A Fast Algorithm for Single Processor Scheduling

Barbara Simons *

Computer Science Division
University of California
Berkeley, California 94720

1. Introduction

Suppose we are given a single processor and a set S of n jobs. For each job X there is a release time r_X and a deadline d_X, with r_X and d_X nonnegative real numbers. A schedule is *feasible* if there is no time at which more than one job is being run and if every job in the schedule is begun no earlier than its release time and is completed by its deadline. The problem is to find a feasible schedule in which each job is run for the same amount of time p. The processing is nonpreemptive in that once a job is started it continues executing until it has run for precisely p units of time.

We present a recursive algorithm which in $O(n^2 \log n)$ time produces a schedule with the earliest possible completion time or determines that none exists. In the special case where all the parameters are integers the algorithm runs for $O(mn \log n)$ time, where $m = \min\{n,p\}$.

This problem is mentioned as an open question by Garey and Johnson [3]. It also arises in enumeration methods for the general job-shop scheduling problem [9,10].

The generalization of this problem, in which each job X has associated with it a running time p_X, is known to be unary NP-complete [11]. If, however, we assume that $r_X < r_Y$ implies that $d_X < d_Y$, then there is a polynomial time algorithm for minimizing the number of tardy jobs [8].

2. A Simple Algorithm

If the release times and deadlines are integer multiples of p, a simple algorithm [2,6] is guaranteed to produce a schedule with the earliest possible completion time, provided that such a schedule exists. This algorithm selects to run at time $t = 0$ the job with the earliest deadline from among all jobs released at time 0. After deleting this job from the pool of unscheduled jobs, it repeats the above procedure for $t = 1, 2, 3$, etc. If at any time all the the jobs which have been released have been scheduled, it sets t to be the earliest release time of the unscheduled jobs and proceeds as before. We shall refer to this algorithm as the *naive algorithm.*

The following example shows that the naive algorithm does not solve the general problem without considerable modification. Suppose that $p = 3$ and we are given jobs A and X with $r_A = 0$, $d_A = 7$, $r_X = 1$, and $d_X = 4$. The naive algorithm will select A to run at time 0. It is clear, however, that X should be scheduled to run at time 1, and A at time 4.

3. The General Algorithm

We now present the algorithm which solves the general scheduling problem. It consists of a main procedure and two recursive subroutines. The actual scheduling of jobs is always done via the naive algorithm. The essential differences between this algorithm and the naive one are that jobs may be removed from the partial schedule and that sets of jobs may be collected together to form scheduling subproblems.

The main procedure begins by scheduling jobs using the naive algorithm. If at some point the naive algorithm attempts to schedule a job X at a time t_X and there is insufficient time to run X (i.e. $t_X + p > d_X$), it designates X to be a *crisis job (c.j.)* and invokes the CRISIS subroutine. After the subroutine has repaired the schedule, the main procedure continues using the

* Supported by NSF grant MCS77-09906.

Presented at the Nineteenth Annual Symposium on Foundations of Computer Science, October 1978

naive algorithm. Either it succeeds in scheduling all the jobs or there is another crisis, in which case it again calls the CRISIS subroutine.

The CRISIS subroutine is called whenever the naive algorithm encounters a c.j. X. Assume for the moment that X is the first job to have a crisis. In this case the subroutine backtracks over the current schedule looking for a job with a deadline greater than d_X. The first such job to be encountered, which is called *Pull (X),* is removed from its current position in the schedule and returned to the pool of unscheduled jobs. The subroutine then uses the naive algorithm to reschedule the set of jobs consisting of X together with those jobs that are jumped over in the backtracking. This set of jobs is called a *restricted set (r.s.).* If $A = \text{Pull}(X)$, then we refer to the restricted set "triggered" by X as $S(A,X]$. (The interval is "open" on the left because A is not included and "closed" on the right because X is included). Note that by definition $d_A > d_X$, and for all Y in $S(A,X]$, $d_Y \leqslant d_X$.

To describe the manner in which the algorithm handles subsequent crises, we need a few more definitions. In what follows S' will always signify either the original set S or some r.s. If $S(B,Z]$ is a r.s. which is properly contained in S', then $S(B,Z]$ is called an *inner r.s.* of S'. A *first level r.s.* of S' is an inner r.s. of S' which is not contained in any other inner r.s. of S'. A job of S' which is not an element of any inner r.s. of S' is called a *first level job* of S'.

