Preemptive Scheduling with Release Times, Deadlines, and Due Times

CHARLES MARTEL

University of California, Davis, California

Abstract Given n jobs, each of which has a release time, a deadline, and a processing requirement, the problem of determining whether there exists a preemptive schedule on m uniform machines which completes each job in the time interval between its release time and its deadline is examined An $O(m^2n^4 + n^5)$ algorithm is presented which uses a generalization of network flow techniques to construct such a schedule whenever one exists This algorithm is then used with search techniques to find a schedule which minimizes maximum lateness.

Categories and Subject Descriptors D 4 1 [**Operating Systems**]· Process Management—*scheduling*; F 2 2 [**Analysis of Algorithms and Problem Complexity**] Nonnumerical Algorithms and Problems—*sequencing and scheduling*, G 2 2 [**Discrete Mathematics**]· Graph Theory—*network problems*

General Terms: Algorithms, Theory

Additional Key Words and Phrases. Preemptive scheduling, uniform machines, identical machines

1. *Introduction*

In this paper we deal with the problem of finding preemptive schedules for jobs with restricted availability on parallel processors. We show that certain scheduling problems of this kind, such as finding a feasible schedule when each job has a release time and a deadline, can be formulated and solved as network flow problems. The flow networks used have capacities associated with sets of arcs rather than single arcs, but a maximum flow can be found using techniques analogous to those for a standard network flow problem.

As part of the general problem formulation we assume that there are m processors indexed $i = 1, 2, \ldots, m$, with speeds $s_1 \geq s_2 \geq \cdots \geq s_m$, and n jobs indexed $j = 1, 2, \ldots, n$, each with a processing requirement p_j. A job run on machine i for t time units completes $s_i \cdot t$ units of processing. Thus if

$$t_{ij} = \text{the time job } j \text{ runs on machine } i,$$

it is necessary that

$$\sum_{i=1}^{i=m} s_i \cdot t_{ij} = p_j$$

This work was supported by the National Science Foundation under Grant MCS 78-20054.

Author's address Department of Electrical and Computer Engineering, University of California, Davis, CA 95616.

in order for job j to be completed. Processors with speed factors of this type are said to be *uniform*.

A processor can work on only one job at a time, and a job can be worked on by only one processor at a time. The processing of a job can be interrupted at any time and resumed immediately on a new processor or at a later time on any processor. There is no cost or time loss associated with such an interruption or preemption.

We first consider the feasibility problem in which every job has a release time r_j and a deadline d_j. We want to find a schedule which completes every job in the interval between its release time and its deadline or to determine that no such schedule exists. Several special cases of this problem have been previously solved. When the processors are identical, that is, all speeds are equal, Horn [4] shows that the problem can be converted to a network flow problem which can be solved in $O(n^3)$ time. Gonzalez and Sahni [3] have an $O(n + m \log n)$ algorithm that solves the problem when all release times are equal and all the deadlines are equal. Sahni and Cho [9] also solve the problem in $O(n^2 + nm^2)$ time when the jobs have variable release times but all jobs have a common deadline.

After solving the feasibility problem, we use the solution to solve a scheduling problem in which jobs have due times rather than deadlines, and we extend the model to allow jobs which have arbitrary availabilities and processors whose speeds vary.

2. *Solution of the Feasibility Problem*

In the case in which each job has a release time and a deadline we can use these release times and deadlines to divide the time line into $2n - 1$ intervals. Let t_i be the ith smallest value among the n release times and n deadlines (for $i = 1, 2, \ldots, 2n$). The ith interval is the time period from t_i to t_{i+1}. Let $\Delta_i = t_{i+1} - t_i$. The jth job is *available* within the ith interval iff $r_j \le t_i$ and $d_j \ge t_{i+1}$. Within each interval the set of available jobs does not change; so, given the amount of processing to be done on each job within an interval, scheduling these jobs within the interval is an instance of the problem where all jobs have the same release time and all jobs have a common deadline. Thus it is sufficient to find the amount of processing to be done on each job within each interval and then use the Gonzalez–Sahni algorithm to construct the actual schedule.

Horvath et al. [5] show that if $q_1 \ge q_2 \ge \cdots \ge q_k$ are the amounts of processing to be done on k jobs, then there is a schedule which completes these processing amounts within an interval of length Δ iff

$$
\begin{aligned}
q_1 &\le s_1 \cdot \Delta, && \text{inequality 1,}\\
q_1 + q_2 &\le (s_1 + s_2) \cdot \Delta, && \text{inequality 2,}\\
&\vdots && \qquad (*)\\
q_1 + q_2 + \cdots + q_{m-1} &\le (s_1 + \cdots + s_{m-1}) \cdot \Delta && \text{inequality } m-1,\\
q_1 + q_2 + \cdots + q_k &\le (s_1 + \cdots + s_m) \cdot \Delta, && \text{inequality } m.
\end{aligned}
$$

Thus we can easily determine if the processing amounts within an interval can be scheduled.

2.1 Converting the Feasibility Problem to a Flow Problem. In order to find the amount of processing to be done on each job within an interval a flow network of the form shown in Figure 1 is used. There is a *J-node* for each job and an *I-node* for each interval. An arc connects a J-node to an I-node if the job correspond-

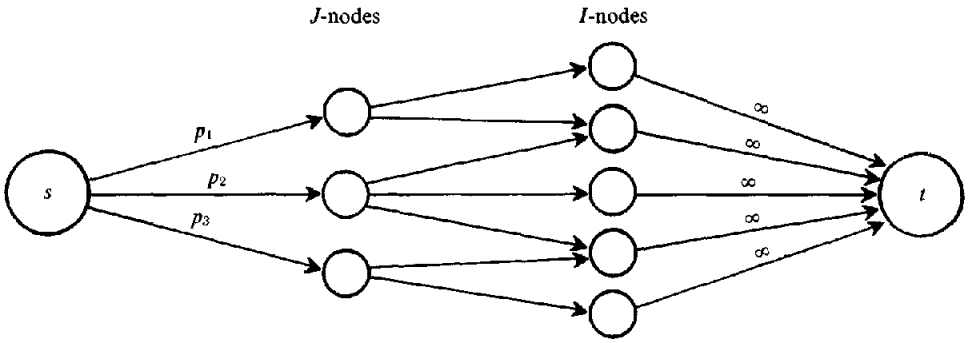

FIGURE 1

ing to the J-node is available in the corresponding interval. The set of arcs directed out of the jth J-node is A_j, and B_i is the set of arcs directed into the ith I-node. There is also a source node s with an arc from s to each J-node and a sink node t with an arc from each I-node to t. We denote the arc from s to the jth J-node as (s, j), the arc from the jth I-node to t as (j, t), and the arc from the jth J-node to the ith I-node as (j, i). We assign flow values to the arcs using a function f which maps the arcs into the nonnegative reals. We apply f to sets of arcs, where

$$f(X) = \sum_{x \in X} f(x).$$

A flow function f is *feasible* iff

(1) $f((s, j)) \leq p_j$, $j = 1, 2, \ldots n$;
(2) for all $X \subseteq B_j$, $f(X) \leq \rho(X)$, $j = 1, 2, \ldots, 2n - 1$, where if $X \subseteq B_j$,

$$\rho(X) = \begin{cases} \Delta_j \cdot \sum_{i=1}^{|X|} s_i & \text{if } |X| < m, \\ \Delta_j \cdot \sum_{i=1}^{m} s_i & \text{if } |X| \geq m; \end{cases}$$

(3) $f(s, j) = f(A_j)$, $j = 1, 2, \ldots, n$,
(4) $f(j, t) = f(B_j)$, $j = 1, 2, \ldots 2n - 1$.

