stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE
(DEPARTMENT OF OPERATIONS RESEARCH)

BW 132/81 NOVEMBER

E.L. LAWLER

PREEMPTIVE SCHEDULING OF PRECEDENCE-CONSTRAINED JOBS ON PARALLEL MACHINES

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, 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.).

1980 Mathematics Subject Classification: 90B35, 68C25

PREEMPTIVE SCHEDULING OF PRECEDENCE-CONSTRAINED JOBS ON PARALLEL MACHINES

E.L. LAWLER

Computer Science Division, University of California, Berkeley, CA 94720, USA

ABSTRACT

Polynomial time-bounded algorithms are presented for solving three problems involving the preemptive scheduling of precedence-constrained jobs on parallel machines: the "intree problem", the "two-machine problem with equal release dates", and the "general two-machine problem". These problems are preemptive counterparts of problems involving the nonpreemptive scheduling of unit-time jobs previously solved by Brucker, Garey and Johnson and by Garey and Johnson. The algorithms and proofs (and the running times of the algorithms) closely parallel those presented in their papers. These results improve on previous results in preemptive scheduling and also suggest a close relationship between preemptive scheduling problems and problems in nonpreemptive scheduling of unit-time jobs.

KEY WORDS & PHRASES: *preemptive scheduling, parallel machines, precedence constraints, polynomial time algorithm.*

NOTE: This report will appear in *Deterministic and Stochastic Scheduling*, edited by M.A.H. Dempster, J.K. Lenstra and A.H.G. Rinnooy Kan, to be published by Reidel, Dordrecht, in 1982.

1. INTRODUCTION

In this paper we present polynomial time-bounded algorithms for solving three problems involving the preemptive scheduling of precedence-constrained jobs on parallel machines. These three problems, which we call the "intree problem", the "two-machine problem with equal release dates", and the "general two-machine problem", are preemptive counterparts of problems involving the nonpreemptive scheduling of unit-time jobs previously solved by Brucker, Garey and Johnson [1] and by Garey and Johnson [2,3]. The algorithms and proofs we present closely parallel those given in their papers, and the running times of the algorithms are nearly comparable. Problems involving nonpreemptive scheduling of unit-time jobs are sometimes formulated as approximations of preemptive scheduling

problems. The results presented in this paper suggest that there is an even closer algorithmic relation between preemptive scheduling and nonpreemptive scheduling of unit-time jobs than had previously been appreciated.

2. PROBLEM DEFINITION

We shall define a general scheduling problem and then indicate how each of the three problems dealt with in this paper is a specialization of this problem.

There are n *jobs* to be scheduled for processing. For each job j, $j = 1,2,\ldots,n$, there are specified a *processing requirement* $p_j > 0$, a *release date* $r_j \geq 0$, prior to which the job is unavailable for processing, and a *due date* $d_j \geq 0$.

The jobs are to be scheduled subject to *precedence constraints* "$\rightarrow$" in the form of a partial order induced by a given acyclic digraph on nodes $j = 1,2,\ldots,n$. If $i \rightarrow j$ then job i must be completed before the processing of job j is begun.

The jobs are to be scheduled on m parallel *machines*. At least m-1 of the machines are identical. One machine is permitted to have the same speed or a strictly slower speed than the others. More specifically let s_i denote the *speed* of machine i, $i = 1,2,\ldots,m$, and assume that $s_1 = s_2 = \ldots = s_{m-1} = 1$, $s_m = s \leq 1$. The *processing capacity* of machine i in a time interval [t,t'] is equal to $s_i(t'-t)$. In order for a job to be completed, the job must be allocated sufficient processing capacity to satisfy its processing requirement.

We make the usual assumptions which apply to the scheduling of parallel machines. A machine can process at most one job at a time and a job can be processed by at most one machine at a time. The schedules we consider are *preemptive*, in that processing of a job can be interrupted at any time and processing resumed at the same time on another machine or at a later time on any machine. There is no penalty for such an interruption or "preemption".

A schedule is *feasible* if no job is processed prior to its release date, if all jobs are completed, and if precedence constraints are observed. A feasible schedule *meets all due dates* if each job is completed no later than its due date.

If in a given feasible schedule the *completion time* of a job j is C_j, then its *lateness* with respect to its specified due date d_j is

$$L_j = C_j - d_j.$$

Our objective is to find a schedule which minimizes

$$L_{max} = \max_j\{L_j\}.$$

(Note that there exists a schedule which meets all due dates if

and only if there exists a schedule for which $L_{max} \leq 0$.)

We now indicate the three special cases of the general problem which are dealt with in this paper.

The Intree Problem: All release dates are zero and the precedence constraints are in the form of a forest of intrees. That is, each job has at most one immediate successor. In the notation of [5], this problem is essentially $P|pmtn,intree|L_{max}$.

The Two-Machine Problem with Equal Release Dates: All release dates are zero and there are exactly two machines. This problem is denoted $Q2|pmtn,prec|L_{max}$.

The General Two-Machine Problem: There are exactly two machines. This is $Q2|pmtn,prec,r_j|L_{max}$.

It should be noted that, by symmetry of release dates and due dates, other specifications are equivalent to the first and second problems above. The "outtree problem" with arbitrary release dates and equal due dates (which may be assumed to be zero) is equivalent to the intree problem. Similarly, the "two-machine problem with equal due dates" (and arbitrary release dates) is equivalent to the two-machine problem with equal release dates. Minimizing maximum lateness for each of these symmetric problems is equivalent to minimizing

$$C_{max} = \max_j\{C_j\}.$$

(This is the same as minimizing "makespan".) The algorithms presented in this paper apply equally well to these symmetric problems, with certain obvious modifications.

3. PREVIOUS RESULTS

A more specialized version of the intree problem was considered by Muntz and Coffman [11] and by Gonzalez and Johnson [4]. In this version, release dates are equal, due dates are equal and the objective is to minimize makespan, *i.e.* the problem is $P|pmtn,tree|C_{max}$. The Muntz-Coffman algorithm requires $O(n^2)$ time and the Gonzalez-Johnson algorithm requires O(n log m) time. Our procedure solves the more general problem of $P|pmtn,tree|L_{max}$ in $O(n^2)$ time. (In both [11] and [4] it is also assumed that the m machines are strictly identical.)

Muntz and Coffman [10] also solved a more specialized version of the two-machine problem in which release dates are equal, due dates are equal and in which the two machines are identical, *i.e.* $P2|pmtn,prec|C_{max}$. Horvath, Lam and Sethi [6] extended the Muntz-Coffman algorithm to allow the two machines to have different speeds, *i.e.* to $Q2|pmtn,prec|C_{max}$. The running time required for

both the Muntz-Coffman and the Horvath-Lam-Sethi algorithms is $O(n^2)$, which is the same as we require for $Q2|pmtn,prec|L_{max}$. (This time bound does not take into account the time required to obtain the transitive closure of the precedence constraints.)

It follows that all three problems dealt with in this paper are strictly more general than those for which polynomial-bounded algorithms have previously been obtained. However, for the more special case of the intree problem considered by Gonzalez and Johnson, the time bound for their algorithm is preferable.