Continuing with the description of the CRISIS subroutine, suppose a job X, which is a first level job of S', has a crisis. The subroutine searches for $\text{Pull}(X)$ as in the initial case, with the added restriction that $\text{Pull}(X)$ be a first level job of S'. In other words, the inner restricted sets of S' are skipped over in the backtracking. Note that this implies that the restricted sets are strictly nested. If the search for $\text{Pull}(X)$ fails, the subroutine reports that no feasible schedule exists and then halts.

Suppose that the search succeeds with $A = \text{Pull}(X)$. Let $\bar{S}(A,X]$ be the set of first level jobs of $S(A,X]$. The subroutine attempts to schedule the jobs of $\bar{S}(A,X]$ without rescheduling the previously scheduled inner restricted sets of $S(A,X]$. As we show in theorem 1, the number of jobs of $\bar{S}(A,X]$ which must be scheduled before the initial first level r.s. of $S(A,X]$, between any two first level restricted sets, or after the final first level r.s. is invariant over all possible schedules. We refer to these spaces in the schedule in which the jobs of $\bar{S}(A,X]$ must be scheduled as *first level intervals* of $S(A,X]$. If a r.s. begins or ends with an inner first level r.s. or if there are no first level jobs between two inner first level restricted sets, then there is a corresponding first level interval which contains no first level jobs. For the CRISIS subroutine the number of jobs of $\bar{S}(A,X]$ which must be scheduled in each first level interval of $S(A,X]$ is, with one exception, the number of jobs of $\bar{S}(A,X]$ which had been scheduled in that interval when X had its crisis. The exception is the initial first level interval, which must be scheduled with one more job of $\bar{S}(A,X]$ than it had contained.

We define $r_{S(A,X]}$ to be the earliest time at which a subroutine can begin scheduling the jobs of $S(A,X]$. For the CRISIS subroutine $r_{S(A,X]}$ is simply the earliest release time of the jobs in $\bar{S}(A,X]$. The subroutine first uses the naive algorithm to schedule the required number of jobs of $\bar{S}(A,X]$ in the initial first level interval starting at time $r_{S(A,X]}$. Assuming that it successfully schedules the jobs in the initial first level interval so that they do not run past the beginning of the following r.s., it then repeats the process for the next first level interval beginning at the time at which the preceding r.s. is completed. It continues in this fashion until all of the first level intervals have been scheduled with the required number of jobs. Crises which occur during the scheduling of $S(A,X]$ are dealt with by recursively calling the CRISIS subroutine.

Because the algorithm must schedule a specified number of first level jobs in each first level interval, some first level job, say Y, may be scheduled to overlap the beginning of an inner first level r.s., say $S(B,Z]$. We say that Y *invades* $S(B,Z]$. The algorithm sets $r_{S(B,Z]}$ to be the time at which Y is completed and calls the INVASION subroutine to reschedule $S(B,Z]$.

The INVASION subroutine counts the number of jobs of $\bar{S}(B,Z]$ in each first level interval of $S(B,Z]$. It then removes the jobs of $\bar{S}(A,X]$ from the schedule and proceedes to schedule the same number of jobs of $\bar{S}(A,X]$ in each first level interval of $S(A,X]$ as had originally been scheduled in that interval. The scheduling portion of the INVASION subroutine is identical to that of the CRISIS subroutine.

A CRISIS call both defines a r.s. and schedules it for the first time. An INVASION call, on the other hand, always involves rescheduling a previously scheduled r.s. to begin at a later start time.

4. An Example

We list the jobs in S. Each job is followed by first its release time and then its deadline. All jobs run for 6 units of time.

A: 0, 74	B: 21, 46	C: 2, 60	D: 50, 68
E: 4, 34	F: 10, 36	G: 28, 38	U: 54, 62
W: 30, 48	X: 52, 68	Z: 25, 4	

The restricted sets are listed below in the order in which they are originally formed.

1. $S(B,Z]=\{G,Z\}$: originally begins at $t=25$; in final schedule begins at $t=28$.

