $(j = 1,\ldots,k)$ and renumber the jobs so that $p_1^{(k)} \ge \cdots \ge p_k^{(k)}$. The subalgorithm to be applied in each interval determines the amounts by which the $p_j^{(k)}$ are to be decreased within R_k. At time r_n, all jobs are available, and it is well known [8] that the minimum time for their completion is given by

$$C^*_{\max} = r_n + \max\{\max_{1\le l\le m-1}\{\textstyle\sum_{j=1}^{l} p_j^{(n)} / \sum_{i=1}^{l} s_i\}, \qquad (1)$$
$$\textstyle\sum_{j=1}^{n} p_j^{(n)} / \sum_{i=1}^{m} s_i\}.$$

The portion of an optimal schedule within any interval R_k can be constructed by applying the Gonzalez-Sahni algorithm [6] to the quantities $p_j^{(k)} - p_j^{(k+1)}$ determined by our subalgorithm. Similarly, a schedule for the final interval $[r_n, C^*_{\max}]$ can be constructed by applying the same algorithm to the quantities $p_j^{(n)}$. Since both the subalgorithm and the schedule construction procedure require $O(n)$ time for each interval, the algorithm requires $O(n^2)$ time overall; it introduces $O(mn)$ preemptions into the optimal schedule.

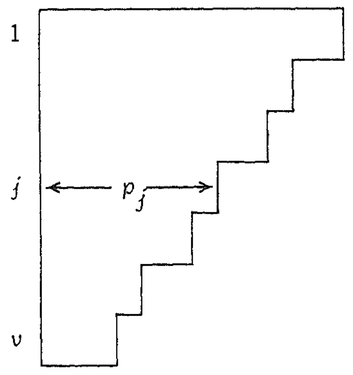
Our algorithm has the property that the remaining execution requirements passed on to the next interval will be as *evenly* distributed as possible. More specifically, for each k there is no way to process the jobs before r_k that could lead to a smaller value for *any* of the partial sums $\sum_{j=1}^{l} p_j^{(k)}$ $(l = 1,\ldots,k)$. This immediately implies the correctness of the algorithm, since each of these partial sums appearing in (1) is as small as it could possibly be.

Rather than giving an inductive proof of this property, we will settle for a simpler correctness proof of the entire algorithm. This proof will also serve to introduce algorithmic refinements, by which the optimum value $C^*_{\max}$ can be determined in $O(n \log n + mn)$ time. An actual schedule can be constructed by applying the Sahni-Cho algorithm [18], using $C^*_{\max}$ as a common deadline for the jobs. This off-line approach requires $O(n \log n + mn)$ time and introduces $O(mn)$ preemptions into the optimal schedule.

Let us consider an interval R_k for fixed k. Given the $p_j^{(k)}$ $(j = 1,\ldots,k)$, we have to determine the $p_j^{(k+1)}$ to be passed on to the next interval R_{k+1}.

Suppose that at time r_k the jobs $J_1, \ldots, J_v$ are available and not yet completed, with $p_1^{(k)} \geq \cdots \geq p_v^{(k)} > 0$. For ease of notation, we drop the superscripts. Thus, denote the given $p_j^{(k)}$ by p_j and the unknown $p_j^{(k+1)}$ by q_j $(j = 1,\ldots,v)$, and let $t = r_{k+1} - r_k$. For purposes of exposition, we assume for the time being that, if $m < v$, machines $M_{m+1}, \ldots, M_v$ with $s_{m+1} = \cdots = s_v = 0$ are added to the model.

The p_j can be viewed as defining a staircase pattern as in Figure 1. The q_j will be chosen in such a way that they define a similar pattern.



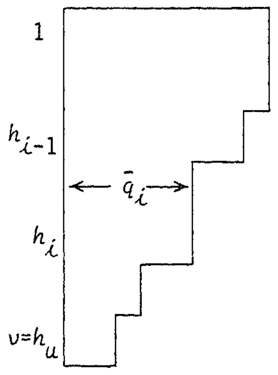


Figure 1. Staircase pattern at r_k. Figure 2. Staircase pattern at r_{k+1}.

