SIAM J. COMPUT.
Vol. 18, No. 4, pp. 690-710, August 1989

A FAST ALGORITHM FOR MULTIPROCESSOR SCHEDULING OF UNIT-LENGTH JOBS*

BARBARA B. SIMONS† AND MANFRED K. WARMUTH‡

Abstract. An efficient polynomial time algorithm for the problem of scheduling n unit length jobs with rational release times and deadlines on m identical parallel machines is presented. By using preprocessing, a running time of $O(mn^2)$ is obtained that is an improvement over the previous best running time of $O(n^3 \log\log n)$. The authors also present new NP-completeness results for two closely related problems.

Key words. scheduling, multiprocessor, release time, deadline, parallel computing, computational complexity, NP-complete

AMS(MOS) subject classifications. 05, 68

1. Introduction. We present an efficient polynomial time algorithm for the problem of scheduling n unit length jobs with rational release times and deadlines on m identical parallel machines. The question of how the requirement that the jobs all have the same length affects the problem was first answered in [10], where a polynomial time algorithm with time complexity $O(n^2 \log n)$ is presented for the single machine case. An alternative algorithm with the same time complexity was subsequently obtained by [1]. Finally, an algorithm with time complexity $O(n \log n)$ for the single machine case was presented in [5].

In the multimachine case, the only previously known polynomial time algorithm has a worst-case running time of $O(n^3 \log\log n)$ [11]. We improve this running time to $O(mn^2)$ by doing some preprocessing before the jobs are actually scheduled. This speedup is obtained by generalizing to an arbitrary number of machines the notion of "forbidden regions," which was developed in [5].

If different integer job lengths are allowed, then by a simple reduction from 3 PARTITION [4], determining whether or not a schedule exists is strongly NP-complete, even if $m = 1$ and all release times and deadlines are integers. If m is arbitrary, then there is a similar reduction in which all jobs are released at time 0 and have the same integer deadline.

We strengthen the above NP-completeness result by allowing only a small number of integer job lengths:

(A) Three integer job lengths (1, 3, and q for some integer q), m machines, integer deadlines, but only one overall release time;

(B) Two integer job lengths (1 and q), m machines, integer release times and deadlines.

2. An overview. The algorithm, called BOUNDED_REGION, has two major sections. The first, called the BACKSEQUENCE Algorithm, is the preprocessing section. It determines a set BR of regions in which only a bounded number of jobs can be started in any feasible schedule. The actual scheduling of the jobs is done by SCHEDULE (BR). This procedure schedules jobs as early as possible subject to the restriction that the regions of BR are not violated. Below is a top level description of the algorithm.

* Received by the editors June 16, 1986; accepted for publication (in revised form) June 15, 1988.

† IBM Almaden Research Center, K53-802, 650 Harry Road, San Jose, California 95120-6099.

‡ Computer and Information Sciences, University of California, Santa Cruz, California 95064. The work of this author was supported by Office of Naval Research grant N00014-86-K-0454.

```
ALGORITHM BOUNDED_REGION.
Begin
  Read input;
  BR:= BACKSEQUENCE;
  If BACKSEQUENCE returns failure condition then HALT in failure;
  Call SCHEDULE (BR);
end BOUNDED_REGION.
```

3. Definitions. We shall assume that there is a set J of *n jobs*, $J(1), J(2), \cdots, J(n)$, and a set M of m *machines*. Each job $J(i)$ has a nonnegative integer *processing requirement*, $p(i)$, a nonnegative rational *release time*, $r(i)$, and a nonnegative rational *deadline*, $d(i)$, with $d(i) \geqq r(i)+p(i)$. When we speak of a job $J(i)$ being *released* by time t, we mean that $r(i) \leqq t$. If job $J(i)$ is *started* at time t, then it is *finished* at time $t+p(i)$ and occupies the interval $[t, t+p(i))$. A *schedule* SCH for a problem instance J and M is an assignment of a nonnegative start time $s(i)$ and a machine $m(i)$, $0 \leqq m(i) \leqq m-1$, for each $J(i) \in J$ such that the following conditions hold.

(1) No job is started before its release time or finished later than its deadline, i.e., $s(i) \geqq r(i)$ and $s(i)+p(i) \leqq d(i)$.

(2) No two jobs overlap, i.e., if job $J(i)$ is started at time t on machine $m(k)$, then no other job assigned to $m(k)$ is started in the interval $[t, t+p(i))$.

Note that in our definition of schedule *preemption* is not allowed, that is, once a job has begun execution it cannot be interrupted and consequently must run until it is completed.

Except for the section on NP-completeness, in which $p(i)$ is allowed to assume one of several integer values, we shall assume that $p(i)=1$. For this case a schedule consists of a *sequence* of start times, where a sequence S for n jobs and m machines is a nondecreasing list of n nonnegative start times with the additional constraint that for $i > m$ the ith start time of the sequence is at least one unit greater than the $(i-m)$th start time. This constraint guarantees that no more than m jobs are being processed simultaneously. We say that such a sequence S is of *length n*, i.e., $|S|=n$. If S is a sequence with $|S|=n$, then S_i, $1 \leqq i \leqq n$, denotes the ith start time counting forward from the beginning of the sequence. Likewise, S^i denotes the ith start time counting backward from the end of the sequence. Note that $S_i = S^{n+1-i}$, for $1 \leqq i \leqq n$, and that S_i is the ith smallest and S^i is the ith largest start time of S. Given a problem instance J and M, a sequence S of length n is an *r-sequence* if there exists a 1-1 mapping s (written $s(i)$ instead of $s(J(i))$) from jobs to elements of S such that $r(i) \leqq s(i)$. Similarly, a sequence S of length n is a *d-sequence* if there exists a 1-1 mapping s from jobs to elements of S such that $s(i) \leqq d(i)-1$. The aim is to produce an *rd-sequence* of length n for the set of n jobs, namely, a sequence for which there exists a 1-1 mapping s such that $r(i) \leqq s(i) \leqq d(i)-1$.

Given a schedule SCH for a problem instance J and M, then the sorted list of start times is clearly an *rd*-sequence. For the opposite direction, assume the m machines are numbered $0, 1, \cdots, m-1$. Given an *rd*-sequence of length n for J, the jobs in J can be assigned start times in SCH by applying the following Earliest Deadline Rule first to S_1, then S_2, then S_3, and so on: the unscheduled job with the smallest deadline from among all jobs released by time S_i is started at time S_i on machine $i \bmod m$.

4. Bounded and forced regions. If the release times are integers, the deadlines rationals, and the processing requirement one unit, then it is possible to construct a schedule in linear time if one exists [3], [7]. (Recall that our definition of schedule is an assignment of start times and machines to jobs.) A simple reduction from sorting shows, however, that the construction of an *rd*-sequence requires $\Omega(n \log n)$ time. This time bound is attained by the Earliest Deadline Algorithm [8], which constructs an *rd*-sequence as well as a schedule for that sequence directly from the problem instance in $O(n \log n)$ time. The Earliest Deadline Algorithm computes start time S_i by setting S_i to be the minimum of the release times of the unscheduled jobs and $S_{i-m}+1$, for $i>m$. It then uses the Earliest Deadline Rule to select the job that is assigned start time S_i and machine $i \bmod m$.

This approach fails if arbitrary rational release times and deadlines are allowed. Intuitively, this happens for two reasons. First, jobs may be released during the time that other jobs are already running. (Note that this can be avoided if the release times are integers, since this implies that the start times can be constrained to be integers.) Second, there may be a set of jobs all of which are released on or after some release time, say $r(j)$, and have deadlines less than or equal to some deadline, say $d(i)$, such that the jobs in this set fill up almost the entire interval between $r(j)$ and $d(i)$. If "too many" jobs are started just before $r(j)$, then these jobs will extend into the interval $[r(j), d(i))$, and there will not be enough space in which to schedule all the jobs from the set. Consequently, it is necessary to bound the number of jobs that start less than one unit prior to $r(j)$. In the single machine case, it may be necessary to construct a forbidden region in which no job can start. Such a region has length no greater than one and has a release time as its right endpoint [5]. The m machine case becomes more complex because there can be up to m such intervals, each interval having a different length and a different restriction on the number of jobs that can start in its interior, but having the same release time as its right endpoint.

A *region* is defined to be an interval, either opened at both ends or closed at both ends. The length of a region is always no greater than one and can equal one only if the region is an open interval. The BACKSEQUENCE Algorithm computes regions in which at least k jobs *must* start in any *rd*-sequence. Because each job runs for one unit of time, this implies regions in which at most $m-k$ jobs *are allowed* to start in any *rd*-sequence. We refer to such a region as a *k-forced start region* and an *$(m-k)$-bounded start region*, respectively. We also say that k (respectively, $m-k$) is the *degree* of the k-forced (respectively, $(m-k)$-bounded) start region. If the region $[\alpha, \beta]$ is a k-forced start region, then the region $(\beta-1, \alpha)$ is an $(m-k)$-bounded start region. Note that if more than $m-k$ jobs start in the region $(\beta-1, \alpha)$, then k jobs cannot start in the region $[\alpha, \beta]$. (We require that the parameter k always lies in the range from one to m.) Lemma 0 below follows directly from the definitions of forced and bounded regions.

LEMMA 0. *If $[\alpha, \beta]$ is a k-forced region, then $(\beta-1, \alpha)$ is an $(m-k)$-bounded region.*

We say that an $(m-k)$-bounded start region is *correct* if there is no *rd*-sequence for the problem instance in which more than $(m-k)$ jobs start in the region. If a sequence or schedule has no more than $m-k$ jobs started in an $(m-k)$-bounded region, we say that the $(m-k)$-bounded region has been *avoided*. If more than $m-k$ jobs are scheduled to start in a $(m-k)$-bounded region, then we say that the $(m-k)$-bounded region is *violated*. The terms *correct*, *avoided*, and *violated* are defined similarly for k-forced regions.

Forced and bounded regions always come in pairs. The reader should bear in mind that we are concerned only with the number of jobs that actually start on all m machines in such a region (as opposed to those that have started earlier and are already running in the region). For reasons that will become apparent later, a k-forced region has a release time at its left endpoint and an $(m-k)$-bounded region has a release time as its right endpoint. Bounded regions are open intervals and forced regions are closed intervals.