The problems solved by Brucker, Garey and Johnson [1] and by Garey and Johnson [2,3] differ from our problems in that jobs with unit processing times ($p_j = 1$ for all j) are to be nonpreemptively scheduled on strictly identical machines. Our algorithms, theorems and proofs are modelled closely after those in [1,2,3], and we achieve fairly similar running time bounds: For $P|intree,p_j=1|L_{max}$ a time bound of O(n log n) is achieved in [1]. (This can be reduced to O(n) time, as shown by Monma [9]). For $P|pmtn,intree|L_{max}$ we require $O(n^2)$ time. In [2], $Q2|prec,p_j=1|L_{max}$ is solved in $O(n^2)$ time, the same as we require for $Q2|pmtn,prec|L_{max}$. (Time for transitive closure not included.) In [3], $Q2|prec,p_j=1,r_j,d_j|-$ is solved in $O(n^3)$ time, the same as we require for $Q2|pmtn,prec,r_j,d_j|-$. However, Garey and Johnson require only $O(n^3 \log n)$ time for $Q2|prec,p_j=1,r_j|L_{max}$, whereas the best time we have achieved for $Q2|pmtn,prec,r_j|L_{max}$ is $O(n^6)$.

4. PLAN OF ATTACK

In the next section we shall describe a priority scheduling algorithm which can be applied to any instance of the three problems we propose to deal with. This priority scheduling algorithm plays the same role as, but is necessarily more complex than, the list scheduling procedure employed for nonpreemptive scheduling of unit-time jobs in [1,2,3].

In general, there is no assurance that the priority scheduling algorithm yields an optimal schedule, or even finds a schedule in which all jobs meet their due dates, if such a schedule exists. Hence we consider special conditions under which the algorithm yields such a schedule. For the problems $P|pmtn,intree|L_{max}$, $Q2|pmtn,prec|L_{max}$, and $Q2|pmtn,prec,r_j|L_{max}$ we obtain successively stronger "consistency" conditions on the problem data and show that if these conditions are met, then the priority scheduling algorithm finds an optimal schedule (or, in the case of the last problem, one in which all due dates are met).

We than describe procedures for modifying due dates so that the appropriate consistency conditions are satisfied. We prove that the priority scheduling algorithm finds a schedule meeting the modified due dates if and only if there exists such a schedule with respect to the original due dates.

The general procedure for solving each of the three scheduling

problems is then:
(1) Compute the transitive closure of the precedence constraints. (Unnecessary for $P|pmtn,intree|L_{max}$.)
(2) Modify the due dates. (More than one application of the due date modification procedure required for $Q2|pmtn,r_j|L_{max}$.)
(3) Apply the priority scheduling algorithm to determine a sequence of time intervals and the amount of each job to be processed in each interval.
(4) Construct an optimal schedule from the output of the priority scheduling algorithm.

5. THE PRIORITY SCHEDULING ALGORITHM

The priority scheduling algorithm schedules jobs in successive time intervals. Having scheduled intervals $[t_1,t_2],\ldots,[t_{k-1},t_k]$, the algorithm determines a schedule for the next interval $[t_k,t_{k+1}]$ from data for the jobs available for scheduling at t_k. A job j is said to be *available* at time t_k if:
(1) $r_j \leq t_k$,
(2) the processing of each of the predecessors of j (with respect to precedence constraints $\rightarrow$) has been completed, and
(3) there is a nonzero amount of processing $p_j^{(k)} > 0$ remaining to be done on j.

The principal differences between our priority scheduling algorithm for preemptive scheduling and the list scheduling algorithm for nonpreemptive scheduling of unit-time jobs in [1,2,3] are:
(1) Priorities are assigned to jobs according to the values $b_j^{(k)} = d_j - p_j^{(k)}$ (where the smallest value has highest priority). In the case of unit-time jobs, priorities are determined by due dates d_j.
(2) Priorities are dynamic. That is, the priority of a job relative to other jobs is not necessarily known in advance of the time the job becomes available. (However, relative priorities of jobs do not change, as long as they remain available.) In the case of unit-time jobs, priorities are static.
(3) The time intervals $[t_k,t_{k+1}]$ vary in length. In the case of unit-time jobs, each interval has unit length.
(4) Typically only a fraction of the total processing required for a job is done in a single interval. In the case of unit-time jobs, the processing of a job is "all or none".

Suppose without loss of generality that there are n jobs available at t_k and that these jobs are ordered in nondecreasing order of priority, *i.e.*

$$b_1^{(k)} \leq b_2^{(k)} \leq \ldots \leq b_n^{(k)}.$$

(Recall $b_j^{(k)} = d_j - p_j^{(k)}$, where $p_j^{(k)} > 0$ is the amount of processing remaining to be done on job j.) Let $x_j^{(k)}$ be the amount of processing of job j to be done in the interval $[t_k,t_{k+1}]$, where

$\Delta = t_{k+1}-t_k$. We propose to determine the values $x_j^{(k)}$ by solving the following optimization problem:

lexicographically maximize $(b_1^{(k+1)}, b_2^{(k+1)}, \ldots, b_n^{(k+1)})$

subject to

(5.1) $\quad b_1^{(k+1)} \le b_2^{(k+1)} \le \ldots \le b_n^{(k+1)}$,

(5.2) $\quad \max_j\{x_j^{(k)}\} \le \Delta$,

(5.3) $\quad \Sigma_j x_j^{(k)} \le (m-1+s)\Delta$,

(5.4) $\quad 0 \le x_j^{(k)} \le p_j^{(k)}$, for all j.

In words, what we propose to do, subject to constraints (5.1)-(5.4), is to process as much as possible of the highest priority job. Then, subject to this amount of processing of the highest priority job, to process as much as possible of the job with second-highest priority, and so forth. Inequalities (5.1) require us to maintain the relative priorities of the jobs which remain uncompleted at time t_{k+1}. Inequalities (5.2) and (5.3) are well-known necessary and sufficient conditions for the processing amounts $x_j^{(k)}$ to be schedulable within an interval of length Δ. Moreover, given values $x_j^{(k)}$, j = 1,2,...,n, satisfying (5.2),(5.3), a feasible schedule for the interval $[t_k,t_{k+1}]$ can be constructed in O(n) time [5]. Inequalities (5.4) simply require us to do no more processing of any job than is required to complete it.

The interval length Δ is chosen to be the minimum of the next release date (if any), *i.e.*

$$\min_j\{r_j | r_j > t_k\},$$

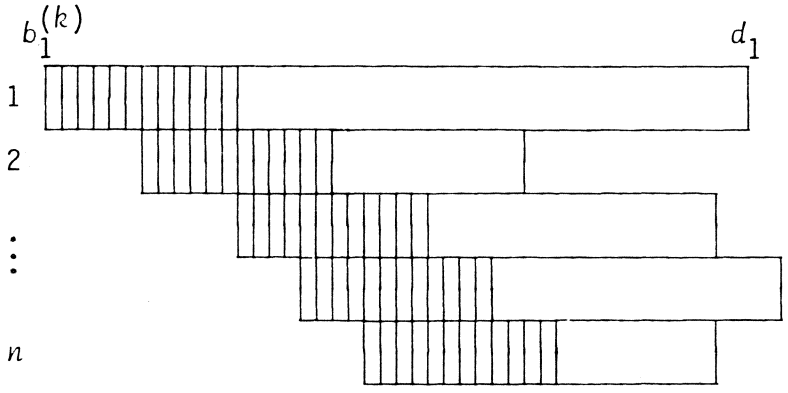
and the smallest value necessary to complete at least one available job, when the $x_j^{(k)}$ values are chosen as above.