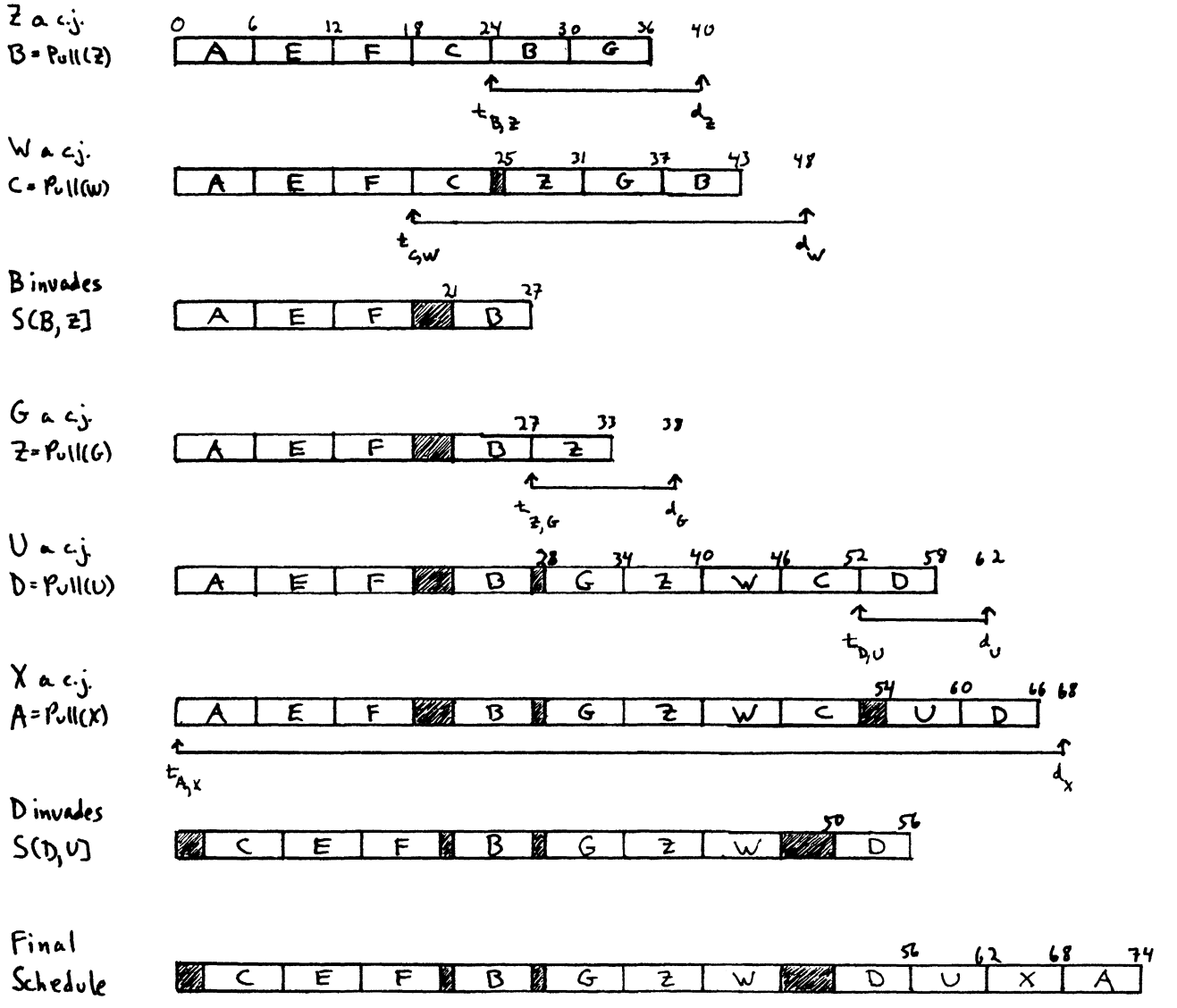
2. $S(C,W]=\{B,Z,G,W\}$: initial first level jobs of $S(C,W]=\{B,W\}$. The initial scheduling of $S(C,X]$ at $t=21$ causes B to invade $S(B,Z]$.

3. $S(Z,G]=\{G\}$: an inner r.s. of $S(B,Z]$ which in turn is an inner r.s. of $S(C,W]$. Originally scheduled to run at $t=28$ and never rescheduled.