4.1. How to avoid bounded regions. We distinguish between two modes of constructing sequences. In the *backward* mode we select the latest possible start time by scanning backward starting from a deadline. In the *forward* mode we select the earliest possible start time for each job by scanning forward starting from a release time. BACKSEQUENCE constructs n sequences using the backward mode starting from each of the deadlines. At the end of each iteration, BACKSEQUENCE uses information from the sequences it has constructed to create a new set of up to m bounded regions. It then calls procedure ADD, which computes additional bounded regions implied by those already constructed. Finally, forward sequencing is used in the procedure SCHEDULE to construct the schedule for the problem instance.

Let S be a sequence, $|S| = v$. The subroutine NEXTFORWARD (S, BR, t), given below, returns the smallest start time S^0 such that $S^0 \geqq t$, $S^0 \geqq S^i$, $1 \leqq i \leqq v$, and all the bounded regions of BR are avoided by the start times of $S|S^0$, where $|$ denotes "appended to." Note that if $S' = S|S^0$, then $S'^i = S^{i-1}$, for $1 \leqq i \leqq v+1$. Similarly, the subroutine NEXTBACKWARD (S, BR, t) returns the largest start time S_0 such that $S_0 \leqq t$, $S_0 \leqq S_i$, for $1 \leqq i \leqq v$, and all the bounded regions of BR are avoided by $S_0|S$. We use both procedures iteratively to construct sequences that avoid all bounded regions of BR. For examples of how bounded regions are avoided, see Figs. 3 and 4 in § 11.

```
SUBROUTINE NEXTFORWARD ($S$, BR, $t$) (* returns minimum start time $S^0 \geqq t$
such that the bounded regions of BR are avoided *)
(0) Let $|S| = v$;
    if $v < m$ then $S^m := t-1$; (* This prevents $S^m$ from being undefined *)
(1) $S^0 := \max(t, S^1, S^m + 1)$; (* This guarantees that $S|S^0$ is a sequence *)
(2) process the bounded regions of BR in order of nondecreasing left endpoints:
    let $R$ be the next bounded region of BR, let $m-k$ be the degree of $R$, and let
    $\beta - 1$ and $\alpha$ be the left and right endpoints, respectively, of $R$;
      if $m - k \leqq v$ then
      if $S^{m-k} \in (\beta - 1, \alpha)$ then $S^0 := \max\{S^0, \alpha\}$; (* $S^0$ is increased by the minimum
      amount such that the $(m-k)$-bounded region $(\beta - 1, \alpha)$ is avoided *)
(3) return ($S^0$).
```

```
SUBROUTINE NEXTBACKWARD ($S$, BR, $t$) (* returns maximum start time $S_0 \leqq t$
such that the bounded regions of BR are avoided *)
(0) Let $|S| = v$;
    if $v < m$ then $S_m := t+1$; (* This prevents $S_m$ from being undefined *)
(1) $S_0 := \min(t, S_1, S_m - 1)$; (* This guarantees that $S_0|S$ is a sequence *)
(2) process the bounded regions of BR in order of nonincreasing right endpoints:
    let $R$ be the next bounded region of BR, let $m-k$ be the degree of $R$, and let
    $\beta - 1$ and $\alpha$ be the left and right endpoints, respectively, of $R$;
      if $m - k \leqq v$ then
      if $S_{m-k} \in (\beta - 1, \alpha)$ then $S_0 := \min(S_0, \beta - 1)$; (* $S_0$ is decreased by the minimum
      amount such that the $(m-k)$-bounded region $(\beta - 1, \alpha)$ is avoided *)
(3) return ($S_0$).
```

LEMMA 1. *Let S be a sequence with $|S| = v$ in which all bounded regions of* BR *are avoided. Then* NEXTFORWARD *computes the minimum start time $S^0 \geqq t$ such that $S|S^0$ is a sequence and the bounded regions of* BR *are avoided.*

Proof. Clearly, $S^0 \geqq t$. Since S is a sequence, $S^0 \geqq S^1$ implies that $S^0 \geqq S^i$, $1 \leqq i \leqq v$. Because $S^0 \geqq S^m + 1$ for $v \geqq m$, it follows that $S|S^0$ is a sequence. The regions are processed according to nondecreasing left endpoint; consequently, S^0 is never assigned a value that violates a region that has been previously processed. (We could have processed the bounded regions by right endpoint instead of left endpoint, as long as the approach was used consistently throughout the processing of the bounded regions.) Since S^0 is increased only if it falls within an $(m-k)$-bounded region already containing $m-k$ jobs, and since S^0 is increased the minimum amount required to avoid the region, S^0 is the minimum start time greater than or equal to t that avoids the bounded regions of BR. □

LEMMA 2. *Let S be a sequence, with $|S| = v$, in which all the bounded regions of* BR *are avoided. Then* NEXTBACKWARD *computes the maximum start time $S_0 \leqq t$ such that $S_0|S$ is a sequence and the bounded regions of* BR *are avoided.*

The proof is similar to the proof of Lemma 1.

5. Computing bounded regions by sequencing backward. Without loss of generality, assume that the jobs are sorted by release times and renamed, so that $r(1) \geqq r(2) \geqq \cdots \geqq r(n)$. In addition, let $d'(1), \cdots, d'(n)$ be the set of deadlines listed in sorted order so that $d'(1) \leqq d'(2) \leqq \cdots \leqq d'(n)$. Note that $r(i)$ remains the release time for $J(i)$ but that $d'(i)$ is almost certainly not the deadline for $J(i)$. We call i the *index* of $J(i)$.

Let $\Sigma(i, j)$ be the set of jobs with index less than or equal to j and deadlines less than or equal to $d'(i)$, $1 \leqq i, j \leqq n$, and let $n_{ij} := |\Sigma(i, j)|$. Note that all jobs in $\Sigma(i, j)$ have release times that are at least as large as $r(j)$. The BACKSEQUENCE Algorithm (presented below) iteratively constructs a set of n sequences, SE $[i]$, $1 \leqq i \leqq n$, and a set of bounded regions BR. SE $[i]$ corresponds to deadline $d'(i)$, and the jth iteration $(1 \leqq j \leqq n)$ corresponds to processing job $J(j)$ with release time $r(j)$. At the end of the jth iteration SE $[i]$ contains n_{ij} start times for the jobs $\Sigma(i, j)$. All these start times are at most $d'(i) - 1$ and are as late as possible subject to the constraint that all bounded regions already constructed are avoided. (Details are given in Theorem 1.) Since no job of $\Sigma(i, j)$ is released before $r(j)$, the algorithm stops in failure if SE $[i]_1 < r(j)$ for some i during the jth iteration. Also, if $r(j) < \text{SE}[i]_1$ and $\text{SE}[i]_1 - r(j) < 1$, for some i, then some job(s) of $\Sigma(i, j)$ must be started in the interval $[r(j), d'(i) - 1)$. This constraint triggers the creation of a bounded region with right endpoint $r(j)$. The bounded region guarantees that not too many jobs are started just prior to $r(j)$ such that they are running in the interval $[r(j), d'(i)]$ and interfering with the scheduling of $\Sigma(i, j)$.

```
SUBROUTINE BACKSEQUENCE
Initialize all sequences of SE [1..n] and BR to ∅;
(* SE [i] is the sequence ending at deadline $d'(i)$ *)
(1) For $j = 1$ to $n$ do (* Update SE [i] so that it contains precisely $n_{ij}$ start times *)
Begin for loop
(2) For all $i$, $1 \leqq i \leqq n$, such that $d'(i) \geqq d(j)$ do
    (* $d(j)$ is the deadline of the job with release time $r(j)$ *)
      begin
      (* since each value of $i$ corresponds to a different list, the order in which the
      $d'(i)$'s are processed is irrelevant *)
         $S_0$ := NEXTBACKWARD (SE [i], BR, $d'(i) - 1$);
         SE [i] := $S_0$|SE [i];
      end;
```

```
(3) For $1 \leqq k \leqq m$ do $f_k := \infty$;
    For $1 \leqq i \leqq n$ do
        $f_k := \min(f_k, \mathrm{SE}[i]_k)$;
(4) If $f_1 < r(j)$, then declare "infeasibility" and halt;
(5) For $k = 1$ to $m$ do
        If $f_k - r(j) < 1$ then
            $\mathrm{BR} := \mathrm{BR} \cup \{$the $(m-k)$-bounded region $(f_k - 1, r(j))\}$.

End for loop;
Return BR;
End BACKSEQUENCE.
```

For the correctness proof below we use the notation BR $[j]$ to denote the set BR after the jth iteration of the global for loop of step (1). Examples of both the construction of the sequences and the creation of the bounded regions by BACKSEQUENCE are given in Figs. 1 and 2 in § 11.

THEOREM 1. *Let j be between one and n. The following statements are true at the completion of the processing of the jth iteration of the global for loop of* BACKSEQUENCE.

(i) *If* BACKSEQUENCE *does not halt in failure, then* $\mathrm{SE}[i]_1 \geqq r(j)$ *for all i such that* $\mathrm{SE}[i]_1$ *is not empty, and there is no d-sequence for* $\Sigma(i, j)$ *in which all regions of* $BR[j-1]$ *are avoided whose kth smallest start time is larger than* $\mathrm{SE}[i]_k$, $1 \leqq k \leqq |\mathrm{SE}[i]|$.

(ii) *All regions of* BR $[j]$ *are correct and are no more than one unit in length.*

(iii) *All regions of* BR $[j]$ *are avoided by* SE $[i]$ *for all i such that* $\mathrm{SE}[i]_1$ *is not empty.*

Proof. The proof is by induction on j. Let $j = 1$. Then clearly $\mathrm{SE}[i]_1 \geqq r(j)$ for all i such that $\mathrm{SE}[i]_1$ is not empty (which in this case is all i such that $d'(i) \geqq d(1)$). Otherwise, BACKSEQUENCE would have halted at step (4). Furthermore, any d-sequence could not have the last job in the sequence start later than $\mathrm{SE}[i]_1 = d'(i) - 1$. Since $\mathrm{BR}[0] = \varnothing$, there are no bounded regions for $\mathrm{SE}[i]_1$ to avoid. This proves condition (i) for the base case.