Inequalities (1) and (2) represent capacity constraints on the flow network, and (3) and (4) are conservation constraints. Inequalities (1) ensure that at most p_j units of processing are assigned to the jth job. The inequalities in (2) ensure that the flows assigned to arcs into an I-node correspond to processing amounts which can be scheduled within the interval.

Any feasible schedule corresponds to a feasible flow, and any feasible flow corresponds to a feasible schedule. The value of a flow is the amount of processing completed in the corresponding schedule. Thus a schedule which completes all jobs exists iff the maximum flow is

$$\sum_{i=1}^{n} p_i.$$

2.2 FINDING THE MAXIMUM FLOW. The constraints (1), (3), and (4) on f describe a classical network flow problem. The addition of the constraints in (2) creates a new kind of flow problem. However, we will now show that a maximum flow can be found using techniques very similar to those used for classical flow networks. Starting

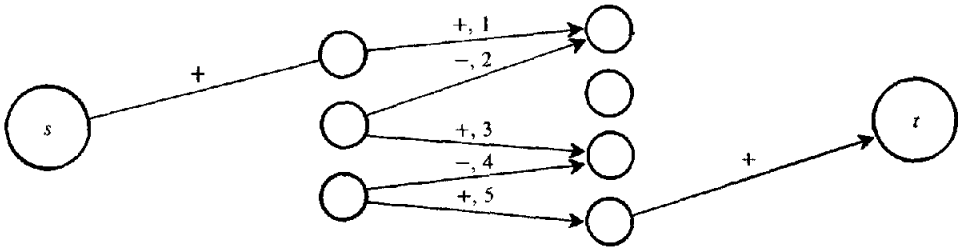

FIGURE 2

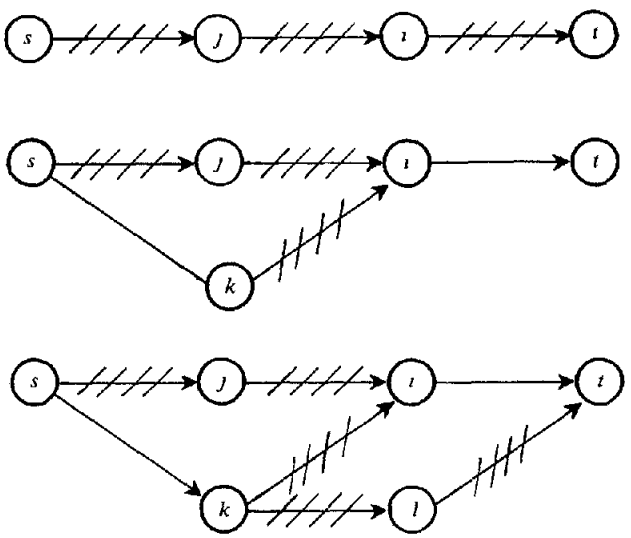

FIGURE 3

with any feasible flow, this flow is successively increased using augmenting paths until a maximum flow is achieved. A modified max-flow–min-cut theorem is used to show that if a flow admits no augmenting path, then the flow is maximum. With respect to a flow, an *augmenting path* is an undirected path from s to t for which there is an amount $\delta > 0$ such that increasing the flow in all forward arcs by δ and decreasing the flow in all backward arcs by δ results in a feasible flow. The maximum such value of δ is the *path capacity*. An augmenting path is shown in Figure 2.

Each augmenting path starts with a forward arc (s, j) such that $f((s, j)) < p_j$. The second arc will be from the jth J-node to an I-node i. If this second arc, (j, i), can have its flow increased without violating any constraints in (2), then $(s, j) \rightarrow (j, i) \rightarrow (i, t)$ is an augmenting path. If increasing the flow in arc (j, i) violates a constraint in (2), then we can try to offset this by decreasing the flow in an arc (k, i). The augmenting path now contains arcs $(s, j)(j, i)(k, i)$, where (k, i) is a backward arc (see Figure 3). The next arc in the path will be a forward arc (k, l). If increasing the flow in (k, l) violates no constraints, then $(s, j)(j, i)(k, i)(k, l)(l, t)$ is an augmenting path. If it does violate a constraint, then the path must be extended further. We will now introduce some terminology to allow us to formally describe which paths are augmenting paths. All of the descriptions are with respect to a given flow f.

If $f((s, i)) = p_i$, then (s, i) *is saturated.* All augmenting paths start with an arc which is not saturated.

If y can follow x on an augmenting path, then (x, y) is an *admissible pair.*

If for $A \subseteq B_i$, $f(A) = \rho(A)$, then A is *saturated* and all arcs in A are *saturated arcs.* When a saturated arc is directed forward in an augmenting path, it must be followed by a backward arc. Also, if $q_1 \geq q_2 \geq \cdots \geq q_k$ are the flows in a saturated set, then

$$\sum_{j=1}^{k} q_j = \begin{cases} \Delta_t \cdot S_k & \text{if } k \geq m, \\ \Delta_t \cdot S_m & \text{if } k > m, \end{cases}$$

where

$$S_k = \sum_{i=1}^{k} s_i.$$

Thus the inequality with index k (m if $k > m$) in (*) holds with equality. Such an inequality is said to be *tight.* Arcs which are directed forward in an augmenting path will be referred to as *plus* arcs, and arcs directed backward will be called *minus* arcs. A + or − superscript may be appended to an arc to indicate its direction in the path. An augmenting path of length k from s to t will be written $(s) \to x_1 \to x_2 \to \cdots \to x_k \to (t)$, where $x_1, x_2, \ldots, x_k$ are arcs connecting J-nodes to I-nodes. The arcs from s and into t are not explicitly indicated. The notation $x \to y$ indicates that (x, y) is an admissible pair.

Let $q_1 \geq q_2 \geq \cdots \geq q_k$ be the flows on arcs in B_i for some i, $1 \leq i \leq 2n - 1$. For an arc x in B_i let j be the least integer such that $f(x) = q_j$. If $j < m$, then x *appears as a plus arc* in inequalities $j, j + 1, \ldots, m$. If $j \geq m$, then x appears as a plus arc in inequality m. Similarly, let j be the largest integer such that $f(x) = q_j$. Then x *appears as a minus arc* in inequalities $j, j + 1, \ldots, m$ if $j < m$ and in inequality m if $j \geq m$.

If x appears as a plus arc in an inequality, then increasing the flow in x will affect that inequality, and if x appears as a minus arc, then decreasing the flow affects the inequality.

We define $H^+(x)$ to be the smallest index of a tight inequality in which x appears as a plus arc, and $H^-(x)$ to be the smallest index of a tight inequality in which x appears as a minus arc. If x is not a saturated arc, it appears in no tight inequality, and we define $H^+(x) = H^-(x) = m + 1$.

Two arcs x, y incident to the same I-node are an admissible pair, $x^+ \to y^-$, iff $H^+(x) \geq H^-(y)$ and $f(y) > 0$. Two arcs x, y incident from the same J-node are an admissible pair, $x^- \to y^+$, iff $f(x) > 0$.

Intuitively, $H^+(x)$ is the index of an inequality which will be violated if the flow in x is increased. If $H^-(y) \leq H^+(x)$, then decreasing the flow in y will compensate for the increase in the flow in x.

2.3 FINDING AUGMENTING PATHS. We will increase the flow by successively finding shortest augmenting paths, using the following labeling algorithm.