As illustrated in Figure 2, such a staircase can be characterized by a sequence $((h_1, \bar{q}_1),\ldots,(h_u, \bar{q}_u))$, where $\bar{q}_i = q_j$ for each J_j with $h_{i-1} + 1 \leq j \leq h_i$ $(i = 1,\ldots,u;\ h_0 = 0;\ h_u = v)$. A first condition for feasibility is that

$$\bar{q}_i > \bar{q}_{i+1} \quad (i = 1,\ldots,u-1). \tag{2}$$

The staircase $((h_1, \bar{q}_1),\ldots,(h_u, \bar{q}_u))$ will be constructed in such a way that, for $i = 1,\ldots,u-1$, the capacity of $M_{h_{i-1}+1}, \ldots, M_{h_i}$ will be fully utilized to decrease $p_{h_{i-1}+1}, \ldots, p_{h_i}$ to $\bar{q}_i$. A second condition for feasibility is therefore that

$$\textstyle\sum_{j=h_{i-1}+1}^{l} q_j = (l-h_{i-1})\bar{q}_i \geq \sum_{j=h_{i-1}+1}^{l} p_j - t\sum_{j=h_{i-1}+1}^{l} s_j \tag{3}$$
$$(l = h_{i-1} + 1,\ldots,h_i;\ i = 1,\ldots,u),$$

with the corners of the staircase, except possibly the last one, corresponding to strict equalities:

$$\textstyle\sum_{j=h_{i-1}+1}^{h_i} q_j = (h_i-h_{i-1})\bar{q}_i = \sum_{j=h_{i-1}+1}^{h_i} p_j - t\sum_{j=h_{i-1}+1}^{h_i} s_j \tag{4}$$
$$(i = 1,\ldots,u-1).$$

A third condition for feasibility is of course that

$$0 \leq q_j \leq p_j \quad (j = 1,\ldots,v). \tag{5}$$

We tentatively construct the first step of the staircase by setting

$$h_1 = 1,\ \bar{q}_1 = p_1 - ts_1.$$

Generally, having found i tentative steps $(h_1, \bar{q}_1),\ldots,(h_i, \bar{q}_i)$ with $h_i < v$ and $\bar{q}_1 > \cdots > \bar{q}_i$, we construct the $(i+1)$-st tentative step by setting

$$h_{i+1} = h_i+1,\ \bar{q}_{i+1} = p_{h_{i+1}} - ts_{h_{i+1}}. \tag{6}$$

If $\bar{q}_i > \bar{q}_{i+1}$ and $\bar{q}_i \geq 0$, the staircase $((h_1, \bar{q}_1),\ldots,(h_{i+1}, \bar{q}_{i+1}))$ satisfies (2) and (4); we increment i by one and, if h_i is still smaller than v, construct the next step.

Suppose now that $\bar{q}_i \leq \bar{q}_{i+1}$ or $\bar{q}_i < 0$. In the latter situation, there is excess capacity on $M_{h_{i-1}+1},\ldots,M_{h_i}$; in both cases, some of the capacity of these machines has to be devoted to processing J_{h_i+1} if (2) and (4) are to be satisfied. We therefore reconstruct the i-th step so as to include J_{h_i+1} as well: h_i is incremented by one, and $\bar{q}_i$ is recalculated according to

$$\bar{q}_i = (\textstyle\sum_{j=h_{i-1}+1}^{h_i} p_j - t\sum_{j=h_{i-1}+1}^{h_i} s_j)/(h_i-h_{i-1}) \tag{7}$$

(cf. (4)). As a result, it may now be that $\bar{q}_{i-1} \leq \bar{q}_i$ ($\bar{q}_{i-1} < 0$ cannot occur). In this case, we reconstruct the $(i-1)$-st step so as to include the current i-th step: h_{i-1} is increased to h_i, and $\bar{q}_{i-1}$ is recalculated as in (7). We continue until once more $\bar{q}_1 > \ldots > \bar{q}_i$; the adjusted staircase $((h_1, \bar{q}_1), \ldots, (h_i, \bar{q}_i))$ includes one more job and may have fewer steps than before. If h_i is still smaller than v, we construct the next step according to (6).