If $\mathrm{BR}[1] = \varnothing$, then conditions (ii) and (iii) also follow. So assume that $\mathrm{BR}[1] \neq \varnothing$. Then $f_1 - r(1) < 1$ in step (5), where $f_1 = d(1) - 1$ (since $\mathrm{SE}[i]_1 = d'(i) - 1$ for $d'(i) \geqq d(1)$). Thus, $[r(1), f_1]$ is a 1-forced region, which by Lemma 0 implies the correctness of the $(m-1)$-bounded region $(f_1 - 1, r(1))$. Note that the latter region is of length no greater than one, since otherwise we would have $f_1 < r(j)$, once again causing the algorithm to halt at step (4). This proves the correctness of (ii) for the base case. Finally, BR [1] is avoided since $\mathrm{SE}[i]_1 \geqq \mathrm{SE}[1]_1 \geqq r(j)$ for all nonempty SE $[i]$, which proves the correctness of (iii).

Assume the lemma holds for $j' - 1$; we now prove it holds for j'.

(i) If SE $[i]$ is unchanged in iteration j', then $j' \notin \Sigma(i, j')$, i.e., $d(j') > d'(i)$ and by the induction assumption (i) holds for $j = j'$. Otherwise, it follows from the induction assumption together with Lemma 2 that the value of S_0 computed by NEXTBACKWARD is at least as large as the smallest feasible start time for SE $[i]$. Furthermore, if one of the other start times of SE $[i]$ were not to satisfy condition (i), then we would have a contradiction to the induction assumption that was made for $j' - 1$. Finally, since by (iii) all regions of $\mathrm{BR}[j'-1]$ are avoided by SE $[i]$ after iteration $j' - 1$, it follows from Lemma 2 that (i) holds for $j = j'$.

(ii) Let $(f_k - 1, r(j))$ be an $(m-k)$-bounded region. Because of the way bounded regions are constructed, we have $r(j) \leqq f_k$, $f_k - r(j) < 1$, and $r(j) - (f_k - 1) < 1$. In addition, for some i, $\mathrm{SE}[i]_1 \leqq \mathrm{SE}[i]_2 \leqq \cdots \leqq \mathrm{SE}[i]_k = f_k$. (Note that $r(j) \leqq \mathrm{SE}[i]_1$.) Consequently, by condition (i), $[r(j), f_k]$ is a k-forced region, which implies by Lemma 0 the correctness of the $(m-k)$-bounded region $(f_k - 1, r(j))$.

(iii) By induction we know that all regions of BR$[j'-1]$ are avoided by SE $[i]$ after iteration $j'-1$ for nonempty SE $[i]_1$. From Lemma 2 we know that if a new start time is computed in step (2), it avoids the regions BR$[j'-1]$. Thus, SE $[i]$ avoids BR $[j'-1]$ after iteration j'. Now, any new regions added to BR $[j']$ have their right endpoints at $r(j')$. But by step (4) of BACKSEQUENCE, the values of SE $[i]$ are at least as large as $r(j')$. Consequently, these new regions are also avoided by SE $[i]$. □

Assume BACKSEQUENCE does not halt in failure. Consider the sequence SE $[i]$ after the completion of BACKSEQUENCE. As the following 1-machine example illustrates, the sequences SE $[i]$ are not necessarily d-sequences for $\Sigma(i, n)$. Let $r(1)=1.2$, $d(1)=4$, $r(2)=1$, and $d(2)=2.5$. Then SE $[2]=\{2, 3\}$. But if $J(2)$ is started at either time 2 or time 3, it will not be completed by its deadline. However, the sequences SE $[i]$ are r-sequences for $\Sigma(i, n)$, as the following trivial algorithm demonstrates. If $J(j)\in\Sigma(i, n)$, i.e., if $d'(i)\geqq d(j)$, then schedule job $J(j)$ at time S_0 computed in step (2) of BACKSEQUENCE during the jth iteration of the global for loop. Clearly, $J(j)$ will start no earlier than $r(j)$ in any of the SE $[i]$ that is updated in step (2), since otherwise BACKSEQUENCE would have halted in step (4).

COROLLARY 1. *For all bounded regions created by* BACKSEQUENCE, $(\beta-1, \alpha)$ *is an* $(m-k)$*-bounded region if and only if* $[\alpha, \beta]$ *is a k-forced region.*

Proof. If $[\alpha, \beta]$ is a k-forced region, then by Lemma 0, $(\beta-1, \alpha)$ is an $(m-k)$-bounded region. So suppose that $(\beta-1, \alpha)$ is an $(m-k)$-bounded region. Since all bounded regions are created in step (5) of BACKSEQUENCE, it follows that for some value of i, SE $[i]$ has k jobs starting in the region $[\alpha, \beta]$. By condition (i) of Theorem 1, it follows that it is necessary for k jobs to begin in the region $[\alpha, \beta]$. Hence, $[\alpha, \beta]$ is a k-forced region. □

COROLLARY 2. *If* BACKSEQUENCE *does not halt in failure, then after the jth iteration of the global loop,* SE $[i]$ *does not violate any bounded regions of the final set of bounded regions computed by* BACKSEQUENCE.

Proof. By (iii) of Theorem 1, SE $[i]$ avoids BR $[j]$; because BACKSEQUENCE does not halt in failure, SE $[i]_1\geqq r(j)$. Since all bounded regions computed at iteration j or later have a right endpoint no greater than $r(j)$, SE $[i]$, as it is computed at the jth iteration, avoids all of BR $[n]$. □

COROLLARY 3. *If* BACKSEQUENCE *halts in failure, then there is no rd-sequence for the problem instance.*

Proof. If the algorithm halts in failure for $i=i^*$, $j=j^*$, then it follows from BACKSEQUENCE together with Theorem 1 that $r(j^*)>f_1=\text{SE}\,[i^*]_1=S_0$ (as computed in step (2) of BACKSEQUENCE). Combining the fact that all jobs in $\Sigma(i^*, j^*)$ have release time at least $r(j^*)$ together with the correctness of (i) of Theorem 1, we get that there is no rd-sequence for $\Sigma(i^*, j^*)$ and hence for the entire problem instance. □

6. Regions may imply additional regions.

If two bounded regions overlap and together cover an interval of length greater than one, then they imply a new bounded region. We call the set of regions created by BACKSEQUENCE *original regions.*

LEMMA 3. *Let* $(\beta-1, \alpha)$ *be an original* $(m-k)$*-bounded region, and let* $(\beta'-1, \alpha')$ *be an original* $(m-k')$*-bounded region such that* $\beta-1<\alpha'<\beta'<\alpha$. *Then* $[\alpha', \beta]$ *is a* $(k+k')$*-forced region,* $(\beta-1, \alpha')$ *is an* $(m-k-k')$*-bounded region, and both regions have length less than* 1.

Proof. Note that the $(m-k)$-bounded region $(\beta-1, \alpha)$ strictly contains the k'-forced region $[\alpha', \beta']$. By Corollary 1, $[\alpha, \beta]$ and $[\alpha', \beta']$ are k- and k'-forced regions, respectively. Since $\beta'<\alpha$, $[\alpha, \beta]$ and $[\alpha', \beta']$ do not overlap. $\beta-1<\alpha'$ implies that

$\beta-\alpha'<1$. Therefore, $[\alpha', \beta]$ is a $(k+k')$-forced region that, by Lemma 0, implies the $(m-k-k')$-bounded region $(\beta-1, \alpha')$ (see Fig. 4).

To prove that the regions are well defined, we must show that $k+k' \leqq m$. Assume for contradiction that $k'>m-k$. Then it follows from BACKSEQUENCE that at the completion of the iteration in which the $(m-k')$-bounded region $(\beta'-1, \alpha')$ is created there is some value I such that there are at least k' elements of SE $[I]$ within the interval $[\alpha', \beta']$. But then at least one of these start times will violate the $(m-k)$-bounded region $(\beta-1, \alpha)$. (Since $\alpha>\alpha'$, we know that the $(m-k)$-bounded region $(\beta-1, \alpha)$ is computed prior to the processing of release time α'.) Therefore, at least one value of SE $[I]$ will be set to $\beta-1$. But $(\beta-1)<\alpha'$, so BACKSEQUENCE will declare infeasibility and halt. Consequently, the region $(\beta'-1, \alpha')$ would not have been declared. Finally, the fact that α' lies between $\beta-1$ and β trivially implies that the regions $(\beta-1, \alpha')$ and $[\alpha', \beta]$ have length less than one. □

The bounded regions implied by Lemma 3 are computed by the subroutine ADD (BR), presented below. Note that the only overlapping regions that are examined by ADD are regions that are in the set BR.

```
SUBROUTINE ADD (BR) (* Computes the additional bounded regions implied by Lemma 3 *)
ADDITIONAL:= ∅:
For all pairs of original bounded regions $(\beta-1, \alpha)$ and $(\beta'-1, \alpha')$ of BR such that $(\beta-1, \alpha)$ is an $(m-k)$-bounded region, $(\beta'-1, \alpha')$ is an $(m-k')$-bounded-region, and $\beta-1<\alpha'=r(j)<\beta'<\alpha$ do

  ADDITIONAL:= ADDITIONAL ∪ {the $(m-k-k')$-bounded region $(\beta-1, \alpha')$}.

BR:= BR ∪ ADDITIONAL;
end ADD.
```

We call the set of regions created by ADD *additional regions*. It follows from the statement of Lemma 3 together with ADD that additional bounded regions are precisely those bounded regions that Lemma 3 proves correct. Note that to incorporate ADD into the algorithm, we need only add the following instruction to the global for loop of step (1) of BACKSEQUENCE.

(6) Call ADD (BR);

A more efficient routine for computing the regions implied by Lemma 3 called FASTADD is given in § 8.

Remark. It is easy to see that Theorem 1 and Corollaries 1, 2, and 3 still hold after the insertion of step (6) in BACKSEQUENCE. Observe that when an additional bounded region $(\beta-1, r(j))$ is created, all entries in the sequences SE [.] are at least $r(j)$. Thus, the additional regions are never violated when they are created, and they are avoided during later iterations in the same manner that the original regions are avoided.

An obvious question is whether or not the additional regions are necessary. As Figs. 3 and 4 in § 11 demonstrate, NEXTFORWARD might return a start time that violates an additional bounded region if the additional bounded regions are not incorporated into the algorithm. Consequently, the resulting sequence is not a d-sequence. Figure 4 in § 11 is an rd-sequence that does not violate the additional bounded region. Thus, additional bounded regions are necessary if BACKSEQUENCE is used to create the regions that are used to construct r-sequences.

