1991

An NC Algorithm for Scheduling Unit-Time Jobs with Arbitrary Release Times and Deadlines

Greg N. Frederickson
Purdue University, gnf@cs.purdue.edu

Susan H. Rodger

Report Number:
90-1005

Frederickson, Greg N. and Rodger, Susan H., "An NC Algorithm for Scheduling Unit-Time Jobs with Arbitrary Release Times and Deadlines" (1991). *Department of Computer Science Technical Reports*. Paper 8.
<https://docs.lib.purdue.edu/cstech/8>

AN NC ALGORITHM FOR SCHEDULING
UNIT-TIME JOBS WITH ARBITRARY
RELEASE TIMES AND DEADLINES

Greg N. Frederickson
Susan H. Rodger

CSD-TR-1005
August 1990
(Revised June 1991)

An NC Algorithm for Scheduling Unit-Time Jobs with Arbitrary Release Times and Deadlines

Greg N. Frederickson*
Department of Computer Science
Purdue University
West Lafayette, IN 47907

Susan H. Rodger†
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

June 25, 1991

Abstract

The problem of scheduling n unit-time jobs with real-valued release times and deadlines is shown to be in NC. The solution is based on characterizations of a canonical schedule and best subset of jobs to be scheduled in a given time interval. The algorithm runs in $O((\log n)^2)$ time and uses $O(n^4/\log n)$ processors.

Key words: parallel algorithm, scheduling, release time, deadline

1 Introduction

A major goal in the study of parallel algorithms is the elucidation of the underlying combinatorial structure of problems. A wealth of insight has been generated by

*Supported by National Science Foundation grants CCR-86202271 and CCR-9001241 and the Office of Naval Research contract N00014-86-K-0689

†Supported by the Office of Naval Research on contract N 00014-86-K-0689. The major portion of this research was done while this author was at Purdue University

designing parallel algorithms for problems in such areas as graph theory, algebra and arithmetic, and computational geometry [KR]. To a lesser extent, parallel algorithms have been reported in scheduling theory [DUW], [HM1], [HM2]. In this paper we focus on a fundamental problem in scheduling theory and identify significant features of it that lead to an NC algorithm. The problem is to find a schedule for a set of jobs on a single processor, where the jobs each have unit-time processing requirements and real-valued release times and deadlines.

Our problem is intermediate in conceptual difficulty between the following two variations. The first variation has integral release times and deadlines in addition to unit processing times, and can be solved sequentially by the earliest deadline first rule [H] and in parallel by a parallel implementation of this rule [AGK], [DS], [R]. The second variation has unequal processing times, and has been shown to be NP-complete [GJ]. Our problem was posed as open in [GJ] as to whether it is polynomially solvable or NP-complete. It was shown in [C], [S] that the problem is polynomially solvable, and an $O(n \log n)$ -time algorithm was presented in [GJST]. These approaches appear inherently sequential, and the problem is challenging in a parallel regimen for the following reason. Since the release times and deadlines are arbitrary real values, the appropriate scheduling choice at a given time might be to schedule no job and allow some fraction of idle time until another job's release time is reached. These choices can be affected by jobs whose release times and deadlines are quite far from such a decision point, making it difficult to resolve such choices "locally". We show how to work around this difficulty, and present an algorithm that uses $O((\log n)^2)$ time and $O(n^4 / \log n)$ processors.

We sketch our approach briefly and indicate the nice features that make this approach possible. The set of jobs is partitioned into subsets based on their release times, such that each subset has associated with it a time interval, which contains the release times of the jobs in the subset. For each such interval, a "best" set of jobs is tentatively chosen, from among those jobs assigned to the interval, to be scheduled within the interval. This best set is such that the jobs that are not chosen have the

largest deadlines among all such sets. (These notions are defined precisely later.) A balanced binary tree structure is imposed on the intervals, taking as leaves of the tree the intervals from earliest to latest, and having the nonleaf nodes represent new intervals that span the intervals of their children. The algorithm then sweeps up through the tree, computing best sets. For two consecutive time intervals I_1 and I_2 , where I_1 precedes I_2 , the set of jobs tentatively chosen to be scheduled in the spanning interval I is generated (roughly) as follows. This set will include the set of jobs tentatively chosen for I_1 plus a best set selected from jobs chosen from I_2 unioned with jobs not chosen from I_1 .

This basic approach is fairly straightforward, but its correctness is not. Choosing best sets allows maximum flexibility in scheduling, since the jobs not chosen to be scheduled in the interval must be scheduled at a later time. But it is far from obvious that such best sets exist for any given time interval. We show in a lengthy proof by contradiction that best sets do in fact exist. The proof uses a nonobvious measure of the size of a problem. Furthermore, it is not obvious that the set of jobs not chosen in I_2 would also not be chosen in I . The correctness of this assertion depends on an involved proof by contradiction that is similar to the proof of the existence of a best set.

There is crucial feature of our solution that we have not yet discussed. In order to be able to insert idle time into the schedule, each time interval mentioned above must actually represent a family of $O(n^2)$ time intervals, whose starting times differ by less than 1, and similarly for ending times. In combining sets of chosen jobs for the families of intervals for I_1 and I_2 , each interval in the family for I results from considering $O(n)$ combinations of individual intervals, one from I_1 and one from I_2 . Some of the combinations do not necessarily result in a schedule. To test feasibility, we use what we call a "template". A template is formed using the set of jobs whose deadlines are in the given interval. From among all sets of these jobs that can be scheduled within the interval, the template is the set of deadlines that is smallest among such sets of jobs. We prove that such a template exists. The proof yields an

elegant mirror image approach to computing the set.

Our paper is organized as follows. In Section 2 we prove the existence of a canonical form for schedules, which is based on certain types of time intervals that we identify. In Section 3 we extend our characterizations to prove the existence of best sets and establish properties that lead to their fast computation. In Section 4 we present our NC algorithm in its entirety and analyze its performance.

2 Properties of Intervals and Schedules

In this section we identify a canonical form for schedules that we use in our parallel divide-and-conquer algorithm. The canonical form contains two types of time intervals induced by a set of jobs. In a “prime interval”, certain jobs must be scheduled within the interval, and these jobs can always be tightly packed together. We identify a maximal set of prime intervals that are “compatible”, called “cover intervals”. The intervals that fall between consecutive cover intervals, called “gaps”, are the more difficult to schedule. Within a gap if there are enough jobs to completely fill the gap, then the jobs can be tightly packed together in two groups, separated by a section of free space in which no job is scheduled. We show that for every schedule, there is a corresponding canonical schedule. This notion of a canonical schedule forms the basis of further characterizations in section 3.

We first define some basic terms and establish some simple properties. Each job i has a release time r_i , a deadline d_i , and it should be processed for one unit of time in the interval $[r_i, d_i)$. The interval is closed on the left end to indicate that the job can start at r_i , and the interval is open on the right end to indicate that the job should be completed by d_i . Let s_i be the start time of job i and c_i be the completion time of job i , $c_i = s_i + 1$. The interval $[s_i, c_i)$ represents the time job i is processed. A *schedule* is an assignment of start times to jobs, such that the difference between the start times of any two jobs is at least one and for each job i , $r_i \leq s_i$ and $c_i \leq d_i$. A set of jobs can be *scheduled in an interval* $[a, b)$ if and only if there is a schedule of

the jobs such that for each job i , $s_i \geq a$ and $c_i \leq b$.

Our approach is based on considering time intervals with certain interesting properties. We say that a job i is *contained* in an interval $[a, b)$ if and only if $a \leq r_i$ and $d_i \leq b$. We consider intervals $[a, b)$ such that $a = r_i$ for some job i contained in $[a, b)$ and $b = d_j$ for some job j contained in $[a, b)$. An interval $[a, b)$ has *looseness* x if and only if there are precisely $b - a - x$ jobs contained in it. If any interval has negative looseness, then no schedule is possible. We shall assume for the remainder of this section that all intervals have nonnegative looseness.

We are now ready to define an interval such that the jobs contained within it are easy to identify and easy to schedule. A *constrained interval* is an interval whose looseness is less than 1. A *prime interval* is a constrained interval $[a, b)$ such that there is no constrained interval $[a', b')$ properly contained in $[a, b)$. Figure 1a shows a set of jobs. Each job is represented by an interval in which the left endpoint is its release time and the right endpoint is its deadline. There are quite a few constrained intervals in the figure. One constrained interval is $[7.2, 11.1)$, which contains three jobs, 4, 5, and 6, and has looseness 0.9. It is not a prime interval because the constrained interval $[8.0, 10.4)$ is contained in it. The prime intervals in the figure are $[8.0, 10.4)$, $[13.0, 15.6)$, and $[13.6, 16.3)$.

A prime interval is quite useful, because the jobs that must be scheduled within the interval can be packed tightly together with no free space between them and with a variable amount of free space on either end of the interval. We prove this in the following lemma.

Lemma 2.1 *Let $[a, b)$ be a prime interval with looseness x . For any y , $0 \leq y \leq x$, the jobs contained in $[a, b)$ can be scheduled in the interval $[a+y, b-x+y)$.*

Proof: Suppose that the jobs contained in $[a, b)$ cannot be scheduled in the interval $[a+y, b-x+y)$ for some value y , $0 \leq y \leq x$. Without loss of generality, assume that $a+y$ is an integer. (Otherwise, we can subtract $a+y - \lfloor a+y \rfloor$ from all values to generate an equivalent problem.) For each job i contained in $[a, b)$, set $r'_i = \lceil r_i \rceil$,

Figure 1: a. A set of jobs with release times and deadlines. b. A cover for this set of jobs. c. A schedule for anchored gaps [5.3, 8.4), [10.4, 13.0) and [15.0, 20.0). d. A schedule for anchored gaps [5.3, 8.0), [10.0, 13.1) and [15.1, 20.0).

and $d'_i = \lfloor d_i \rfloor$, and call the resulting jobs *modified*. Since all r'_i and d'_i are integers, and there is no schedule for the modified jobs, then there must be an interval $[a', b']$ properly contained in $[a, b]$, with a' and b' integers, such that there are at least $b' - a' + 1$ modified jobs contained in $[a', b']$. Let a'' be the earliest release time of an original job whose corresponding modified job is contained in $[a', b']$, and let b'' be the latest deadline of an original job whose corresponding modified job is contained in $[a', b']$. Then there are at least $b' - a' + 1$ original jobs in the interval $[a'', b'']$, which is of length $b'' - a'' < (b' + 1) - (a' - 1) = b' - a' + 2$. Thus the interval $[a'', b'']$ has looseness less than 1, *i.e.*, it is a constrained interval. Furthermore, interval $[a'', b'']$ is properly contained in $[a, b]$, for the following reason. Since $b' - a' + 1 \leq (b - x + y) - (a + y)$, and $a', b', a + y$ and $b - x + y$ are all integers, either $a' \geq a + y + 1$ or $b' \leq b - x + y - 1$. Thus either $a'' > a' - 1 \geq a + y$ or $b'' < b' + 1 \leq b - x + y$, and it follows that $[a'', b'']$ is properly contained in $[a, b]$. This is a contradiction to the original assumption that $[a, b]$ is a prime interval. It follows that the original jobs can be scheduled in $[a + y, b - x + y]$. $\square$.

Within isolated prime intervals, jobs are easy to schedule, but when these intervals overlap, a schedule in one interval affects the schedule in the other. Two prime intervals $[a, b]$ and $[a', b']$, with $a < a'$, are *compatible* if and only if $b - a' < 2$. If two prime intervals are compatible, then they do not contain a common job. This can be shown as follows. Suppose each interval contained job i . Then $a' \leq r_i < d_i \leq b$, which implies that $d_i - r_i < 2$, and thus $[r_i, d_i]$ would be a constrained interval contained inside an interval $[a, b]$ claimed to be prime. This is not possible.

Because of the incompatibility of certain prime intervals, we focus on a subset of the set of all prime intervals. A maximal set of prime intervals that are pairwise compatible is a *cover* for the set of jobs. A cover is shown in Figure 1b. Jobs 5 and 6 must be scheduled in $[8.0, 10.4]$ (indicated by $\{5, 6\}$ in the figure), and jobs 11 and 12 must be scheduled in $[13.0, 15.6]$. Not all prime intervals are compatible. The prime interval $[13.6, 16.3]$ is not part of the cover since $[13.0, 15.6]$ and $[13.6, 16.3]$ are not

compatible. Both of these prime intervals contain job 12. For any two compatible prime intervals that overlap, the jobs in the prime interval with the smaller left endpoint are scheduled before the jobs in the other prime interval. For example, if job 4 in Figure 1 had deadline 8.6, then there would be an additional prime interval in the cover, $[7.2, 8.6)$, that contains job 4. The compatible prime intervals $[7.2, 8.6)$ and $[8.0, 10.4)$ overlap, so job 4 would have to be scheduled before jobs 5 and 6.

