stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE
(DEPARTMENT OF OPERATIONS RESEARCH)

BW 99/79 MAART

J. LABETOULLE, E.L. LAWLER,
J.K. LENSTRA, A.H.G. RINNOOY KAN

PREEMPTIVE SCHEDULING OF UNIFORM MACHINES SUBJECT TO RELEASE DATES

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

AMS (MOS) subject classification scheme (1970): 90B35, 68A20

PREEMPTIVE SCHEDULING OF UNIFORM MACHINES SUBJECT TO RELEASE DATES

J. LABETOULLE
IRIA Laboria, Rocquencourt, France

E.L. LAWLER
University of California, Berkeley, U.S.A.

J.K. LENSTRA
Mathematisch Centrum, Amsterdam, The Netherlands

A.H.G. RINNOOY KAN
Erasmus University, Rotterdam, The Netherlands

ABSTRACT

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 the maximum completion time on an arbitrary number of uniform machines. The second algorithm minimizes the maximum lateness with respect to due dates for the jobs on an arbitrary number of identical machines or on two uniform machines. NP-hardness is established for the problem of minimizing the total weighted completion time on a single machine.

KEY WORDS & PHRASES: *preemptive scheduling, uniform machines, identical machines, single machine, release dates, maximum completion time, maximum lateness, total weighted completion time, polynomial-time algorithm, NP-hardness.*

NOTE: This report is not for review. It will appear in the Proceedings of the Summer School in Combinatorial Optimization, Sogesta, Urbino, Italy, July 10-21, 1978.

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 *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* $\sum w_jC_j = \sum_{j=1}^{n} w_jC_j$.

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 [9] derived a closed form expression for the optimum value of C_{max}. Gonzalez and Sahni [7] 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 [16] 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 a polynomial-time nearly on-line algorithm to minimize C_{max}; it can also be used to minimize L_{max} in the case of equal release dates. Sahni and Cho [17] independently developed a similar algorithm for this problem.

In Section 3 we consider the minimization of L_{max} on m identical or two uniform machines. For the case of equal release dates, Horn [8] 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}.

In Section 4 we consider the minimization of $\sum C_j$ and $\sum w_j C_j$. For the case of equal release dates, Bruno and Gonzalez [6] 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 $\sum w_j C_j$ [14]. It follows that $\sum w_j C_j$ is minimized on a single machine by scheduling the jobs in order of nonincreasing ratios w_j/p_j [18], 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 $\sum w_j C_j$.

In Section 5 we conclude by mentioning an important recent contribution to the theory of preemptive scheduling and indicating a major open problem in this area.

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 $r_1 \le \ldots \le r_n$ and $s_1 \ge \ldots \ge s_m$.

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,...,n-1), denote the remaining execution requirement of J_j at r_k by $p_j^{(k)}$ (j = 1,...,k) and renumber the jobs so that $p_1^{(k)} \ge \ldots \ge p_k^{(k)}$. 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 [9] that the minimum time for their completion is given by

$$(1) \quad C_{max}^* = r_n + \max\{\max_{1\le\ell\le m-1}\{\textstyle\sum_{j=1}^{\ell} p_j^{(n)} / \sum_{i=1}^{\ell} s_i\}, \sum_{j=1}^{n} p_j^{(n)} / \sum_{i=1}^{m} s_i\}.$$

The portion of an optimal schedule within any interval R_k can be constructed by applying the Gonzalez-Sahni algorithm [7] 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 $\sum_{j=1}^{\ell} p_j^{(k)}$ (ℓ = 1, ...,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 [16], 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,...,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 $p_1^{(k)} \geq \ldots \geq p_v^{(k)} > 0$. 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,...,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 $s_{m+1} = \ldots = s_v = 0$ 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. As

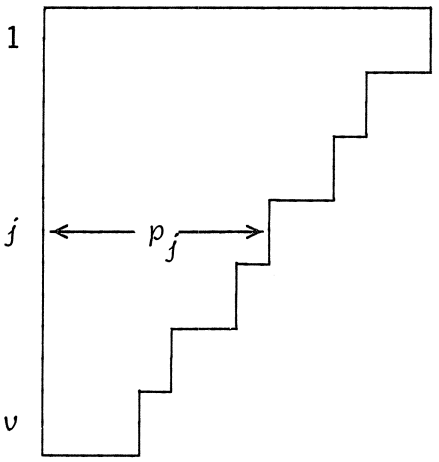


Figure 1 Staircase pattern at r_k.