7. Producing the final schedule.

In this section we show how to produce an rd-sequence and a schedule using the set of regions created by BACKSEQUENCE and ADD.

```
SUBROUTINE SCHEDULE (BR) (* constructs an $rd$-sequence by iteratively computing the earliest possible start time and then assigning a job to the newest start time *)
$S := \emptyset$;
For $i = 1$ to $n$ do
    (1) Let $r$ be the minimum release time of all unscheduled jobs;
    (2) $S_i :=$ NEXTFORWARD ($S$, BR, $r$); $S := S | S_i$;
    (3) Use the Earliest Deadline Rule to select the job that starts at time $S_i$ on
        machine $i$ mod $m$;
end SCHEDULE.
```

LEMMA 4. SCHEDULE *produces an r-sequence in which all bounded regions are avoided.*

Proof. The selection of r in step (1) guarantees that there is always some unscheduled job that has been released by the time returned by NEXTFORWARD. In addition, by Lemma 1 all bounded regions are avoided. □

Let S be a sequence, and let $J(i)$ be the job that is assigned start time S_j by the Earliest Deadline Rule. We say that $J(i)$ is in *slot* j of sequence S. We also say that slot j *contains* $J(i)$ in sequence S. When speaking of slots, we shall omit the sequence name if the reference is unambiguous.

LEMMA 5. *Let S be the r-sequence produced by* SCHEDULE, *and let $S_0 = -\infty$. If some job in the schedule produced by* SCHEDULE *is completed after its deadline, then there exist v, w, i, and j such that*

(1) *The jobs in slots $v, v+1, \cdots, w$ consist exactly of the set $\Sigma(i, j)$;*

(2) *$S_w + 1 > d'(i)$, i.e., some job of $\Sigma(i, j)$ is finished after its deadline;*

(3) *$S_{v-1} < r(j) \leqq S_v$.*

Proof. Let X be the job in S that is finished later than its deadline and is in the largest numbered slot, say w, of all such jobs. That is, $d(X) = d'(i)$ for some i, and $S_w + 1 > d'(i)$. It follows from the choice of X that all jobs with deadlines no greater than $d'(i)$ are in slots numbered no greater than w.

If all jobs in slots 1 through w have deadlines no greater than $d'(i)$, then by setting $v = 1$ and $j = n$, we get that conditions (1)-(3) of the lemma are satisfied. Otherwise, let $v-1$ be the largest numbered slot less than w that contains a job with deadline greater than $d'(i)$. Let r be the minimum release time of the jobs in slots v through w, and let j be the maximum such that $r = r(j)$. Note that $\Sigma(i, j)$ consists of all jobs with release time at least $r(j)$ and deadline at most $d'(i)$. Since S is an r-sequence, it follows that $r(j) \leqq S_v$. Furthermore, since slot $v-1$ contains a job with a deadline larger than the deadlines of the jobs in slots v through w, it follows from the Earliest Deadline Rule that none of the jobs in slots v through w was available at time S_{v-1}. Therefore, $S_{v-1} < r(j) \leqq S_v$, and condition (3) of the lemma is proved. All the jobs in slots v through w are in $\Sigma(i, j)$, i.e., they must have deadlines at most $d'(i)$ (from the choice of v) and release time at least $r(j)$ (from the choice of j). Also no jobs of $\Sigma(i, j)$ are scheduled in slots 1 through $v-1$, since they are not yet released. Similarly, no jobs of $\Sigma(i, j)$ appear in slots $w+1, \cdots, n$: the jobs of $\Sigma(i, j)$ have deadlines at most

$d'(i)$, and thus they would be late if they were scheduled in slots $w+1$ through n; but w is the last slot with a late job. This shows that condition (1) holds, and completes the proof of the lemma. □

LEMMA 6. *If v, w, i, j are chosen as in Lemma 5 with the additional constraint that n_{ij} is minimal, then the following condition holds*:

(4) *If a job from slots v through w starts at its release time, then it starts at S_v.*

Proof. Assume that Lemma 5 holds for v, w, i, and j, with n_{ij} being minimal, and that condition (4) does not hold. In other words, there is some slot v', $v < v' \leqq w$, such that the job in slot v' has release time $S_{v'}$. Let j' be maximum such that $r(j') = S_{v'}$. Now conditions (1), (2), and (3) of Lemma 5 hold for v', w, i, and j', and $n_{ij'} < n_{ij}$. This contradicts the minimality of n_{ij}. □

For the remainder of this section we assume that S is the sequence constructed by SCHEDULE and that some job in the schedule produced by SCHEDULE is completed after its deadline. Let v, w, i, and j be chosen to satisfy Lemmas 5 and 6. Recall that SE$[i]$ corresponds to deadline $d'(i)$ and that the jobs in $\Sigma(i, j)$ all have deadlines less than or equal to $d'(i)$.

To simplify the notation, let F_{q-v+1} denote S_q, $0 \leqq q \leqq n$. Then the relevant start times in S for the jobs in $\Sigma(i, j)$ are $F_1, F_2, \cdots, F_{n_{ij}}$ instead of $S_v, S_{v+1}, \cdots, S_w$. Condition (3) of Lemma 5 now states that $F_0 < r(j) \leqq F_1$. (Recall that $S_0 = -\infty$.)

We compare the start times F_q to the start times of the sequence SE$[i]$ computed by BACKSEQUENCE in iteration j. Again for the sake of notation, let $B_k = \text{SE}[i]_k$ after iteration j, $1 \leqq k \leqq n_{ij}$. It follows from the construction of SE$[i]$ and from the fact that BACKSEQUENCE did not halt at step (4) that $r(j) \leqq B_1 \leqq B_2 \leqq \cdots \leqq B_{n_{ij}} \leqq d'(i) - 1$.

LEMMA 7. $F_q \leqq B_q$, $1 \leqq q \leqq n_{ij}$.

Before we prove Lemma 7, observe that since its correctness implies that $F_{n_{ij}} \leqq B_{n_{ij}} \leqq d'(i) - 1$, we get an immediate contradiction to the assumption that job X, which is scheduled in slot w in S, is finished later than its deadline of $d'(i)$. Therefore, Lemmas 4 and 7 imply the following theorem.

THEOREM 2. SCHEDULE *produces an rd-sequence and a schedule.*

Proof of Lemma 7. Assume q is the minimum value such that $F_q > B_q$. Since the value of F_q is assigned either in step (1) or in step (2) of NEXTFORWARD and there are three different possible assignments for step (1), we break the analysis into four cases.

Case 1. $F_q = F_{q-1}$ ($S^0 := S^1$ in step (1) of NEXTFORWARD). Since condition (3) of Lemma 5 implies that $F_1 > F_0$, we have in this case that $q > 1$. But $F_q > B_q \geqq B_{q-1}$ leads to $F_{q-1} > B_{q-1}$, and this contradicts the minimality of q.

Case 2. $F_q = F_{q-m} + 1$ ($S^0 := S^m + 1$ in step (1) of NEXTFORWARD). If $q - m \geqq 1$, then $F_{q-m} \leqq B_{q-m} \leqq B_q < F_q = F_{q-m} + 1$. This implies that $B_q - B_{q-m} < 1$, which contradicts the fact that SE$[i]$ is a sequence (Lemma 2).

If $q - m < 1$, then by condition (3) of Lemma 5, $F_{q-m} \leqq F_0 < r(j)$. Now, $F_q > B_q$ and $F_q = F_{q-m} + 1$ imply that $B_q < r(j) + 1$. Since $r(j) \leqq B_1 \leqq \cdots \leqq B_q < r(j) + 1$, $[r(j), B_q]$ is a q-forced region and $(B_q - 1, r(j))$ is an original $(m-q)$-bounded region. Since $B_q - 1 < F_q - 1 = F_{q-m}$ and since $F_0 < r(j)$, we have that $F_{q-m}, F_{q-m+1}, \cdots, F_0$ start in the $(m-q)$-bounded region $(B_q - 1, r(j))$. Therefore, there are $m - q + 1$ start times in the $(m-q)$-bounded region $(B_q - 1, r(j))$, and this region is violated by S. This contradicts Lemma 4, which states that the sequence produced by SCHEDULE does not violate any bounded regions.

Case 3. The job started at F_q in S has release time F_q ($S^0 := t$ and $t = r =$ minimum release time of all unscheduled jobs when NEXTFORWARD is called by

SCHEDULE). By condition (4) of Lemma 6, q can only be 1. Therefore, when NEXTFORWARD is called, all jobs of $\Sigma(i, j)$ are unscheduled. Thus, $r = F_q = r(j)$. Now the assumption that $F_1 > B_1$ implies $B_1 < r(j)$, contradicting condition (i) of Theorem 1.

Case 4. F_q received its final value in step (2) of NEXTFORWARD (to avoid violating a bounded region). This implies both that there exists an original p-bounded region $(\beta - 1, F_q)$ and that S has p start times within that region. These start times are $F_{q-p}, F_{q-p+1}, \cdots, F_{q-1}$. If $q > p$, then $F_{q-p} \leqq B_{q-p} \leqq B_{q-p+1} \leqq \cdots \leqq B_q < F_q$. This implies that SE$[i]$ violates a bounded region, which contradicts Corollary 2.

Assume that $q \leqq p$ which implies that $\beta - 1 < F_{q-p} < r(j)$. Recall that $(\beta - 1, F_q)$ is a p-bounded region. Since all $B_1, B_2, \cdots, B_q$ lie within $[r(j), B_q]$, $(B_q - 1, r(j))$ is an original $(m-q)$-bounded region. Combining inequalities, we have $\beta - 1 < r(j) < B_q < F_q$. Therefore, the conditions of Lemma 3 hold, with $\alpha = F_q$, $m - k = p$, $\beta' = B_q$, $\alpha' = r(j)$, and $k' = q$. Thus, $(\beta - 1, r(j))$ is a $(p-q)$ additional bounded region. We know that the interval $[r(j), F_q)$ contains $q - 1$ start times from S, namely $F_1, \cdots, F_{q-1}$, and that $F_0 < r(j)$. Because the p-bounded region $(\beta - 1, F_q)$ contains p start times from S, it follows that $(\beta - 1, r(j))$ contains $p - q + 1$ start times from S, which violates the additional $(p-q)$-bounded region $(\beta - 1, r(j))$. □

COROLLARY 4 (to Theorem 2). *There is no rd-sequence that has a qth smallest time smaller than the qth smallest time of the sequence S produced by* SCHEDULE, $1 \leqq q \leqq n$.

Proof. Suppose for contradiction that S' is an *rd*-sequence with start times $S'_1, S'_2, \cdots, S'_n$, and let q be the minimum value such that $S'_q < S_q$. As in Lemma 7, we analyze the four cases for the assignment of the value of S_q.