The precise scheduling of jobs contained in the prime intervals of a cover, called *cover intervals*, is dependent on the scheduling of the jobs not contained in those intervals. Given a cover, let $[a, b)$ and $[a', b')$ be two consecutive cover intervals. The *gap* between intervals $[a, b)$ and $[a', b')$ is the interval $[a'', b'')$, where $a'' = a + \lfloor b - a \rfloor$ and $b'' = b' - \lfloor b' - a' \rfloor$. The value a'' is the theoretically earliest possible time after a at which a job not contained in $[a, b)$ can be started in a schedule, and b'' is the latest possible time before b' at which a job not contained in $[a', b')$ can be completed. Note that if the looseness of $[a, b')$ is 0, then gap $[a'', b'')$ constitutes the empty interval. Of course, whether a job can actually start at a'' in a schedule depends on whether the jobs contained in $[a, b)$ are scheduled to complete by a'' . It is important to maintain this flexibility in the definition of a gap. If two cover intervals overlap, then in any schedule the gap between them will either be empty or will contain precisely one job, which is not contained in either of the cover intervals. If $b'' - a'' < 1$, then in any schedule a job that is not contained in a cover interval will not be scheduled in the gap.

For uniformity, we require that each gap is always surrounded by two cover intervals. This is easily taken care of by adding a cover interval with looseness 0 at the beginning and end of the schedule. Let $r_{\min}$ be the minimum release time and $d_{\max}$ the maximum deadline in the problem. Two new jobs are introduced, job $n+1$ with $r_{n+1} = r_{\min} - 1$ and $d_{n+1} = r_{\min}$, and job $n+2$ with $r_{n+2} = d_{\max}$ and $d_{n+2} = d_{\max} + 1$. This forces two new cover intervals $[r_{n+1}, d_{n+1})$ and $[r_{n+2}, d_{n+2})$ to be included in the cover. Clearly, the original n jobs can be scheduled if and only if the new set of $n+2$ jobs can be scheduled. In Figure 1, two additional jobs would be added on the ends

of the overall interval, job 14 with $r_{14} = 4.3$ and $d_{14} = 5.3$ and job 15 with $r_{15} = 20.0$ and $d_{15} = 21.0$. This would result in cover intervals on the ends of the overall interval; these are not shown.

We restrict our attention to specific subintervals of gaps. For a given cover, let $[a, b)$ and $[a', b')$ be two consecutive cover intervals with looseness x and x' , respectively. An *anchored gap* is an interval $[a'', b'')$, where $b - x \leq a'' \leq b$, $a' \leq b'' \leq a' + x'$, a'' differs from some release time by an integer. At least one anchored gap $[b - x, a' + x')$ for gap $[b, a')$ exists, since the looseness of $[a, b')$ is by assumption greater than 0. Since each of a'' and b'' can be one of at most n values, there are at most n^2 anchored gaps for any gap.

If there is a schedule of jobs in an anchored gap that almost fills the anchored gap, then there is a schedule in which the jobs are packed together in two groups with free space between the two groups. For a given schedule, a *hole* in an anchored gap $[a'', b'')$ is a nonempty interval $[a''', b''')$ contained in $[a'', b'')$, such that no jobs are scheduled in $[a''', b''')$ and both $a''' - 1$ and b''' are start times for jobs in the schedule.

Lemma 2.2 *For any set of jobs that has a schedule S , and for any cover for the set of jobs, there is a schedule S' such that for each anchored gap $[a, b)$ that has $\lfloor b - a \rfloor$ jobs scheduled within it, there is at most one hole in the anchored gap.*

Proof: Let $[a, b)$ be an anchored gap in S that has $\lfloor b - a \rfloor$ jobs scheduled in it. We claim that there is a schedule of the jobs in the anchored gap such that there is only one hole within it. The proof of the claim is by induction on h , the number of jobs scheduled between first and last holes in $[a, b)$. For the basis, with $h = 0$, the claim is trivially satisfied. For the induction step, with $h > 0$, assume that the claim holds whenever there are fewer than h jobs scheduled between the first and last holes of $[a, b)$. Let $[a_1, b_1)$ be the first hole in $[a, b)$ and let $[a_l, b_l)$ be the last hole in $[a, b)$. We shall show that some of the jobs scheduled in $[b_1, a_l)$ can be scheduled starting at a_1 or finishing at b_l , thus reducing the number of jobs scheduled between the first and last holes.

For each job i scheduled in $[b_1, a_l]$, temporarily reset the release time r_i to be $r'_i = \max\{r_i, a_1\}$ and the deadline d_i to be $d'_i = \min\{d_i, b_l\}$. Since each such job i is scheduled in $[b_1, a_l]$, the schedule would still be valid if we had reset r_i to be $\max\{r_i, b_l\}$ and d_i to be $\min\{d_i, a_l\}$. Since $a_1 < b_1$ and $b_l > a_l$, the resetting we actually do is no more restrictive, and thus the schedule is still valid. Let $a' = \min\{r'_i\}$ and $b' = \max\{d'_i\}$. Since $[a, b)$ contains $\lfloor b - a \rfloor$ jobs, and $(b_1 - a_1) + (b_l - a_l) < 1$, we have $(b_1 - a') + (b' - a_l) < 1$. Thus $[a', b')$ is a constrained interval with respect to the modified release times and deadlines. Note that if both $a' > a_1$ and $b' < b_l$, then $[a', b')$ would be a constrained interval with respect to the original release times and deadlines, and would be compatible with all cover intervals, a contradiction to the cover being maximal.

Without loss of generality, assume $a' = a_1$. (The argument for $b' = b_l$ is similar.) Now identify the smallest value $b'' < b'$ such that $[a_1, b'')$ is a constrained interval with respect to the modified release times and deadlines. Any interval that is a constrained interval with respect to the modified release times and deadlines and is contained within $[a', b')$ must have either a' or b' as an endpoint. Thus $[a_1, b'')$ does not properly contain a constrained interval, and is thus a prime interval with respect to the modified release times and deadlines. The set of jobs whose scheduled positions in S overlap with $[a_1, b'')$ is precisely the set of jobs contained in $[a_1, b'')$ with respect to the modified release times and deadlines. By Lemma 2.1, this set of jobs can be scheduled in $[a_1, a_1 + \lfloor b'' - a_1 \rfloor]$. The remaining jobs are scheduled as they were in S . Thus the first hole will now begin at $a_1 + \lfloor b'' - a_1 \rfloor$ rather than a_1 , and there will be $h - \lfloor b'' - a_1 \rfloor$ jobs between the first and last holes. By the induction hypothesis, the jobs in $[a, b)$ can be rescheduled to yield just one hole. This completes the proof of the claim.

The proof of the lemma follows by handling in turn each anchored gap $[a, b)$ that has $\lfloor b - a \rfloor$ jobs scheduled in it and has more than one hole. $\square$

A schedule of the jobs for the anchored gaps $[5.3, 8.4)$, $[10.4, 13.0)$ and $[15.0, 20.0)$

is shown in Figure 1c. A schedule of the jobs for anchored gaps $[5.3, 8.0)$, $[10.0, 13.1)$ and $[15.1, 20.0)$ is shown in Figure 1d. Lemma 2.2 is illustrated by these schedules. In each anchored gap the jobs can be scheduled so that there is at most one hole within the gap. Note that there is no schedule when the first anchored gap is $[5.3, 8.2)$. This follows since jobs 3 and 4 would have to be scheduled in the first anchored gap, and jobs 1, 7, 9, 11, 12 and 13 would have to be scheduled by 16.3, with 16.2 being the earliest at which they could all be finished. But then job 10 cannot start before 16.2, and thus cannot finish by its deadline.

We now show that if there is a schedule, then there is a schedule such that the jobs are nicely packed and the starting times of the jobs are convenient values. Given a set J of jobs, a *breakpoint* is any number x such that for some job j , $x - r_j$ is an integer. Given a cover, those jobs that must be scheduled within the cover intervals are called *cover jobs*, and those jobs that are scheduled within anchored gaps are called *gap jobs*. A *canonical schedule* is a schedule in which each job starts at a breakpoint, the cover jobs are scheduled tightly together within the cover intervals, and there is at most one hole in any anchored gap $[a, b)$ in which $\lfloor b - a \rfloor$ gap jobs are scheduled. We show in the next theorem that we can restrict our algorithm to finding a canonical schedule.

Theorem 2.1 (Canonical Schedule) *For any set of jobs that has a schedule and for any cover for the set of jobs, there is a corresponding canonical schedule.*

Proof: Consider any schedule S and any cover for the set of jobs. Consider any cover interval $[a, b)$ with looseness x . The jobs contained in $[a, b)$ will be the only jobs completely scheduled in $[a, b)$, since $x < 1$. Let a' be the earliest start time of any of these jobs. Let $y = a' - a$. By Lemma 2.1, the jobs contained in $[a, b)$ can be scheduled in $[a + y, b - x + y)$, which means that they are scheduled without any free space between them. It follows that there is a schedule S' such that for each cover interval, the cover jobs are scheduled without any free space between them within the cover interval. Given S' , Lemma 2.2 establishes that there is a schedule S'' such that

there is only one hole in each anchored gap $[a, b)$ that has $\lfloor b - a \rfloor$ jobs scheduled in it.

We derive a canonical schedule S''' from S'' that preserves the same order of the jobs, though it may shift their start times. For this discussion, consider there to be one additional hole, that starts when the last job completes. Consider the first hole $[a_1, b_1)$ in the schedule, and let J_1 be the set of jobs with start times before a_1 . Let y_1 be the maximum value that can be subtracted from the start of each job in J_1 such that a schedule still remains. Subtract y_1 from each such start time. Clearly, some job in J_1 is starting at its release time, and all the rest start at a time that differs from this time by an integer. For each succeeding hole $[a_i, b_i)$, $i > 1$, let J_i be the set of jobs scheduled between this hole and the preceding one. Let y_i be the maximum value that can be subtracted from the start of each job in J_i such that a schedule still remains. Subtract y_i from each such start time. Clearly, either some job in J_i is starting at its release time, or the start time of the first job in J_i equals the completion time of the last job in J_{i-1} . In the latter case, each job starts at a time that differs from some release time by an integer, by transitivity. Thus, each job in S''' starts at a time that differs from some job's release time by an integer. It follows that S''' is a canonical schedule. $\square$.

By Theorem 2.1, we can limit deadlines to being breakpoints. Thus we may assume as preprocessing that each deadline d_j is reset to the largest breakpoint no larger than d_j . Alternatively, breakpoints could be defined in terms of deadlines, and each release time r_j could be reset to the smallest breakpoint no smaller than r_j .

Consider the schedules in Figure 1c and Figure 1d. In both schedules, the cover jobs are tightly scheduled together within the cover intervals, and there is at most one hole in any anchored gap. Since $r_1 = 5.3$, $r_7 = 10.4$ and $r_{11} = 13.0$, every job in the schedule in Figure 1c starts at a breakpoint. Thus the schedule in Figure 1c is canonical. However, since there is no job whose release time has fractional part .1, the schedule in Figure 1d is not canonical.

3 Best r-Sets, Best d-sets, and Templates

The characterization of canonical schedules in the last section is not sufficient for designing a fast parallel algorithm. In particular, no method is implied to choose an appropriate set of jobs to be scheduled in a gap, and no method is identified for choosing suitable endpoints of a gap, in the case that its bracketing cover intervals have nonzero looseness. In this section we discuss the existence and computation of what we shall define as a “best r-set”, a best choice of a subset of jobs to be scheduled in an interval. To make best r-sets unique, we shall transform problem instances so that all release times are distinct and all deadlines are distinct. A best r-set is easy to compute when the interval is a gap, but is more complicated to compute when the interval contains a collection of gaps and cover intervals. In the latter case, we first establish the existence of the best r-set, and then show how to select a subset of the jobs that will form the best r-set if there is a schedule. To identify suitable endpoints for a gap, all possible choices can be considered, with a test performed to determine if the selected jobs can be scheduled. We examine what we call a “modified mirror image problem”, and identify a template of deadlines that represents the minimum set of deadlines that will result in a schedule. The template can be compared to a set of selected jobs to determine if the endpoints were suitable.

We first discuss a transformation that will give us uniqueness with respect to the best sets that we shall introduce shortly. Let a set of jobs be *simple* if all release times are distinct and all deadlines are distinct. Given a set of jobs for which a schedule exists, we can reset release times and deadlines so as to make the set of jobs simple. While either of the following operations apply, perform it. If $r_j = r_k$ and $d_j \leq d_k$ for jobs j and k , then reset r_k to be $r_j + 1$. If $r_j < r_k$ and $d_j = d_k$, then reset d_j to be $d_k - 1$. Clearly, performing these operations does not affect whether or not a schedule exists. We assume for the remainder of this section that the set of jobs in the problem instance has been transformed so as to be simple.