Start Label 1 all arcs x such that x is incident from J-node i and (s, i) *is not saturated*;
$L = 2$,
While there is a newly labeled arc Do
Begin ·

Plus· For each newly labeled arc x which is not saturated and incident to I-node t, label (t, t) with L,
For each newly labeled saturated arc x, give label L to all unlabeled arcs y such that $x^+ \to y^-$,
If no new arcs were labeled, or an arc incident to t was labeled then Halt;
$L := L + 1$,

Minus For all newly labeled arcs x such that $f(x) > 0$, give label L to all unlabeled arcs y such that y and x are incident from the same J-node,
$L = L + 1$;
End,

Note that each arc is only labeled as either a plus arc or a minus arc, not both. To show that all shortest paths are found by the labeling procedure, we prove the following lemma.

LEMMA 2.3.1. *If* $(s) \to \cdots \to w^- \to x^+$ *is a shortest path consisting of admissible pairs and ending at* x*, then no shortest* s*–*t *augmenting path uses* x *as a minus arc.*

PROOF. Suppose contrary to the assertion that $P = (s) \to \cdots \to v^+ \to x^- \to y^+ \to \cdots \to (t)$ is a shortest augmenting path and $(s) \to \cdots \to w^- \to x^+$ is a shortest augmenting path to x. Since w, x, and y are all incident from the same J-node and $f(w) > 0$, $w^- \to y^+$. Now any augmenting path to a minus arc has an odd number of arcs, and any augmenting path to a plus arc has an even number of arcs. Thus $|(s) \to \cdots \to v^+ \to x^-| > |(s) \to \cdots \to w^- \to x^+|$, and therefore $|(s) \to \cdots \to w^- \to y^+ \to \cdots \to (t)| < |P|$. This contradicts our assumption that P was a shortest path; so, no shortest path uses x as a minus arc.

A similar proof can be used to show that if an arc is labeled in the minus part of the label procedure, no shortest path uses x as a plus arc. □

A shortest path can be extracted from a labeled graph by finding a labeled arc incident to t and working backward. At a given stage, if x is the current arc with label L, we next find an arc y such that (y, x) is an admissible pair and y's label is $L - 1$. In order to help the analysis we will break ties among possible arcs which could precede x by choosing y such that $f(y)$ is maximum. A shortest path which is chosen using this tie break rule will be called a *minimal augmenting path.*

2.4 PROOF OF OPTIMALITY

THEOREM 2.4.1. *If a flow admits no augmenting path from* s *to* t*, then the flow is maximum.*

PROOF. Suppose that with respect to a given flow we have labeled the network and no augmenting path was found. We will partition the arcs and show that the total flow on arcs within each set is maximum.

Let $L = \{x \mid \text{arc } x \text{ was labeled and } x \text{ connects a } J\text{-node to an } I\text{-node}\}$, and $L_i = L \cap B_i$. Let $U = \{x \mid x \text{ connects a } J\text{-node to an } I\text{-node and } x \text{ is unlabeled}\}$. Similarly, let $U_i = U \cap A_i$.

We now show that the flow in each set L_i is maximum. We first show that every arc in L is saturated. Suppose an arc x in L were not saturated. If x has a plus label, then $(s) \to \cdots \to x \to (t)$ would be an augmenting path, and if x has a minus label, then there must be an arc y which labeled x, and such an arc must have a plus label and not be saturated; so, the above argument applies to y. Therefore all arcs in L are saturated.

Each saturated arc appears as a plus arc in a tight inequality. Thus for each set L_i there is a tight inequality in which all the arcs of the set appear as plus arcs. Let j be the smallest index of such an inequality. There must be an arc y in L_i which does not appear as a plus arc in any inequality of smaller index; so, $j = H^+(y)$. Now let $q_1 \geq q_2 \geq \cdots \geq q_k = f(y)$ be the flows on the arcs in L_i ($|L_i| = k$). Then

$$\sum_{l=1}^{k} q_l = \begin{cases} S_k \cdot \Delta_i & \text{if } j = k < m, \\ S_m \cdot \Delta_i & \text{if } k > m \text{ and } j = m, \end{cases}$$

which is the maximum amount of flow possible in a set of cardinality k.

If $j > k$, then there is an arc x which is not in L_i and is incident to the ith I-node, and $f(x) > f(y)$. But then if y has a plus label, x would have been labeled from y, and if y has a minus label, then there is an arc z in L_i with a plus label which labeled

y. But since $H^+(z) \geq j$, z would also have labeled x. Thus no such arc x can exist; so, $j = k$ or $k > m$, and the flow L_t is maximum.

Since the sets L_t are a partition of L and the flow in each set L_t is maximum, the flow in L is maximum.

We now show that the flow on the arcs in U is maximum. We start by showing that each set U_t is either empty or contains all arcs incident from the tth J-node. Suppose an arc in A_t is in L. Then either (s, t) is not saturated, in which case all arcs in A_t are in L, or some arc y in A_t was given a minus label. If y were given a minus label, it would cause all unlabeled arcs in A_t to be given a plus label. Thus if one arc in A_t is in L, all arcs must be in L.

Now if U_t is empty, the flow is trivially maximum. If U_t is not empty, it contains all arcs in A_t, and the arc (s, t) is saturated. Thus $f(U_t) = p_t$, which is again a maximum flow for the arcs in U_t. Since the flow in each set U_t is maximum, the flow in U is maximum.

Since the arcs in U and L form an s–t cutset, and the total flow on these arcs is maximum, the flow in the network is maximum. This result is analogous to the max-flow–min-cut theorem for a classical flow network. □

The previous results allow us to prove a further result about the scheduling problem. For a set of jobs R, we define the *M*aximum *A*vailable *C*omputation on R in the tth interval, $MAC(R, t)$, to be

$$S_k \cdot \Delta_t, \qquad k = \min\{m, \text{number of jobs in } R \text{ available in the } t\text{th interval}\}.$$

This is the amount of computation that would be done on the jobs in R if the k jobs available in the tth interval were scheduled on the k fastest machines for all of the tth interval. We also define MAC(R) to be

$$\sum_{t=1}^{2n-1} \text{MAC}(R, t).$$

MAC(R) is an upper bound on the processing that can be on the jobs in R. We also define $P(R)$ to be the sum of the processing amounts of the jobs in R. Clearly if MAC(R) $< P(R)$, then the jobs in R cannot be scheduled, but we can now prove a stronger fact.

THEOREM 2.4.2. *A set of jobs J has a feasible schedule iff every subset R of J has $MAC(R) \geq P(R)$.*

PROOF. The necessity is immediate. To prove sufficiency, we show that if J has no feasible schedule, then we can find a set R such that

$$\text{MAC}(R) < P(R).$$

If J has no schedule, then there is a maximum flow which does not saturate all arcs incident from s. Let L and L_t be defined for this flow as in Theorem 2.4.1, and let R be the set of all jobs such that the job's J-node is incident to an arc in L.

We proved in Theorem 2.4.1 that if there is one arc incident from a J-node in L, then all arcs incident from the J-node are in L. Thus L is the set of all arcs incident from J-nodes corresponding to jobs in R. Therefore,

$$|L_t| = \text{the number of jobs in } R \text{ available in interval } t.$$

Now if k is $\min\{m, |L_t|\}$, we know that

$$f(L_t) = S_k \cdot \Delta_t = \text{MAC}(R, t).$$

Thus $f(L) = \text{MAC}(R)$. Since some arcs from s are not saturated, there are sets A_i with $f(A_i) < p_i$. All arcs in such sets A_i are labeled; so, $f(L) < P(R)$. Therefore $\text{MAC}(R) < P(R)$. □

2.5 PROOF OF TERMINATION. We now show that if we successively augment along a minimal augmenting path, then $O(m^2n^2 + n^3)$ augmentations are required to produce a flow which admits no augmenting path, and we know that a flow which admits no augmenting path is maximum. We begin by proving several lemmas about paths and augmentations.