The process is terminated as soon as $h_i = v$. If $\bar{q}_i < 0$, we reset $\bar{q}_i = 0$ and note that only in this situation the last corner of the staircase does not correspond to a strict equality.

We have to verify that the resulting staircase $((h_1, \bar{q}_1), \ldots, (h_u, \bar{q}_u))$ and the corresponding remaining execution requirements $q_1, \ldots, q_v$ indeed satisfy the feasibility conditions (2) - (5). For (2) and (4), this is obvious. To see that (3) must be true, note that each $\bar{q}_i$ is initially defined by an equality constraint and can only increase thereafter. To verify (5), it is sufficient to show that $\bar{q}_i \leq p_{k_i}$. Subtracting (3) for $l = h_i - 1$ from (4), we find $\bar{q}_i \leq p_{k_i} - ts_{k_i}$, which implies the desired result.

Let us now analyze the running time of the subalgorithm. The number of step constructions as in (6) is exactly v. The number of step reconstructions as in (7) is at most $v-1$, since during each adjustment two steps are collapsed into one. It follows that the process terminates in $O(v)$ time. This presupposes that the given values p_j are ordered; but since the relative order of the remaining execution requirements does not change, we can maintain an ordered list of these values and insert the value of the job that becomes available at r_k in $O(v)$ time. Hence the subalgorithm determines the values q_j for each interval in $O(v)$ time. As has been indicated above, the Gonzalez-Sahni algorithm [6] can be applied to construct an actual schedule in each interval in $O(v)$ time as well. We thus have arrived at a nearly on-line algorithm that requires $O(n^2)$ time overall.

We now intend to prove the correctness of the algorithm.

We note first that not only does the relative order of the remaining execution requirements remain invariant, but also the following stronger property holds: as soon as two remaining execution requirements become equal, they will remain equal. To see this, suppose that $p_j = p_{j+1}$ at time r_k, and let $h_i = j$. According to (6), we set $\bar{q}_{i+1} = p_{j+1} - ts_{j+1}$. But $\bar{q}_i \leq p_j - ts_j \leq p_j - ts_{j+1} = \bar{q}_{i+1}$, and we have to reconstruct the i-th step so as to include J_{j+1} as well.

This leads us to define the rank of an available job J_j at time r_k as the value h_i for which $h_{i-1}+1 \leq j \leq h_i$. The rank of a job at time r_n is defined analogously as its step height that would be found if the subalgorithm were to be applied in the interval $[r_n, C^*_{\max}]$. A job will be called *critical* if its rank is at most $m-1$ and *noncritical* otherwise. The rank of a job cannot decrease; in particular, once a job becomes noncritical, it never becomes critical again. It follows from (4) that in any interval the fastest h_l machines are exclusively processing the longest h_l critical jobs. A critical job is processed continuously from its release date until it either is completed or becomes noncritical.

These observations suggest the following correctness proof for the algorithm. First, suppose that the schedule ends at $C^*_{\max}$ with the simultaneous completion of l critical jobs $(l < m)$. At any time when l' of these jobs are available, they are processed by the fastest l' machines. In this case, the schedule is clearly optimal.

Alternatively, suppose that the schedule ends with the simultaneous completion of m noncritical jobs. Let r_k be the last release date just prior to which there is idle time on some machine. Ignoring the jobs that are available but noncritical at time r_{k-1}, we conclude that the portion of the schedule for the remaining jobs has a structure as illustrated in Figure 3.

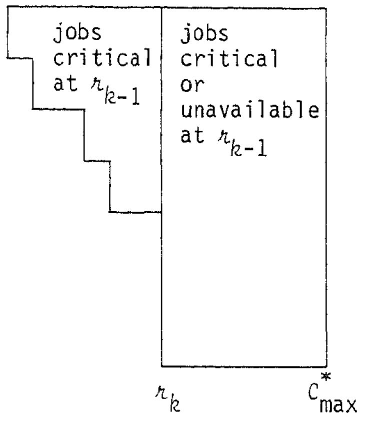

jobs critical at r_{k-1}
jobs critical or unavailable at r_{k-1}
r_k
$C^*_{\max}$

Figure 3 - Simultaneous completion of noncritical jobs.