Let us now consider the form of a solution to the optimization problem posed above. We begin with the case in which Δ is fixed and the values $p_j^{(k)}$ are suitably large, *e.g.* $p_j^{(k)} \ge \Delta$. If there are m-1 or fewer available jobs, the choice of the $x_j^{(k)}$ values is simple:

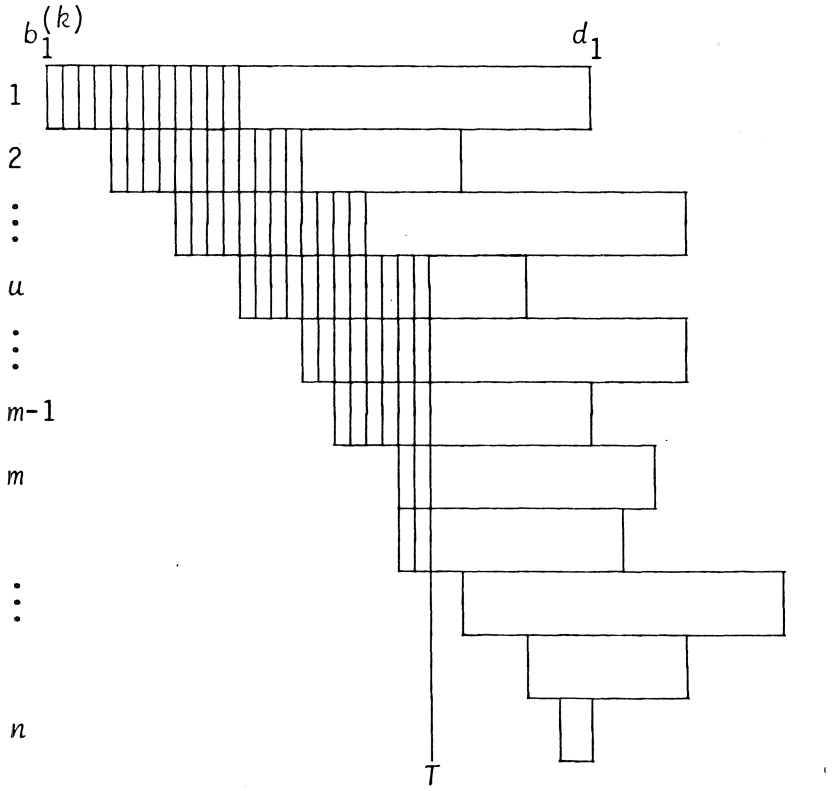
$$x_j^{(k)} = \Delta, \quad \text{for all } j.$$

This solution is indicated in Figure 1(a), in which each job j is represented by a bar extending from $b_j^{(k)}$ to d_j. The shaded portion of bar j represents $x_j^{(k)}$.

Now suppose that there are at least m jobs available at time t_k. We assert that the x_j values are determined by two parameters: an integer u, $1 \le u \le m$, and a real number T. For given u and T,



(a) $n \leq m-1$



(b) $n \geq m$

Figure 1. Scheduling when t_{k+1} determined by release date.

$$x_j^{(k)} = \begin{cases} \Delta, & j < u, \\ \max\{0, T-b_j^{(k)}\}, & j \geq u. \end{cases}$$

A solution of this form is illustrated in Figure 1(b).

For convenience, let v be the index such that $b_v^{(k)} \leq T < b_{v+1}^{(k)}$ (where $b_{n+1}^{(k)} = +\infty$). We assert that u and T provide an optimal solution to our optimization problem if they are such that

$$b_{u-1}^{(k)} + \Delta \leq T, \tag{5.5}$$

$$\sum_{j=u}^{v} (T - b_j^{(k)}) = (m-u+s)\Delta. \tag{5.6}$$

Inequality (5.5) insures that the ordering conditions (5.1) are satisfied. Equation (5.6) implies that the total capacity of machines u,u+1,...,m is completely utilized in processing jobs u,u+1,...,v (while machines 1,2,...,u-1 are completely utilized in processing jobs 1,2,...,u-1). There is thus no idle time on any of the m machines; in order to increase the amount of processing of any job, we would need to decrease the amount of processing of some other job, thereby violating conditions (5.1). We assert that it is intuitively obvious that such a solution is optimal.

The procedure below computes u, v and T for an interval length determined by a fixed value of t_{k+1}, under the assumption that processing requirements are large, *e.g.* $p_j^{(k)} \geq t_{k+1} - t_k$, for all j. The procedure begins with $\Delta = 0$, $u = v = m$ and $T = b_m^{(k)}$. T is then increased in steps, where at each step the increase in T is limited by the amount required before (i) u must be decremented, (ii) v must be incremented, or (iii) the corresponding value of Δ becomes equal to $t_{k+1} - t_k$. (This latter is determined by the total amount of processing P which must be done on jobs u,u+1,...,v and the capacity of machines u,u+1,...,m in a time interval of length $t_{k+1} - t_k$.) There are at most n steps and the procedure requires at most O(n) time.

```
FIXED INTERVAL PROCEDURE
Input:    m,s, specifying machine environment;
          b_j, j = 1,2,...,n, specifying job priorities;
          t_k,t_{k+1}, specifying a fixed interval.
Output:   u,v,T, specifying processing amounts in interval (it is
          assumed that processing requirements are large).
begin procedure
     Δ := 0;
     u := m;
     v := m;
     T := b_m;
     P := 0;
     while Δ < t_{k+1}-t_k
     do   while T = b_{u-1}+Δ
          do   u := u-1;
               P := P+Δ
          od;
          while T = b_{v+1}
          do   v := v+1 [If v = n then let b_{n+1} = +∞.]
          od;
```

T_1 := $((b_{u-1}+\Delta)(m-u+s)-(v-u+1)T)/((m-u+s)-(v-u+1))$;
T_2 := b_{v+1};
T_3 := $T + (t_{k+1}-t_k-\Delta)(m-u+s)/(v-u+1)$;
[T_1,T_2,T_3 are limits on the new size of T, as described in text.]
T' := $\min\{T_1,T_2,T_3\}$;
P := $P + (v-u+1)(T'-T)$;
Δ := $P/(m-u+s)$

od
end procedure.

Now let us consider how to determine Δ in the case that the interval length is to be just large enough so that one or more jobs are completed. Clearly, when Δ is properly determined and u,v,T are found as above, we should have

(5.7) $$b_j+\Delta \le d_j, \quad j = 1,2,\ldots,u-1,$$

(5.8) $$T \le d_j, \quad j = u,u+1,\ldots,m,$$

with equality in at least one case.