Case 1. $S_q = S_{q-1}$. This is contradicted by $S_{q-1} \leqq S'_{q-1} \leqq S'_q < S_q$.

Case 2. $q > m$ and $S_q = S_{q-m} + 1$. Again we get a contradiction since $S_{q-m} \leqq S'_{q-m} \leqq S'_q < S_q = S_{q-m} + 1$ implies that $S'_q < S'_{q-m} + 1$.

Case 3. S_q is the minimum release time of all the jobs in slots q through n in S. None of the jobs in slots q through n in S can be scheduled in slots 1 through q of S' since $S'_q < S_q$. But this is impossible since it is not possible to place $n - q + 1$ jobs in slots $q + 1$ through n of S'.

Case 4. There exists a k-bounded region $(\beta - 1, S_q)$ and there are k jobs starting in $(\beta - 1, S_q)$ in S. It follows from the definition of q that $S_{q-k} \leqq S'_{q-k} \leqq S'_{q-k+1} \leqq \cdots \leqq S'_q < S_q$, and hence $(\beta - 1, S_q)$ is violated by S', contradicting the assumption that S' is an *rd*-sequence. □

In particular, Corollary 4 implies that the output of SCHEDULE has minimum makespan over all schedules that observe the release time and deadline constraints.

8. An $O(mn^2)$ implementation of BACKSEQUENCE.

We first indicate how to speed up the running time of ADD. For each value of j for which BR(j) has been computed by BACKSEQUENCE, only the largest of each of the (at most) m regions is recorded in step (5). Thus, the total number of regions is $O(mn)$. If $(\beta - 1, r(j))$ is an original k-bounded region, then all possible additional regions that are implied by $(\beta - 1, r(j))$ can be computed in $O(mn)$ time. However, since there might be m original regions computed by BACKSEQUENCE for $r(j)$, the running time could be $O(m^2n^2)$ for the entire algorithm. This can be avoided by methodically checking for overlapping regions.

Let B-REGION be an $m \times n$ array. When an original $(m-k)$-bounded region $(f_k - 1, r(j))$ is created by BACKSEQUENCE, set B-REGION$(m-k, j) := f_k - 1$, $1 \leqq k \leqq m$. Note that B-REGION$(m-1, j) \leqq B$-REGION$(m-2, j) \leqq \cdots \leqq B$-REGION$(0, j)$.

```
SUBROUTINE FASTADD ($j$) (* $j$ is the $j$-value that has just been processed by BACKSEQUENCE *)
(1) $q := j-1$; (* start by comparing with $B$-REGION ($*, j-1$) *)
(2) while $q>0$ and $0<r(q)-r(j)<1$ do (* $B$-REGION ($*, q$) is the next one to check *)
    (3) Compute $K$ and $W$ such that:
        (3a) $K :=$ minimum value of $k$ such that $B$-REGION $(m-k, q) < r(j)$;
        (3b) $W :=$ minimum value of $w$ such that $B$-REGION $(m-w, j) < r(q)-1$;
    (4) If $K$ and $W$ are both defined then (* the necessary condition for additional
        bounded regions *)
        for $k := K$ downto 1 do
        If $f_{(m-W-k)} - 1 > B$-REGION $(m-k, q)$ then replace the $(m-W-k)$-
        bounded region $(f_{(m-W-k)}-1, r(j))$ with ($B$-REGION $(m-k, q)$, $r(j)$)
        (* update $f_{(m-W-k)}$ because of additional region *)
    (5) $q := q-1$; (* compare with the next set of regions *)
end while statement;
end FASTADD.
```

LEMMA 8. *The total amount of time required by all the calls to* FASTADD *is* $O(mn^2)$.

Proof. There are $O(n)$ calls to FASTADD. For each call to FASTADD there are at most $O(n)$ iterations of the while loop with each iteration requiring $O(m)$ time for step (3) and $O(m)$ time for step (4). □

The speedup of FASTADD depends on the following observation.

(*) If $(\beta-1, \alpha)$ is both an $(m-k)$-bounded region and an $(m-k')$-bounded region, for $k' \leqq k$, then the $(m-k')$-bounded region is redundant.

The correctness of (*) follows immediately from the definition of bounded regions and implies that the $(m-k')$-bounded region can be discarded.

LEMMA 9. *The additional bounded regions created by* FASTADD *are correct. Furthermore, all additional bounded regions created by* ADD *that are not created by* FASTADD *are redundant.*

Proof. Lemma 3 implies that any $(m-W-K)$-bounded region created by FASTADD for $k=K$ is correct. Since B-REGION $(m-k, q) \leqq B$-REGION $(m-K, q)$ for $k \leqq K$, it follows that B-REGION $(m-K, q) < r(j)$ implies that B-REGION $(m-k, q) < r(j)$, $k \leqq K$. Since B-REGION $(m-W, j) < r(q)-1$, the conditions of Lemma 3 hold for $k \leqq K$, and therefore the $(m-W-k)$-bounded regions created by FASTADD are correct. By (*) any bounded region implied by B-REGION $(m-w, j)$ for $w \leqq W$ is redundant. The definition of W implies that there are no additional regions created by B-REGION $(m-w, j)$ for $w > W$. □

We remind the reader that ALGORITHM BOUNDED_REGION, referred to below, is the main procedure that is presented at the beginning of this paper.

THEOREM 3. *The running time of* ALGORITHM BOUNDED_REGION *is* $O(mn^2)$.

Proof. We first show how to implement SCHEDULE in $O(mn \log n)$ time. Steps (1) and (3) of SCHEDULE cost only $O(n \log n)$ overall to implement. To implement step (2) efficiently, we make m sorted lists of bounded regions with each list containing no more than n regions as follows. For $0 \leqq k \leqq m-1$, sort the $O(n)$ k-bounded regions by their left endpoints, with ties being broken by the right endpoint. This takes $O(mn \log n)$ time. The ordered lists of regions are examined in step (2) of NEXTFORWARD starting from the 0-bounded regions and ending with the $(m-1)$-bounded regions. A pointer associated with each list of regions denotes the region most recently examined. This pointer is updated to the next region on the list whenever a slot is moved through a region. If the start time of S^0 is increased because of a k-bounded region, step (1) of NEXTFORWARD guarantees that all start times that are subsequently computed will be at least as large as S^0. Therefore, once the start time of a slot is moved through a region, that region will never again have to be examined. Consequently, the total number of times that all regions are examined by NEXTFORWARD is $O(mn)$, and each region is examined in $O(1)$ time.

We now show that BACKSEQUENCE can be implemented in $O(mn^2)$ time. This is easy to verify for all but the calls to NEXTBACKWARD. Note that we have already shown how to implement ADD efficiently (Lemma 9). When we use the same technique as was used to implement step (2) of SCHEDULE, all calls associated with SE $[i]$ can be processed in $O(mn)$ time. Note that the bounded regions are created according to the sorted right endpoint, which is the order required by NEXTBACKWARD. For each of the n sequences SE $[i]$ the list of bounded regions needs to be scanned only once. Thus, BACKSEQUENCE has running time $O(mn^2)$. □

9. NP-completeness results. The NP-completeness proofs are reductions from the following problem that has been shown to be strongly NP-complete in [6].[1]

1/1/1 SCHEDULING.

Instance: Set T of triplets that are to be executed in the time interval $[0, 3f)$, with $|T| = t$. A triplet consists of three unit-length jobs, each of which has an integer release time that is at most $3f-1$.

Question: Is it possible to schedule f triplets of T on one machine in the time interval $[0, 3f)$ such that all $3f$ jobs are started no earlier than their release times?

For the sake of completeness of this paper we present an alternative reduction for the 1/1/1-scheduling problem to that given in [6].

THEOREM 4. *1/1/1 scheduling is strongly* NP-*complete.*

Proof. The reduction is from the strongly NP-complete 3-partition problem [4].

3-PARTITION.

Instance: A multiset Q of $3m$ natural numbers $\{n_i: 0 \leqq i \leqq 3m-1\}$ and a natural number B, with $\sum_{i=0}^{3m-1} n_i = mB$.

Question: Can Q be partitioned into m 3-element multisets $Q_1, Q_2, \cdots, Q_m$ such that the numbers in each Q_i sum to B?

Given an arbitrary instance of 3-partition as defined above, we assume without loss of generality that $n_{p-1} \leqq n_p$, for $1 \leqq p \leqq 3m-1$. We define Q' to be the set of all possible combinations of three elements of Q that sum to B. Each triple of numbers (n_i, n_j, n_k) of Q' corresponds to a triplet of jobs $(J(i), J(j), J(k))$ with release times i, j, and k, respectively. Let T be all such corresponding triplets of jobs. The instance of 1/1/1 scheduling consists of T and the interval $[0, 3m)$, with $f = m$.

Note that $|T| < (3m)^3$ and $r(i) \leqq 3m-1$, for all jobs of T. Thus, even with a unary encoding of the release times this reduction is polynomial.

Given a solution to the 3-partition problem, it is easy to produce a schedule for the 1/1/1-scheduling problem. Each Q_i in the solution to the 3-partition instance corresponds to a triplet of T. We simply schedule the jobs in the corresponding triplets at their release times. The resulting schedule maps the jobs onto the start times $\{0, \cdots, 3m-1\}$.

[1] The definition of 1/1/1 scheduling given in [6] uses deadlines instead of release times as is done here. However, the definitions are symmetric.

Given triplets $T_1, \cdots, T_m$ and a one-processor schedule SCH of the corresponding $3m$ jobs onto $\{0, \cdots, 3m-1\}$, we have

$$\sum_{i=0}^{3m-1} n_i = \sum_{k=1}^{m} \sum_{J(i) \in T_k} n_{s(i)} = mB.$$

Since no job starts before its release time and since $n_{p-1} \leqq n_p$, it follows that

$$\sum_{J(i) \in T_k} n_{s(i)} \geqq \sum_{J(i) \in T_k} n_{r(i)} = B \quad \text{for } 1 \leqq k \leqq m.$$

We conclude that $\sum_{J(i) \in T_k} n_{s(i)} = B$, for $1 \leqq k \leqq m$, and thus the m sets $\{n_{s(i)}: J(i) \in T_k\}$ constitute a solution to the instance of 3-partition. □

PROBLEM A. *Instance*: m identical machines and n jobs, each job having a release time of 0, an integer deadline, and length 1, 3, or q, for some integer q.