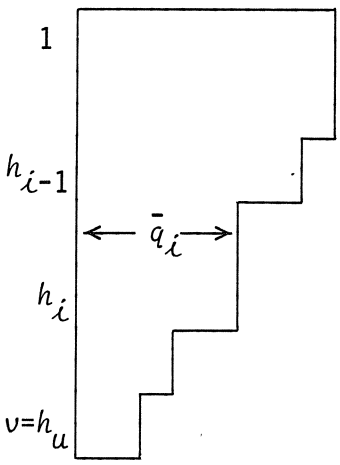


Figure 2 Staircase pattern at r_{k+1}.

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,...,u; $h_0 = 0$; $h_u = v$). A first condition for feasibility is that

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

The staircase $((h_1,\bar{q}_1),\ldots,(h_u,\bar{q}_u))$ will be constructed in such a way that, for i = 1,...,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

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

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

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

A third condition for feasibility is of course that

$$(5)\quad 0 \le q_j \le p_j \quad (j = 1,\ldots,v).$$

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 $\bar{q}_1 > \ldots > \bar{q}_i$, we construct the (i+1)-st tentative step by setting

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

If $\bar{q}_i > \bar{q}_{i+1}$ and $\bar{q}_i \ge 0$, 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 $\bar{q}_i \le \bar{q}_{i+1}$ 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

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

(cf. (4)). As a result, it may now be that $\bar{q}_{i-1} \le \bar{q}_i$ ($\bar{q}_{i-1} < 0$ 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 $\bar{q}_i \leq p_{h_i}$. Subtracting (3) for $\ell = h_i-1$ from (4), we find $\bar{q}_i \leq p_{h_i}-ts_{h_i}$, 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 [7] 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_i machines are exclusively processing the longest h_i 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 ($\ell < m$). 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.

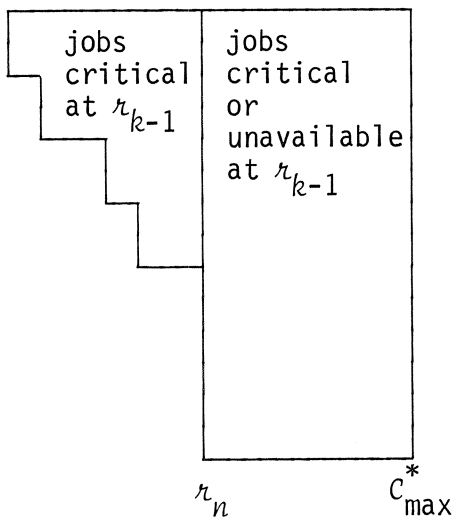


Figure 3 Simultaneous completion of noncritical jobs.

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 care-

fully 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. The noncritical jobs of lowest rank, i.e.,

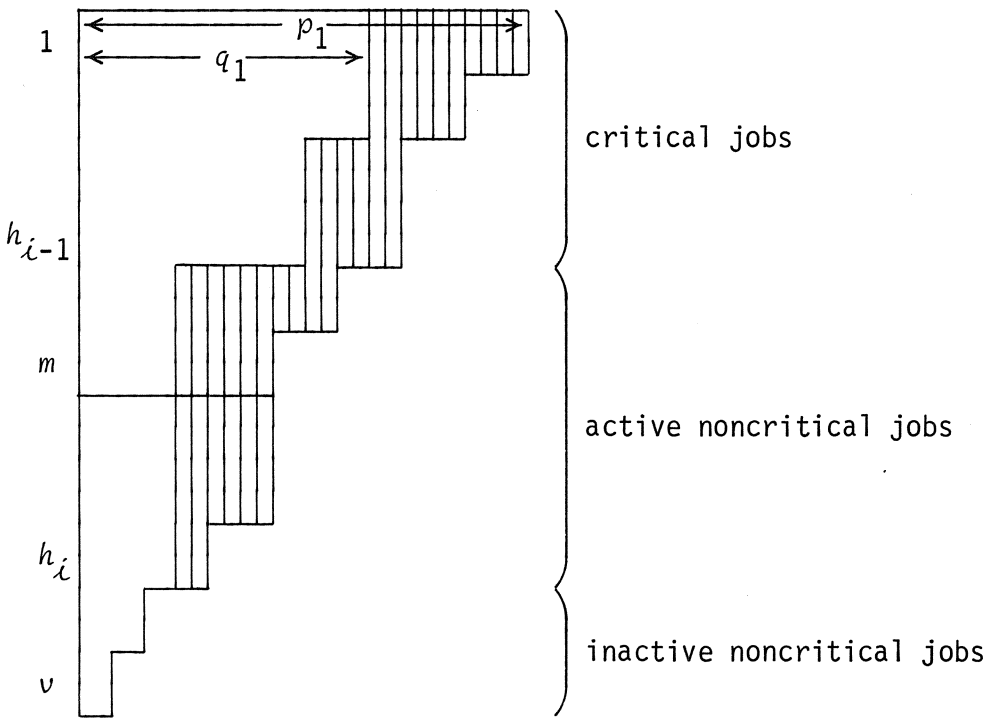


Figure 4 Staircase patterns at r_k and r_{k+1}.