LEMMA 2.5.1. *A shortest augmenting path has length 2n or less.*

PROOF. If a path has more than $2n$ arcs, then it must have two plus arcs, x, y, incident from the same J-node. Assume that x occurs before y on the path. If x is the first arc on the path, then the arc from s to the J-node is not saturated, and the path could start with y and delete the part of the path from x to y. If x is not the first arc, then a minus arc v incident from the same J-node precedes x. Since $f(v) > 0$, $v^- \to y^+$; so, the path between x and y can again be eliminated. Thus if a path has more than $2n$ arcs, then a shorter path exists. □

We will frequently use the technique of finding admissible pairs, which allows us to splice together two pieces of a path into a new shorter path.

LEMMA 2.5.2. *If a shortest augmenting path contains admissible pairs $x_1^+ \to y_1^-$, $x_2^+ \to y_2^-, \ldots, x_k^+ \to y_k^-$, all incident to the same I-node and appearing on the path in the order given, then*

$$H^-(y_1) \le H^+(x_1) < H^-(y_2) < \cdots < H^-(y_k) \le H^+(x_k).$$

PROOF. If for $i < j \le k$, $x_i^+ \to y_j^-$, then there is a shorter augmenting path which deletes the path between x_i and y_j. Since $H^+(x_i) \ge H^-(y_j)$ would imply such an admissible pair,

$$H^+(x_i) < H^-(y_{i+1}) \qquad \text{for} \quad i = 1, 2, \ldots k-1. \tag{1}$$

Since $x_i^+ \to y_j^-$, by definition,

$$H^-(y_i) \le H^+(x_i) \qquad \text{for} \quad i = 1, 2, \ldots, k. \tag{2}$$

The result follows from (1) and (2). □

COROLLARY 2.5.1. *If a plus arc on an augmenting path appears in a tight inequality, then that inequality will remain tight after augmentation.*

PROOF. If any other minus arc appears in the inequality, then the preceding plus arc also appears. □

LEMMA 2.5.3. *Suppose x, y, v, and w are distinct arcs incident to the same interval node, and with respect to flow f^0, $x^+ \to y^-$ and $v^+ \not\to w^-$. If by only increasing x's flow by σ and decreasing y's flow by σ we create flow f', and $v^+ \to w^-$ in f', then with respect to f^0: (a) $x^+ \to w^-$, (b) $v^+ \to y^-$, (c) $f^0(w) < f^0(y)$, and (d) $f^0(x) < f^0(v)$.*

PROOF. Since under f^0, $v^+ \not\to w^-$, there exists one or more tight inequalities in which v appears as a plus arc and w does not appear as a minus arc. Such an inequality is called a *blocking* inequality. Since under f', $v^+ \to w^-$, no blocking inequalities exist. We now show that this implies that every blocking inequality contains y as a minus arc and does not contain x as a plus arc.

Proof by contradiction. First suppose there is a blocking inequality in which y does not appear as a minus arc. Then, since $x^+ \to y^-$, x does not appear in this inequality as a plus arc. Therefore none of the flows in this inequality change; so, it is a blocking inequality under f'. This is impossible; so, every blocking inequality must contain y as a minus arc. Assertions (b) and (c) follow directly from this.

Now suppose there is a blocking inequality which contains x as a plus arc. Since the inequality contains both x as a plus arc and y as a minus arc, it will remain tight, still contain v as a plus arc, and not contain w as a minus arc. Thus it will still be a blocking inequality under f'. Thus no blocking inequality contains x as a plus arc. Therefore $H^+(x) \geq H^-(w)$, which implies (a), and $H^+(v) < H^+(x)$, which proves (d). □

LEMMA 2.5.4. *If augmenting with respect to a minimal path P creates an arc pair $v^+ \to w^-$, then there is an arc pair $x^+ \to y^-$ on P whose flow changes alone created the arc pair $v^+ \to w^-$.*

PROOF. Let $x_1^+ \to y_1^-$, $x_2^+ \to y_2^-$, ..., $x_k^+ \to y_k^-$ be the arc pairs on P which are incident to the node that v and w are incident to. Since (v^+, w^-) is not an admissible arc pair with respect to the original flow, there is at least one blocking inequality for this pair of arcs. Let l be the smallest index of a blocking inequality and r the largest index. In order for inequality l not to be a blocking inequality after augmentation there must be an arc pair $x_j^+ \to y_j^-$ such that $H^-(y_j) \leq l$ and $H^+(x_j) > l$. By Lemma 2.5.2, only one arc pair has this property, and, by Corollary 2.5.1, inequalities in which x_j appears as a plus arc remain tight after augmentation. Thus if $H^+(x) \leq r$, then inequality r will remain a blocking inequality. Therefore $H^+(x_j) > r$, and, by Lemma 2.5.2, for each arc pair $x_i^+ \to y_i^-$, $i \neq j$, x_i appears as a plus arc in a blocking inequality iff y_i also appears as a minus arc in the inequality. Therefore, if a blocking inequality is not tight after augmentation, it is caused solely by changes in x_j's and y_j's flow. □

LEMMA 2.5.5. *Let P be a minimal augmenting path w. r. t. f^0 and f^1 the flow obtained by augmenting w. r. t P. Then any augmenting path Q w. r. t. f^1 is such that $|Q| \geq |P|$, and if $|Q| = |P|$, then Q existed w. r. t. f^0.*

PROOF. Suppose f^1 admits a path Q such that $|Q| < |P|$. We show that no such Q can exist even if Q can contain arc pairs w. r. t. both f^0 and f^1. The proof is by induction on the number of arc pairs in Q which are not arc pairs w. r. t. f^0. These are called *new arc pairs.* Note that the only new arc pairs are plus–minus pairs.

Basis. Suppose there is one new arc pair, $v^+ \to w^-$, in Q. By Lemma 2.5.4, this arc pair was created by changes made in the flows of a single arc pair in P, $x^+ \to y^-$. Since x, y, v, and w are all incident to the same interval node, we know from Lemma 2.5.3 that $x^+ \to w^-$ and $v^+ \to y^-$ w. r. t. f^0. Thus both $M = (s) \to \cdots \to x^+ \to w^- \to \cdots \to (t)$ and $N = (s) \to \cdots \to v^+ \to y^- \to \cdots \to (t)$ are paths w. r. t. f^0, which were obtained by splicing together prefixes and suffixes of P and Q. Now if the prefix $(s) \to \cdots \to x^+$ is longer than $(s) \to \cdots \to v^+$, then $|N| < |P|$. If the suffix $y^- \to \cdots \to (t)$ is longer than $w^- \to \cdots \to (t)$, then $|M| < |P|$. Thus since P is minimal and $|Q| \leq |P|$, the two suffixes must have the same length and the two prefixes must have the same length. But in this case both N and P are shortest paths with a common suffix $y^- \to \cdots \to (t)$, and, by Lemma 2.5.3, the arc pairs $x^+ \to y^-$ and $v^+ \to y^-$ exist w. r. t. f^0 and $f^0(v) > f^0(x)$. This violates the assumption that P is minimal; so, no path Q can exist.