Question: Does there exist a valid schedule for the set of jobs?

THEOREM 5. *Problem* A *is strongly* NP-*complete.*

Proof. Let $T = \{T(i): 0 \leqq i \leqq t-1\}$ and f be an instance of 1/1/1 scheduling. Denote the triplet $T(i)$ as $(T(i, 1), T(i, 2), T(i, 3))$, and the release time of $T(i, j)$ as $r(i, j)$. Without loss of generality we assume that $r(i, 1) < r(i, 2) < r(i, 3)$. (If $r(i, j) = r(i, k)$, then one of these release times can be increased by one since the jobs are identical.)

The corresponding instance of Problem A consists of t machines, numbered $0, 1, \cdots, t-1$, and the following set of jobs (see Fig. 5 in § 11):

(1) A set of *filler jobs* $F = \bigcup_{i=1}^{4(t-1)} F(i)$. The subset $F(i)$ contains $t - \lceil i/4 \rceil$ filler jobs $F(i, j)$ each of which has length one such that $d(F(i, j)) = i, 1 \leqq j \leqq t - \lceil i/4 \rceil$.

(2) A set of *chain jobs* $C = \bigcup_{i=0}^{t-1} C(i)$. The ith chain $C(i)$ corresponds to the triplet $T(i)$. It contains $3f+2$ chain jobs $C(i, j)$, $1 \leqq j \leqq 3f+2$, each of which has length $q = 4t$ such that

$$\begin{aligned} d(C(i, j)) &= 4i + jq && \text{for } 1 \leqq j \leqq r(i, 1) + 1, \\ &= 4i + jq + 1 && \text{for } r(i, 1) + 1 < j \leqq r(i, 2) + 1, \\ &= 4i + jq + 2 && \text{for } r(i, 2) + 1 < j \leqq r(i, 3) + 1, \\ &= 4i + jq + 3 && \text{for } r(i, 3) + 1 < j \leqq 3f + 2. \end{aligned}$$

(3) A set of *interval jobs* $I = \{I(j): 1 \leqq j \leqq 3f\}$ each of which has length 1, with $d(I(j)) = (j+1)q - 1$. $I(j)$ corresponds to the interval $[j-1, j)$ in the 1/1/1 schedule.

(4) A set of *pusher jobs* $P = \{P(i): 1 \leqq i \leqq t-f\}$ all of which have length 3, and deadline $(3f+2)q - 1$. The pusher jobs correspond to the unscheduled triplets.

Let T' be a solution to the instance of 1/1/1 scheduling, that is, T' is a subset of f triplets of T for which there exists a schedule $\text{SCH}_{1/1/1}$ on one machine with all jobs appearing in the interval $[0, 3f)$. We construct a schedule SCH_A for the corresponding instance of Problem A. (See Fig. 5 in § 11.)

Machine i, $0 \leqq i \leqq t-1$, contains $4i$ filler jobs in the interval $[0, 4i)$. The chain $C(i)$, for $0 \leqq i \leqq t-1$, is scheduled on machine i. We distinguish between two cases:

Case $T(i) \notin T'$. Job $C(i, j)$, for $1 \leqq j \leqq 3f+1$, is scheduled on machine i finishing at $4i + jq$. The only exception is the last chain job $C(i, 3f+2)$ of $C(i)$. It finishes at $4i + (3f+2)q + 3$ instead of three time units earlier so that a pusher job can be scheduled in the interval $[4i + (3f+1)q, 4i + (3f+1)q + 3)$ on machine i.

Case $T(i) \in T'$. In this case three interval jobs of length 1 are scheduled on machine i instead of one pusher job. Without loss of generality we can assume that $T(i,1)$ precedes $T(i,2)$ and $T(i,2)$ precedes $T(i,3)$ in $\mathrm{SCH}_{1/1/1}$, since $r(i,1) < r(i,2) < r(i,3)$. Otherwise, we could swap the jobs of $T(i)$ to make them appear in the above order.

Assume that $T(i,1)$, $T(i,2)$, and $T(i,3)$ are scheduled in the intervals $[j-1, j)$, $[k-1,k)$, and $[l-1,l)$, respectively, in $\mathrm{SCH}_{1/1/1}$. Schedule the interval jobs $I(j)$, $I(k)$, and $I(l)$ on machine i such that they finish at times $4i+jq+1$, $4i+kq+2$, and $4i+lq+3$, respectively. Schedule the chain jobs of $C(i,r)$ on machine i such that they finish at the following times:

$$\begin{aligned} &4i+rq && \text{for } 1 \leqq r \leqq j,\\ &4i+rq+1 && \text{for } j < r \leqq k,\\ &4i+rq+2 && \text{for } k < r < 3f+2,\\ &4i+rq+3 && \text{for } r = 3f+2. \end{aligned}$$

This completes the construction of $\mathrm{SCH}_{\mathrm{A}}$. Note that all jobs of the instance of A are completed before or at their deadlines.

To show the reverse, suppose we are given a schedule $\mathrm{SCH}_{\mathrm{A}}$ for the instance of A. We want to show that $\mathrm{SCH}_{\mathrm{A}}$ determines a subset T' of f triplets of T such that there exists a schedule $\mathrm{SCH}_{1/1/1}$ for T' on one machine in the interval $[0, 3f)$.

CLAIM A. (i) The jobs $C(i,1)$, for $0 \leqq i \leqq t-1$, are scheduled on different machines in $\mathrm{SCH}_{\mathrm{A}}$. Without loss of generality, assume $C(i,1)$ is scheduled on machine i.

(ii) The filler jobs are scheduled in $\mathrm{SCH}_{\mathrm{A}}$ such that the interval $[0, 4i)$ of machine i is occupied with filler jobs and $C(i,1)$ is scheduled on machine i such that it finishes at $4i+q$, $0 \leqq i \leqq t-1$. (See Fig. 5 in § 11.)

(iii) All jobs of the chain $C(i)$ are scheduled on the machine i in $\mathrm{SCH}_{\mathrm{A}}$. The job $C(i,j)$ can have one of at most four different finishing times in $\mathrm{SCH}_{\mathrm{A}}$: $4i+jq$, $4i+jq+1$, $4i+jq+2$, or $4i+jq+3$.

(iv) Machine i, $0 \leqq i \leqq t-1$, contains either three interval jobs of length one, or one pusher job of length 3.

(v) The set $T' = \{T(i)$: machine i in $\mathrm{SCH}_{\mathrm{A}}$ contains interval jobs$\}$ is a solution of the instance of 1/1/1 scheduling. Assume $\mathrm{SCH}_{\mathrm{A}}$ is normalized, where by normalized we mean that no interval job can be moved to the right by *swapping* it with the chain job or interval job that is following it on the same machine. Then $I(j)$, for $1 \leqq j \leqq 3f$, is scheduled in the interval $[jq, (j+1)q-1)$ in $\mathrm{SCH}_{\mathrm{A}}$. We can construct a schedule $\mathrm{SCH}_{1/1/1}$ for T' as follows. If $I(j)$ is the kth interval job on machine i, $1 \leqq k \leqq 3$, then schedule job $T(i,k)$ in the interval $[j-1, j)$ of $\mathrm{SCH}_{1/1/1}$.

Proof of (i). Assume two jobs $C(k,1)$ and $C(k',1)$, $k < k'$, are scheduled on the same machine in $\mathrm{SCH}_{\mathrm{A}}$. Since chain jobs have length q, the earliest time both jobs can finish is time $2q$. This contradicts the assumption that there is a feasible schedule for Problem A since, for $0 \leqq i \leqq t-1$, $d(C(i,1)) \leqq d(C(t-1,1)) = 4(t-1)+q < 2q$.

Proof of (ii). The job $C(0,1)$ has to start at time 0 and finish at its deadline q. Assume $1 \leqq j \leqq 4$. Then, since $|F(j)| = t-1$, and the jobs of $F(j)$ have deadline j, the intervals $[j-1, j)$ of machines 1 through $t-1$ of $\mathrm{SCH}_{\mathrm{A}}$ must be occupied with the filler jobs of $F(j)$. This implies that $C(1,1)$ is started at time 4 in $\mathrm{SCH}_{\mathrm{A}}$ and finished at its deadline $4+q$. Simple induction on j implies that the jobs of $F(j)$ are scheduled on machine $\lceil j/4 \rceil$ through t and that $C(i,1)$ starts at $4i$ and finishes at its deadline $4i+q$.

Proof of (iii). We define the lexicographic order $\leqq$ on the tuples (i,j) as follows. If $i < t-1$, then (i,j) immediately precedes $(i+1, j)$; otherwise, $i = t-1$ and $(t-1, j)$

immediately precedes $(0, j+1)$. We use this lexicographic order to do an induction on the index tuple (i, j) of $C(i, j)$.

The base case that (iii) holds for $(i, j) \leqq (t-1, 1)$ follows from claim (ii). Assume that (iii) holds for all jobs $C(i, j)$ such that $(i, j) \leqq (i', j')$. Let (i^*, j^*) be the next tuple after (i', j') in the lexicographic ordering. By induction we know that machine i is busy in the interval $[4i+(j-1)q+3, 4i+jq)$, for $(i, j) \leqq (i', j')$, since it is executing $C(i, j)$. In particular, all machines i such that $i \neq i^*$ are busy in the interval $[4i^*+(j^*-1)q+3, 4i^*+(j^*-1)q+4)$. Since $d(C(i^*, j^*)) \leqq 4i^*+j^*q+3$, we conclude that $C(i^*, j^*)$ has to be executed on machine i^*. Clearly, $C(i^*, j^*)$ has to be completed by its deadline, since SCH_A is a valid schedule. The fact that $C(i^*, j^*)$ has to be finished at time $4i^*+j^*q$ or later follows by induction, since $C(i^*, j^*-1)$ is finished at time $4i^*+(j^*-1)q$ or later. This completes the proof of (iii).

Proof of (iv). From (i) and (iii), we know that machine i of SCH_A receives $4i$ filler jobs of length 1 and $3f+2$ chain jobs of length q. The deadlines of the pusher jobs and interval jobs are no greater than $(3f+2)q-1$. Therefore, the last job on machine i, for $0 \leqq i \leqq t-1$, is the chain job $C(i, 3f+2)$. Since $d(C(i, 3f+2)) = 4i+(3f+2)q+3$, it follows that there are exactly three units available on machine i for interval and pusher jobs. This gives us $3t$ time units for interval and pusher jobs on all t machines. Since there are $3f$ interval jobs of length 1 and $t-f$ pusher jobs of length 3, we conclude that each machine i of SCH_A contains either three interval jobs or one pusher job.