Before r_k, the available critical jobs are processed by the fastest machines. Between r_k and $C^*_{\max}$, there is no idle time. It follows that the schedule is optimal for the jobs under consideration and *a fortiori* that $C^*_{\max}$ is the minimum time to complete all the jobs.

Let us use the new terminology to describe a more efficient implementation of the subalgorithm. We will reduce the running time by dealing more carefully with the noncritical jobs, circumventing the need to introduce machines of speed zero.

Consider the situation after a typical application of the subalgorithm, as illustrated in Figure 4.

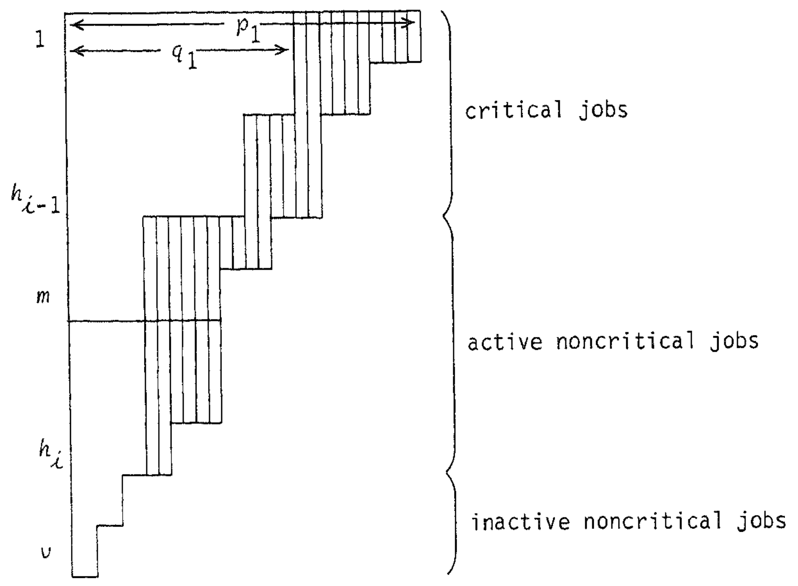


Figure 4 - Staircase patterns at r_k and r_{k+1}.

The noncritical jobs of lowest rank, i.e., $J_{h_{i-1}+1},\ldots,J_{h_i}$ where $h_{i-1}+1 \leq$, $m \leq h_i$, will be called *active*. In the interval R_k, their remaining execution requirements are reduced by machines $M_{h_{i-1}+1}, \ldots, M_m$ to a common amount $\bar{q}_i$. The remaining noncritical jobs, i.e., $J_{h_i+1},\ldots,J_v$, will be called *inactive*. In R_k, their remaining execution requirements are not reduced at all, since $\bar{q}_i > \bar{q}_{i+1} = p_{h_{i+1}}$ (note that $s_{h_{i+1}} = 0$).

As a first refinement, the subalgorithm does not have to deal with the active noncritical jobs separately, since their remaining execution requirements will remain equal throughout. They can easily be handled simultaneously by straightforward generalizations of (6) and (7). As a second refinement, the subalgorithm can be terminated as soon as either $h_i = v$ or $h_i \geq m$ and $\bar{q}_i > p_{h_i+1}$.

Rather than maintaining an ordered list of all remaining execution requirements, we have to do so only for the largest $m-1$ of them. We simply record the number of active noncritical jobs, their common remaining execution requirement, and the lowest index of any of them. Finally, we maintain a priority queue for the remaining requirements of the inactive noncritical jobs.