$J_{h_{i-1}+1},\ldots,J_{h_i}$ 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 execution 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 [16] 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 subject to arbitrary release dates for the jobs.

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 $L_{max} \leq y$. This condition is equivalent to the requirement that no job J_j is completed after an *induced deadline* d_j+y. Sahni [15] 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 [8] 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 $\{e_1,\ldots,e_{2n}\}$ $(e_1 \leq \ldots \leq e_{2n})$ 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 $r_j \leq e_k$ and $e_{k+1} \leq d_j+y$. 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 [11]. It should be evident that the trial value y is feasible if and only if the maximum flow value is $P = \sum_{j=1}^{n} p_j$. If the maximum flow value is indeed P, a feasible schedule is easily constructed: for each interval E_k, read off the flows through the arcs (J_j,E_k) and apply McNaughton's schedule-construction procedure [14]. The resulting schedule contains at most $O(n^2)$ preemptions.

Notice that there are certain *critical* trial values of L_{max}. These are the n^2 values y such that $d_j+y = r_k$ for some pair J_j and J_k. The vertex-arc structure of the network remains unchanged for all trial values between two

successive critical values.

We propose to find the optimum value of L_{max} in two phases. In the first phase, the largest infeasible critical value y_0 is determined. A bisection search for y_0 requires the testing of $\log_2 n^2 = O(\log n)$ trial values, or $O(n^3 \log n)$ time overall.

In the second phase, a maximum value flow and a minimum capacity cut are found in the network with capacities induced by the value y_0. Next, a value $y_1 > y_0$ is determined in such a way that the capacity of this cut is increased to exactly P. The procedure is then repeated in the network induced by y_1. This process yields a sequence of increasing trial values y_i. It terminates when the minimum cut capacity is exactly P, i.e., at an iteration z where y_z is the first feasible trial value and therefore the optimum value of L_{max}. We shall show that $z = O(\min\{n^2, \log n + \log p_{max}\})$, where $p_{max} = \max_{1 \le j \le n}\{p_j\}$. Hence the two phases require $O(n^3 \min\{n^2, \log n + \log p_{max}\})$ time overall.

Suppose a minimum cut with capacity $P_0 < P$ is found in the network for y_0. Consider how the capacity of this cut is changed when y_0 is increased by some positive amount δ. The capacity $e_{k+1}-e_k$ of an arc (J_j,E_k)

(a) stays the same if e_k and e_{k+1} are both release dates or both deadlines;

(b) increases by δ if e_k is a release date and e_{k+1} is a deadline;

(c) decreases by δ if e_k is a deadline and e_{k+1} is a release date.

A similar situation holds for the capacities of the arcs (E_k,T), except that they change by $m\delta$ or $-m\delta$ rather than by δ or $-\delta$. It is not hard to establish that the capacity of the cut is increased by $\mu_0\delta$, where μ_0 is an integer *multiplier* with $1 \le \mu_0 \le 2n^2$. Accordingly, we set $\delta = (P-P_0)/\mu_0$, $y_1 = y_0+\delta$, and repeat.

Each cut in the network can be characterized by a pair (μ,P'), where μ is its multiplier and P' its capacity. When y_i is increased to y_{i+1}, the multipliers of cuts do not change, although their capacities, of course, do. To obtain bounds on the number of iterations, suppose that the minimum cut found at iteration i has multiplier μ_i and capacity P_i, and consider the replacement of y_i by y_{i+1}.

Each cut with multiplier $\mu \ge \mu_i$ will have its capacity increased to at least P. Hence $\mu_{i+1} < \mu_i$, unless $z = i+1$. It follows that there can be at most $O(n^2)$ iterations.