We first introduce the notion of feasibility with respect to an interval, and then

we define what we call an r -set. Let J be a set of jobs, and $[a, b)$ an interval. Recall that a job is contained in an interval $[a, b)$ if $r_j \geq a$ and $d_j \leq b$. Let $J[a, b)$ be the subset of all jobs in J that are contained in $[a, b)$. A set J is $[a, b)$ -feasible if there is a schedule for $J[a, b)$. If there is no schedule for $J[a, b)$, then there cannot be a schedule for J . We next consider a partition of J based on release times. Let $J_r[a, b)$ be the subset of all jobs j in J such that $r_j \in [a, b)$. Note that $J[a, b) \subseteq J_r[a, b)$. A set A is an r -set for $[a, b)$ with respect to J if and only if J is $[a, b)$ -feasible and A is a maximum-cardinality subset of $J_r[a, b)$ such that $J[a, b) \subseteq A$ and the jobs in A can be scheduled in $[a, b)$. We choose the name r -set, where r denotes release time, to emphasize that jobs are partitioned by their release times.

As some r -sets are better than others when constructing a schedule, we define the notion of a "best" r -set. Consider an interval $[a, b)$. Clearly, the jobs that are contained in $[a, b)$ must be scheduled in $[a, b)$. For those jobs with release times in $[a, b)$ but with deadlines greater than b , it is preferable to choose to schedule in $[a, b)$ those jobs with the smallest deadlines that can be scheduled. This strategy allows the jobs that are not chosen to have a better chance of being scheduled in a later time interval, since they have larger deadlines. We define the partial order relation $\leq_d$ in order to compare the deadlines of sets of jobs. If A and B are sets of jobs and $|A| = |B|$, then $A \leq_d B$ if and only if $d_{j'_i} \leq d_{j''_i}$ for $i = 1, 2, \dots, |A|$, where j'_i is the job with the i -th smallest deadline in A and j''_i is the job with the i -th smallest deadline in B . If we deal with a set of jobs in which all deadlines are distinct, equality will hold if and only if subsets A and B are identical. An r -set A for $[a, b)$ with respect to J is a *best r -set* for $[a, b)$ with respect to J if and only if for any other r -set B for $[a, b)$ with respect to J , $A \leq_d B$. If J is simple, then A is a unique best r -set.

We discuss examples of best r -sets, using the set of jobs in Figure 1a. There are 6 jobs whose release times lie in $[5.3, 8.2]$, jobs 1, 2, 3, 4, 5 and 6. We do not consider jobs 5 and 6 since they obviously cannot be scheduled within this interval. Jobs 3 and 4 form the best r -set for $[5.3, 8.2)$ as $\{3, 4\} \leq_d B$, for B that is any of $\{1, 3\}$, $\{1, 2\}$, $\{2, 3\}$, $\{1, 4\}$, and $\{2, 4\}$. Similarly, jobs 7 and 9 form the best r -set for $[10.2, 13.0)$.

The best r -set for $[15.0, 20.0)$ is empty since there are no jobs with a release time in the interval.

We discuss the existence and computation of the best r -set for two types of intervals, the simpler interval that does not contain a constrained interval and the more complex interval that can contain constrained intervals. The first type corresponds to an anchored gap in our algorithm, and the second type corresponds to what we shall call an anchored multiple gap, which we consider in the combining step of our algorithm.

We first discuss the existence and computation of the best r -set with respect to a set J of jobs for an interval $[a, b)$ that contains no constrained intervals. Clearly, J is $[a, b)$ -feasible. Computing the best r -set for $[a, b)$ is straightforward. We define the *discrete earliest deadline rule* applied to the jobs in $J_r[a, b)$ as follows. For each job i , let r'_i be the smallest value no smaller than r_i such that $b - r'_i$ is an integer. Then apply the earliest deadline first rule using modified release times on the interval $[b - \lfloor b - a \rfloor, b)$. Using the earliest deadline first rule results in a set of jobs A such that $A \leq_d B$ for any subset B of $J_r[a, b)$ that can be scheduled in $[a, b)$.

Lemma 3.1 *Let J be a set of jobs, and $[a, b)$ an interval, such that there is no constrained interval contained in $[a, b)$. The subset of $J_r[a, b)$ scheduled in $[a, b)$ by the discrete earliest deadline rule is a best r -set for $[a, b)$ with respect to J .*

Proof: Consider any subset J' of $J_r[a, b)$ that can be scheduled in $[a, b)$. For each job $j \in J'$, let r'_j be the smallest value no smaller than r_j such that $b - r'_j$ is an integer. Suppose that there is an interval $[a', b')$ that contains, with respect to the modified release times, more than $\lfloor b' - a' \rfloor$ jobs from J' . Let $a'' = \min\{r_j | j \in J', r'_j \geq a' \text{ and } d_j \leq b'\}$. Since $r'_j - r_j < 1$ for each $j \in J'$, $(a' - a'') < 1$. Thus there are at least $\lfloor b' - a'' \rfloor$ jobs contained in $[a'', b')$ with respect to the original release times. Since $J_r[a, b)$ can be scheduled in $[a, b)$, there are at most $\lfloor b' - a'' \rfloor$ jobs contained in $[a'', b')$. Thus $[a'', b')$ is a constrained interval with respect to the original release times, which is a contradiction to the assumption that $[a, b)$ does not contain a constrained interval.

Thus there is no such interval $[a', b']$. It follows that all jobs in J' can be scheduled with respect to the modified release times. Thus modifying release times as in the discrete earliest deadline first rule does not eliminate any r -sets. Then choosing the jobs with earliest deadlines first clearly produces a best r -set. $\square$.

We now discuss the existence of a best r -set for an interval that can contain constrained intervals. A *multiple gap* is an interval $[a, b)$ such that a is the left endpoint of a gap, b is the right endpoint of a different gap, and thus there is at least one cover interval within $[a, b)$. An *anchored multiple gap* is an interval $[a, b)$ such that a is the left endpoint of an anchored gap, b is the right endpoint of a different anchored gap, and thus there is at least one cover interval within $[a, b)$. Our algorithm will compute best r -sets, if they exist, for certain anchored multiple gaps. We now prove the existence of the best r -set for an unrestricted interval $[a, b)$ and an $[a, b)$ -feasible set of jobs that is simple.

Theorem 3.1 (Best r -set) *Let $[a, b)$ be an interval, and J be an $[a, b)$ -feasible set of jobs that is simple. Then there is a (unique) best r -set for $[a, b)$ with respect to J .*

Proof: The proof is by contradiction.

We define an r -problem P to consist of an interval $[a, b)$ and an $[a, b)$ -feasible set of jobs J with distinct release times and distinct deadlines. Define a P -breakpoint to be either a or a breakpoint with respect to J . For any interval $[a, b)$, $a < b$, define $b \ominus a$, the *size* of $[a, b)$, to be the number of distinct P -breakpoints that lie in this interval. (We view the operation $\ominus$ as discretized subtraction, and use the symbol $\ominus$ as we would the minus sign. Thus we assume that the first operand is no smaller than the second.) Let the *size* of job i be the size of the interval $[r_i, d_i)$. Then the *size* of P is the size of interval $[a, b)$ plus the sum of the sizes of all jobs in J plus the sum of the product of sizes for all pairs of jobs in J :

$$\text{size}(P) = (b \ominus a) + \sum_{i \in J} (d_i \ominus r_i) + \sum_{i, j \in J, j \neq i} (d_i \ominus r_i) * (d_j \ominus r_j)$$

We consider an r -problem P , consisting of interval $[a, b)$ and set J of jobs, that is of smallest size among those r -problems that do not satisfy the theorem. Since P is of smallest size, $J_r[a, b) = J$. We shall eliminate all but two r -sets from consideration, and deduce various properties about schedules for these r -sets. We then use these properties to generate a contradiction.

Note that some job j in J has release time a , otherwise there would be a corresponding smaller r -problem P' , consisting of interval $[a', b)$ and set of jobs J , where a' is the next P -breakpoint after a . P' is a smaller r -problem than P , so P' would have a best r -set A' in interval $[a', b)$. The set A' would be the best r -set for P in interval $[a, b)$, which is a contradiction to the assumption that there is no best r -set in P . Thus, there is some job j with $r_j = a$.

We restrict our attention to two r -sets in P as follows. Consider all r -sets in $[a, b)$ that have every schedule in $[a, b)$ starting at a . Then any such r -set must contain job j , and j is scheduled starting at a in such a schedule. We infer a smaller r -problem P' with interval $[a + 1, b)$ and set of jobs $J - \{j\}$. There is a best r -set A' for P' . Let $A_1 = A' \cup \{j\}$. A_1 is an r -set for P . We next consider all r -sets in $[a, b)$ that have some schedule in $[a, b)$ that does not start at a . Let a' be the next breakpoint after a . We infer a smaller r -problem P'' with interval $[a', b)$ and set of jobs J . (Note that job j will not be in any r -set for P'' .) There is a best r -set A_2 for P'' . If $|A_2| < |A_1|$, then A_1 is the best r -set for P , contradicting our assumption that there is no best r -set for P . Thus $|A_2| = |A_1|$, and A_2 is an r -set for P .

Any r -set A for P must satisfy either $A_1 \leq_d A$ or $A_2 \leq_d A$, since any schedule for A in $[a, b)$ either starts at a or it doesn't. Thus we can confine our attention to the r -sets A_1 and A_2 . Note that each job in J must appear in either A_1 , A_2 , or both. Otherwise, we could remove such a job and get a smaller r -problem. The smaller r -problem would have a best r -set, which would then be the best r -set for P , a contradiction. Let S_1 be a canonical schedule for A_1 in $[a, b)$, and S_2 be a canonical schedule for A_2 in $[a', b)$. By Theorem 2.1, canonical schedules exist for S_1 and S_2 . We note several properties of S_1 and S_2 .

Claim 3.1 *Every job that is scheduled in both S_1 and S_2 starts at its release time in one of the schedules and completes at its deadline in the other schedule. If a job is scheduled in just one of S_1 and S_2 , then the job starts at its release time in that schedule, and its deadline is greater than b .*

Proof of Claim: Let k be a job in J that is scheduled in either S_1 , S_2 , or both. Let s_k be k 's earliest starting time in the two schedules. Suppose $r_k < s_k$. We generate a new r-problem P' by resetting r_k to s_k . Note that resetting r_k does not generate any new breakpoints, since choosing S_1 and S_2 to be canonical schedules guarantees that s_k is a breakpoint. Also note that if this causes J to no longer be simple, we can just apply the appropriate operations to reset release times. P' is a smaller r-problem since the size of job k (and possibly some other jobs) has been reduced. Since P' is smaller than P , there is a best r-set A' in P' . Schedules S_1 and S_2 are both schedules in P' . Thus A_1 and A_2 are both r-sets in P' , so $A' \leq_d A_1$ and $A' \leq_d A_2$. Thus, A' is the best r-set for P , which is a contradiction to the assumption that there is no best r-set in P . Thus, $r_k = s_k$ for all k in J .

Suppose that k is scheduled in both S_1 and S_2 and let c_k be k 's latest completion time in the two schedules. Suppose $d_k > c_k$. We generate a smaller r-problem P' by resetting d_k to c_k . Again we perform any additional operations needed to keep J simple. Since P' is smaller than P , there is a best r-set A' in P' . Note that any job whose deadline is reset must have its deadline be at most b , and is thus in every r-set for P and for P' . Since both A_1 and A_2 also contain any job whose deadline is reset, $A' \leq_d A_1$ for P' if and only if $A' \leq_d A_1$ for P , and similarly for A' and A_2 . Since for any r-set A for P , either $A_1 \leq_d A$ or $A_2 \leq_d A$, A' is a best r-set for P , which is a contradiction to the assumption that there is no best r-set in P . Thus, $d_k = c_k$ for all k in J .

Suppose job k is scheduled in just one of S_1 and S_2 . Since $J = J_r[a, b]$ and J is $[a, b]$ -feasible, any job in J with deadline at most b must be scheduled in both S_1 and S_2 . Since job k is not in both S_1 and S_2 , $d_k > b$. This completes the proof of

Claim 3.1.

In the remainder of the proof, if resetting a release time or deadline causes J to no longer be simple, we apply the appropriate operations, as in the proof of the last claim, to make J simple once again. We note that whenever a deadline is changed, the job will appear in every corresponding r -set. Thus comparisons between r -sets using $\leq_d$ are not affected.

For schedule S_p , where $p = 1$ or $p = 2$, and for any job h in J that is scheduled in S_p , let s_h^p be the start time of h in S_p and c_h^p be the completion time of h in S_p . We say that the scheduled position of a job g in S_1 *overlaps* that of job h in S_2 if and only if $\max\{s_g^1, s_h^2\} < \min\{c_g^1, c_h^2\}$.

Claim 3.2 *For any job that is scheduled in both S_1 and S_2 , its scheduled positions in the two schedules overlap.*

Proof of Claim: Suppose the claim is false. Let k be the job with the smallest deadline that is scheduled in both S_1 and S_2 such that its scheduled positions in the two schedules do not overlap.