Induction step. Assume the theorem is true for any new augmenting path with k or fewer new arc pairs. Let P be a minimal augmenting path and Q an augmenting path with $k+1$ new arc pairs. Let $v^+ \to w^-$ be the last new arc pair in Q. Using Lemma 2.5.3 on this arc pair as in the proof of the basis, we can combine P and Q to get an augmenting path N such that either N has no new arc pairs or $|N| \leq |P|$, and N was not an augmenting path w. r. t. f^0. The second case violates the induction assumption, and the first violates the minimality of P. Thus no augmenting path Q with $k+1$ new arc pairs exists, and the lemma follows. □

We now show how to find the capacity of an augmenting path. We will first show that given an arc pair $x^+ \to y^-$, we can compute the maximum amount σ such that increasing the flow in x by σ and decreasing the flow in y by σ results in a feasible flow. Call this the *capacity* of $x^+ \to y^-$, which we write cap(x, y).

Suppose $x^+ \to y^-$ is an arc pair with x and y incident to the ith I-node. If $f(y) > f(x)$, then we can interchange x and y's flows and retain feasibility. Once this has been done, or if $f(y) \leq f(x)$, we can increase x's flow while decreasing y's flow until one of four conditions occurs:

(1) y's flow is decreased to zero.
(2) x appears as a plus arc in a new inequality which is tight.
(3) x appears as a plus arc in inequality i, y doesn't appear as a minus arc in inequality i, and increasing x's flow makes i a tight inequality.
(4) y "drops out" of an inequality which is tight and in which x occurs.

Thus cap(x, y) is the smallest amount σ which causes one of conditions (1)–(4) to occur. To describe these conditions algebraically, let $q_1 \geq q_2 \geq \cdots \geq q_k$ be the flows on arcs incident to the I-node to which x and y are incident, and let $f(y) = q_r$ and $f(x) = q_p$. The *slack* in the jth inequality, SL(j), is

$$\mathrm{SL}(j) = \begin{cases} S_j \cdot \Delta_i - \sum_{l=1}^{l=j} ql & \text{for } j = 1, 2, \ldots, m-1, \\ S_m \cdot \Delta_i - \sum_{l=1}^{l=k} ql & \text{for } j = m. \end{cases}$$

Thus if $f(y) > f(x)$, then cap$(x, y) = f(y) - f(x)$ + minimum of

(1) $f(x)$;
(2) $(q_j - q_r) + \mathrm{SL}(j)$, $j = 1, 2, \ldots, r-1$;
(3) $\mathrm{SL}(j)$, $j = r, r+1, \ldots, p-1$;
(4) $(q_p - q_{j+1}) + \mathrm{SL}(j)$, $j = p, p+1, \ldots, m$.

If $f(y) \leq f(x)$, then cap(x, y) is the minimum of

(1a) $f(y)$;
(2a) $(q_j - q_p) + \mathrm{SL}(j)$, $j = 1, 2, \ldots, p-1$;
(3a) $\mathrm{SL}(j)$, $j = p, p+1, \ldots, r-1$;
(4a) $(q_r - q_{j+1}) + \mathrm{SL}(j)$, $j = r, r+1, \ldots, m$.

We now show that the capacity of an arc pair on a shortest augmenting path is independent of changes in the flows of other arcs on the path.

LEMMA 2.5.6. *If $x^+ \to y^-$ and $v^+ \to w^-$ are both on a minimal augmenting path and are incident to the same I-node, then increasing the flow in the plus arc and decreasing the flow in the minus arc in either arc pair by an amount less than or equal to its capacity will not affect the capacity of the other arc pair.*

PROOF. Assume that $x^+ \to y^-$ occurs first. Then by Lemma 2.5.2 we know that $H^+(x) < H^+(v)$. Let $f(v) = q_p$, $f(w) = q_r$, and $H^+(x) = L$. Now if $f(v) > f(w)$, then $\text{cap}(v, w) \leq (q_L - q_p) + \text{SL}(L) = q_L - q_p$. If $f(w) \geq f(v)$, then $\text{cap}(v, w) \leq (f(w) - f(v)) + (q_L - q_r)$. Thus the augmented value of v is at most q_L, and therefore the L largest flow values remain unchanged when $v^+ \to y^-$ is augmented. Since $\text{cap}(x, y)$ is a function of the L largest flow values, it is unchanged.

Now augmenting $x^+ \to y^-$ only changes the L largest flow values. Since inequality L is tight, $\text{cap}(v, w)$ does not depend on the L largest flow values. Thus $\text{cap}(v, w)$ will not be affected by augmenting the arc pair (x, y). □

Since the capacity of an arc pair is independent of the other arc pairs, the maximum amount we can augment along a minimal augmenting path $s \to x_1^+ \to y_1^- \to \cdots \to y_{k-1}^- \to x_k^+ \to t$ is the minimum of

(1) the amount we can increase the flow in the arc directed out of s before it becomes saturated:
(2) the amount we can increase the flow in x_k before it becomes saturated;
(3) $\min\{\text{cap}(x_i, y_i), i = 1, 2, \ldots, k - 1\}$.

If an augmenting path P contains an arc pair $x^+ \to y^-$ and $\text{cap}(P) = \text{cap}(x, y)$, then $x^+ \to y^-$ is a *critical* arc pair on P.

LEMMA 2.5.7. *In a maximum flow computation at most $O(m^2n + n^2)$ augmenting paths of a given length can have a critical arc pair.*

PROOF. We prove this bound by showing that only $O(m^2 + n)$ critical arc pairs can occur at each node. Now if $x^+ \to y^-$ is a critical arc pair, then either

(1) x is loose
(2) $H^+(x) = m$, or
(3) $H^+(x) < m$.

If case (1) applies, then we will increase x and decrease y until x becomes tight or y's flow becomes zero. If x becomes tight, it will never be used in case (1) again, and if y's flow is zero, it will not be used in case (1) until the path length changes. Thus case (1) occurs $O(n)$ times at a given node, without increasing the path length.

If case (2) applies, then we increase x and decrease y until $H^+(x) < m$ or y's flow becomes zero. Once $H^+(x) < m$, it will never be used in case (2) until the path length changes, and once $f(y) = 0$, y is not used until the path length changes. Thus case (2) occurs at most $O(n)$ times at a node before the path length changes.

Now if case (3) applies, then there are at most $H^+(x) - 1$ arcs y such that $x^+ \to y^-$, since only the $H^+(x)$ largest arc flows can have $H^-(y) \leq H^+(x)$. By Lemma 2.5.5, no new arc pairs are used until the path length changes. Thus each plus arc is used in at most m critical arc pairs of type (3). If case (3) applies to a plus arc x, then $x^+ \to y^-$ only if y's flow is one of the $m - 1$ largest flows at the I-node. As more plus arcs are in case (3), the number of minus arcs with flows among the $m - 1$ largest decreases; so, only $O(m^2)$ cases of type (3) can occur at a node.

Therefore the total number of critical arc pairs at a node is $O(m^2 + n) \cdot O(n)$ nodes; so, there are $O(m^2n + n^2)$ critical arc pairs which can be used before the path length changes. □

THEOREM 2.5.1. *After $O(m^2n^2 + n^3)$ augmentations along minimal augmenting paths no augmenting paths will exit.*

PROOF. Each time we augment along a minimal augmenting path, one of three things happens:

(a) The first arc in the path becomes saturated.
(b) The next to last arc becomes saturated.
(c) A critical arc pair is removed.

Once a plus arc becomes saturated, it remains saturated until the path length increases; so, (a) and (b) can each occur only $O(n^2)$ times per path length, and $O(n^3)$ times in total.