Our strategy is as follows. We first carry out a computation very much as in the fixed interval procedure, but with two modifications:
(1) When T is stepped, T is not permitted to violate conditions (5.8). When $T = d_j$, for some $j \ge u$, the computation stops.
(2) When u is decremented, the condition $b_{u-1}+\Delta > d_{u-1}$ is checked. When this is found to hold, the computation stops.

We shall call the modified procedure the "variable interval procedure". (We leave implementation of the modifications to the reader.) At the conclusion of the variable interval procedure we shall have either:
(1) determined u,v,T for the interval $[t_k,t_{k+1}]$, without violating (5.7) or (5.8);
(2) found values u,v,T and $\Delta < t_{k+1}-t_k$ such that (5.7),(5.8) are satisfied, and one of the constraints (5.8) is satisfied with equality;
(3) found values u,v,T and $\Delta < t_{k+1}-t_k$ such that conditions (5.8) are satisfied, but at least one constraint (5.7) is violated.

In the first two cases we are done: the value $t_{k+1} = t_k+\Delta$ found by the procedure is correct. In the third case, the correct value of t_{k+1} is simply

$$t_{k+1} = t_k + \min\{p_j^{(k)} \mid 1 \le j \le u-1\}.$$

Running the procedure one more time with the correct value of t_{k+1} completes the computation.

We are now ready to indicate the complete priority scheduling procedure.

```
PRIORITY SCHEDULING PROCEDURE
Input:    m,s, specifying machine environment;
          triples (p_j,r_j,d_j), j = 1,2,...,n, specifying jobs;
          an acyclic digraph specifying precedence constraints.
Output:   intervals [t_k,t_{k+1}], k = 1,2,...,N-1, where N ≤ 2n-1;
          values x_j^(k) ≥ 0, indicating the amount of processing of
          job j to be performed in interval [t_k,t_{k+1}].
begin procedure
     compute the in-degree of each job;
     create a priority queue Q (by release date) containing all
     jobs with in-degree zero;
     create a list A of available jobs in nondecreasing order of
     b_j (job priority) values; initially A is empty;
     k := 0;
     while Q is nonempty
     do   k := k+1;
          t_k := min{r_j | j ∈ Q};
          add to A and remove from Q all jobs in Q such that
          r_j = t_k;
          while A is nonempty
          do   t_{k+1} := min{r_j | j ∈ Q};
               apply the variable interval procedure to the jobs
               in A; if the resulting values of Δ,u,v,T violate
               conditions (5.7), rerun the procedure with t_{k+1} =
               t_k + min{p_j^(k) | 1 ≤ j ≤ u-1}; [This determines u,v,T,
               and x_j^(k), for all j.]
               remove from A all jobs completed in the interval
               [t_k,t_{k+1}]; for each such job decrement the in-degree
               of each of its successors and place in Q each job
               whose in-degree is reduced to zero;
               add to A and remove from Q all jobs in Q such that
               r_j ≤ t_{k+1};
               k := k+1
          od
     od
end procedure.
```

We assert that the priority scheduling procedure can be implemented to run in $O(n^2)$ time overall. As we commented above, the output of the procedure can also be transformed into an actual schedule in $O(n^2)$ time.

6. A USEFUL LEMMA

We wish to consider the effect of applying the priority scheduling procedure to jobs whose due dates satisfy the *consistency conditions*

(6.1) $\quad d_j \le b_k$ whenever $j \to k$

and

(6.2) $\quad r_j \le b_j$ for all j.

The following is a self-evident property of the priority scheduling procedure.

PROPOSITION. *The procedure fails to schedule all jobs to meet due dates if and only if at some interval* $[t_{\ell-1},t_\ell]$ *there is a job* j *(available at* $t_{\ell-1}$*) for which* $b_j^{(\ell)} < t_\ell$.

LEMMA 6.1. *Suppose the priority scheduling algorithm is applied to a set of jobs for which* (6.1) *and* (6.2) *hold. Let* $[t_{\ell-1},t_\ell]$ *be the earliest interval at which there is a job* j *with* $b_j^{(\ell)} < t_\ell$. *Then* $b_j^{(\ell)} = T < t_\ell$ *for all jobs scheduled for processing in the interval and there is no idle time on any of the machines in the interval.*

Proof. Assume the jobs are numbered so that $b_1^{(\ell-1)} \le b_2^{(\ell-1)} \le \ldots \le b_n^{(\ell-1)}$, $b_1^{(\ell)} \le b_2^{(\ell)} \le \ldots \le b_n^{(\ell)}$. Then $b_1^{(\ell)} < t_\ell$. If $n \ge m$ and $u = 1$ the result follows immediately.

So let us suppose that either $n \le m-1$ or $n \ge m$ and $u \ge 2$. Then we would have $b_1^{(\ell)} = b_1^{(\ell-1)}+t_\ell-t_{\ell-1}$ and $b_1^{(\ell-1)} < t_{\ell-1}$. If $\ell \ge 2$ and job 1 was available at $t_{\ell-2}$, then $[t_{\ell-1},t_\ell]$ would not be an earliest interval at which there is a job j with $b_j^{(\ell)} < t_\ell$. If $\ell = 1$ or if job 1 was not available at $t_{\ell-2}$, then job 1 first became available at $t_{\ell-1}$. This implies that either $r_1 = t_{\ell-1} > b_1^{(\ell-1)} = b_1$, in contradiction to (6.2), or that some job j, with $j \to 1$, was completed in the interval $[t_{\ell-2},t_{\ell-1}]$. But if this latter were the case, then, by (6.1), $b_j^{(\ell-1)} = d_j \le b_1 = b_1^{(\ell-1)} < t_{\ell-1}$ and $[t_{\ell-1},t_\ell]$ would not be the earliest interval with a $b_j^{(\ell)} < t$. It follows that $n \ge m$ and $u = 1$, and the lemma is proved. □

7. THE INTREE PROBLEM

We now turn to the in-tree problem, $P|pmtn,intree|L_{max}$. Each job has at most one immediate successor and $r_j = 0$ for all jobs.

We first consider the question of whether or not there exists a schedule meeting all due dates. If a job k is to meet its due date d_k, then the latest possible time at which its processing can begin is $b_k = d_k-p_k$. If $j \to k$, job j must be completed by time b_k, else k will be late. Thus not only must job j be completed by time d_j, it must be completed by time b_k. Since the processing of job j must obey both constraints, we can replace due date d_j by a new due date $d_j' = \min\{d_j,b_k\}$ without changing the problem in any essential way. A schedule meets all due dates in the new problem if and only if it meets all due dates in the original problem. This type of due date modification can be applied repeatedly until we finally obtain an equivalent problem having *modified due dates* d_j' satisfying the consistency conditions

(7.1) $\quad d'_j \le b'_k = d'_k - p_k$, whenever $j \to k$.

The following simple algorithm constructs such a set of modified due dates and can be implemented to run in time O(n).

DUE DATE MODIFICATION PROCEDURE
Input: m,s, specifying machine environment;
ordered pairs (p_j,d_j), $j = 1,2,\ldots,n$, specifying jobs;
an acyclic digraph in which each node has out-degree at most one.
Output: modified due dates d'_j, $j = 1,2,\ldots,n$.
begin procedure
for each job j which has no successor
do $d'_j := d_j$
od;
while there is a job which has not been assigned its modified due date and whose immediate successor k has had its due date modified, select such a job j and
do $d'_j := \min\{d_j, b'_k = d'_k - p_k\}$
od
end procedure.