We introduce five transformations for generating smaller r -problems that result in contradictions. The first is called an *exchange-d* transformation. Let g and h be jobs in J with $r_h < r_g$, $d_h < d_g$, and $s_h^p < s_g^p \leq s_h^{3-p}$, where $p = 1$ or $p = 2$. We transform r -problem P into a new r -problem P' by replacing jobs h and g with jobs h' and g' , where $r_{h'} = r_h$, $d_{h'} = d_g$, $r_{g'} = r_g$, and $d_{g'} = d_h$. We call this transformation an *exchange-d*, as we exchange the deadlines for h and g . Schedules S_{3-p} and S_p for P are easily transformed into corresponding schedules S'_{3-p} and S'_p for P' in the following way. Schedule S'_{3-p} will be the same as S_{3-p} , except that g' will be in the position of h , and if g is in S_{3-p} then h' will be in the position of g . Schedule S'_p will be the same as S_p , except that h' will be in the position of h , and g' in the position of g . Thus the sets A'_{3-p} and A'_p corresponding to A_{3-p} and A_p are r -sets in P' .

A schedule S' in P' can be transformed to a schedule S in P in the following way. Any job in S' that is not h' or g' is scheduled in the same position in S . Note

that g' must be scheduled in S' , and h' may or may not be scheduled in S' . If h' is scheduled overlapping in $[r_{h'}, r_{g'}]$, then h is scheduled in S in the position of h' in S' and g is scheduled in S in the position of g' in S' . Otherwise, h is scheduled in S in the position of g' in S' , and if h' is scheduled in S' , then g is scheduled in S in the position of h' in S' . Any set A in P whose corresponding set A' in P' is an r -set in P' is itself an r -set in P . The size of P' is smaller than the size of P since:

$$\begin{aligned}
& (d_g \ominus r_h) * (d_h \ominus r_g) < ((d_g \ominus r_h) * (d_h \ominus r_g)) + ((d_g \ominus d_h) * (r_g \ominus r_h)) \\
& = ((d_g \ominus r_h) * (d_h \ominus r_g)) + (d_g \ominus d_h) * ((d_g \ominus r_h) - (d_g \ominus d_h) - (d_h \ominus r_g)) \\
& = ((d_g \ominus r_h) - (d_g \ominus d_h)) * ((d_h \ominus r_g) + (d_g \ominus d_h)) \\
& = (d_h \ominus r_h) * (d_g \ominus r_g)
\end{aligned}$$

Thus, P' has a best r -set A' . For set B as any of A_{3-p} , A_p or A , h is in B if and only if g' is in B' , and g is in B if and only if h' is in B' . Thus for set B as any of A_{3-p} , A_p or A , $B \leq_d B'$ and $B' \leq_d B$. Since $A' \leq_d A'_{3-p}$ and $A' \leq_d A'_p$, we have $A \leq_d A_{3-p}$ and $A \leq_d A_p$, and thus A is a best r -set for P , which is a contradiction to the assumption that P does not have a best r -set. Thus whenever an exchange-d transformation can be applied, a contradiction can be achieved. This concludes the discussion of an exchange-d transformation.

The second transformation is called a *compress* transformation. Let g and h be jobs in J with $c_g^p = s_h^p$, $c_g^{3-p} = s_h^{3-p}$, and $s_h^p \leq s_h^{3-p} < c_h^p$, where $p = 1$ or $p = 2$. We generate a new r -problem P' in which the jobs g and h are compressed into one job h' such that $r_{h'} = r_h$ and $d_{h'} = d_h$. For any job i with $r_i < r_g$, reset r_i to $r_i + 1$. For any job i with $d_i < d_g$, reset d_i to $d_i + 1$. Reset a to $a + 1$. Note that the sets A'_{3-p} and A'_p in P' that correspond to A_{3-p} and A_p in P are r -sets in P' . The new r -problem P' is smaller than P , and thus has a best r -set A' . Let $A = A' \cup \{g, h\} - \{h'\}$. Then A is an r -set for P . Note that any job other than g or h that has its deadline changed must be in every r -set for P' and in every r -set for P . It follows that $A \leq_d A_{3-p}$ and $A \leq_d A_p$.

Thus, A is a best r -set for P , which is a contradiction to our initial assumption. Thus whenever a compress transformation can be applied, a contradiction can be achieved.

The third transformation is a variation of the compress, called an *inverted compress*. Let g and h be jobs in J with $c_g^p = s_h^p$, $c_h^{3-p} = s_g^{3-p}$, and $s_g^p \leq s_h^{3-p} < c_g^p$, where $p = 1$ or $p = 2$. We generate a new r -problem P' in which the jobs g and h are compressed into one job h' such that $r_{h'} = s_h^p$ and $d_{h'} = d_g$. For any job i with $r_i < r_g$, reset r_i to $r_i + 1$. For any job i with $d_i < c_h^{3-p}$, reset d_i to $d_i + 1$. Reset a to $a + 1$. In a fashion analogous to that of the compress transformation, a contradiction can be achieved whenever an inverted compress transformation can be applied.

The fourth transformation is called an *increase- r* transformation. Let g and h be jobs in J with g scheduled before h in S_p , $c_g^p < s_h^p$, no other job is scheduled between g and h in S_p , and $c_g^p < d_g$, where $p = 1$ or $p = 2$. By Claim 3.1, $r_g = s_g^p$. We reset r_g to be the next larger breakpoint, giving a smaller r -problem P' , for which there would be a best r -set. This set would also be a best r -set for P , contradicting our initial assumption about P . Thus whenever an increase- r transformation can be applied, a contradiction can be achieved.

The fifth transformation is called a *decrease- d* transformation. Let g and h be jobs in J with g scheduled before h in S_p , $c_g^p < s_h^p$, no other job is scheduled between g and h in S_p , and $r_h < s_h^p$, where $p = 1$ or $p = 2$. By Claim 3.1, $d_h = c_h^p$. We reset d_h to be the next smaller breakpoint, giving a smaller r -problem P' , for which there would be a best r -set. Since job h must be in any r -set for P , changing its deadline does not affect whether or not an r -set is the best r -set for P . Hence the best r -set for P' would also be the best r -set for P , contradicting our initial assumption about P . Thus whenever a decrease- d transformation can be applied, a contradiction can be achieved.

We now proceed with a case analysis. Assume that $c_k^2 \leq s_k^1$. (The argument for $c_k^1 \leq s_k^2$ is essentially the same). By Claim 3.1, $r_k = s_k^2$ and $d_k = c_k^1$. Suppose as assumption (a1) that there is no job scheduled in S_2 during any part of the interval $[s_k^1, c_k^1]$. Then we can reset r_k to be s_k^1 , and get a smaller r -problem P' , for which

A_2 can still be scheduled, with job k in interval $[s_k^1, c_k^1)$. This would mean that there would be a best r -set for P' . This set would also be the best r -set for P , contradicting the assumption about P . Thus (a1) cannot hold, and there is a job h that is scheduled in S_2 during some part of the interval $[s_k^1, c_k^1)$.

Either $s_h^2 \leq s_k^1 < c_h^2 \leq c_k^1$, or both $s_k^1 < s_h^2 < c_k^1 < c_h^2$ and there is no job i such that $s_i^2 \leq s_k^1 < c_i^2$. Suppose as (a2) that $s_h^2 \leq s_k^1 < c_h^2 \leq c_k^1$. By Claim 3.1 either $r_h = s_h^2$ or $d_h = c_h^2$. Suppose as (a2.1) that $r_h = s_h^2$. Then an exchange-d transformation can be applied to jobs h and k . Thus (a2.1) does not hold, and $r_h \neq s_h^2$.

Thus we have that $d_h = c_h^2$. Since deadlines are distinct, $d_h < d_k$. Since k is the job with smallest deadline whose positions in S_1 and S_2 do not overlap, job h is scheduled in positions in S_1 and S_2 that overlap. Thus job h is the job that precedes k in S_1 . Furthermore, $c_h^1 = s_k^1$, since otherwise we could apply an increase- r transformation. Let g be the job that precedes job h in S_2 . We have that $c_g^2 = s_h^2$, since otherwise we could apply an decrease-d transformation. Either $g \neq k$ or $g = k$. Suppose as (a2.2) that $g \neq k$. Either $d_g = c_g^2$ or $r_g = s_g^2$. Suppose as (a2.2.1) that $d_g = c_g^2$. By choice of job k , the positions of g in S_1 and S_2 overlap. Thus job g is the job that precedes h in S_1 . Furthermore, $c_g^1 = s_h^1$ since otherwise we could apply an increase- r transformation. A compress transformation can now be applied to jobs g and h . Thus (a2.2.1) does not hold, and $r_g = s_g^2$. Then $s_g^1 \geq c_k^1$ and $r_k < r_g$. An exchange-d operation can now be applied to jobs k and g . Thus (a2.2) does not hold, so $g = k$. Since $g = k$, we can apply an inverted compress transformation for k and h , which leads to a contradiction.

Thus (a2) does not hold, which means that there is a job h such that $s_k^1 < s_h^2 < c_k^1 < c_h^2$ and there is no job i such that $s_i^2 \leq s_k^1 < c_i^2$. Let job m be the job that precedes job h in S_2 . Since there is no job i such that $s_i^2 \leq s_k^1 < c_i^2$, $c_m^2 < s_h^1$. We have that $r_h = s_h^2$, since otherwise we could apply a decrease-d transformation on jobs m and h . Let job l be the job that precedes job k in S_1 . Note that $c_l^1 = s_k^1$, since otherwise we could apply a decrease-d transformation on jobs l and k . Also, $d_l < d_k$, since otherwise we could apply an exchange-d transformation for k and l . By

choice of k , the scheduled positions of l in S_1 and S_2 overlap, and since there is no job i such that $s_i^2 \leq s_k^1 < c_i^2$, it follows that $m = l$. Since there is no job i such that $s_i^2 \leq s_k^1 < c_i^2$, we have $c_l^2 \leq s_k^1$, which means that $c_l^2 \leq c_l^1$, from which it follows that $d_l = c_l^1$. Then $c_l^2 = c_l^1$, since otherwise we could apply an increase- r transformation to l and h . Since $c_l^1 = d_l$ and $c_l^2 = c_l^1$, it follows that $s_l^2 = s_l^1$.

Let g be the job that precedes job l in S_2 . Either $g = k$ or $g \neq k$. Suppose as (a3) that $g = k$. We must have $c_g^2 = s_l^2$, since otherwise we could apply an increase- r transformation for g and l . We can then apply an inverted compress transformation for k and l . This transformation leads to a contradiction. Thus (a3) does not hold, and $g \neq k$. Let f be the job that immediately precedes job l in S_1 . Either $f = g$ or $f \neq g$. Suppose as (a4) that $f = g$. It follows that $r_g = d_g - 1$, by an argument similar to the one that showed that $r_l = d_l - 1$. We can then apply a modified compress transformation to g and l , with the only difference being the following. For any i with $r_i < r_g$, reset r_i to $r_i + r_l - r_g$. For any i with $d_i < d_l$, reset d_i to $d_i + r_l - r_g$. This leads to a contradiction.

Thus (a4) does not hold, and $f \neq g$. Either $s_g^2 = r_g$ or $c_g^2 = d_g$. Suppose $s_g^2 = r_g$. Then we can perform an exchange- d operation on jobs g and k , which leads to a contradiction. Thus $c_g^2 = d_g$. By choice of k the positions of g in S_1 and S_2 overlap, which means that the positions of f in S_1 and S_2 cannot overlap. Thus $s_f^1 = r_f$ and $d_f > d_k$. But then we can reset d_k to be c_f^1 and r_f to be s_k^1 . This gives a smaller r -problem P' , which will thus have a best r -set. It follows that this set will also be a best r -set for P , a contradiction. At this point, all cases have been exhausted. Thus, there can be no job k whose scheduled positions in S_1 and S_2 do not overlap. This completes the proof of Claim 3.2.

We are now ready to generate the contradiction to the assumption that the theorem does not hold. Let j be the job scheduled at a in S_1 and let h be the first job scheduled in S_2 . If $j = h$, then we generate a new r -problem P' by deleting h and resetting a to $a + 1$. Then P' is smaller than P and thus it has a best r -set A' . Let $A = A' \cup \{h\}$. Then $A \leq_d A_1$ and $A \leq_d A_2$, which yields a contradiction, so $j \neq h$.

Since no other job in S_2 can overlap j in S_1 , job j is not in S_2 , by Claim 3.2. Either $s_h^2 \geq c_j^1$ or $s_h^2 < c_j^1$. If $s_h^2 \geq c_j^1$, then $A_2 \cup \{j\}$ can be scheduled in $[a, b)$, since j can be scheduled in $[a, a+1)$ and A_2 can be scheduled in $[a+1, b)$. But this would contradict A_2 being an r -set for P . Thus $s_h^2 < c_j^1$. It follows that $s_h^2 = r_h$. Either h is scheduled in S_1 or it isn't. If h is not scheduled in S_1 , then $d_h > b$. We generate a new r -problem P' by removing j and h and resetting a to $a+1$. In a similar manner to the argument above, this yields a contradiction. If h is scheduled in S_1 , then $s_h^1 < c_h^2$. Since $d_j > b$, we generate a new r -problem P' by removing h and resetting r_j to r_j+1 and a to $a+1$. In a similar manner to the argument above, this yields a contradiction. At this point we have exhausted all cases. The theorem then follows that there is a best r -set for $[a, b)$. $\square$.