At each release date, the execution requirement of the job that becomes available is, depending on its size, inserted either in the ordered list in $O(m)$ time or in the priority queue in $O(\log n)$ time. The staircase computations for the longest $m-1$ jobs and the active noncritical jobs require $O(m)$ time in each interval and $O(mn)$ time overall. The queue operations require $O(\log n)$ time in each interval and $O(n \log n)$ time overall, since once an inactive job becomes active and is withdrawn from the queue, it remains active throughout. Hence successive applications of the modified subalgorithm determine the value $C^*_{\max}$ in $O(n \log n + mn)$ time. As has been indicated above, the Sahni-Cho algorithm [18] can be applied to construct an actual schedule in the interval $[r_1, C^*_{\max}]$ in $O(n \log n + mn)$ time as well. We thus have arrived at an off-line algorithm that requires $O(n \log n + mn)$ time overall.

3. Maximum Lateness

We now consider the problem of minimizing the *maximum lateness* $L_{\max}$ on m identical machines.

A relaxed version of this problem is to test a *trial value* of $L_{\max}$ for feasibility. That is, for a given value y, one has to determine whether or not there exists a schedule for which $L_{\max} \le y$. This condition is equivalent to the requirement that no job J_j is completed after an *induced deadline* $d_j + y$. Sahni [16] proposed an off-line algorithm for the case of equal deadlines that requires $O(n \log mn)$ time and introduces at most $n-2$ preemptions. He also showed that there can be no nearly on-line algorithm for the case of arbitrary deadlines. Horn [7] proposed a network flow algorithm for the latter case. He suggested that one might conduct a search for the optimum value of $L_{\max}$, but offered no upper bound on the number of trial values that have to be tested. Our contribution here is to obtain such a bound and to show that it is polynomial in the problem size.

Horn's approach is as follows. Suppose y is a trial value for $L_{\max}$. Let $\{e_1, \ldots, e_{2n}\}$ $(e_1 \le \cdots \le e_{2n})$ be the ordered collection of release dates r_j and induced deadlines $d_j + y$; if a release date and a deadline are equal, the smaller index is to be assigned to the release date. Further, define the time interval $E_k = [e_k, e_{k+1}]$ for $k = 1,\ldots,2n-1$.

A flow network is constructed with job vertices $J_1,\ldots,J_n$, interval vertices $E_1,\ldots,E_{2n-1}$, a source vertex S and a sink vertex T. There is an arc (J_j, E_k) of capacity $e_{k+1} - e_k$ if and only if $r_j \le e_k$ and $e_{k+1} \le d_j + y$. In addition, there is an arc (S, J_j) of capacity p_j for $j = 1,\ldots,n$ and an arc (E_k, T) of capacity $m(e_{k+1} - e_k)$ for $k = 1,\ldots,2n-1$. Now, a maximum value flow is found in $O(n^3)$ time

$\{J_j | j \in X\}$ and each vertex E_k, it is required that the total flow through the set of arcs $\{(J_j, E_k) | j \in X\}$ is no more than $\rho(k, |X|)$, where

$$\rho(k, l) = (e_{k+1} - e_k) \sum_{i=1}^{\min\{l,m\}} s_i .$$

Further, each arc (E_k, T) has capacity $\rho(k, m)$. This is a special case of the *polymatroidal* network flow model [11], in which the capacities are defined by nonnegative and submodular functions, one for each set of arcs entering (or leaving) a specific vertex; the model derives its name from the fact that such a set function corresponds to the rank function of a polymatroid. Traditional notions such as augmenting paths and labeling techniques can be extended to find a maximum value flow for the scheduling model in $O((m^2n^3 + n^4)(m + \log n))$ time [13,14].

To determine the optimum value of $L_{\max}$, one again uses the concept of critical trial values so as to arrive at a polynomial-time algorithm that requires $O(n^2 + n \log s_1 + \log(\max_j\{d_j\} + P))$ calls to the feasibility routine [14]. Admittedly, the degree of the polynomial is on the high side. By way of compensation, we should add that the investigation of polymatroidal network flow models was inspired by this scheduling problem and has yielded a useful generalization and unification of classical network flow theory and much of the theory of matroid optimization [11].

4. Total Weighted Completion Time

We finally consider the problem of minimizing the *total completion time* $\sum C_j$ or the *total weighted completion time* $\sum w_jC_j$.