The arguments above establish the following result.

LEMMA 7.1. *There exists a schedule meeting the original due dates if and only if there exists a schedule meeting the modified due dates.*

THEOREM 7.2. *The schedule obtained by applying the priority scheduling algorithm to the problem with modified due dates meets all original due dates if and only if such a schedule exists.*

Proof. Since $d'_j \le d_j$, for all j, if the schedule obtained by applying the priority scheduling algorithm meets the modified due dates then it meets the original due dates. So suppose the schedule obtained does not meet the modified due dates. We shall show that this implies that there exists no schedule meeting the modified due dates. Then, by Lemma 7.1, there is no schedule meeting the original due dates.

So suppose the schedule does not meet the modified due dates and let us apply Lemma 6.1. Let $[t_{\ell-1},t_\ell]$ be the earliest interval at which there is a job j with $b_j^{(\ell)} = T < t_\ell$. We assert that in each earlier interval $[t_{k-1},t_k]$, $k = 1,2,\ldots,\ell-1$, there is no idle time on any machine and $b_j^{(k)} \le T$ for each job processed in the interval. For suppose this were not the case. Then at least one job j processed in $[t_{\ell-1},t_\ell]$ was unavailable at t_{k-1}, else the algorithm would have scheduled it (in preference to idle time or to a job i with $b_i^{(k)} > T$). But j could only have been unavailable at t_{k-1} because one or more of its predecessors was not yet completed. But for each predecessor i of j, $d'_i \le b'_j$, so necessarily

$b_i^{(k)} \leq T$. No two jobs processed in $[t_{\ell-1},t_\ell]$ have a common predecessor (a property of intree orders), so we have established our assertion.

Let

$$p_j(t) = \begin{cases} p_j, & t \geq d_j', \\ \max\{0,p_j-(d_j'-t)\}, & t \leq d_j'. \end{cases}$$

Then $p_j(t)$ denotes an amount of processing that must be done on job j prior to time t, if its due date is to be met. The above analysis shows that

$$\Sigma_j\, p_j(T) \geq (m+s-1)t_\ell > (m+s-1)T.$$

In other words, there is more processing which must be performed before time T than the m machines have processing capacity. Hence there can be no schedule meeting the modified due dates and the theorem is proved. □

As it turns out, not only have we solved the problem of constructing a schedule meeting all due dates, if such a schedule exists, but we have also solved the problem of minimizing maximum lateness.

THEOREM 7.3. *The schedule obtained by applying the priority scheduling algorithm to the problem with modified due dates minimizes maximum lateness with respect to the original due dates.*

Proof. The problem of minimizing L_{max} is clearly equivalent to the problem of determining the smallest possible value of L such that when the original due dates d_j are replaced by due dates d_j+L, $j = 1,2,\ldots,n$, there exists a schedule meeting the new due dates. But for any such L the modified due dates are simply changed by the same constant L. That is, if modified due dates d_j' are obtained from the original due dates d_j and d_j'' are obtained from due dates d_j+L, then $d_j'' = d_j'+L$, $j = 1,2,\ldots,n$. (This can be proved by straightforward induction.) Moreover, the priority scheduling algorithm yields precisely the same schedule when applied to due dates d_j'' and d_j'. (We also omit proof of this fact, considering it to be obvious.) It follows that the schedule constructed by the algorithm when applied to the problem with modified due dates d_j' minimizes maximum lateness. □

Notice that if all the original due dates are the same, our procedure constructs a schedule which minimizes makespan. However, as we pointed out earlier, the algorithm of Gonzalez and Johnson [4] provides a running time of $O(n \log m)$, whereas ours is $O(n^2)$.

8. THE TWO-MACHINE PROBLEM WITH UNIFORM RELEASE DATES

Now let us consider the two-machine problem with uniform release dates, $Q2|pmtn,prec|L_{max}$.

We first consider the question of whether or not there exists a schedule which meets all due dates. As in the case of the intree problem, we note that due dates can be modified so as to observe condition (6.1). However, additional modifications are necessary, as well as more notation.

As in the proof of Theorem 7.2, let us define

$$p_j(t) = \begin{cases} p_j, & t \geq d_j, \\ \max\{0, p_j-(d_j-t)\}, & t < d_j. \end{cases}$$

In words, $p_j(t)$ is a lower bound on the amount of processing that must be done on job j prior to time t, if job j is to meet its due date. Let S(j) denote the set of all successors (immediate or not) of job j. Then in any schedule in which each successor k of j meets its due date, it must be the case that

$$C_j \leq t - \frac{1}{1+s} \Sigma_{k\in S(j)} p_k(t), \quad \text{for all } t \geq d_j.$$

The reason for this is that the processing capacity of the two machines in the interval $[C_j,t]$ is $(1+s)(t-C_j)$ and this must not be less than the total amount of processing of the successors of j which must be performed prior to t.

This suggests the additional consistency condition

$$d_j \leq t - \frac{1}{1+s} \Sigma_{k\in S(j)} p_k(t), \quad \text{for all } j \text{ and } t \geq d_j. \tag{8.1}$$

Fortunately, in order to modify due dates so that they conform with this consistency condition, it is not necessary to check (8.1) for all values of $t \geq d_j$.

LEMMA 8.0. *If for given* j, *conditions* (8.1) *are satisfied for all*

$$d \in \{d_k | j \to k\},$$

then condition (8.1) *is satisfied for all* $t \geq d_j$.

Proof. Let $t \geq d_j$ and let

$$\bar{d} = \min\{d_k | d_k \geq t,\ j \to k\},$$

$$\underline{d} = \max\{d_k | d_k \leq t,\ j \to k\}.$$

If both sets above are empty, then j has no successors and (8.1) is clearly satisfied. If the first set is empty, but not the second, $\underline{d} = \max\{d_K | j \to k\}$ and it is easy to verify that satisfaction

of (8.1) for $\underline{d}$ implies satisfaction for t. There is a similar argument if the second set is empty. So assume neither set is empty and $\underline{d},\bar{d}$ are both well defined.