Corollary 3.1.1 *Let $[a, b)$ be an interval, and J be an $[a, b)$ -feasible set of jobs with distinct deadlines. Then there is a (unique) best r -set for $[a, b)$ with respect to J .*

Proof: If deadlines are distinct, then there do not exist two different subsets A and B of jobs such that $A \leq_d B$ and $B \leq_d A$. If there are any jobs in J with the same release time, do the following. For the subset of jobs in J with release times less than a , arbitrarily reset all release times to be distinct values less than a . This cannot change any r -set for $[a, b)$. Next, perform the appropriate operations to reset release times until all remaining release times are distinct. Note that a best r -set for $[a, b)$ will remain a best r -set for $[a, b)$. By Theorem 3.1 there will be a unique best r -set for $[a, b)$ in the transformed problem. Thus there will be a unique best r -set for $[a, b)$ in the original problem. $\square$.

We consider the problem of determining a best r -set for an anchored multiple gap, given the best r -sets for two adjacent anchored multiple gaps that span it. We first consider the simpler problem of recomputing the best r -set for an interval $[a, b)$ when one additional job with release time a is inserted into the set of jobs. We show that

if the best r -set for $[a, b)$ changes at all, then the only change is that the new job replaces one of the jobs in the best r -set.

Lemma 3.2 *Let $[a, b)$ be an interval, and J and $J' = J \cup \{j'\}$ be $[a, b)$ -feasible sets of jobs with distinct deadlines, where j' is a job not in J with $r_{j'} = a$. Let A_1 be the best r -set with respect to J , and A_2 the best r -set with respect to J' . Then $A_2 \subseteq A_1 \cup \{j'\}$.*

Proof: The proof is by contradiction and is similar in structure to the proof of Theorem 3.1. We first note that by Corollary 3.1.1, best r -sets A_1 and A_2 exist.

We define an r^+ -problem P to consist of an interval $[a, b)$, a set of jobs J , and an additional job $j' \notin J$ with $r_{j'} = a$ such that $J' = J \cup \{j'\}$ is $[a, b)$ -feasible. We define P -breakpoint, size of an interval, and size of a job as in the proof of Theorem 3.1. The size of r^+ -problem P is the size of interval $[a, b)$ plus the sum of the sizes of all jobs in J' plus the sum of the product of sizes of all pairs of jobs in J' .

We consider an r^+ -problem P , consisting of interval $[a, b)$, set J of jobs, and additional job j' , that is of smallest size among those r^+ -problems that do not satisfy the lemma. Since P is of smallest size, $J_r[a, b) = J$ and $J'_r[a, b) = J'$. Suppose that $A_2 \not\subseteq A_1 \cup \{j'\}$. Clearly, $j' \in A_2$, since otherwise $A_2 = A_1$.

Let S_1 be any schedule for A_1 , and S_2 be any schedule for A_2 . We note several properties of S_1 and S_2 .

Claim 3.3 *Every job that is scheduled in both S_1 and S_2 starts at its release time in one of the schedules and completes at its deadline in the other schedule. Job j' completes in S_2 at its deadline. If a job in J is scheduled in just one of S_1 and S_2 , then the job starts at its release time in that schedule, and its deadline is greater than b .*

Proof of Claim: Let j be a job in J that is scheduled in either S_1 , S_2 , or both. Let s_j be j 's earliest starting time in the two schedules. Suppose $r_j < s_j$. We generate a new r^+ -problem P' by resetting r_j to s_j . Clearly A_1 is the best r -set in P' with respect to J and A_2 is the best r -set in P' with respect to J' . P' is a smaller r^+ -problem since

the size of job j has been reduced. Since P' is smaller than P , $A_2 \subseteq A_1 \cup \{j'\}$, which contradicts the assumption that P does not satisfy the lemma. Thus, $r_j = s_j$.

Suppose that job j is scheduled in both S_1 and S_2 and let c_j be j 's latest completion time in the two schedules. Suppose $d_j > c_j$. We generate a new r^+ -problem P' by resetting d_j to c_j . If this causes two jobs to have the same deadline, then apply the appropriate operation to reset deadlines, as was discussed in the proof of Theorem 3.1. Any job whose deadline is changed must have its deadline be at most b , and thus will be in every r -set for P' with respect to J and with respect to J' , and similarly for P . Thus A_1 and A_2 remain best r -sets for P' with respect to J and J' , respectively. Since P' is smaller, it follows that $A_2 \subseteq A_1 \cup \{j'\}$, which gives a contradiction. Thus, $d_j = c_j$.

If job j' completes in S_2 before its deadline, then $d_{j'}$ can be reset to $c_{j'}$. Note that this does not affect the comparisons for best r -set A_2 , since j' is in A_2 . This once again gives a smaller r^+ -problem, yielding a contradiction.

Suppose job $j \in J$ is scheduled in one of S_1 and S_2 . Since $J = J_r[a, b]$ and J is $[a, b]$ -feasible, any j in J with deadline at most b must be scheduled in both S_1 and S_2 . Since job j is not in both S_1 and S_2 , $d_j > b$. This completes the proof of the claim.

Claim 3.4 *For any job in J that is scheduled in both S_1 and S_2 , their scheduled positions in the two schedules overlap.*

Proof of Claim: The proof of this claim is similar to the proof of Claim 3.2 in Theorem 3.1, but in this lemma we are addressing r^+ -problems and derive contradictions to the assumption of this lemma. The proof of this claim is a straightforward transformation of the proof of Claim 3.2, and is omitted.

We are now ready to generate the contradiction to the assumption that the lemma does not hold. Suppose j' is not the last job scheduled in S_2 . Let h be the last job scheduled in S_2 and k be the last job scheduled in S_1 . If $k = h$, then we can generate a smaller r^+ -problem by deleting this job and subtracting 1 from b , and thus achieve

a contradiction. Thus, $k \neq h$. Suppose $s_k^1 \geq s_h^2$. Then $s_k^1 = r_k$ and $d_k > b$. If $d_h > b$, then replace h by k in S_2 , and generate a smaller r^+ -problem, since there is one less job, and achieve a contradiction. Thus $d_h \leq b$, which means that the job scheduled before k in S_1 is h . But then we can generate a smaller r^+ -problem by removing h , resetting the release time of k to $r_k - 1$ and resetting the interval to $[a, b - 1)$. This again leads to a contradiction. The argument is similar if $s_k^1 < s_h^2$.

Thus j' is the last job scheduled in S_2 . Let h be the first job scheduled in S_2 and k be the first job scheduled in S_1 . If $k = h$, then we can generate a smaller r^+ -problem, which gives a contradiction, so $k \neq h$. Suppose $s_h^2 \leq s_k^1$. Then $s_h^2 = r_h$ and $d_h > b$. Then h and j' can be swapped in S_2 and the r_h reset to $s_{j'}^2$, before the swap. The size of a job has been reduced, resulting in a smaller r^+ -problem, which gives a contradiction, so $s_h^2 > s_k^1$. Then, $d_k > b$. If $d_h > b$, then again, h and j' can be swapped in S_2 , generating a smaller r^+ -problem, which leads to a contradiction. Thus, $d_h \leq b$, which means that the second job scheduled in S_1 is h . But this leads to a smaller r^+ -problem. This achieves the final contradiction, as we have shown that $s_h^2 \leq s_k^1$, $s_h^2 \geq s_k^1$, and $s_h^2 \neq s_k^1$. Thus, all jobs except $j' \in S_2$ appear in S_1 . $\square$

We now consider recomputing the best r -set when a set of additional jobs is introduced. We show that a job that is not in the original best r -set cannot appear in the recomputed best r -set.

Theorem 3.2 *Let $[a, b)$ be an interval, and J' and $J \subset J'$ be $[a, b)$ -feasible sets of jobs with distinct deadlines, and where each job $j' \in J' - J$ has $r_{j'} = a$. Let A_1 be the best r -set with respect to J , and let A_2 be the best r -set with respect to J' . Then $A_2 \subseteq A_1 \cup (J' - J)$.*

Proof: We first note that by Corollary 3.1.1, best r -sets A_1 and A_2 exist. Our proof is by induction on $|J' - J|$. For the basis, we have $|J' - J| = 1$. The basis case holds by Lemma 3.2. For the induction step, we have $|J' - J| > 1$. Assume as the induction hypothesis that the theorem holds for all values smaller than $|J' - J|$. Let

j' be a job in $J' - J$. Let A_1 be the best r -set for $[a, b)$ with respect to J . By the induction hypothesis, the best r -set A_2 for $[a, b)$ with respect to $J' - \{j'\}$ satisfies $A_2 \subseteq A_1 \cup ((J' - J) - \{j'\})$. By Lemma 3.2, the best r -set A_3 for $[a, b)$ with respect to J' satisfies $A_3 \subseteq A_2 \cup \{j'\}$, which implies $A_3 \subseteq A_1 \cup (J' - J)$. $\square$.

In the last part of this section we concentrate on computing the best r -sets for intervals that contain cover intervals. As shown in Lemma 3.1, an interval $[a, b)$ that has no constrained interval contained within it (and hence no cover interval contained within it) is $[a, b)$ -feasible and the corresponding schedule can be computed easily using the discrete earliest deadline rule. Determining $[a, b)$ -feasibility is more involved when the interval contains cover intervals because we do not know in advance when the first job of a cover interval should be started. For certain choices of the starting time of the first job in a cover interval, there may be no schedule. Our approach is to consider all possible starting times, and compute a set of jobs that is the best r -set if the set is $[a, b)$ -feasible. Then the $[a, b)$ -feasibility of the set can be tested by comparing the set of deadlines of its jobs with a “template” generated from a “mirror image problem”. The template is composed of the smallest allowable deadlines for $[a, b)$ that result in a schedule. If the deadlines of the jobs in the proposed best r -set are greater than or equal to those in the template, then the interval is $[a, b)$ -feasible.

First we define the notions of d -set, best d -set, and template. Let $[a, b)$ be an interval, J a set of jobs, and $J_d[a, b)$ be the subset of all jobs j in J such that $d_j \in (a, b)$. The set A is a d -set for $[a, b)$ with respect to J if and only if J is $[a, b)$ -feasible and A is a maximum-cardinality subset of $J_d[a, b)$ that includes all jobs in $J[a, b)$, and the jobs in A can be scheduled in $[a, b)$. We choose the name d -set, where d denotes deadline, to stress the partitioning of jobs by their deadlines. A d -set A for $[a, b)$ with respect to J is a *best d -set* for $[a, b)$ with respect to J if for any other d -set B for $[a, b)$ with respect to J , $A \leq_d B$. The set of deadlines of the best d -set for $[a, b)$ with respect to J is called a *template* for $[a, b)$ with respect to J .

We discuss examples of templates using the set of jobs in Figure 1a. There are

five jobs whose deadlines lie within $(15.0, 20.0]$, jobs 2, 8, 9, 10, and 13. The best d-set for $[a, b)$ consists of the four jobs 2, 8, 9 and 10. Thus the template for $[15.0, 20.0)$ consists of the deadlines of jobs 2, 8, 9, and 10. The deadline of job 13 is not in the template since only one of jobs 9 and 13 can be scheduled in $(15.0, 16.4)$, and job 13 has a later deadline. Similarly, the template for $[5.3, 8.2)$ consists of the deadline of job 3, and the template for $[10.2, 13.0)$ consists of the deadlines of jobs 7 and 1.

A template can be used to test for feasibility as follows. Let $[a, b)$ and $[a', b')$ be two consecutive anchored multiple gaps, where $a' - b$ equals the number of jobs in the cover interval separating them. Note that the set $J_r[b, a')$ includes the cover jobs that lie in the cover interval and possibly some anomalous gap jobs that don't start within a gap. Assume that J is $[a, b)$ -feasible, $[a', b')$ -feasible, and $[b, a')$ -feasible. We want to determine if J is $[a, b')$ -feasible. Let J' be all jobs in $J_r[a, b')$ except those in the best r-set for $[a, b)$ and those in the cover interval, where each job j in J' with $r_j < a'$ has its release time reset to a' . (This resetting does not change the problem, since any job in J' will not be chosen to be scheduled before a' .) Then J' is $[a', b')$ -feasible if and only if $A \leq_d B$, where A is the best d-set for $[a', b')$ with respect to J and B is the best r-set for $[a', b')$ with respect to J' . In other words, jobs in J' that are contained in $[a', b')$ can be scheduled if and only if the template for $[a', b')$ with respect to J precedes (in $\leq_d$) the deadlines for the best r-set for $[a', b')$ with respect to J' .