Let us first assume that all release dates are equal. Bruno and Gonzalez [5] proposed a simple algorithm to minimize $\sum C_j$ on m uniform machines: order the jobs according to nondecreasing execution requirements, and schedule each successive job preemptively so as to minimize its completion time. This algorithm is illustrated in Figure 5. Obviously, it requires $O(n \log n + mn)$ time and introduces at most $(m-1)(n - \frac{m}{2})$ preemptions.

The Bruno-Gonzalez algorithm not only minimizes $\sum C_j$ but also $\sum_{j=1}^{l} C_j$ for $l = 1,\ldots,n-1$. Further, it minimizes $\sum w_jC_j$ provided that the weights are *agreeable*, i.e., $p_j < p_k$ implies $w_j \geq w_k$ [5].

A characteristic feature of the algorithm is that at each point in time the fastest machines are working on the jobs with the shortest remaining execution requirements. One may consider a straightforward extension to the case of arbitrary release dates, in which at each subsequent release date the above rule is applied to the available jobs.

$m = 3$, $s_1 = 3$, $s_2 = 2$, $s_3 = 1$

$n = 4$, $p_1 = 3$, $p_2 = 8$, $p_3 = 8$, $p_4 = 10$

optimal schedule obtained by Bruno-Gonzalez algorithm:

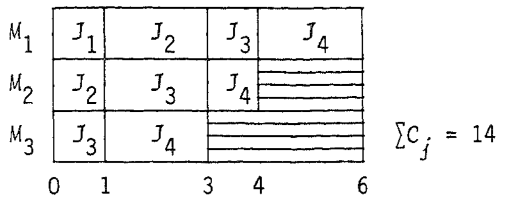


Figure 5 - Example with m uniform machines, all $r_j = 0$, $\sum C_j$ criterion.

In contrast to the algorithm described in Section 2, the resulting algorithm has the property that the remaining execution requirements passed on to the next interval will be as *unevenly* distributed as possible. Unfortunately, it may produce non-optimal schedules, as is illustrated in Figure 6. The example shows in fact that no on-line algorithm will be able to minimize $\sum C_j$ even on two identical machines.

For the case of a single machine, it has been pointed out in Section 1 that when all release dates are equal $\sum w_jC_j$ is minimized in $O(n \log n)$ time by scheduling the jobs in order of nonincreasing ratios w_j / p_j [19]. Again, an obvious extension to the case of arbitrary release dates is to apply the ratio rule at each release date to the remaining execution requirements of the available jobs. This on-line algorithm yields an optimal schedule when the weights are equal or agreeable [1]. Surprisingly [1,p.82], the problem is *NP*-hard when the weights are arbitrary, as will be shown below.

This result will be obtained by a reduction from the following *NP*-complete problem [4]:

PARTITION: Given a set $T = \{1,\ldots,t\}$ and positive integers $a_1,\ldots,a_t,b$ with $\sum_{j \in T} a_j = 2b$, does there exist a subset $S \subset T$ such that $\sum_{j \in S} a_j = b$?

Given any instance of PARTITION, we define a corresponding instance of the problem of minimizing $\sum w_jC_j$ on a single machine subject to arbitrary release dates as follows:

$$n = t + 1;$$

$$r_j = 0,\ p_j = w_j = a_j \quad (j \in T);$$

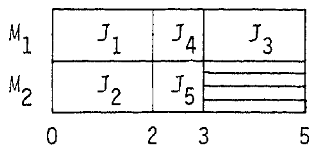
$$r_n = b,\ p_n = 1, w_n = 2.$$

$n = 5$, $r_1 = r_2 = r_3 = 0$, $r_4 = r_5 = r$

$p_1 = p_2 = p_3 = 2$, $p_4 = p_5 = 1$

(a) $r = 2$

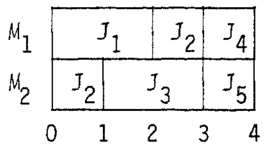
optimal schedule obtained by extended Bruno-Gonzalez algorithm:



$\sum C_j = 15$

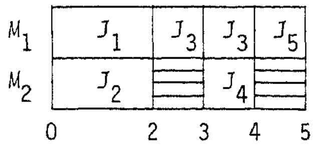
(b) $r = 3$

optimal schedule:



$\sum C_j = 16$

non-optimal schedule obtained by extended Bruno-Gonzalez algorithm:



$\sum C_j = 17$

Figure 6 - Example with two identical machines, $\sum C_j$ criterion.