If $\Sigma p_k(\underline{d}) \geq \Sigma p_k(t) - (1+s)(t-\underline{d})$ or if $\Sigma p_k(\bar{d}) \geq \Sigma p_k(t) + (1+s)(\bar{d}-t)$, then satisfaction of (8.1) for $\underline{d},\bar{d}$, respectively, implies satisfaction for t. Suppose that $\Sigma p_k(\underline{d}) < \Sigma p_k(t) - (1+s)(t-\underline{d})$ and $\Sigma p_k(\bar{d}) < \Sigma p_k(t) + (1+s)(\bar{d}-t)$. Examination of the definition of $p_j(t)$ reveals that

$$\frac{\Sigma p_k(t) - \Sigma p_k(\underline{d})}{t-\underline{d}} \leq \frac{\Sigma p_k(\bar{d}) - \Sigma p_k(t)}{\bar{d}-t},$$

which yields a contradiction. □

The following algorithm constructs a set of modified due dates satisfying both (6.1) and (8.1). It can be implemented to run in $O(n^2)$ time, provided precedence constraints are already in transitively closed form.

```
DUE DATE MODIFICATION PROCEDURE
Input:    s, 0 < s ≤ 1, specifying speed of second machine;
          ordered pairs (p_j,d_j), j = 1,2,...,m, specifying jobs;
          a transitively closed acyclic digraph specifying prece-
          dence constraints →.
Output:   modified due dates d'_j, j = 1,2,...,m.
begin procedure
     for each job j which has no successors
     do   d'_j := d_j
     od;
     create two ordered lists of jobs whose due dates have been
     modified, one ordered by modified due dates d'_j, the other
     ordered by the values b'_j = d'_j-p_j;
     while there is a job which has not been assigned its modified
     due date and all of whose successors have had their due dates
     modified, select such a job j and
     do   scan the two lists of d'_j and b'_j values, extracting all
          the successors of j;
          merge the two sublists and use the merged list to compute
          the sum Σ_{k∈S(j)} p_k(d'), for each modified due date d'
          assigned to a successor of j; [All such sums can be com-
          puted in O(n) time.]
          d'_j := min{d_j,min_d'{d' - Σp_k(d')/(1+s)}};
          for each successor k of j
          do   d'_j := min{d'_j,b'_k = d'_k-p_k}
          od;
          insert d'_j, b'_j = d'_j-p_j into ordered lists
     od
end procedure.
```

LEMMA 8.1. *There exists a schedule meeting the original due dates if and only if there exists a schedule meeting the modified due dates.*

THEOREM 8.2. *The schedule obtained by applying the priority scheduling algorithm to the problem with modified due dates meets all the original due dates if and only if such a schedule exists.*

Proof. As in the proof of Theorem 7.2, if the schedule obtained by applying the priority scheduling algorithm meets the modified due dates, then it meets the original due dates. So, as before, suppose the schedule obtained does not meet the modified due dates and let us show that this implies that there exists no schedule meeting the modified due dates.

Let $[t_{\ell-1},t_\ell]$ be the first interval in which $b_j^{(\ell)} < t_\ell$ for some job j. By Lemma 6.1, there is no idle time on any machine in this interval and $b_j^{(\ell)} = T < t_\ell$ for all jobs j processed in $[t_{\ell-1},t_\ell]$.

Case 1. In each earlier interval $[t_{k-1},t_k]$ there is no idle time on any machine and $b_j^{(\ell)} \le T$ for each job processed in the interval. Then we have

$$\Sigma_j\, p_j(T) \ge (1+s)t_\ell > (1+s)T,$$

and there can be no schedule meeting the modified due dates.

Case 2. There is an earlier interval $[t_{k-1},t_k]$ in which there is either idle time or a job j is processed with $b_j^{(\ell)} > T$. Let $[t_{k-1},t_k]$ be the latest such interval. It must be the case that the jobs processed in $[t_{\ell-1},t_\ell]$ are unavailable for processing in $[t_{k-1},t_k]$, from which it follows that some job j is processed and completed in $[t_{k-1},t_k]$ which is a predecessor of all such jobs. Let j be this job. By (8.1),

$$d_j' \le T - \frac{1}{1+s}\,\Sigma_{k\in S(j)} p_k(T).$$

But

$$\Sigma_{k\in S(j)} p_k(T) \ge (1+s)(t_\ell - t_k) > (1+s)(T-t_k),$$

which implies $d_j' < t_k$. But $d_j' = b_j^{(k)} \ge t_k$, which yields a contradiction. Hence Case 1 must apply and there can exist no schedule meeting the modified due dates. □

THEOREM 8.3. *The schedule obtained by applying the priority scheduling algorithm to the problem with modified due dates minimizes maximum lateness with respect to the original due dates.*

Proof. Similar to that for Theorem 7.3. □

9. THE GENERAL TWO-MACHINE PROBLEM

Now consider the general two-machine problem, $Q2|pmtn,prec,r_j|L_{max}$: there are arbitrary release dates and precedence constraints, but of course only two machines.

Once again we first consider the question of whether or not there exists a schedule which meets all due dates. Consistency conditions (6.1) and (8.1) apply as in the previous section, but are not strong enough for our purposes. More restrictive conditions can be formulated as follows.

Let $p_j(t)$ be some known lower bound on the amount of processing of job j which must be done prior to time t. A function defining lower bounds of this type will be required to satisfy the conditions

$$(9.0) \qquad p_j(t) \begin{cases} = p_j, & t \geq d_j, \\ \geq \max\{0, p_j(t')-(t'-t)\}, & t \leq t' \leq d_j. \end{cases}$$

For given j,r,t, with $r_j \leq r \leq t$, define

$$P(j,r,t) = \Sigma_{k\in S(j)}\, p_k(t) + \Sigma_{k\notin S(j), r_k \geq r, k\neq j}\, p_k(t).$$

If $P(j,r,t) > (1+s)(t-r)$, the amount of processing to be done prior to time t and after the completion of job j exceeds the total processing capacity of the two machines in the interval $[r,t]$. It follows that if all due dates are to be met we must have

$$C_j \leq t - \frac{1}{1+s} P(j,r,t).$$

Similarly, if $P(j,r,t) \leq (1+s)(t-r)$, $(1+s)(t-r)-P(j,r,t)$ is an upper bound on the amount of processing of job j which can be done in the interval $[r,t]$. It follows that we should have

$$p_j(r) \geq p_j(t)-(1+s)(t-r)+P(j,r,t).$$

Motivated by these observations, we frame the following definition.

Definition. Release dates r_j and due dates d_j, $j = 1,2,\ldots,n$, are *internally consistent* if there exist functions $p_j(t)$ satisfying (9.0), such that for all j,r,t, with $r_j \leq r \leq t$,

$$(9.1) \qquad r_j \geq r_i + p_i \qquad \text{whenever } i \to j,$$

$$(9.2) \qquad d_j \leq t - p_k(t) \qquad \text{whenever } j \to k,$$

$$(9.3) \qquad p_j(r_j) = 0 \qquad \text{for all } j,$$

$$(9.4) \qquad d_j \leq t - \frac{1}{1+s} P(j,r,t) \qquad \text{if } P(j,r,t) > (1+s)(t-r),$$

(9.5) $p_j(r) \geq p_j(t)-(1+s)(t-r)+P(j,r,t)$ if $P(j,r,t) \leq (1+s)(t-r)$.

The following lemma shows that in order to establish internal consistency of due dates, it is unnecessary to verify conditions (9.0)-(9.5) for all j,r,t, with $r_j \leq r \leq t$. It is sufficient to consider only $r \in R$, $t \in RUD$, where $R = \{r_k | k = 1,2,\ldots,n\}$, $D = \{d_k | k = 1,2,\ldots,n\}$.