We illustrate how a template is used. Consider the set of jobs in Figure 1a, but with $r_9 = 12.1$. The best r-set for anchored multiple gap $[5.3, 13.0)$ will have jobs 1 and 3 scheduled in the gap $[5.3, 8.0)$, and jobs 4, 7 and 2 in gap $[10.0, 13.0)$. It is not possible to schedule the remaining jobs 8, 9, 10, and 13 in $[15.0, 20.0)$. Job 13, with second smallest deadline, has a deadline smaller than the second smallest value, 17.1, in the template. Since $\{9, 10, 2, 8\} \not\leq_d \{9, 13, 10, 8\}$, there is no schedule for this choice of anchored gaps.

We next discuss how the existence of best r-sets relates to the existence of best d-sets. Given a set J of jobs that constitute a problem P , the *mirror image* problem P^M is defined as follows. For each job i in J with release time r_i and deadline

d_i , there is a job i in J^M with release time $r_i^M = r_{\min} + (d_{\max} - d_i)$ and deadline $d_i^M = d_{\max} - (r_i - r_{\min})$. It would be convenient if i were in the best d-set in P^M if and only if i were in the best r-set in P . Unfortunately, this is not the case, since in both problems P and P^M , the relation $\leq_d$ is based on deadlines. We show the existence of the best d-set below in Theorem 3.3, and also show how to generate it using a modified mirror image problem.

Theorem 3.3 (Best d-set) *Let $[a, b]$ be an interval. Let J be an $[a, b]$ -feasible set of jobs that is simple. Then there is a best d-set for $[a, b]$ with respect to J .*

Proof: Let $[a, b]$ be an interval, J an $[a, b]$ -feasible set of jobs, and let P be a d-problem consisting of finding the best d-set for the interval $[a, b]$ with respect to J , if such a set exists. We generate a new d-problem $\hat{P}$ with job set $\hat{J}$ in the following way. For each job $j \in J$ with release time r_j and deadline d_j there is a job j in $\hat{J}$ with deadline $\hat{d}_j = d_j$ and release time $\hat{r}_j = r_j$ if $r_j \geq a$, and $\hat{r}_j = a - (d_j - a)$ otherwise. It follows that the release times $\hat{r}_j$ in $\hat{P}$ are all distinct, as are the deadlines $\hat{d}_j$. The d-problem $\hat{P}$ is to find the best d-set for the interval $[a, b]$ with respect to $\hat{J}$. A set is the best d-set for P if and only if it is the best d-set for $\hat{P}$, since the only differences in jobs in J and $\hat{J}$ are modified release times that are outside of the interval $[a, b]$. The release times are modified in such a way that for any two jobs j and k in J with $r_j, r_k < a$, if $d_j < d_k$, then $\hat{r}_j > \hat{r}_k$.

Let $\hat{P}^M$ be an r-problem formed by taking the mirror image of d-problem $\hat{P}$, as follows. For each job j let release time $r_j^M = \hat{r}_{\min} + (\hat{d}_{\max} - \hat{d}_j)$ and deadline $d_j^M = \hat{d}_{\max} - (\hat{r}_j - \hat{r}_{\min})$. Let $a^M = \hat{r}_{\min} + (\hat{d}_{\max} - b)$ and $b^M = \hat{d}_{\max} - (a - \hat{r}_{\min})$. Since J is $[a, b]$ -feasible, $\hat{J}$ is $[a, b]$ -feasible, and $\hat{J}^M$ is $[a^M, b^M]$ -feasible. Thus there is an r-set for $[a^M, b^M]$ with respect to $\hat{J}^M$. Since the release times $\hat{r}_j$ in $\hat{P}$ are distinct, and the deadlines $\hat{d}_j$ are also distinct, it follows that the release times r_j^M in $\hat{P}^M$ are distinct, and the deadlines d_j^M are also distinct. By Theorem 3.1, there is a best r-set for $[a^M, b^M]$ with respect to $\hat{J}^M$. This best r-set is equivalent to the best d-set for $[a, b]$ with respect to $\hat{J}$, which is equivalent to the best d-set for $[a, b]$ with respect to

J . $\square$.

The proof of Theorem 3.3 identifies a method for computing d-sets. When an interval $[a, b)$ does not contain a prime interval, computing the best d-set for it is easy. As in the previous proof, modify the release times of the jobs, transform the problem into a mirror image problem, and compute an r-set using the discrete earliest deadline rule. This best r-set is equivalent to the best d-set in the original problem.

When an interval $[a, b)$ does contain a prime interval, we can construct its best d-set by combining the best d-sets of two subproblems. We need a lemma similar to Theorem 3.2, which applied to r-sets. When we combine the best d-sets of two subproblems, we will reset the deadlines of certain jobs to b . To preserve some way of breaking ties in finding a best d-set, we introduce the notion of an *original deadline* $\bar{d}_j$. Among best d-sets for $[a, b)$, we choose that one whose set of original deadlines is smallest to be the *best d-set subject to original deadlines*.

Lemma 3.3 *Let $[a, b)$ be an interval, and J' and $J \subset J'$ be $[a, b)$ -feasible sets of jobs, where jobs in J' have distinct release times and distinct original deadlines $\bar{d}_j$, and $\bar{d}_j = d_j$ for each job j in J and $\bar{d}_{j'} \geq d_{j'} = b$ for each job $j' \in J' - J$. Let A_1 be the best d-set with respect to J , and let A_2 be the best d-set with respect to J' , subject to original deadlines. Then $A_2 \subseteq A_1 \cup (J' - J)$.*

Proof: The proof is similar to that of Theorem 3.3 except that it appeals to Theorem 3.2 rather than Theorem 3.1. Let $\widehat{J}'$ be the set of jobs with release times modified as follows. For each job $j \in J'$ there is a job $\hat{j}$ in $\widehat{J}'$ with deadline $\hat{d}_j = d_j$ and release time $\hat{r}_j = r_j$ if $r_j \geq a$, and $\hat{r}_j = a - (\bar{d}_j - a)$ otherwise. Note that all $\hat{r}_j$ will be distinct. Let $\widehat{J}$ be the subset of $\widehat{J}'$ corresponding to the subset J of J' . Let $a^M = \hat{r}_{\min} + (\hat{d}_{\max} - b)$ and $b^M = \hat{d}_{\max} - (a - \hat{r}_{\min})$. Let $\widehat{J}^M$ be the mirror image set of jobs corresponding to $\widehat{J}'$, and $\widehat{J}^M$ be the mirror image subset of jobs corresponding to $\widehat{J}$. Since J' is $[a, b)$ -feasible, $\widehat{J}'$ is $[a, b)$ -feasible, and $\widehat{J}^M$ is $[a^M, b^M)$ -feasible. Thus $\widehat{J}^M$ is $[a^M, b^M)$ -feasible, and there is an r-set for $[a^M, b^M)$ with respect to $\widehat{J}^M$. Note

that all $\hat{d}_j^M$ will be distinct, since all $\hat{r}_j$ are distinct. By Corollary 3.1.1, there is a best r-set $\hat{A}_1^M$ for (a^M, b^M) with respect to $\hat{J}^M$, and a best r-set $\hat{A}_2^M$ for (a^M, b^M) with respect to $\hat{J}'^M$. By Theorem 3.2 $\hat{A}_2^M \subseteq \hat{A}_1^M \cup (\hat{J}'^M - \hat{J}^M)$. These best r-sets are equivalent to best d-sets $\hat{A}_1$ and $\hat{A}_2$ for $[a, b]$ with respect to $\hat{J}$ and $\hat{J}'$, resp., which are equivalent to best d-sets A_1 and A_2 for $[a, b]$ with respect to J and J' , resp. The theorem then follows. $\square$.

The computation of best d-sets for intervals that contain prime intervals is similar to the computation of r-sets, and will be discussed in more detail in the next section.

4 The NC Algorithm

In this section we describe a parallel divide-and-conquer algorithm for determining if there is a schedule, and if so, generating it. The algorithm consists of four steps, plus a preprocessing step. The preprocessing step replaces the original set of jobs with an equivalent set of jobs, in which all release times are distinct, all deadlines are distinct, and each deadline is a breakpoint. The first step uses the characterization of section 2 to form a cover and to label the jobs as either cover jobs or gap jobs. Then the jobs are partitioned, assigning each job to either a cover interval or a gap based on release times. A second partition is also generated, based on deadlines. The second step imposes a balanced binary tree structure on the problem, with the leaves representing gaps in order from earliest to latest, and with each nonleaf node representing a multiple gap containing the gaps represented by its leaf descendants. The characterizations of section 3 are used to compute best r-sets and best d-sets for anchored gaps and anchored multiple gaps corresponding to the tree nodes. If there is a best r-set that includes all jobs, then there is a schedule; otherwise, the algorithm halts. If a schedule exists, then the third step obtains a schedule of the gap jobs by starting with the largest anchored multiple gap and its best r-set, and repeatedly selecting two constituent anchored multiple gaps of an anchored multiple gap and

splitting the corresponding set of jobs into smaller sets until only anchored gaps and their corresponding sets remain. The jobs within each of these sets can easily be scheduled within their assigned anchored gap. Given the endpoints of the anchored gaps, the fourth step schedules the cover jobs in the cover intervals, using Lemma 2.1. We discuss each step carefully, and analyze its time and processor requirements.

The preprocessing step first makes every deadline a breakpoint. For each job j , the fractional parts $u_j = r_j - \lfloor r_j \rfloor$ and $v_j = d_j - \lfloor d_j \rfloor$ of its release time and deadline are determined. The multiset of values u_j is then sorted. For each j , a binary search is performed in the sorted list to find the largest u_i no larger than v_j . (If there is no such u_i , then u_i is taken to be -1 plus the largest value in the list.) Then d_j is reset to be $d_j - v_j + u_i$.

Next the preprocessing step increases certain release times so that all release times are distinct, and decreases certain deadlines so that all deadlines are distinct. As stated in section 3, we wish to perform the following operations apply, while they apply. If $r_j = r_k$ and $d_j \leq d_k$ for jobs j and k , then reset r_k to be $r_j + 1$. If $r_j < r_k$ and $d_j = d_k$, then reset d_j to be $d_k - 1$. We first describe how to handle all instances of the first operation in parallel, and then all instances of the second.

First we describe how to make all release times distinct. For each release time r_j , its fractional part $u_j = r_j - \lfloor r_j \rfloor$ is extracted. The multiset of the values u_j are then sorted. Then the jobs j are partitioned into sets R_u such that $u_j = u$. For each set R_u , the following is done. The parallel version of the discrete earliest deadline first rule [AGK, R] is applied to the set. If no schedule is possible, then our algorithm halts with failure. Otherwise, reset the release time of job j to be its starting time in the schedule.

To make all deadlines distinct first convert the problem into a mirror image problem. The release time r_j^M for job j in P^M is set to $r_{\min} + (d_{\max} - d_j)$, and the deadline d_j^M for job j in J^M is set to $d_{\max} - (r_j - r_{\min})$. Then the above algorithm for resetting release times is run, but without resetting the release times. Instead, the deadline of job j is reset to $d_j - (s_j - (r_{\min} + d_{\max} - d_j)) = d_{\max} + r_{\min} - s_j$, where s_j is the

starting time of job j in the corresponding schedule for the mirror image problem.

Lemma 4.1 *Given a set J_0 of n unit-time jobs with arbitrary release times and deadlines, an equivalent simple set J with all deadlines being breakpoints can be found in $O(\log n)$ time using $O(n)$ processors.*

Proof: The correctness of the above procedure is established as follows. By Theorem 2.1, deadlines need only be breakpoints. Next consider generating simple set J . When applying the first of the above operations, the only jobs that can have equal release times at some point are those that have an equal fractional part. The discrete earliest deadline rule schedules correctly for any set of jobs, all of whose release times differ from some value by an integral amount. The discrete earliest deadline rule always schedules a job at the earliest possible release time, subject to no other job being available and having an earlier deadline. Thus the starting time of the job corresponds to the release time generated by the repeated application of the above operation. The application of the discrete earliest deadline rule to the mirror image problem gives a schedule in which every job starts as late as possible. Correctness then follows.

We analyze the resource bounds as follows. Sorting will use $O(\log n)$ time on $O(n)$ processors, and performing n binary searches in parallel, as well as the parallel version of the discrete earliest deadline rule [AGK, R], will use the same resources. $\square$.

The first step identifies subproblems that can be solved independently. The subproblems are formed by finding a cover and its associated gaps, and partitioning the jobs into sets that are associated with either a cover interval or a gap. The cover is found by forming the set of all possible constrained intervals and then deleting those that are neither prime nor compatible. First the constrained intervals are identified. For each pair consisting of a release time r_i and a deadline d_j , where $r_i < d_j$, let $n_{i,j}$ be the number of jobs contained in this interval. If $d_j - r_i < n_{i,j}$, then the algorithm

halts, as no schedule exists. If $d_j - r_i - n_{i,j} < 1$, then $[r_i, d_j)$ is a constrained interval. Second, for each release time r_i , if there is more than one constrained interval starting at r_i , then all such intervals except for the one with the smallest deadline are deleted. Similarly, for each deadline d_j , if there is more than one constrained interval ending at d_j , then all such intervals except for the one with the largest release time are deleted. At most n constrained intervals will remain.