We claim that PARTITION has a solution if and only if there exists a schedule with value $\sum w_jC_j \leq y$, where

$$y = \sum_{1 \leq j \leq k \leq t} a_j a_k + 3b + 2.$$

With respect to $\{J_j | j \in T\}$, any nonpreemptive schedule without machine idle time is optimal and has value $\sum_{1 \leq j \leq k \leq t} a_j a_k$. Inserting the unit-time job J_n in a schedule for $\{J_j | j \in T\}$ increases the contribution to $\sum w_jC_j$ of the latter set by the total weight of all jobs completed after J_n. Let us denote the index set of all jobs completed before J_n by S, the length of the interval from b until J_n starts by c ($c \geq 0$), and the length of the interval from the last completion before J_n until J_n starts by d ($d \geq 0$). We then have for any schedule that

$$C_n = b + c + 1,$$

$$\sum_{j \in S} w_j = b + c - d,$$

$$\sum w_jC_j = \sum_{1\le j\le k\le t} a_ja_k + 2b - \sum_{j\in S} w_j + 2C_n = y + c + d$$

(cf. Figure 7). It follows that there exists a schedule with value y if and only if PARTITION has a solution.

Since PARTITION can be solved in $O(tb)$ time, the above reduction does not exclude the existence of a similar *pseudopolynomial* algorithm [4] for the single machine problem. However, the latter problem is NP-hard even with respect to a *unary* encoding [12] (NP-hard in the *strong* sense [4]), which implies that it cannot be solved in pseudopolynomial time unless $P = NP$.

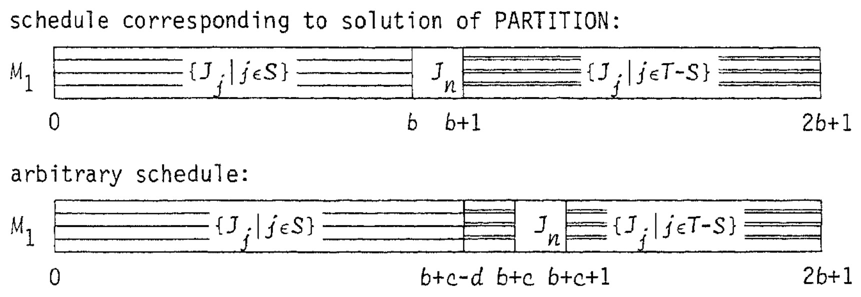


Figure 7 - Reduction from PARTITION to single machine problem, $\sum w_jC_j$ criterion.

This stronger result can be obtained by a reduction from the following unary NP-complete problem [4]:

3-PARTITION: Given a set $T = \{1,\ldots,3t\}$ and positive integers $a_1,\ldots,a_{3t},b$ with $\frac{1}{4}b < a_j < \frac{1}{2}b\,(j\in T)$ and $\sum_{j\in T} a_j = tb$, do there exist t pairwise disjoint subsets $S_i \subset T$ such that $\sum_{j\in S_i} a_j = b$ for $i = 1,\ldots,t$?

The reduction is as follows:

$$n = 4t - 1;$$

$$r_j = 0,\ p_j = w_j = a_j \quad (j\in T);$$

$$r_j = (j-3t)(b+1)-1,\ p_j = 1,\ w_j = 2 \quad (j = 3t+1,\ldots,4t-1);$$

$$y = \sum_{1\le j\le k\le 3t} a_ja_k + (t-1)t(\tfrac{3}{2}b+1).$$

The equivalence proof is left to the reader.

5. Concluding Remarks

The major open problem in the area of preemptive scheduling of uniform machines subject to release dates involves the minimization of $\sum C_j$. It has been pointed out that this problem cannot be solved by an on-line algorithm. We suspect that it cannot be proved *NP*-hard either and conjecture that it is solvable in polynomial time.