LEMMA 9.0. *Let* $p_j(t)$ *be functions with domain* RUD. *If these functions satisfy* (9.0)-(9.5) *for* $r \in R$, $t \in RUD$, *then the due dates are internally consistent.*

Proof. Extend the functions by the rule

$$p_j(t) = \begin{cases} p_j, & t \geq d_j, \\ \max\{0, p_j(t')-(t'-t)\}, & t < d_j. \end{cases}$$

The lemma follows by reasoning similar to that used in the proof of Lemma 8.0. □

Lemma 9.0 suggests that it is possible to construct an algorithm for modifying due dates, based on the construction of functions $p_j(t)$ over the domain RUD. We shall employ three loops in this algorithm. The outer loop considers values of $t \in RUD$, in decreasing order. The middle loop considers values of j, in arbitrary order, and the inner loop considers values of $r \in R$ in increasing order.

For a given triple t,j,r, the algorithm compares $P(j,r,t)$ with $(1+s)(t-r)$ and modifies either d_j or $p_j(r)$, in accordance with (9.4),(9.5), as necessary. When the processing of j is completed for a given value of t, the values $p_j(u)$, $u \leq t$, are revised to conform with condition (9.0). The algorithm then halts if $p_j(r_j) > 0$, in violation of (9.3). The due dates of predecessors of j are then modified in accordance with (9.2).

Each value t considered in the outer loop is either a (fixed) release date or a due date which remains unchanged by any further processing. Because of the maintenance of conditions (9.2), if $t = d_j$ then $p_k(t) = 0$ for all successors k of j. Hence if $P(j,r,t) > (1+s)(t-r)$ (which is the only condition under which d_j can be changed), then $P(k,r,t) > (1+s)(t-r)$ for any job k such that $r = r_k$, and $p_k(r_k) > 0$, in violation of (9.3). It follows that at most $2n$ values of t are considered by the outer loop.

We next note that $P(j,r,t)$ is determined only by the values of $p_k(t)$, $k = 1,2,\ldots,n$. Since values of t are processed in decreasing order, the values $p_k(t)$ are not changed by further repetitions of the loop. It follows that if the due date modification algorithm runs to termination without halting because of violation of condition (9.3), the functions $p_j(t)$ constructed by the algorithm satisfy conditions (9.0)-(9.5). By Lemma 9.0, the modified

due dates are then internally consistent.

DUE DATE MODIFICATION PROCEDURE

Input: m,s, specifying machine environment;
triples (p_j,r_j,d_j), $j = 1,2,\ldots,n$, specifying jobs;
a transitively closed acyclic digraph specifying precedence constraints $\rightarrow$.
[We assume that

$$r_j \geq \max\{r_i+p_i \mid i \rightarrow j\},$$

$$d_j \leq \min\{d_i-p_i \mid j \rightarrow i\};$$

if this is not so, release and due dates should be modified accordingly.]

Output: modified due dates d_j, $j = 1,2,\ldots,n$, or indication that release dates and due dates are not internally consistent.

```
begin procedure
     t := +∞;
     R := {r_j | j = 1,2,...,n};
     D := {d_j | j = 1,2,...,n};
     loop if all t ∈ RUD have been considered
          then stop
          fi;  [Modified due dates are internally consistent.]
          set t to the largest value in RUD less than current t;
          for all j with d_j ≤ t
          do   r := r_j;
               P(j,r,t) := Σ_{k∈S(j)} p_k(t) + Σ_{k∉S(j),r_k≥r} p_k(t);
               while there is an r' in R such that r < r' ≤ t, let
               r' be the smallest value in R greater than r and
               do   P(j,r,t) := P(j,r,t) - Σ_{k∉S(j),r≤r_k<r'} p_k(t);
                    r := r';
                    if P(j,r,t) ≤ (1+s)(t-r)
                    then p_j(r) := max{p_j(r),p_j(t)-(1+s)(t-r)+P(j,r,t)}
                    else d_j := min{d_j,t - P(j,r,t)/(1+s)};
                         update D, as necessary
                    fi
               od;
               let r_j = u_1 < u_2 < ... < u_ℓ = t be the different
               values in RUD between r_j and t;
               if p_j(u_i) has not previously been computed
               then p_j(u_i) := 0
               fi;
               for i = ℓ-1,ℓ-2,...,1
               do   p_j(u_i) := max{p_j(u_i),p_j(u_{i+1})-(u_{i+1}-u_i)}
               od;
               if p_j(r_j) > 0
               then stop
               fi;  [Due dates are not internally consistent.]
               t' := min{u ∈ RUD | p_j(u) > 0},
               for each k such that k → j
```

```
                do   d_k := min{d_k, t'-p_j(t')}
                od
          od
     forever
end procedure.
```

Notice that the for loop on j is executed $O(n^2)$ times and that the computation of $P(j,r,t)$ requires only $O(n)$ time. The while loop on r is executed $O(n^2)$ times and the revision of $P(j,r,t)$ requires at most one reference to a given job k, $k \notin S(j)$, for each combination of t and j. Hence only $O(n^3)$ time is required for the revision of $P(j,r,t)$. A similar analysis of other details confirms that the algorithm runs in $O(n^3)$ time.

We now wish to show that if due dates are internally consistent, then there exists a schedule meeting all due dates. We shall do this, not by applying the priority scheduling algorithm directly to the n jobs with modified due dates, but by applying the priority scheduling algorithm to a new problem with an expanded set of $O(n^2)$ jobs.

Suppose at the termination of the due date modification algorithm there is a job j and a $t \in RUD$, $t < d_j$, such that $p_j(t)$, as computed by the algorithm, is such that

$$p_j(t) > p_j-(d_j-t),$$

where d_j is the modified due date. If we were simply to apply the priority scheduling algorithm to the original set of jobs with modified due dates, it might fail to schedule a sufficient amount of processing for job j prior to time t, thereby constructing a schedule which fails to meet due dates, though such a schedule exists. To overcome this difficulty, we replace each job j by a chain of smaller jobs, as follows.

Let $p_j(t)$, $j = 1,2,\ldots,n$, $t \in RUD$, be the values computed by the due date modification algorithm. For each job j, let

$$\{u \mid u \in RUD,\ p_j(u) > 0,\ u \le d_j\} = \{u_1,u_2,\ldots,u_\ell\},$$

with $u_1 < u_2 < \ldots < u_\ell = d_j$. Create ℓ new jobs, $j(1),j(2),\ldots,j(\ell)$. Set $p_{j(1)} = p_j(u_1)$, $d_{j(1)} = u_1$, $r_{j(1)} = r_j$, and set $p_{j(i+1)} = p_j(u_{i+1})-p_j(u_i)$, $d_{j(i+1)} = u_{i+1}$, $r_{j(i+1)} = d_{j(i)}$, for $i = 2,3,\ldots,\ell$. Replace job j by the ℓ new jobs, and modify the precedence constraints so that $h \to j(1)$, for all $h \to j$, $j(\ell) \to k$, for all $j \to k$, and $j(i) \to j(i+1)$, for $i = 1,2,\ldots,\ell-1$.

PROPOSITION. *There exists a schedule meeting the modified due dates of the* n *jobs if and only if there exists a schedule meeting the due dates of the expanded set of jobs, as obtained above.*

We also note that any feasible schedule for the $O(n^2)$ jobs is easily transformed into a feasible schedule for the original n jobs.