4. $S(D,U]=\{U\}$: originally scheduled to run at $t=54$; rescheduled in the scheduling of $S(A,X]$ to begin at $t=56$.

5. $S(A,X]=\{E,F,B,G,Z,W,C,U,D,X\}$: initial first level jobs of $S(A,X]=\{E,F,C,D,X\}$; initial first level restricted sets are $S(C,W]$ and $S(D,U]$. Going from left to right the initial first level intervals of $S(A,X]$ must be scheduled with 3, 1, and 1 jobs respectively; the scheduling of the middle first level interval causes D to invade $S(D,U]$. The first level intervals are started at times 2, 50, and 62.

The final order of the jobs is C,E,F,B,G,Z,W,D,U,X,A.



We now summarize the algorithm.

Main algorithm.

1. While S has not been completely scheduled, schedule the jobs of S using the naive algorithm; otherwise halt (S has been successfully scheduled):
   a) If some job X has a crisis, call CRISIS(X).

CRISIS subroutine (X).

1. Backtrack over the first level jobs of S' looking for Pull(X). Let A be Pull(X) and define $S(A,X]$ to be a restricted set. If no Pull(X) exists, report failure and halt.
2. Count the number of jobs of $\bar{S}(A,X]$ in each first level interval of $S(A,X]$. Increase the count of the initial first level interval by 1.
3. Remove the jobs of $\bar{S}(A,X]$ from the schedule.
4. $i \leftarrow 1$.
5. While the required number of jobs of $\bar{S}(A,X]$ have not been scheduled in the i^{th} first level interval, schedule the jobs of $\bar{S}(A,X]$ using the naive algorithm. (If $i=1$, the first level interval begins at $r_{S(A,X]}$; otherwise, the interval begins at the time at which the preceding r.s. is completed).
   a) If some job Z has a crisis, call CRISIS(Z).
   b) If some job Y invades the following r.s. $S(C,W]$, set $r_{S(C,W]}$ to be the time at which Y is completed and call INVASION($S(C,W]$, $r_{S(C,W]}$).
6. If all the jobs of $S(A,X]$ have been scheduled then return; otherwise $i \leftarrow i+1$.
7. Go to step 5.

INVASION subroutine ($S(A,X]$, $r_{S(A,X]}$).

1. Count the number of jobs of $\bar{S}(A,X]$ in each first level interval of $\bar{S}(A,X]$.

   Steps 2-6 are identical to steps 3-7 of the CRISIS subroutine.

5. Analysis

Lemma 1. If a job X has a crisis more than once, then the algorithm fails.

Proof. When a r.s. is formed, all the jobs in it have deadlines no greater than the deadline of the crisis job which triggered the r.s. Consequently, if that job has a second crisis, the algorithm will not find a job to pull and will fail at step 1 of the CRISIS subroutine. □

Corollary 1. The algorithm always halts.

Proof. By lemma 1 the algorithm will halt if more than n crises occur. □

Corollary 2. The algorithm fails if and only if it is unable to find Pull(X) for some c.j. X.

A *division hole* occurs when a job has just been completed and none of the jobs which remain to be scheduled has been released. A division hole is a *first level division hole* of S' if the jobs which are being scheduled when the division hole occurs are first level jobs of S'.

Removing a first level job of S' which is scheduled to the left of a first level division hole of S' cannot affect the scheduling of first level jobs of S' to the right of the hole. In particular suppose that X is a first level job of S' when it becomes a c.j., $A = \text{Pull}(X)$, and $t_{A,X}$ is the time at which A had been scheduled to run when X had its crisis. If there is a first level division hole of S' between $t_{A,X}$ and t_X, then none of the first level jobs of S' which follows the division hole can be begun before the end of the division hole. Hence, when the CRISIS subroutine attempts to schedule $S(A,X]$, these jobs will be scheduled just as they had been and X will have another crisis. Therefore, by lemma 1 the algorithm will fail. We have just proved the following lemma:

Lemma 2. Suppose when X becomes is a crisis job it is a first level job of S'. If there is a first level division hole of S' between X and Pull(X), then the algorithm will fail.