By Lemma 2.5.7, (c) occurs at most $O(m^2n + n^2)$ times per path length; so, the total number of augmenting paths used is $O(m^2n^2 + n^3)$. □

2.6 COMPLEXITY ANALYSIS AND IMPLEMENTATION. We will first give an $O(m^2n^5 + n^6)$ implementation of the maximum flow algorithm and then show how to improve this to an $O(m^2n^4 + n^5)$ algorithm.

```
    Maxflow1   While an augmenting path exists do
(1)            begin Compute all the admissible pairs,
(2)                  Label the network,
(3)                  Find a minimal augmenting path;
(4)                  Compute the path capacity,
(5)                  Augment the flow,
               end
```

We will do step (1) by creating for each arc x a *successor* list and a *predecessor* list. The successor list contains all arcs y such that $x^+ \to y^-$. The predecessor list contains all arcs z such that $z^+ \to x^-$. These two lists are constructed by computing $H^+(x)$ and $H^-(x)$ for each arc x which connects a J-node to an I-node. This takes $O(n \log n)$ time per interval node, or $O(n^2 \log n)$ in all. The lists can be constructed from the H^+ and H^- values in $O(n)$ time per arc, or $O(n^3)$ total time. All admissible pairs at the I-nodes can be easily derived from these lists. The admissible pairs at the J-nodes can be gotten from a list of arcs at the J-node which have nonzero flow. Thus (1) takes $O(n^3)$ time.

In the labeling procedure in Section 2.3, each arc gets a label at most once, and processing a newly labeled arc requires $O(n)$ to scan its successor list. Thus the labeling, step (2), takes $O(n^3)$ time. We then update the successor and predecessor lists so each arc y in the successor list of x has a label one greater than x's label and each arc in the predecessor list has label one less than x.

We can find a minimal augmenting path by scanning one predecessor list for each arc in the path; so, (3) takes $O(n^2)$ time.

The capacity of an arc pair can be computed in $O(n)$ time, since we have already sorted the flow values at each I-node in step (1). Thus step (4) takes $O(n^2)$ time.

Once the path capacity is known, we need to update only $O(n)$ flow values; so, step (5) is $O(n)$. Thus the loop has complexity $O(n^3)$ and by Lemma 2.5.7 is executed $O(m^2n^2 + n^3)$ times for a total complexity of $O(m^2n^5 + n^6)$. Steps (1) and (2) are the bottleneck in the loop, and we now show how to speed them up.

By Lemma 2.4.5 we know that no new augmenting paths as short as the one used are created. Thus no new admissible pairs are used until the path length changes. This means it is sufficient to update the labeling by deleting admissible pairs which are removed by augmentation and only do steps (1) and (2) of Maxflow1 when the length of a shortest path increases. Thus (1) and (2) are done only $O(n)$ times; so, the total work on (1) and (2) is $O(n^4)$.

We now show how to update the labeling. If an admissible pair $x^+ \to y^-$ is removed, we check to see if y's label is one greater than x's label. If it is greater, then we remove y from x's successor list and remove x from y's predecessor list. If y's predecessor list becomes empty, then we remove all admissible pairs $y^- \to z^+$ such

that z is on y's successor list and change y's label to infinity. We continue until no more admissible pairs are removed.

The update of the labeling requires $O(m)$ time per interval node to determine which admissible pairs are removed and $O(n)$ time for each arc removed from a successor list or predecessor list. However, each time we remove an arc from a predecessor or successor list, we reduce by one the number of arc pairs which could be a critical arc pair on an augmenting path of the current length. Thus, by Lemma 2.4.6, at most $O(m^2n + n^2)\cdot O(n)$ work per path length is needed to update the successor and predecessor lists. Therefore updating the labeling takes $O(m\cdot n)$ to find the admissible pairs removed times $O(m^2n^2 + n^3)$ augmentations for $O(m^3n^3 + mn^4)$ total.

The bottleneck is now the $O(m^2n^4 + n^5)$ required for (3) and (4) in Maxflow1; so, the algorithm is $O(m^2n^4 + n^5)$.

2.7 Bounding the Number of Preemptions. The schedule which is constructed from a maximum flow will have two classes of preemptions. Those caused by scheduling a job more than once within an interval will be called *internal preemptions*, and those caused by scheduling a job in more than one interval will be called *external preemptions*.

The Gonzalez–Sahni algorithm, which is used to construct the schedule for each interval, produces at most $2(m - 1)$ preemptions within each interval [3]. Thus there are at most $2(m - 1)\cdot(2n - 1)$ internal preemptions. A schedule constructed from a maximum flow can have as many as $n^2 - n$ external preemptions; however, we will now show how to modify the output of the maximum flow algorithm to produce a schedule with at most $(m - 1)\cdot(2n - 1) + 2n - 2$ external preemptions.

2.7.1 *Modifying the Maximum Flow.* If x is an arc which connects the jth J-node to the ith I-node and with respect to flow f, $f(x) > 0$, then in the schedule constructed from f the jth job will be scheduled in the ith interval. Thus if we bound the number of arcs which have nonzero flow, we will bound the number of external preemptions.

We will modify a maximum flow by finding *augmenting cycles.* An augmenting cycle with respect to a flow f is an undirected cycle $x_1 \to x_2 \to \cdots \to x_k$ such that (x_i, x_{i+1}), $i = 1, 2, \cdots, k - 1$, and (x_k, x_1) are admissible pairs. The capacity of this cycle is

$$\min\{\text{cap}(x_i, x_{i+1}),\ i = 1, 2, \ldots, k - 1;\ \text{cap}(x_k, x_1)\}.$$

We now define the following two sets of arcs with respect to a flow f:

$$C = \{x \mid H^+(x) < m\},$$
$$D = \{x \mid x, f(x) > 0, \text{ and } x \text{ connects a } J\text{-node to an } I\text{-node}\}.$$

The number of external preemptions in the schedule produced from a flow f is

$$|C| + |D| - n.$$

We now prove several lemmas about the number of arcs in C and D.

Lemma 2.7.1. *C contains at most $m\cdot(2n - 1)$ arcs.*

Proof. Within each set B_i, $i = 1, 2, \cdots 2n - 1$, only the m arcs which have the largest flow values could be in C. Thus at most $m\cdot(2n - 1)$ arcs are in C. □

Lemma 2.7.2. *With respect to a flow f, if $|D| > 3n - 2$, then there is an augmenting cycle consisting of arcs in D.*

PROOF. There are n J-nodes and $2n-1$ I-nodes incident to the arcs in D. Thus if there are more than $3n-2$ arcs in D, then the graph consisting of the J-nodes, the I-nodes, and D must contain a cycle.

Now for any pair of arcs x, y in D, $H^+(x) \geq m$, $f(x) > 0$, and $f(y) > 0$. Therefore if x and y are incident to the same I-node, then $x^+ \to y^-$, and if they are incident to the same J-node, $x^- \to y^+$. Thus any undirected cycle consisting of arcs in D is an augmenting cycle. □

Once an augmenting cycle is found we compute the capacity δ of the cycle and increase the flow in all forward arcs by δ and decrease the flow in all backward arcs by δ. This produces a new feasible flow but does not change the total flow from s to t.

LEMMA 2.7.3. *If f is a flow which admits an augmenting cycle P consisting of arcs in D, then the flow f' produced by augmenting using P is such that*

$$|D'| < |D|,$$

where D and D' are defined with respect to f and f'.

PROOF. P contains some arc pair (x, y) such that $\text{cap}(P) = \text{cap}(x, y) = \delta$. If $x^+ \to y^-$ on P, then either $\delta = f(y)$ or adding δ to x's flow causes x to appear as a plus arc in an inequality with index less than m. In the first case, $y \notin D'$, and in the second, $x \notin D'$.