Further, each cut with multiplier $\mu < \mu_i$ and capacity $P' \geq P-(P-P_i)\mu/\mu_i$ will have its capacity increased to at least P. It is not hard to verify that at each iteration the number of possible pairs (μ,P') with $P' \leq P$ is decreased by a factor of at least two. It follows that there can be at most $\log_2(n^2P) = O(\log n + \log p_{max})$ iterations. We have arrived at the desired result.

Bruno and Gonzalez [3] showed that essentially the same feasibility test can be employed in the case of two uniform machines. Under the assumption that $s_1 \geq s_2$, each arc (J_j,E_k) has capacity $s_1(e_{k+1}-e_k)$ and each arc (E_k,T) has capacity $(s_1+s_2)(e_{k+1}-e_k)$.

The first phase in our search for the optimum value of L_{max} is as before. In the second phase, the arc capacities change by $\pm s_1\delta$ and $\pm(s_1+s_2)\delta$, instead of $\pm\delta$ and $\pm m\delta$. Since there are now s_1n^2 possible values for the multiplier μ, the first bound on the number of iterations is $O(s_1n^2)$. Since there are s_1n^2P possible pairs (μ,P') with $P' \leq P$, the second bound is $O(\log s_1 + \log n + \log p_{max})$. Hence the algorithm requires $O(n^3\min\{s_1n^2, \log s_1 + \log n + \log p_{max}\})$ time overall.

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* $\sum w_j C_j$.

Let us first assume that all release dates are equal. Bruno and Gonzalez [6] 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.

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

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

optimal schedule obtained by Bruno-Gonzalez algorithm:

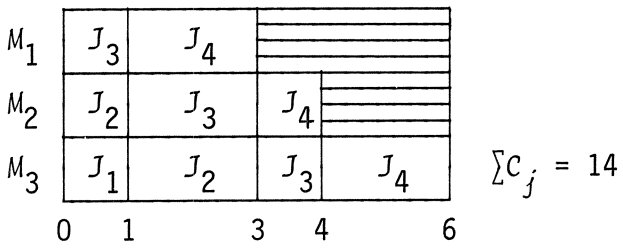


Figure 5 Example with m uniform machines, all $r_j = 0$, $\sum C_j$ criterion.

The Bruno-Gonzalez algorithm not only minimizes $\sum C_j$ but also $\sum_{j=1}^{\ell} C_j$ for $\ell = 1,\ldots,n-1$. Further, it minimizes $\sum w_j C_j$ provided that the weights are *agreeable*, i.e., $p_j < p_k$ implies $w_j \geq w_k$ [6].

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. 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.

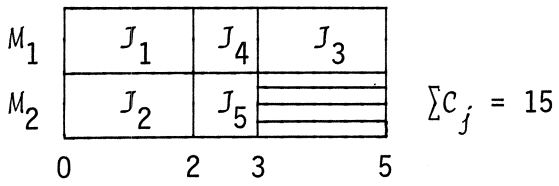
For the case of a single machine, it has been pointed out in Section 1 that when all release dates are equal $\sum w_j C_j$ is minimized in O(n log n) time by scheduling the jobs in order of nonincreasing ratios w_j/p_j [18]. Again,

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

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

(a) $r = 2$

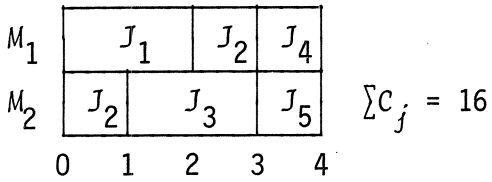
optimal schedule obtained by extended Bruno-Gonzalez algorithm:



$\sum C_j = 15$

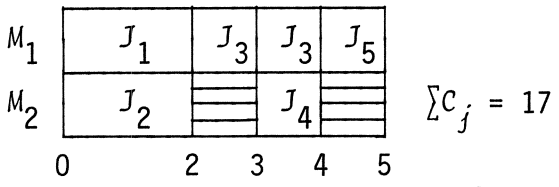
(b) $r = 3$

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.