Proof of (v). Assume SCH_A is normalized and assume by contradiction that $I(j)$ is an interval job that is scheduled in the interval $[lq, (l+1)q-1)$, for some $l<j$. If $I(j)$ is scheduled on machine i, then by (iii), it has to be scheduled between the chain jobs $C(i, l)$ and $C(i, l+1)$ in the interval $[4i+lq, 4i+lq+3)$. Since $d(C(i, l+1)) \leqq d(C(t-1, l+1)) \leqq (l+2)q-1 \leqq d(I(j))$, we can move $I(j)$ past $C(i, l+1)$ as follows. Iteratively swap $I(j)$ with the interval jobs between $I(j)$ and $C(i, l+1)$, and finally, swap $I(j)$ with $C(i, l+1)$. The fact that $I(j)$ could be swapped with a chain or interval job to its right contradicts the fact that SCH_A is normalized. We conclude that $l=j$ and that $I(j)$ is scheduled in the interval $[jq, (j+1)q-1)$ in SCH_A.

To complete the proof of (v), we need to show that the constructed schedule $\text{SCH}_{1/1/1}$ is a valid schedule for T', that is $r(i, k) \leqq j-1$, for $1 \leqq k \leqq 3$. By (iii), we know that $I(j)$ is scheduled between $C(i, j)$ and $C(i, j+1)$ in the interval $[4i+jq, 4i+jq+3)$ on machine i. Since $k-1$ interval jobs are scheduled before $I(j)$ and $3-k$ after $I(j)$ on machine i, it follows that $I(j)$ is executed in the interval $[4i+jq+k-1, 4i+jq+k)$ on machine i. This implies that $d(C(i, j+1)) \geqq 4i+(j+1)q+k$ and therefore by the definition of the deadlines of the chain jobs, we have $r(i, k)+1<j+1$, which is equivalent to $r(i, k) \leqq j-1$. This completes the proof of Claim A and the theorem. □

In Problem B instead of having jobs of length 3 as in Problem A we have arbitrary integer release times as well as arbitrary integer deadlines.

PROBLEM B. *Instance*: m identical machines and n jobs, each job having a release time and deadline that are arbitrary integers, and having length 1 or q, for some integer q.

Question: Does there exist a valid schedule for the set of jobs?

The reduction for B is similar to the reduction for A. The jobs of length 3 in the reduction of A are replaced in B by an additional set of jobs R, all of which have length 1 or q.

THEOREM 6. *Problem* B *is strongly* NP-*complete.*

Proof. In Problem A the length-3 jobs were used to guarantee that either three or zero interval jobs were scheduled on a machine. In the latter case the machine received a pusher job of length 3. In the reduction below the same set of jobs is used as in Theorem 5 except that the length-3 pusher jobs are replaced by R.

The *slack* $\Delta(J)$ of job J is defined to be the difference $d(J)-r(J)-p(J)$. The jobs of R have release time at least $(3f+2)q$. Thus, for any machine i, $0 \leqq i \leqq t-1$, in a valid schedule all jobs of R follow all other jobs. The set R consists of the following sets of jobs:

(1) A set of *frame jobs* $A=\{A(i)\colon 0 \leqq i \leqq t-1\}$, with $r(A(i))=(3f+t+3)q+4i+3$, $d(A(i))=r(A(i))+q$, and $p(A(i))=q$. Note that $\Delta(A(i))=0$.

(2) A set of *glider jobs* $G=\bigcup_{i=1}^{t} G(i)$, with $G(i)=\{G(i,j)\colon j \neq t-i, 0 \leqq j \leqq t-1\}$, $r(G(i,j))=(3f+2+j)q+4i$, $d(G(i,j))=r(G(i,j))+q+3$, and $p(G(i,j))=q$. Note that $\Delta(G(i,j))=3$.

(3) A set of *early jobs* $E=\{E(i)\colon 0 \leqq i \leqq t-1\}$, with $r(E(i))=(3f+2)q+4i$, $d(E(i))=(3f+t-i+3)q+4i+3$, and $p(E(i))=q$.

(4) A set of *late jobs* $L=\{L(i)\colon 0 \leqq i \leqq t-1\}$ with $r(L(i))=r(E(i))+3$, $d(L(i))=d(E(i))-3$, and $p(L(i))=q$.

(5) A set of *stuffer jobs* $U=\{U(i)\colon 1 \leqq i \leqq 3(t-f)\}$, with $r(U(i))=(3f+t+3)q$, $d(U(i))=(3f+t+4)q-1$, and $p(U(i))=1$.

The jobs of R can be scheduled such that $A(i)$, $G(i)$, $E(i)$, and $L(i)$, for $0 \leqq i \leqq t-1$, appear on machine i in one of the following two ways. (See Fig. 6 in § 11.)

Case 1. $E(i)$ starts at $(3f+2)q+4i$ on machine i. $E(i)$ is followed by $G(i,1)$, $G(i,2),\cdots,G(i,t-i-1)$, $L(i)$, $G(i,t-i+1),\cdots,G(i,t)$, three stuffer jobs, and $A(i)$.

Case 2. $L(i)$ starts at $(3f+2)q+4i+3$ on machine i. $L(i)$ is followed by $G(i,1)$, $\cdots,G(i,t-i-1)$, $E(i)$, $G(i,t-i+1),\cdots,G(i,t)$, $A(i)$.

Since $|U|=3(t-f)$, case 1 occurs exactly $t-f$ times and case 2 occurs f times.

To show that R has to be scheduled as outlined above, we prove the following claim (see also Fig. 6 in § 11). Note that the jobs of R have the same function as the pusher jobs of the previous reduction.

CLAIM B. For $0 \leqq i \leqq t-1$, (i) Machine i receives $t+2$ jobs of length q of R. The last job of R of length q on machine i is $A(i)$.

(ii) The vth job of R of length q on machine i starts either at time $(3f+2+v-1)q+4i$ or no more than three time units later. The job $G(i,j)$, for $j \neq t-i$, $1 \leqq j \leqq t$, is the $(j+1)$st job of R of length q on machine i.

(iii) The jobs $E(i)$ and $L(i)$ share the first and $(t-i+1)$st position of the length q jobs of R on machine i.

(iv) The first job of R is an early job on eactly $(t-f)$ machines, and a late job on the remaining f machines. In the case where the first job on machine i is $E(i)$, the start time for $E(i)$ is $(3f+2)q+4i$, and three stuffer jobs are scheduled before $A(i)$. In the case where the first job on machine i is $L(i)$, $L(i)$ begins at $(3f+2)q+4i+3$.

Proof of (i). Since all frame jobs $A(i)$ start at $(3f+t+3)q+4i+3$ and run for $q=4t$ units of time, exactly one of the t frame jobs has to be scheduled on each of the t machines. In statement (iii) of Claim A (of the reduction for problem A), it was shown that the last job not in R on machine i finishes either at $(3f+2)q+4i$ or no more than three time units later. Thus, if machine i were to run more than $t+2$ jobs of R of length q, then the $(t+3)$rd job could not be finished earlier than time $(3f+t+5)q+4i$. Since all deadlines of the jobs of R are smaller than $(3f+t+5)q$, we conclude that each machine runs at most $t+2$ jobs of R of length q. Because R contains t^2+2t jobs of length q and there are t machines, each machine receives exactly $t+2$ jobs of R of length q.

From part (iii) of Claim A, we know that the first job of R of length q on machine i can start no earlier than $(3f+2)q+4i$. Thus the $(t+2)$nd such job can start no earlier than $(3f+t+3)q+4i$. By a simple induction, it can be shown that this implies that $A(i)$ is scheduled on machine i.

Proof of (ii). Since $r(G(i,j))=(3f+j+2)q+4i$ and $\Delta(G(i,j))=3$, the second part of statement (ii) is implied by the first part. To prove the first part, note that by statement (iii) of Claim A the first job of length q of R can start no earlier than $(3f+2)q+4i$. Additionally, by statement (i) above, machine i receives $t+2$ jobs of R of length q, with the last such job starting at time $(3f+t+3)q+4i+3$.

Proof of (iii). By (i) and (ii) above, we know that every job of E and L is scheduled as the first or $(t-i+1)$st job of R of length q on some machine i. It is easy to see that the first and tth job of R of length q of machine 0 are $E(0)$ and $L(0)$. No other job of E and L is released early enough to be the first, and no job of E or L has a deadline large enough to be the $(t+1)$st job of machine i. By a simple induction it can be shown that $E(i)$ and $L(i)$ are the first and the $(t-i+1)$st jobs of R of length q on machine i.

Proof of (iv). If $L(i)$ is the first job of machine i, then by (ii) above, it must start at its release time of $(3f+2)q+4i+3$. In this case, no stuffer jobs can be scheduled on machine i.

If $L(i)$ is the $(t-i+1)$st job on machine i, then by (ii) above it must finish at its deadline of $(3f+t-i+3)q+4i$. This implies that $E(i)$ is started at its release time of $(3f+2)q+4i$ and that no more than three stuffer jobs can be scheduled in the interval $[(3f+t+3)q+4i, (3f+t+3)q+4i+3)$. Since there are $3(t-f)$ stuffer jobs of length 1, $E(i)$ must be the first job of R on at least $t-f$ machines.

Using arguments similar to the reduction of Problem A, the existence of $3f$ interval jobs implies that there are at least f machines whose last job finishes later than $(3f+2)q+4i$. We conclude that exactly $t-f$ machines start with a job from $E(i)$, and the remaining f machines start with a job from $L(i)$.

This completes the proof of Claim B. It follows from the proof for Problem A that the f machines starting with a job from $L(i)$ will contain the interval jobs. □

10. Open problems.

(1) We conjecture that ALGORITHM BOUNDED_REGION can be modified to obtain a running time of $O(mn \log n)$. We have already shown that SCHEDULE runs in $O(mn \log n)$ time. It is also possible, using a technique of [5] to avoid computing the additional regions. This technique determines whether or not start times being computed by BACKSEQUENCE *might* violate a newly computed bounded region by comparing the start times mod 1. If the answer is determined to be yes, then the start times are decremented accordingly. It can be shown that this process will not cause a feasible problem instance to be incorrectly designated infeasible. It is also straightforward to show that this precomputation corresponds to the additional regions.