Because the jobs of $\bar{S}(A,X]$ are rescheduled each time $S(A,X]$ is rescheduled, a job which is an element of $\bar{S}(A,X]$ when a subroutine is called to schedule $S(A,X]$ may have become a member of an inner r.s of $S(A,X]$ upon the return from the subroutine. Consequently, when we speak of first level jobs or intervals of $S(A,X]$, we shall always have in mind a particular point in time in the execution of the algorithm.

Lemma 3. Suppose the algorithm has just *completed* a successful call to a subroutine to schedule a r.s. $S(A,X]$. Let S_i = {jobs of $\bar{S}(A,X]$ in the i^{th} first level interval of $S(A,X]$} and let $k = |S_i|$. Suppose t is the completion time of the r.s. preceding the first job of S_i and t' is the completion time of the last job of S_i to be scheduled. Then there is no feasible schedule in which

a set of k jobs of $\bar{S}(A,X]$ can be begun at time t or later and completed before t'.

Proof. In the schedule returned by the subroutine the jobs in S_i are all members of $\bar{S}(A,X]$. Hence if there is any unscheduled time between t and t', it must be a first level division hole of $S(A,X]$. If there are no division holes, then the lemma follows trivially. So suppose that there are division holes between t and t', and let $\bar{t}$ be the completion time of the last such division hole. None of the jobs of $\bar{S}(A,X]$ which are scheduled after $\bar{t}$ are released before $\bar{t}$. Therefore, the jobs in S_i jobs are completed as soon as possible. □

Theorem 1. Let $S(A,X]$ be a r.s. If a feasible schedule exists then assertions 1-4 hold for all feasible schedules. Each time a subroutine is about to schedule $S(A,X]$ (for the CRISIS subroutine this time is immediately after step 3, for the INVASION subroutine it is immediately after step 2) assertions 1-3 hold:

1. The first job of $S(A,X]$ is always scheduled to begin in $(t_{A,X}, t_{A,X}+p)$.

2. Only jobs in $S(A,X]$ can be scheduled totally in $(t_{A,X}, d_X]$.

3. $S(A,X]$ can not be scheduled to begin before $r_{S(A,X]}$.

When the program returns from a subroutine call the following assertion holds:

4. $S(A,X]$ can not be completed any earlier than the time at which it is currently scheduled to be completed.

Proof. The proof is by induction on k, where $k \leftarrow 1$ the first time a subroutine is about to schedule a r.s., and $k \leftarrow k+1$ whenever either a subroutine is about to schedule a r.s. or the program returns from a subroutine call. By lemma 2 we can assume that when X has its crisis there are no division holes at the same level as X between $t_{A,X}$ and t_X. We also know that for all first level jobs Y of $S(A,X]$, $r_Y > t_{A,X}$. Otherwise, since $d_Y < d_A$, the naive algorithm would have scheduled Y instead of A.

Basis, $k=1$. The CRISIS subroutine has been called for the first time with X the first job to have had a crisis and $A = \text{Pull}(X)$.

Suppose there are m jobs in $S(A,X]$. Since there were no division holes or restricted sets between X and A when X had its crisis, $d_X - t_{A,X} < (m+1)\times p$, and assertions 1 and 2 follow. Because $r_{S(A,X]} = \min\{r_Y \mid Y \text{ is a first level job of } S(A,X]\}$, assertion 3 follows trivially.

Induction step. We first consider the case where a subroutine has just been called on $S(A,X]$.

If the call is an INVASION call, then assertions 1 and 2 held for the CRISIS call on X and therefore they still hold. So to prove assertions 1 and 2 assume that $S(A,X]$ is being scheduled for the first time. Note that it may already have inner restricted sets.

Claim. If $S(A,X]$ is being scheduled for the first time, the only first level interval of $S(A,X]$ which can be scheduled with more jobs of $\bar{S}(A,X]$ than it had originally contained is the one in which A had been scheduled.