THEOREM 9.2. *If due dates and release dates are internally consistent, then there exists a schedule meeting all due dates.*

Proof. Apply the priority scheduling algorithm to the expanded set of $O(n^2)$ jobs obtained as above. Suppose it fails to obtain a schedule meeting all due dates, and let $[t_{\ell-1},t_\ell]$ be the first interval in which there is a job j with $b_j^{(\ell)} = T < t_\ell$.

Case 1. In all earlier intervals $[t_{k-1},t_k]$ there is no idle time and there is no job i with $b_i^{(k)} > T$. Then

$$\Sigma_i\ p_i(T) \geq (1+s)(t_\ell - r_1) > (1+s)(T-r_1),$$

where r_1 is the earliest release date. But this is not possible, else $p_1(r_1) > 0$.

Case 2. There is an earlier interval $[t_{k-1},t_k]$ in which there is either idle time or some job i is processed with $b_i^{(k)} > T$. Let $[t_{k-1},t_k]$ be the latest such interval.

(a) Suppose that a job h is completed at time t_k. Then each job processed between time t_k and time t_ℓ is either a successor of job h or is a job with release date t_k or later. (Note that condition (9.1) insures that each successor of a job with release date t_k or later has a release date later than t_k.) It follows that

$$P(h,t_k,T) \geq (1+s)(t_\ell - t_k) > (1+s)(T-t_k).$$

But this is not possible, else $d_h < t_k$ and $[t_{\ell-1},t_\ell]$ would not be an earliest interval with a job j such that $b_j^{(\ell)} < t_\ell$.

(b) Suppose that no job is completed at time t_k, but t_k is the release date r_h of a job h. By an argument similar to the above, it follows that $p_h(r_h) > 0$, which is not possible by internal consistency. □

We note that although the priority scheduling algorithm is applied to $O(n^2)$ jobs, the time required for the computation is bounded by $O(n^3)$. The precedence constraints are such that at most n jobs are available at any given time and the number of intervals is $O(n^2)$. It follows that $O(n^3)$ time is sufficient to determine whether or not there exists a schedule meeting all due dates.

Unfortunately, it is not possible to minimize maximum lateness as easily as in the previous two sections. It is clearly *not* the case that if L is added to all due dates then all modified due dates are changed by the same constant L.

One way to minimize maximum lateness is to apply the technique of Meggido [8]. In this approach, the due date modification algorithm is applied to due dates d_j+L, where L is maintained as a symbolic variable. Each time a numerical comparison is made whose outcome depends upon the value of L, there is an easily computed "critical" value L* such that one outcome occurs if $L \leq L^*$ and another if $L > L^*$. If the minimum value of L yielding an internally consistent set of due dates is already known to be larger or

smaller than L* there is no ambiguity about the outcome of the comparison. Otherwise, the due date modification algorithm is applied to due dates d_j+L^*. If the resulting due dates are internally consistent, the desired value of L is known to be no greater than L*; otherwise, the desired value is known to be larger. Proceeding in this way, one can determine the minimum value of maximum lateness in $O(n^3)$ applications of the due date modification algorithm, or $O(n^6)$ time overall.

10. EXTENSIONS AND GENERALIZATIONS

The possibilities for finding polynomial time algorithms for more general cases of the problems considered in this paper appear to be limited. NP-hardness proofs exist for several more general problems.

In the context of nonpreemptive scheduling of unit jobs, Brucker, Garey and Johnson [1] showed that the intree problem becomes NP-hard if outtree constraints replace intree constraints (but release dates remain equal and due dates are arbitrary). The problem $P|pmtn,outtree|L_{max}$ is NP-hard [7], since there is no advantage to preemption for the subclass of problem instances of $P|outtree,p_j=1|L_{max}$ shown to be NP-hard in [1].

One question which remains unresolved is the status of the intree problem when generalized to allow the m machines to have different speeds instead of m-1 identical machines and one possibly slower one. The algorithmic approach taken here appears to be not quite adequate for this case.

Also with reference to nonpreemptive scheduling of unit jobs, Ullman [13] proved that it is NP-hard to schedule jobs to minimize makespan, when scheduling jobs for an arbitrary number of machines subject to arbitrary precedence constraints. The preemptive version of this problem is also NP-hard [14].

Ullman's result leaves open the question of whether NP-hardness holds for any *fixed* number of machines and, in particular, for three machines. It is reasonable to believe that if a polynomial time algorithm can be found for nonpreemptive scheduling of unit jobs on three machines, a similar algorithm can be found for the preemptive version of the three-machine problem.

ACKNOWLEDGMENTS

This research was supported in part by NSF grant MCS78-20054. The author wishes to acknowledge the help of Harold Gabow, in pointing out an error in [3]. (It is necessary to require that release dates be consistent, as in (9.1).) He also wishes to thank Jan Karel Lenstra for his advice, and for his patience and encouragement in the preparation of this paper.

REFERENCES

1. P. BRUCKER, M.R. GAREY, D.S. JOHNSON (1977) Scheduling equal length tasks under tree-like precedence constraints to minimize maximum lateness. *Math. Oper. Res.* 2,275-284.
2. M.R. GAREY, D.S. JOHNSON (1976) Scheduling tasks with non-uniform deadlines on two processors. *J. Assoc. Comput. Mach.* 23,461-467.
3. M.R. GAREY, D.S. JOHNSON (1977) Two-processor scheduling with start times and deadlines. *SIAM J. Comput.* 6,416-426.
4. T.F. GONZALEZ, D.B. JOHNSON (1980) A new algorithm for preemptive scheduling of trees. *J. Assoc. Comput. Mach.* 27, 287-312.
5. R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* 5, 287-326.
6. E.C. HORVATH, S. LAM, R. SETHI (1977) A level algorithm for preemptive scheduling. *J. Assoc. Comput. Mach.* 24,32-43.
7. J.K. LENSTRA. Private communication.
8. N. MEGGIDO (1979) Combinatorial optimization with rational objective functions. *Math. Oper. Res.* 4,414-423.
9. C.L. MONMA (1979) Linear-time algorithm for scheduling equal-length tasks with due dates subject to precedence constraints. Technical Memorandum 79-1712, Bell Laboratories, Holmdel, NJ.
10. R.R. MUNTZ, E.G. COFFMAN, JR. (1969) Optimal preemptive scheduling on two-processor systems. *IEEE Trans. Comput.* C-18, 1014-1020.
11. R.R. MUNTZ, E.G. COFFMAN, JR. (1970) Preemptive scheduling of real-time tasks on multiprocessor systems. *J. Assoc. Comput. Mach.* 2,324-338.
12. S. SAHNI, Y. CHO (1979) Nearly on line scheduling of a uniform processor system with release times. *SIAM J. Comput.* 8, 275-285.
13. J.D. ULLMAN (1975) *NP*-Complete scheduling problems. *J. Comput. System Sci.* 10,384-393.
14. J.D. ULLMAN (1976) Complexity of sequencing problems. In: E.G. COFFMAN, JR. (ed.) (1976) *Computer & Job/Shop Scheduling Theory*, Wiley, New York, 139-164.