In [5], it was shown that the running time for the single machine version of this problem is $O(n \log n)$. This running time was obtained by eliminating the need to compute all sequences for those jobs with deadline at least as great as the deadline of the job whose release time is currently being processed. We believe that similar techniques should work for the general problem, but we have not been able to solve the problem of how to capture the notion of different degrees of boundedness, i.e., the fact that we have regions with restrictions on the number of jobs that can start within them.

(2) Can the NP-completeness proofs be made tighter, or are there polynomial time algorithms for the more constrained problems? In particular, what can be said about variations of Problems A and B in which there is only a single machine or there

are a constant number of machines. There ia polynomial time algorithm for the single machine version of Problem B [2], but we have been unable to generalize the algorithm to the case where the running time of a job is one of two arbitrary integers, as opposed to being either one or an arbitrary integer.

11. Figures.

We use the example below to illustrate the BOUNDED_REGION Algorithm.

Example $m=2$, $n=7$. The release time and deadline of a job are listed as an ordered pair after the job:

$$A(0, 4.4),\ B(0.2, 2.2),\ C(0.3, 2.3),\ D(0.5, 1.8),\ E(1.6, 3.4),\ F(2.4, 3.6),\ G(2.4, 4.0).$$

The table constructed for the above example is given in Fig. 2. In column j we list only those entries that are changed in iteration j.

$\mathrm{BR}[1]=((1.6, 2.4), \varnothing)$ $\quad \mathrm{BR}[2]=((1.6, 2.4), (2, 2.4))$
$\mathrm{BR}[3]=((1, 1.6), \varnothing)$ $\quad \mathrm{BR}[4]=((-0.2, 0.5), \varnothing)$
$\mathrm{BR}[5]=((-0.2, 0.3), \varnothing)$ $\quad \mathrm{BR}[6]=((-0.7, 0.2), (0.0, 0.2))$
$\mathrm{BR}[7]=((-0.7, 0.0), \varnothing)$

FIG. 1. *The original bounded regions computed by* BACKSEQUENCE.

release times / deadlines	F	G	E	D	C	B	A
	2.4	2.4	1.6	0.5	0.3	0.2	0.0
D 1.8				0.8			
B 2.2				1.2		1.0[3]	
C 2.3				1.3	1.0[2]	0.3	
E 3.4			2.4	2.4	1.4	1.0[4]	
F 3.6	2.6		2.6	1.6	1.6	0.6	
G 4.0	3.0	3.0	2.0	1.6[1]	1.0	0.6	
A 4.4	3.4	3.4	2.4	2.4	1.4	1.0[4]	0.4

[1] was 2.0
[2] was 1.3
[3] was 1.2
[4] was 1.4

FIG 2. *The sequences computed by* BACKSEQUENCE.

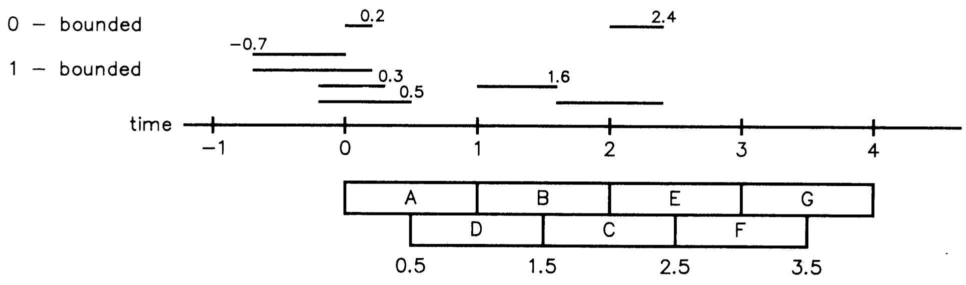


FIG 3. *Output of* SCHEDULE (BR), *where* BR *consists only of original regions.*

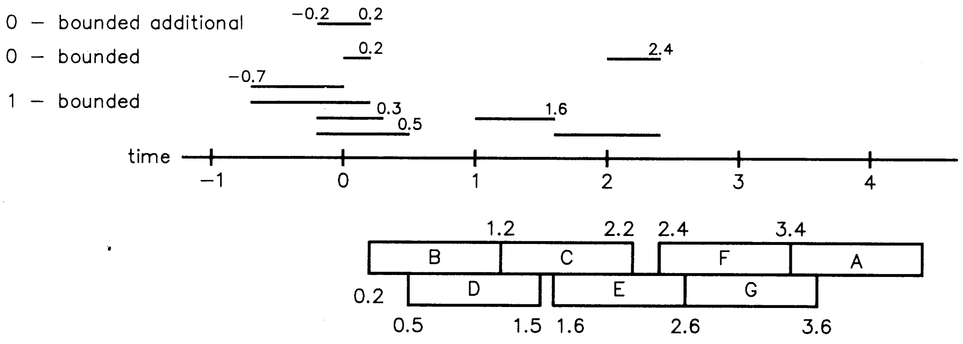
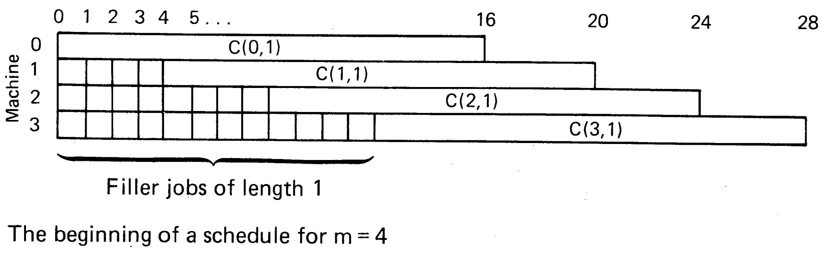


FIG. 4. *The final rd-sequence produced by* SCHEDULE (BR), *with additional regions included in* BR.



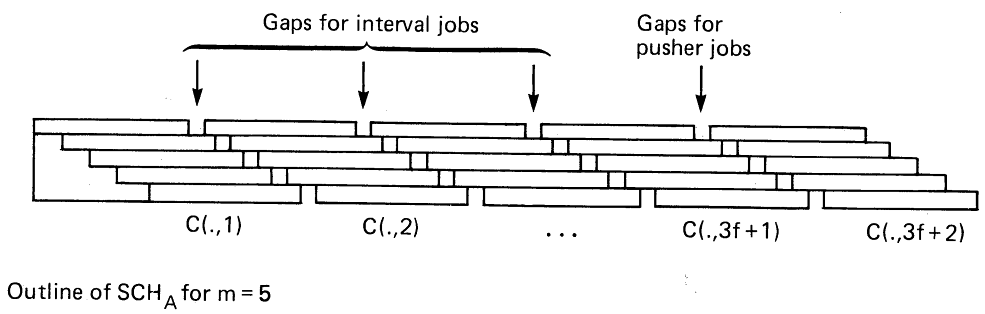


FIG. 5. SCH_A.

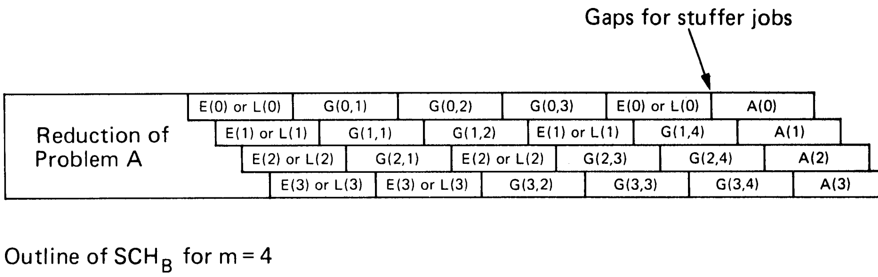


FIG. 6. SCH_B.

The bounded regions computed at each iteration of the for loop of BACK-SEQUENCE are listed in Fig. 1. The first entry in the ordered pair is the 1-bounded region, the second is the 0-bounded region. The symbol $\varnothing$ is used when there is no bounded region.

Figure 3 shows how NEXTFORWARD avoids the original bounded regions. For the purpose of illustration, we ignore the additional regions. Observe that if only the original bounded regions listed above are used, C is completed later than its deadline, so the sequence is not a d-sequence. This follows because BR[4] and BR[6] satisfy the conditions of Lemma 3 and hence create the additional 0-bounded region $(-0.2, 0.2)$.

Acknowledgments. The authors are very grateful to Danny Dolev for the many hours he spent discussing the algorithm with us. We also are very appreciative of the comments and criticisms we received from the two very thorough referees.

REFERENCES

[1] J. CARLIER, *Problème à une machine dans le cas où les tâches ont des durées égales*, Tech. Report, Institut de Programmation, Université de Paris VI, Paris, France, 1979.
[2] D. DOLEV, B. SIMONS, AND M. WARMUTH, private communication.
[3] G. FREDERICKSON, *Scheduling unit-time tasks with integer release times and deadlines*, Inform. Process. Lett., 16 (1983), pp. 171-173.
[4] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*, W. H. Freeman, San Francisco, CA, 1978.
[5] M. R. GAREY, D. S. JOHNSON, B. B. SIMONS, AND R. E. TARJAN, *Scheduling unit-time tasks with arbitrary release times and deadlines*, SIAM J. Comput., 10 (1981), pp. 256-269.
[6] H. GABOW AND M. STALLMANN, *Scheduling multitask jobs with deadlines on one processor*, manuscript.
[7] H. GABOW AND R. E. TARJAN, *A linear-time algorithm for a special case of disjoint set union*, J. Comput. System Sci., 30 (1985), pp. 209-221.
[8] J. R. JACKSON, *Scheduling a production line to minimize maximum tardiness*, Res. Report 43, Management Science Research Project, University of California, Los Angeles, CA, 1955.
[9] J. K. LENSTRA, A. G. H. RINNOOY KAN, AND P. BRUCKER, *Complexity of machine scheduling problems*, Ann. Discrete Math., 1 (1977), pp. 343-362.
[10] B. B. SIMONS, *A fast algorithm for single processor scheduling*, in 19th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Long Beach, CA, 1980, pp. 246-252.
[11] ———, *Multiprocessor scheduling of unit length jobs with arbitrary release times and deadlines*, SIAM J. Comput., 12 (1983), pp. 294-299.
[12] J. D. ULLMAN, NP-*complete scheduling problems*, J. Comput. System Sci., 10 (1975), pp. 384-393.