Third, the prime intervals are identified. Each constrained interval is compared with every other constrained interval and deleted if it contains such an interval. The fourth step is to form a cover. The prime intervals $[a_i, b_i)$ are sorted on the values a_i . Since no interval is contained in another, they are also sorted by b_i . For each $[a_i, b_i)$ binary search is used to determine the prime interval $[a_j, b_j)$ with $i < j$ such that $[a_i, b_i)$ and $[a_j, b_j)$ are compatible and for any k , $i < k < j$, $[a_i, b_i)$ and $[a_k, b_k)$ are not compatible. Using recursive doubling, a maximal set of prime intervals that are compatible is identified, and those prime intervals that are not compatible with one of the selected prime intervals are deleted. The remaining prime intervals constitute a cover.

Having identified a cover, the gaps are then identified. The set of jobs are then partitioned in the two partitions as follows. Any job contained in a cover interval is a cover job, and is assigned to the cover interval in both partitions. The remaining jobs are gap jobs, and are assigned as follows. For the partition based on release times, if the release time of a gap job falls within a gap, then the job is assigned to that gap. Otherwise, the gap job is called *anomalous*, and it is assigned to the cover interval containing its release time. For the partition based on deadlines, if the deadline of a gap job falls within a gap, then the job is assigned to that gap; otherwise, the gap job is assigned to the cover interval containing its deadline.

Lemma 4.2 *Given a set of n unit-time jobs with arbitrary release times and deadlines, a cover can be computed in $O(\log n)$ time using $O(n^2/\log n)$ processors.*

Proof: The correctness of the above procedure follows from the definition of a cover.

We next analyze the time complexity. Assuming that jobs are indexed by nondecreasing deadlines, the $n_{i,j}$ are computed as follows. Let $v_{i,j}$ indicate whether job j lies within $[r_i, d_j)$, so $v_{i,j} = 1$ if $r_j \geq r_i$ and $v_{i,j} = 0$ if $r_j < r_i$. The $n_{i,j}$ are computed by performing a prefix sum over the $v_{i,j}$'s, $n_{i,j} = v_{i,1} + v_{i,2} + \dots + v_{i,j}$. Using Brent's Theorem [B], this uses $O(n^2/\log n)$ processors. Reducing the number of constrained intervals under consideration to at most n also uses $O(n^2/\log n)$ processors. Assigning one processor to $\log n$ pairs of constrained intervals, identifying prime intervals and finding a compatible set uses $O(n^2/\log n)$ processors. Each of the above activities uses $O(\log n)$ time. $\square$.

We next discuss the second step in our algorithm. It first imposes a balanced binary tree structure on the problem, with the leaves representing gaps in order from earliest to latest, and with each nonleaf node representing a multiple gap containing the gaps represented by its leaf descendants. It then computes best r-sets and best d-sets, if they exist, for anchored gaps and anchored multiple gaps corresponding to the tree nodes, using a bottom-up sweep through the tree. The final result at the root of the tree will be the best r-set and the best d-set for $[r_{min}, d_{max})$. The best r-sets are in sorted order by deadlines, not in scheduled order, so that they can be merged with other best r-sets quickly. Best d-sets are also in sorted order by deadlines.

We first discuss computing best r-sets and best d-sets for anchored gaps. Let $[a, b)$ be a gap, preceded by a cover interval of looseness x and followed by a cover interval of looseness x' . Associated with gap $[a, b)$ are the anchored gaps $[a_i, b_h)$, where $a_i \in [a, a+x)$, and $b_h \in (b-x', b]$, and a_i and b_h are breakpoints. Assume that the a_i , and also the b_h , are indexed in increasing order. Thus there are at most n^2 anchored gaps associated with each gap. Consider one anchored gap $[a_i, b_h)$ associated with gap $[a, b)$. Recall that $J_r[a, b)$ is the set of gap jobs assigned to gap $[a, b)$ in the release time partition. We shall understand $J_r[a_i, b_h)$ to be $J_r[a, b)$. (For any job assigned to $[a, b)$ whose release time is less than a_i , we are implicitly assuming that its release time is modified to be a_i for anchored gap $[a_i, b_h)$, for the purposes of computing best

r-sets. We make a similar assumption for any anchored multiple gap that starts at a_i .) A similar understanding applies for $J_d[a_i, b_h)$ and the deadline partition. For each anchored gap $[a_i, b_h)$ and set of jobs $J_r[a_i, b_h)$, the best r-set for $[a_i, b_h)$, denoted $J_r^*[a_i, b_h)$, is computed. Those jobs from $J_r[a_i, b_h)$ that are not chosen for $J_r^*[a_i, b_h)$ are placed into $J_r^-[a_i, b_h)$, the set of remaining jobs not chosen yet. Similarly, considering the jobs $J_d[a_i, b_h)$, the best d-set for $[a_i, b_h)$, denoted $J_d^*[a_i, b_h)$, is computed and the remaining jobs from $J_d[a_i, b_h)$ not chosen for $J_d^*[a_i, b_h)$ are placed into $J_d^-[a_i, b_h)$.

There is no need to compute the best r-set for each of the at most n^2 anchored gaps, since there are at most $2n$ distinct best r-sets for these anchored gaps. For a given b_h and all possible a_i , there are at most two distinct best r-sets for all of the corresponding anchored gaps $[a_i, b_h)$. This follows since $\lfloor b_h - a_1 \rfloor \leq \lfloor b_h - a_k \rfloor + 1$. If there are two best r-sets, then there is some a_l such that all $[a_i, b_h)$, $i \leq l$, have the same best r-set, and all $[a_i, b_h)$, $i > l$, have the same best r-set. The two best r-sets can be found by computing best r-sets for $[a_1, b_h)$ and $[a_k, b_h)$ using the discrete earliest deadline rule. Computing at most $2n$ best r-sets instead of n^2 best r-sets reduces the number of processors needed for this activity by a factor of n .

We show how to compute best r-sets for anchored gaps contained in gap $[a, b)$. First compute $J_r^*[a_1, b_h)$ for all valid indices h . Apply the parallel version of the discrete earliest deadline rule [AGK, R] for the jobs in $J_r[a_1, b_h)$ and the interval $[a_1, b_h)$. Set $J_r^-[a_1, b_h)$ to $J_r[a_1, b_h) - J_r^*[a_1, b_h)$. In the same manner, compute $J_r^*[a_k, b_h)$ and $J_r^-[a_k, b_h)$ for all valid indices h , where k is the largest index for the a_i . The set $J_r^*[a_i, b_h)$ is set to $J_r^*[a_1, b_h)$ if $\lfloor b_h - a_i \rfloor = \lfloor b_h - a_1 \rfloor$; otherwise it is set to $J_r^*[a_k, b_h)$. In the first case, $J_r^-[a_i, b_h) = J_r^-[a_1, b_h)$ and in the second case $J_r^-[a_i, b_h) = J_r^-[a_k, b_h)$. These additional best r-sets and remaining sets do not need to be computed as they are just duplicates of other sets.

In a similar manner, the best d-sets for anchored gaps contained in gap $[a, b)$ are computed. For a given a_i , there are at most two best d-sets. The release times of each job $j \in J_d[a_i, b_h)$ with $r_j < a_i$ is reset to $a_i - (d_j - a_i)$, the mirror image of this problem is formed, and then solved by the discrete earliest deadline first rule. The

maximum number of jobs that can be scheduled in $[a_i, b_h)$ is stored as $C[a_i, b_h)$. For anchored gap $[a_i, b_h)$, this value is $\lfloor b_h - a_i \rfloor$.

Lemma 4.3 *For all anchored gaps, determining best r-sets and best d-sets whenever they exist uses $O(\log n)$ time and $O(n^2)$ processors.*

Proof: Correctness of the above procedure follows from Lemma 3.1, Theorem 3.3, and the above discussion. To analyze the time and processor complexity, let the l -th gap have n_l jobs associated with it. A best r-set or best d-set is computed for at most $2n$ anchored gaps. The parallel version of the discrete earliest deadline rule [AGK, R] uses $O(\log n)$ time and $O(n)$ processors to schedule n jobs. Thus computing one best r-set or one best d-set in parallel takes $O(\log n_l)$ time and $O(n_l)$ processors, so the total number of processors needed to compute the $2n$ best r-sets is $O(n * n_l)$. Since $n = n_1 + n_2 + \dots + n_g$, the total number of processors is $O(n^2)$. $\square$.

We next discuss computing best r-sets and best d-sets for anchored multiple gaps. Let $[a, b')$ be a multiple gap composed of the two consecutive multiple gaps $[a, b)$ and $[a', b')$. In general multiple gaps overlap their two surrounding cover intervals, so that the cover interval between $[a, b)$ and $[a', b')$ is not $[b, a')$. For convenience we shall abuse our notation slightly and refer to this cover interval as $[b, a')$. Assume that best r-sets and best d-sets have already been computed for all anchored multiple gaps associated with $[a, b)$ and $[a', b')$. Let $[a_i, b_h)$ be an anchored multiple gap for $[a, b)$, and let $[a'_g, b'_f)$ be an anchored multiple gap for $[a', b')$, where $a'_g - b_h$ equals the number of cover jobs contained in the cover interval $[b, a')$, and such that $J_r^*[a_i, b_h)$ and $J_r^*[a'_g, b'_f)$ exist. Among r-sets that have a schedule in which a cover job starts at b_h , if one exists, let $J_r^{*h}[a_i, b'_f)$ be the best such r-set for $[a_i, b'_f)$. Define $J_d^{*h}[a_i, b'_f)$ similarly. Let $J_r^{-h}[a_i, b'_f) = J_r[a_i, b'_f) - J_r^{*h}[a_i, b'_f)$, and similarly for $J_d^{-h}[a_i, b'_f)$. For every pair $[a_i, b_h)$ and $[a'_g, b'_f)$ such that $a'_g - b_h$ equals the number of cover jobs contained in $[b, a')$, sets $J_r^{*h}[a_i, b'_f)$ and $J_d^{*h}[a_i, b'_f)$ are computed if they exist. Then, for each anchored multiple gap $[a_i, b'_f)$ of gap $[a, b')$, the various values of b_h are examined, and from

among the corresponding $J_r^{*h}[a_i, b_f]$, if there are any, the best r-set is identified as $J_r^*[a_i, b_f]$. Then $J_r^-[a_i, b_f]$ is chosen to be the corresponding $J_r^{-h}[a_i, b_f]$. Similarly, from among the corresponding $J_d^{*h}[a_i, b_f]$, if there are any, the best d-set is identified as $J_d^*[a_i, b_f]$. The value $C[a_i, b_f]$ is set to $C[a_i, b_h] + (a'_g - b_h) + C[a'_g, b'_f]$.

We next discuss how to compute $J_r^{*h}[a_i, b_f]$ if it exists. Note that $J_r^*[a_i, b_h] \subseteq J_r^{*h}[a_i, b_f]$, since the jobs in $J_r^*[a_i, b_h]$ can be scheduled in the $[a_i, b_h]$ portion of $[a_i, b_f]$, and none of the jobs from $J_r^*[a'_g, b'_f]$ or $J_r^-[a'_g, b'_f]$ can be considered in $[a_i, b_h]$. Since the set of cover jobs of a cover interval is unaffected by the exact positioning of the gaps on either side of it, we let $J^*[b, a']$ be the set of cover jobs contained in $[b, a']$ and let $J_r^-[b, a']$ be the set of anomalous gap jobs whose release times lie in $[b, a']$. Clearly, $J^*[b, a'] \subseteq J_r^{*h}[a_i, b_f]$. We now focus on computing the jobs for the $[a'_g, b'_f]$ portion of $J_r^{*h}[a_i, b_f]$. Let $J' = J_r^*[a'_g, b'_f] \cup J_r^-[a_i, b_h] \cup J_r^-[b, a']$. Let J'' be the $\min\{|J'|, C[a'_g, b'_f]\}$ jobs with smallest deadlines from set J' . Then the deadlines of the two sets $J_d^*[a'_g, b'_f]$ and J'' are compared, truncating, for the comparison only, the larger of the two sets. If $J_d^*[a'_g, b'_f] \leq_d J''$, then the jobs in J'' can be scheduled within $[a'_g, b'_f]$ and J'' is a best r-set for $[a'_g, b'_f]$. Otherwise, there is no schedule using anchored multiple gap $[a_i, b_f]$ and starting a cover job at b_h . Thus, if $J_d^*[a'_g, b'_f] \leq_d J''$, then $J_r^{*h}[a_i, b_f] = J_r^*[a_i, b_h] \cup J^*[b, a'] \cup J''$, and $J_r^{-h}[a_i, b_f] = J_r^-[a'_g, b'_f] \cup (J' - J'')$.