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 [10]:

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

$$n = t+1;$$
$$r_j = 0,\ p_j = w_j = a_j \quad (j \in T);$$
$$r_n = b,\ p_n = 1,\ w_n = 2.$$

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

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

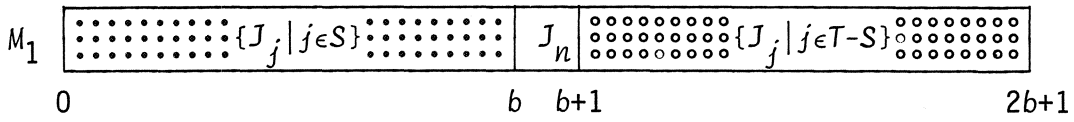
With respect to $\{J_j \mid j \in T\}$, any nonpreemptive schedule without machine idle time is optimal and has value $\sum_{1\le j\le k\le t} a_ja_k$. Inserting the unit-time job J_n in a schedule for $\{J_j \mid j \in T\}$ increases the contribution to $\sum w_jC_j$ of the latter set by the total weight of all jobs completed after J_n. Denoting the index set of all jobs completed before J_n by S, we therefore have for any schedule that

$$C_n = b+c+1 \qquad \text{for some } c \ge 0,$$
$$\sum_{j\in S} w_j = b+c-d \quad \text{for some } d \ge 0,$$
$$\sum w_jC_j = \sum_{1\le j\le k\le t} a_ja_k + 2b - \sum_{j\in S} w_j + 2C_n = y + c + d$$

(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 [5] 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 [5]), which implies that it cannot be solved in pseudopolynomial time unless $P = NP$.

schedule corresponding to solution of PARTITION:



arbitrary schedule:

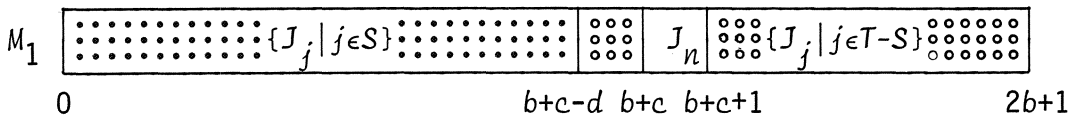


Figure 7 Reduction from PARTITION to single machine problem, $\sum w_j C_j$ criterion.

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

3-PARTITION: Given positive integers $a_1,\ldots,a_{3t},b$ with $\sum_{j=1}^{3t} a_j = tb$, do there exist t pairwise disjoint 3-element subsets $S_i \subset \{1,\ldots,3t\}$ such that $\sum_{j\in S_i} a_j = b$ for $i = 1,\ldots,t$?

The reduction is as follows:

$$n = 4t-1;$$

$$r_j = 0,\ p_j = w_j = a_j \qquad (j = 1,\ldots,3t);$$

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

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

The equivalence proof is left to the reader.

5. CONCLUDING REMARKS

An important recent contribution to the theory of preemptive scheduling of uniform machines subject to arbitrary release dates is the development of a polynomial-time algorithm to minimize L_{max} by Martel [13]. His method proceeds along the same lines as our algorithm for the case of identical machines described in Section 3 and is based on a "generalized" network flow model.

The remaining major open problem in this area 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.

ACKNOWLEDGMENTS

This research was partially supported by NSF Grant MCS76-17605 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 (1975) Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.* 4,397-411.
5. M.R. GAREY, D.S. JOHNSON (1978) "Strong" NP-completeness results: motivation, examples and implications. *J. Assoc. Comput. Mach.* 25,499-508.
6. T. GONZALEZ (1977) Optimal mean finish time preemptive schedules. Technical Report 220, Computer Science Department, Pennsylvania State University.
7. T. GONZALEZ, S. SAHNI (1978) Preemptive scheduling of uniform processor systems. *J. Assoc. Comput. Mach.* 25,92-101.
8. W.A. HORN (1974) Some simple scheduling algorithms. *Naval Res. Logist. Quart.* 21,177-185.
9. E.C. HORVATH, S. LAM, R. SETHI (1977) A level algorithm for preemptive scheduling. *J. Assoc. Comput. Mach.* 24,32-43.
10. R.M. KARP (1975) On the computational complexity of combinatorial problems. *Networks* 5,45-68.
11. A.V. KARZANOV (1974) Determining the maximal flow in a network by the method of preflows. *Soviet Math. Dokl.* 15,434-437.
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 (1979) Personal communication.
14. R. McNAUGHTON (1959) Scheduling with deadlines and loss functions. *Management Sci.* 6,1-12.
15. S. SAHNI (1977) Preemptive scheduling with due dates. Technical Report 77-4, Department of Computer Science, University of Minnesota, Minneapolis.
16. S. SAHNI, Y. CHO (1979) Scheduling independent tasks with due times

on a uniform processor system. *J. Assoc. Comput. Mach.*, to appear.

17. S. SAHNI, Y. CHO (1979) Nearly on line scheduling of a uniform processor system with release times. *SIAM J. Comput.*, to appear.

18. W.E. SMITH (1956) Various optimizers for single-stage production. *Naval Res. Logist. Quart.* 3,59-66.