An important generalization of the models considered in this paper is the addition of *precedence constraints* between the jobs. It turns out that a number of results for the nonpreemptive scheduling of unit-time jobs subject to precedence constraints can be extended to the preemptive scheduling of jobs with arbitrary processing requirements. For example, polynomial-time algorithms have been obtained for the minimization of $C_{\max}$ on an arbitrary number of identical machines subject to release dates and outtree constraints, and for the minimization of $L_{\max}$ on two uniform machines subject to release dates and general precedence constraints. Also some *NP*-hardness proofs carry through, e.g., for the above $C_{\max}$ problem with intree rather than outtree constraints. The reader is referred to [10] for further details.

Another challenge is to investigate the *stochastic* counterparts of these models, in which the job parameters are random variables and an expected objective value is to be minimized. Initial results for such models are reported in [20].

Acknowledgments

The referee's comments led to an improvement of our running time analysis in Section 3. This research was partially supported by NSF Grants MCS76-17605 and MCS78-20054 and by NATO Special Research Grant 9.2.02 (SRG. 7).

References

[1] K.R. Baker (1974) *Introduction to Sequencing and Scheduling*. Wiley, New York.

[2] J. Bruno, E.G. Coffman, Jr., R. Sethi (1974) Scheduling independent tasks to reduce mean finishing time. *Comm. ACM* 17, 382-387.

[3] J. Bruno, T. Gonzalez (1976) Scheduling independent tasks with release dates and due dates on parallel machines. Technical Report 213, Computer Science Department, Pennsylvania State University.

[4] M.R. Garey, D.S. Johnson (1979) *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

[5] T. Gonzalez (1977) Optimal mean finish time preemptive schedules. Technical Report 220, Computer Science Department, Pennsylvania State University.

[6] T. Gonzalez, S. Sahni (1978) Preemptive scheduling of uniform processor systems. *J. Assoc. Comput. Mach.* 25, 92-101.

[7] W.A. Horn (1974) Some simple scheduling algorithms. *Naval Res. Logist. Quart.* 21, 177-185.

[8] E.C. Horvath, S. Lam, R. Sethi (1977) A level algorithm for preemptive scheduling. *J. Assoc. Comput. Mach.* 24, 32-43.

[9] A.V. Karzanov (1974) Determining the maximal flow in a network by the method of preflows. *Soviet Math. Dokl.* 15, 434-437.

[10] E.L. Lawler (1982) Preemptive scheduling of precedence-constrained jobs on parallel machines. In: M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan (eds.) (1982) *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht, 101-123.

[11] E.L. Lawler, C.U. Martel (1982) Computing maximal "polymatroidal" network flows. *Math. Oper. Res.* 7, 334-347.

[12] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker (1977) Complexity of machine scheduling problems. *Ann. Discrete Math.* 1, 343-362.

[13] C. Martel (1982) Preemptive scheduling with release times, deadlines, and due times. *J. Assoc. Comput. Mach.* 29, 812-829.

[14] C. Martel (1982) Scheduling uniform machines with release times, deadlines and due times. In: M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan (eds.) (1982) *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht, 89-99.

[15] R. McNaughton (1959) Scheduling with deadlines and loss functions. *Management Sci.* 6, 1-12.

[16] S. Sahni (1979) Preemptive scheduling with due dates. *Oper. Res.* 27, 925-934.

[17] S. Sahni, Y. Cho (1979) Nearly on line scheduling of a uniform processor system with release times. *SIAM J. Comput.* 8, 275-285.

[18] S. Sahni, Y. Cho (1980) Scheduling independent tasks with due times on a uniform processor system. *J. Assoc. Comput. Mach.* 27, 550-563.

[19] W.E. Smith (1956) Various optimizers for single-stage production. *Naval Res. Logist. Quart.* 3, 59-66.

[20] G. Weiss (1982) Multiserver stochastic scheduling. In: M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan (eds.) (1982) *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht, 157-179.