If $x^- \to y^+$ on P, then $\delta = f(x)$ and $x \notin D'$. Thus at least one arc in D is not in D'. Since no arc with flow zero has its flow increased and no arc in C has its flow decreased, there are no arcs in D' that were not in D. □

THEOREM 2.7.1. *A solution to the feasibility scheduling problem requires at most $2(m-1)\cdot(2n-1) + m\cdot(2n-1) + 2n - 2$ preemptions.*

PROOF. By Lemmas 2.7.2 and 2.7.3 we can always find a maximum flow f such that $|D| \leq 3n-2$. By Lemma 2.7.1, $|C| \leq m\cdot(2n-1)$; so, the number of external preemptions $= |C| + |D| - n \leq m\cdot(2n-1) + (3n-2) - n = m\cdot(2n-1) + 2n - 2$. Since there are at most $2(m-1)\cdot(2n-1)$ internal preemptions, the result follows. □

2.7.2 *Complexity Analysis.* It follows from Lemmas 2.7.2 and 2.7.3 that $O(n^2)$ augmenting cycles are required to convert a maximum flow f into a maximum flow f' with $|D'| \leq 3n-2$. We now show how to find these augmenting cycles.

(1) Given an initial maximum flow f we sort the flow values at each I-node, compute the sets C and D, and throw away all other arcs. This takes $O(n^2 \log n)$ time.
(2) An augmenting cycle is found using depth-first search on the arcs in D. This takes $O(n^2)$ time.
(3) Computing the cycle capacity takes $O(n^2)$ time.
(4) Updating the flow values and the sorted lists of flow values takes $O(n^2)$ time.
(5) Computing the new C and D sets takes $O(n^2)$ time.

Step (1) is done once and steps (2)–(5) are done $O(n^2)$ times; so, the total time is $O(n^4)$.

3. *The Lmax Problem*

The feasibility algorithm can also be used to solve another scheduling problem. Suppose that each job has a due time rather than an absolute deadline. The lateness

of a job is defined to be its completion time minus its due time. The Lmax problem is to find a schedule which minimizes the maximum value of lateness. To solve this problem, we find the smallest value L such that adding L to all the due times and treating the sums as deadlines yields a feasible scheduling problem. Call this the scheduling problem induced by L. The minimum value of L, which we denote L^*, can be found to within any desired precision using binary search, but if all the data are integers, it is possible to find this value exactly.

There are $O(n^2)$ *critical* values of L such that

$$d_i + L = r_j$$

for some release time and due time. The intervals within which a job can be processed only change when a critical value of L is reached. Thus we can determine which arcs connect J-nodes to I-nodes by doing a binary search over the $O(n^2)$ critical values. This search will find two consecutive critical values L_0 and L' such that L_0 induces an infeasible scheduling problem and L' induces a feasible one.

When the feasibility algorithm is run on the scheduling problem induced by L_0, we will get a set of jobs R_0 such that

$$\mathrm{MAC}(R_0) < P(R_0).$$

We will then calculate the minimum value L_1 such that the problem induced by L_1 has

$$\mathrm{MAC}(R_0) = P(R_0).$$

We then run the feasibility algorithm on the scheduling problem induced by L_1. This will either yield a feasible schedule, in which case $L_1 = L^*$, or a new set of jobs R_1 such that

$$\mathrm{MAC}(R_1) < P(R_1).$$

Continue with L_2, L_3, ... until a feasible schedule is found.

Given a value L_i and its associated set R_i, we now show how to calculate L_{i+1}. Let $\mathrm{MAC}^j(R) = \mathrm{MAC}(R)$ in the schedule induced by L_j, where R is any set of jobs. Associated with the set R_i is a *multiplier* M_i such that

$$M_i \cdot \delta = \mathrm{MAC}^{i+1}(R_i) - \mathrm{MAC}^i(R_i) \qquad \text{for} \quad \delta = L_{i+1} - L_i \quad \text{and} \quad L_{i+1} \le L'.$$

Recall that

$$\mathrm{MAC}(R_i, j) = S_k \cdot \Delta_j, \qquad k = \min\{m, \text{number of jobs in } R \text{ available in interval } j\}.$$

When L_i is increased to $L_i + \delta$, S_k does not change if $\delta \le L' - L_i$. Thus the change in $\mathrm{MAC}(R_i, j)$ is

$$\begin{array}{rl} S_k \cdot \delta & \text{if interval } j \text{ is a release time followed by a due time,} \\ -S_k \cdot \delta & \text{if interval } j \text{ is a due time followed by a release time,} \\ 0 & \text{otherwise.} \end{array}$$

Thus we can easily add up the changes in $\mathrm{MAC}(R_i, j)$ for all intervals to get M_i. By definition,

$$\delta = L_{i+1} - L_i = \frac{P(R_i) - \mathrm{MAC}^i(R_i)}{M_i}.$$

So we can compute L_{i+1} using L_i and M_i.

For any set of jobs R the increase in $\mathrm{MAC}(R, j)$ is at most $S_m \cdot \delta$, and the total increase in $\mathrm{MAC}(R)$ is at most $2n \cdot S_m \cdot \delta$. Thus all multipliers are no larger than

$2n \cdot S_m$, and since each multiplier is a sum of machine speeds, the multipliers will be integral.

LEMMA 3.1. *A feasible schedule will be found after computing $O(n \cdot S_m)$ L_t values.*

PROOF. We need only show that multipliers decrease in size. Now suppose, contrary to our assumption, that (a) $M_{t+1} \geq M_t$ and (b) $L_{t+1} < L^*$. Then

(1) $\text{MAC}^{t+1}(R_{t+1}) - \text{MAC}^t(R_{t+1}) = M_{t+1} \cdot (L_{t+1} - L_t)$, by definition;
(2) $\text{MAC}^{t+1}(R_t) - \text{MAC}^t(R_t) = M_t \cdot (L_{t+1} - L_t) = P(R_t) - \text{MAC}(R_t)$;
(3) $\text{MAC}^{t+1}(R_{t+1}) - \text{MAC}^t(R_{t+1}) \geq \text{MAC}^{t+1}(R_t) - \text{MAC}^t(R_t)$ by (a), (1), and (2);
(4) $\text{MAC}^{t+1}(R_{t+1}) < P(R_{t+1})$, by (b);
(5) $\text{MAC}^{t+1}(R_t) = P(R_t)$, by definition;
(6) $P(R_{t+1}) - \text{MAC}^t(R_{t+1}) > P(R_t) - \text{MAC}^t(R_t)$, putting (4) and (5) into (3).

But R_t contains all unfinished jobs in the schedule induced by L_t; so, for any set of jobs R,

(7) $P(R_t) - \text{MAC}^t(R_t) \geq P(R) - \text{MAC}^t(R)$.

Thus (6) contradicts (7), and therefore $M_{t+1} < M_t$ for $L_{t+1} < L^*$. Since $M_t \leq 2n \cdot S_m$, at most $O(n \cdot S_m)$ multipliers will be used. □

Since L^* will be found after $O(n \cdot S_m)$ calls to the feasibility algorithm, the total time complexity is $O((m^2n^5 + n^6) \cdot S_m)$. This time bound is polynomial in the number of jobs but is not polynomial in the problem description. However, since we have found the topology of the network for the schedule induced by L^*, the Lmax problem can be formulated as a linear programming problem. This formulation will allow us to construct a polynomial search for L^*.