We first prove that the first level interval which had contained A has room for at most one additional job of $\bar{S}(A,X]$. Let $S(B,Z]$ be the earliest scheduled first level r.s. of $S(A,X]$. When X has its crisis there can be no division holes between $t_{A,X}$ and $t_{B,Z}$. Also, as was mentioned above, all the jobs of $\bar{S}(A,X]$ must have release times greater than $t_{A,X}$. The induction assumption that the first job of $S(B,Z]$ must be scheduled in $(t_{B,Z}, t_{B,Z}+p)$ implies that $S(B,Z]$ cannot be started after $t_{B,Z}+p$. Therefore, there is room before the first job of $S(B,Z]$ to schedule at most the number of jobs which had previously been scheduled in $[t_{A,X}, t_{B,Z}]$, counting A as one of the jobs, and then only if the first of these jobs is scheduled in $(t_{A,X}, t_{A,X}+p)$.

To show that the rest of the claim is true, let $S(C,W]$ and $S(D,U]$ be adjacent first level restricted sets of $S(A,X]$, with $S(C,W]$ preceding $S(D,U]$ in the schedule. Then $S(C,W]$ and $S(D,U]$ both satisfy the induction assumption, and by lemma 2 there are no division holes between them when X has its crisis. Note that by assertion 4, $S(C,W]$ cannot be rescheduled with an earlier completion time. Thus, by lemma 3 no more jobs can be scheduled in the first level interval between $S(C,W]$ and $S(D,U]$ than had previously been scheduled in that interval. The last first level interval of $S(A,X]$ does have some additional time for scheduling jobs, namely $d_X - t_X$, which is less than p. But because of the absence of division holes and the fact that the preceding first level r.s. cannot end any earlier, this first level interval also does not have room for more jobs than it had originally contained. Therefore, the claim follows. Note that the proof of the claim implies that the total amount of idle time in each first level interval of $S(A,X]$ is always less than p.

By induction assumption 2 it is not possible to insert an additional job in the space allocated to a r.s. of $S(A,X]$. Therefore, each first level interval of $S(A,X]$ must be scheduled with the maximum possible number of jobs of $\bar{S}(A,X]$. This proves assertions 1 and 2.

Next we prove assertion 3. If $S(A,X]$ is being scheduled by the CRISIS subroutine, then assertion 3 follows trivially. So assume that $S(A,X]$ is being scheduled because of an INVASION call and that $S(A,X]$ is a first level r.s. of $S(D,U]$. Since $S(A,X]$ is being rescheduled, there is a job Y of $S(D,U]$ which invades $S(A,X]$. If Y is an element of an earlier inner r.s. of $S(D,U]$, then that r.s. satisfies assertion 4 and conse-

quently Y cannot end any earlier. Therefore, we need prove assertion 3 only for the case where Y is a job of $\bar{S}(D,U]$.

We first show that there can't be fewer than the current number of jobs of $\bar{S}(D,U]$ scheduled before $S(A,X]$. If $S(D,U]$ is being scheduled by the CRISIS subroutine, then by the above claim there is a required number of first level jobs for each first level interval, and the total amount of idle time in each first level interval is always less than p. If $S(D,U]$ is being scheduled by the INVASION subroutine, these first level intervals may have been subdivided by newly formed restricted sets. Clearly the total amount of idle time in any of the smaller intervals cannot exceed p. So it is not possible to take a job from an earlier first level interval and schedule it instead in a later one. By assertion 2 of the induction assumption, it is not possible to expand the space allocated to a r.s. of $S(D,U]$ to make room for a first level job of $S(D,U]$. Consequently, each first level interval of $S(D,U]$ must be scheduled with the same number of first level jobs that it had previously contained.

The proof of assertion 3 is completed by demonstrating that the first level interval which contains Y ends as soon as possible. By the induction assumption the first job of $S(D,U]$ cannot begin before $r_{S(D,U]}$. Nor can any r.s. which precedes Y end any earlier. Hence, the first level interval containing Y cannot be started any earlier. Therefore, by lemma 3 there is no feasible schedule in which $S(A,X]$ can begin before $r_{S(A,X]}$.

We now assume that a subroutine has just returned from a call on $S(A,X]$, and we will show that assertion 4 holds. All first level restricted sets of $S(A,X]$ satisfy the induction assumptions and hence cannot end any earlier. So if the last job of $S(A,X]$ in the schedule belongs to an inner r.s. of $S(A,X]$, then $S(A,X]$ ends as soon as possible.

