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

005

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 mod 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 α 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 α 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 ≦ i ≦ n, such that d'(i) ≧ 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
BR := 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 BR$[0] = \varnothing$, there are no bounded regions for $\mathrm{SE}[i]_1$ to avoid. This proves condition (i) for the base case.

If BR$[1] = \varnothing$, then conditions (ii) and (iii) also follow. So assume that 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 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* j*th 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