Let q_{ijk} be the amount of time that job "j" runs on machine "i" in interval "k." Gonzalez and Sahni [2] have shown that the processing amounts within an interval can be scheduled iff the total time spent on each job is no greater than the interval length and the total time each processor is used is no greater than the interval length. Now, to determine the interval length, note that if Δ_k is the length of the kth interval in the schedule induced by L_0, then the length of that interval in the schedule induced by $L_0 + L$ is

Δ_k	if the endpoints of the interval are both release times or are both deadlines;
$\Delta_k + L$	if the first endpoint is a release time and the second is a deadline;
$\Delta_k - L$	if the first endpoint is a deadline and the second is a release time.

Define RR to be $\{i \mid$ the ith interval has release times for both of its endpoints$\}$. The sets DD, RD, and DR are similarly defined. Define $r(j)$ to be the index of the interval in which the jth job is first available and $d(j)$ to be the index of the interval in which it is last available.

Thus we can find L^* by solving the following linear program: Minimize L subject to the following constraints:

(1) For $j = 1, 2, \ldots, n$,

$$\sum_{i=1}^{i=m} \sum_{k=r(j)}^{k=d(j)} q_{ijk} \cdot s_i = p_j.$$

These constraints ensure that each job is completed.

(2) For $i = 1, 2, \ldots, m$,

$$\sum_{j=1}^{j=n} q_{ijk} \leq \Delta_k \qquad \text{for each } k \text{ in } RR \text{ or } DD;$$

$$\sum_{j=1}^{j=n} q_{ijk} - L \leq \Delta_k \qquad \text{for each } k \text{ in } RD;$$

$$\sum_{j=1}^{j=n} q_{ijk} + L \leq \Delta_k \qquad \text{for each } k \text{ in } DR.$$

These constraints ensure that the total time per machine is no greater than the interval length.

(3) For $j = 1, 2, \ldots, n$,

$$\sum_{i=1}^{i=m} q_{ijk} \leq \Delta_k \qquad \text{for each } k \text{ in } RR \text{ or } DD.$$

$$\sum_{i=1}^{i=m} q_{ijk} - L \leq \Delta_k \qquad \text{for each } k \text{ in } RD.$$

$$\sum_{i=1}^{i=m} q_{ijk} + L \leq \Delta_k \qquad \text{for each } k \text{ in } DR.$$

These constraints ensure that the total time per job is no greater than the interval length.

(4) All $q_{ijk} \geq 0$.

This system can be represented as: Minimize x_L subject to

$Ax = b$, A is an $O(n^2 \cdot n^3)$ matrix, x an $O(n^3)$ vector, b an $O(n^2)$ vector;
$x \geq 0$.

The solution of this system will be

$$x = B^{-1}b,$$

where B is an $O(n^2 \cdot n^2)$ basis of A. By Cramer's rule,

$$x_i = \frac{|B_i|}{|B|},$$

where B_i is B with the ith column replaced by b.

For $(i_1, i_2, \ldots, i_k)$ a permutation of the integers $1, 2, \ldots, k$, define $f(i_1, \ldots, i_k) = \prod_{j=1}^{j=k} b_{i_j j}$. A permutation is even if it is obtained using an even number of interchanges, otherwise it is odd. We compute the determinent of a k by k matrix by summing f applied to all even permutations and then subtracting f applied to all odd permutations. Thus an upper bound on $|B|$ can be obtained by taking the maximum value f achieves and multiplying this by the number of permutations which result in nonzero values.

Constraints (1)–(3) contribute at most one nonzero coefficient to each column in A. Therefore each column in B has at most three nonzero entries, and there are only $3^{O(n^2)}$ ways we can pick a nonzero element from each column in B. Only $O(n)$ rows of A contain elements greater than one, and s_1 is the largest element of A; so, f cannot achieve a value larger than $s_1^{O(n)}$. Thus the determinent of B is at most $3^{O(n^2)} \cdot s_1^{O(n)}$, and therefore $\log|B|$ is $O(n^2 + n \cdot \log(s_1))$. The largest element of b is at most the largest due date plus the sum of the processing requirements. Call this value $b_{\max}$. Therefore, $\log(|B_i|)$ is $O(n^2 + n \cdot \log(s_1) + \log(b_{\max}))$.

Papadimitriou has shown that if for a rational number $x = y/z$ a number M is known such that $\max\{y, z\} \leq M$, then x can be found using a search with at most $\log M$ steps [8]. Thus we can solve the Lmax problem with $O(n^2 + n \cdot \log(s_1) + \log(b_{\max}))$ calls to the feasibility routine. This time bound is polynomial in the size of the problem description.

4. *Extensions to the Model*

The feasibility algorithm can be used to solve several scheduling problems which are more general than the one considered. Rather than having a release time and a deadline, each job has associated with it a list of time intervals within which it can be processed. Also, rather than having the processors available at all times, each processor has associated with it a list of the time intervals during which it can be used. Using these lists of time intervals, we can divide the time line into intervals such that within each interval the jobs and the processors which are available do not change. It is then easy to construct a flow network to solve this problem. The Lmax problem can also be solved when jobs have release times and due times and each processor has a list of intervals during which it is available.

The techniques used to find a maximum flow can be extended to find a maximum flow in a more general network. In this model each node of the network has constraints on subsets of arcs directed into the node and on subsets of arcs directed from the node. This model is described in [7].

ACKNOWLEDGMENTS. Several people contributed to the final version of this paper. I would like to thank Eugene Lawler, who suggested the problem and contributed several useful ideas; Barbara Simons, who helped improve the exposition; and the referee, who suggested the construction to reduce the number of preemptions from $O(n^2)$ to $O(mn)$.

REFERENCES

(Note References [1, 6, 10] are not cited in the text)

1 BRUNO, J., AND GONZALEZ, T. Scheduling independent tasks with release dates and due dates on parallel machines. Tech. Rep. 213, Computer Science Dep., Pennsylvania State Univ., College Park, Pa., 1976.

2 GONZALEZ, T., AND SAHNI, S. Open shop scheduling to minimize finish time. *J. ACM 23*, 4 (Oct. 1976), 665–679.

3 GONZALEZ, T., AND SAHNI, S. Preemptive scheduling of uniform processor systems. *J. ACM 25*, 1 (Jan. 1978), 92–101.

4. HORN, W. Some simple scheduling algorithms. *Naval Res. Log. Q. 21* (1974), 177–185.

5 HORVATH, E. C., LAM, S., AND SETHI, R. A level algorithm for preemptive scheduling. *J. ACM 24*, 1 (Jan. 1977), 32–43.

6 LABETOULLE, J., LAWLER, E. L., LENSTRA, J. K., AND RINNOOY KAN, A. H.G. Preemptive scheduling of uniform machines subject to release dates. Tech. Rep. BW 99/79, Mathematisch Centrum, Amsterdam, The Netherlands, Sept. 1979.

7 LAWLER, E. L., AND MARTEL, C. Computing maximal 'polymatroidal' network flows. Tech. Rep., Electronics Research Lab., Univ. of California, Berkeley, Calif., Dec. 1980.

8. PAPADIMITRIOU, C. Efficient search for rationals," *Inf. Proc. Lett. 8* (Jan. 2, 1979), 1–9.

9. SAHNI, S., AND CHO, Y. Nearly on line scheduling of a uniform processor system with release times. *SIAM J. Comput. 8* (1979), 275–285.

10 SAHNI, S., AND CHO, Y. Scheduling independent tasks with due times on a uniform processor system. *J. ACM 27*, 3 (July 1980), 550–563.

RECEIVED DECEMBER 1979; REVISED SEPTEMBER 1981, ACCEPTED OCTOBER 1981