Suppose the last job of $S(A,X]$ to be scheduled is not in an inner r.s. of $S(A,X]$. If $S(A,X]$ has first level restricted sets, then the one which precedes the final first level interval ends as soon as possible. It follows from previous arguments that each first level interval contains the maximum possible number of first level jobs. So the final first level interval starts as soon as possible. Otherwise, $S(A,X]$ consists only of a single first level interval which by assertion 3 begins as soon as possible. In either case by lemma 3 this final first level interval of $S(A,X]$ is completed as soon as possible. □

Theorem 2. If the algorithm successfully schedules a set of jobs S, it does so with the earliest possible completion time.

Proof. If the final schedule of S has restricted sets, then by theorem 1 they all have the earliest possible completion time and cannot contain any additional jobs. By arguments identical to those used in lemma 3, the final first level interval of S is completed as soon as possible. Therefore the jobs of S are scheduled with the earliest possible completion time. □

Theorem 3. If the algorithm fails to schedule a set of jobs S, then there is no feasible schedule for S.

Proof. By corollary 2 if the algorithm fails then there is a c.j. Y for which Pull(Y) does not exist. Suppose Y is a first level job of S'. Since it is not possible to insert an additional job in the space allocated to any r.s., a feasible schedule for S' could exist only if Y can be inserted in some first level interval of S'. But the failure of the algorithm implies that all first level jobs of S' which had been scheduled before t_Y have deadlines no greater than d_Y and hence cannot be scheduled at a later time. Certainly first level division holes of S' cannot be used as space in which to schedule Y. Therefore, Y cannot be inserted in any first level interval and there can be no feasible schedule of S' and consequently of S. □

Theorem 4. The algorithm has worst case running time of $O(n^2 \log n)$.

By lemma 1 each job can have at most one crisis if the algorithm is to succeed. Each time a new r.s. is formed the jobs within it are rescheduled at most once unless they become part of still another r.s. If we charge the crisis job for the cost of scheduling the job which was removed, we get that each job is charged for scheduling no more than n times. Multiplied by n jobs this gives n^2 schedulings. Since there is an overhead of $\log n$ for scheduling a job because of the use of heaps [1], we get that in the worst case the running time is $O(n^2 \log n)$.

If the release times and deadlines are all integers, then by assertion 1 of lemma 4 there are only $p-1$ possible start times for any restricted set. Since each time a r.s. is rescheduled it has a later start time, no r.s. can be scheduled more than m times, where $m = \min\{n,p\}$. This gives the bound for the integer case of $O(mn \log n)$. □

6. Acknowledgements.

The author would like to thank Eugene Lawler for suggesting this problem and Michael Sipser and Allan Adler for their many helpful comments.

References.

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading Mass., 1974.

[2] K.R. Baker and Z.S. Su, "Sequencing with due-dates and early start times to minimize maximum tardiness", *Naval Res. Logist. Quart.,* 21 (1974), pp. 171-176.

[3] M.R. Garey and D.S. Johnson, "Two processor scheduling with start-times and deadlines", *SIAM Journal on Computing,* 6 (1977), pp. 416-426.

[4] M.R. Garey and D.S. Johnson, *Computers and Intractibility: A Guide to the Theory of NP Completeness,* W.H. Freeman, San Francisco, California, 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", to appear.

[6] W.A. Horn, "Some simple scheduling algorithms", *Naval Res. Logist. Quart.,* 21 (1974), pp. 177-185.

[7] R.M. Karp, "Reducibility among combinatorial problems", *Complexity of Computer Computations,* R.E. Miller and J.M. Thatcher, eds., Plenum Press, New York, 1972, pp. 85-104.

[8] H. Kise, T. Ibaraki, and H. Mine, "A solvable case of one-machine scheduling problem with ready and due times," *Operations Research,* 26, (1978), pp. 121-126.

[9] B.J. Lageweg, J.K. Lenstra, A.H.G. Rinnooy Kan, "Minimizing maximum lateness on one machine: computational experience and some applications", *Statistica Neerandica,* 39 (1976), pp. 25-41.

[10] B.J. Lageweg, J.K. Lenstra, and A.H.G. Rinnooy Kan, "Job-shop scheduling by implicit enumeration", *Management Science,* 24, (1977), pp. 441,450.

[11] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems", *Annals of Discrete Mathematics,* 1, (1977), pp. 343-362.