We next discuss how to compute $J_d^{*h}[a_i, b_f]$ if it exists. Clearly, $J_d^*[a'_g, b'_f] \subseteq J_d^{*h}[a_i, b_f]$, and $J^*[b, a'] \subseteq J_d^{*h}[a_i, b_f]$. We now focus on computing the jobs for the $[a_i, b_h]$ portion of $J_d^{*h}[a_i, b_f]$. Let $J' = J_d^*[a_i, b_h] \cup J_d^-[a'_g, b'_f] \cup J_d^-[b, a']$. A set $\hat{J}$ with the following modified release times is generated as follows. For each job $j \in J'$, let $\hat{r}_j$ be the modified release time and let r_j be the original release time. If r_j is less than a_i then $\hat{r}_j$ is set to $a_i - (d_j - a_i)$, otherwise $\hat{r}_j$ is set to r_j . Then $\hat{J}$ is sorted by release times. The $\min\{|J'|, C[a_i, b_h]\}$ jobs in $\hat{J}$ with largest release times are identified, with J'' being the set of corresponding jobs in J' . If $J_r^{*h}[a_i, b_f]$ exists, then $J_d^{*h}[a_i, b_f] = J'' \cup J_d^*[a'_g, b'_f] \cup J^*[b, a']$, and $J_d^{-h}[a_i, b_f] = J_d^-[a_i, b_h] \cup (J' - J'')$. Otherwise, there is no set $J_d^{*h}[a_i, b_f]$.

This step is complete when it has been determined if there is an r-set for the root

of the tree. If there is an r -set for $[r_{\min}, d_{\max})$, then a schedule of the jobs exists.

Lemma 4.4 *For all anchored multiple gaps, computing best r -sets and best d -sets whenever they exist uses $O((\log n)^2)$ time and $O(n^4/\log n)$ processors.*

Proof: As the crucial step in computing $J_r^{*h}[a_i, b_f]$, consider the computation of J'' . Since none of the jobs in J' can be scheduled in $[a_i, b_h)$, those jobs with release times less than a'_g can have their release times reset to a'_g . By Theorem 3.2, if J' is $[a'_g, b'_f]$ -feasible, then there is a best r -set for $[a'_g, b'_f)$ that will contain no jobs from $J_r^-[a'_g, b'_f]$. It follows that the computation of $J_r^{*h}[a_i, b_f]$ is correct. A similar argument using Lemma 3.3 establishes the correctness of the computation of $J_d^{*h}[a_i, b_f]$.

We next consider time and processor complexity. Let $[a, b)$ and $[a', b')$ be the multiple gaps associated with two sibling nodes, and let there be n_i gap jobs whose release time lies within $[a, b)$. There are at most n^2 anchored multiple gaps $[a_i, b_h)$ for $[a, b)$. Each anchored multiple gap $[a_i, b_h)$ must be matched against an anchored multiple gap $[a'_g, b'_f)$ for $[a', b')$, where $a'_g - b_h$ equals the number of cover jobs that lie within the cover interval that lies between these anchored multiple gaps. Thus for any anchored multiple gap $[a_i, b_h)$, there are at most n anchored multiple gaps $[a'_g, b'_f)$ with which it must be checked. Thus at most n^3 pairs of anchored multiple gaps $[a_i, b_h)$ and $[a'_g, b'_f)$ must be checked. Each pair of anchored multiple gaps $[a_i, b_h)$ and $[a'_g, b'_f)$ can be checked in $O(\log n)$ time using $n_i/\log n$ processors. Since the sum of n_i for all gaps at one level of the tree is $O(n)$, the total number of processors needed is $n^4/\log n$. Since there are at most $\log n$ levels in the tree, the total time for this activity is $O((\log n)^2)$. $\square$.

We next discuss the third step in our algorithm. With respect to the balanced binary tree structure imposed on the previous step, this step selects one anchored multiple gap or anchored gap for each node in the tree, along with a corresponding set of jobs, using a top-down pass through the tree. We call these anchored multiple gaps and anchored gaps *selected multiple gaps* and *selected gaps*, resp. We call the

set $J^*[a_i, b_h)$ that corresponds to a selected multiple gap or selected gap $[a_i, b_h)$ to be the *selected set* for $[a_i, b_h)$. The final result is a partition of the gap jobs into selected sets for anchored gaps, from which a schedule of the gap jobs within selected gaps is obtained.

This step begins by noting that $[r_{min}, d_{max})$ is a selected multiple gap, and the best r -set for $[r_{min}, d_{max})$ is a selected set for $[r_{min}, d_{max})$. Let $[a, b')$ be a multiple gap, and $[a, b)$ and $[a', b')$ its two constituent gaps or multiple gaps, based on the structure of the binary tree. Let $[a_i, b_f)$ be the selected multiple gap for $[a, b')$. Let $[a_i, b_h)$ and $[a'_g, b'_f)$ be anchored multiple gaps for the multiple gaps $[a, b)$ and $[a', b')$, resp. This step considers the at most n pairs such that $a'_g - b_h$ is equal to the number of cover jobs that are contained in the cover interval that lies between $[a_i, b_h)$ and $[a'_g, b'_f)$. Each pair $[a_i, b_h)$ and $[a'_g, b'_f)$ is examined to determine if there is a partition of $J^*[a_i, b'_f)$ into two sets such that the first can be scheduled in $[a_i, b_h)$ and the second can be scheduled in $[a'_g, b'_f)$. A pair is then chosen for which there is such a partition, with the first set being $J^*[a_i, b_h)$, and the second set being $J^*[a'_g, b'_f)$.

We next discuss how to test a given pair $[a_i, b_h)$ and $[a'_g, b'_f)$ to see if there is the desired partition of $J^*[a_i, b'_f)$. First, all gap jobs in $J^*[a_i, b'_f)$ that could be scheduled in $[a_i, b_h)$ are identified. Let J' be the set of all jobs in $J^*[a_i, b'_f)$ whose release times are less than a_i . Let $J'' = (J^*[a_i, b'_f) \cap J_r^*[a_i, b_h)) \cup J'$. Let J''' be the $\min\{|J''|, C[a_i, b_h)\}$ jobs of J'' with smallest deadlines. Then the deadlines of the two sets $J_d^*[a_i, b_h)$ and J''' are compared, truncating, for the comparison only, the larger of the two sets. If $J_d^*[a_i, b_h) \leq_d J'''$, then the jobs in J''' can be scheduled within $[a_i, b_h)$. Let $J'''' = J^*[a_i, b'_f) - J^*[a_i, b_h) - J'''$ be the set of jobs that must be scheduled in $[a'_g, b'_f)$. Then the deadlines of the two sets $J_d^*[a'_g, b'_f)$ and J'''' are compared, truncating, for the comparison only, the larger of the two sets. If $J_d^*[a'_g, b'_f) \leq_d J''''$, then the jobs in J'''' can be scheduled within $[a'_g, b'_f)$. If both tests succeed then $[a_i, b_h)$ and $[a'_g, b'_f)$ can be selected, and $J^*[a_i, b_h)$ and $J^*[a'_g, b'_f)$ would be J''' and J'''' , resp.

Lemma 4.5 *Given best r -sets and best d -sets for all anchored gaps and anchored*

multiple gaps for which these sets exist, identifying selected gaps and selected multiple gaps and the corresponding selected sets uses $O((\log n)^2)$ time and $O(n^2/\log n)$ processors.

Proof: We first discuss correctness. Since $J^*[a_i, b'_f)$ exists, there must be some choice of b_h such that there is a partition of $J^*[a_i, b'_f) - J^*[b, a')$ into two sets that can be scheduled in $[a_i, b_h)$ and $[a'_g, b'_f)$, resp. Suppose we are examining such a choice b_h . We verify that we make this partition correctly. It does not alter the results to assume that any job in J' has modified release time a_i . By Theorem 3.2, there is a best r -set for $[a_i, b_h)$ with respect to J (with modified release times) that contains only jobs in $J' \cup J_r^*[a_i, b_h)$. By the approach that we employ in the algorithm, a job in $J_r^*[a_i, b_h)$ will be found in $J^*[a_i, b'_f)$ unless there is a corresponding job with release time before a that has replaced it. Thus $|J''| \geq |J_r^*[a_i, b_h)|$. Thus given $J^*[a_i, b'_f)$, our algorithm makes a best choice of jobs to be scheduled in $[a_i, b_h)$.

We next analyze the time and processors used to determine $J^*[a_i, b_h)$ and $J^*[a'_g, b'_f)$. At most n pairs $[a_i, b_h)$ and $[a'_g, b'_f)$ are checked, and these can all be checked in $O(\log n)$ time using $O(n |J^*[a_i, b'_f)| / \log n)$ processors. Thus $O(n^2/\log n)$ processors are used for one level in the tree. The time for each level is $O(\log n)$ and there are at most $\log n$ levels in the tree, so the total time is $O((\log n)^2)$. $\square$.

We next discuss scheduling the gap jobs in gaps. For each gap, there is a selected gap and a corresponding selected set. Unfortunately, the jobs in the selected set are in sorted order by deadlines, not in scheduled order. The schedule of jobs in the selected set is computed using a modified earliest deadline algorithm.

Let $[a_i, b_h)$ be the selected gap of gap $[a, b)$, with corresponding selected set $J^*[a_i, b_h)$. The location of the first hole in $[a_i, b_h)$ is determined, and $J^*[a_i, b_h)$ is partitioned into two sets of jobs, those jobs that will be scheduled before the first hole in the gap, and those jobs that will be scheduled after the first hole. If $|J^*[a_i, b_h)| = \lfloor b_h - a_i \rfloor$, then there is only one hole, otherwise there can be several holes. To compute the position of the first hole, the smallest c is found such that there is no job available

to be scheduled at position $a_i + c$. Thus the starting position of the first hole is at $a_i + c$. The c jobs with release times at most $a_i + c - 1$ are scheduled, using the discrete earliest deadline algorithm [AGK, R]. Then the remaining $\lfloor b_h - a_i \rfloor - c$ jobs in $[b_h - (\lfloor b_h - a_i \rfloor - c), b_h)$ are scheduled, again using the discrete earliest deadline algorithm.

Lemma 4.6 *Computing the schedule of the jobs in selected sets within their selected gaps uses $O(\log n)$ time and $O(n)$ processors.*

Proof: First we discuss correctness. By definition of selected set, a schedule exists. There are c jobs with release time at most $a_i + c - 1$. The discrete earliest deadline algorithm schedules these jobs in $[a_i, a_i + c)$. The remaining jobs must be scheduled in $[a_i + c, b_h)$. Since there is a schedule of these jobs within this interval, we may reset any deadline greater than b_h to be b_h . If any constrained interval is created, it must end at b_h . Consider such a constrained interval $[a'', b_h)$ that contains no other such constrained interval. Only the jobs contained in this interval can be scheduled so as to overlap with interval $[b_h - \lfloor b_h - a'' \rfloor, b_h)$. By Lemma 2.1, these jobs can be scheduled in $[b_h - \lfloor b_h - a'' \rfloor, b_h)$. The same argument may be applied inductively to any remaining jobs and interval $[a_i + c, b_h - \lfloor b_h - a'' \rfloor)$. All such jobs scheduled start at a time that differs from b_h by an integer. When no constrained intervals remain, the remaining jobs can be scheduled by the discrete earliest deadline rule to also satisfy this requirement. But then there exists a schedule of remaining jobs in which all jobs start at a time that differs from b_h by an integer. The discrete earliest deadline rule will find such a schedule.

Next we analyze the time and processors used. The first hole in the l -th gap is computed by sorting the jobs in nondecreasing release time order, and performing a prefix sum on the number of release times less than or equal to unit spaced positions in the interval. This takes $O(\log n_l)$ time and $O(n_l)$ processors, where n_l is the number of jobs to be scheduled in the selected gap. Applying the parallel earliest deadline algorithm to the two pieces of the gap also takes $O(\log n_l)$ time and $O(n_l)$

processors. Computing one schedule for each gap in parallel takes $O(\log n)$ time and $O(n)$ processors. $\square$.

The fourth step in the algorithm schedules the cover jobs. The choice of selected gaps dictates the position of the cover jobs within the cover intervals. By Lemma 2.1, there is a schedule with the jobs packed tightly together. Let $[a, b)$ and $[a', b')$ be consecutive gaps, and $[a_i, b_h)$ and $[a'_g, b'_f)$ the corresponding selected gaps. Consider the cover interval $[b, a')$. Note that $a'_g - b_h = |J^*[b, a')|$. The discrete earliest deadline algorithm is applied to the set $J^*[b, a')$ in interval $[b_h, a'_g)$.

Lemma 4.7 *Scheduling the cover jobs within the cover intervals takes $O(\log n)$ time and uses $O(n)$ processors.*

Proof: The parallel earliest deadline algorithm [AGK, R] applied to the i -th cover interval that contains n_i jobs takes $O(\log n_i)$ time and uses $O(n_i)$ processors. Thus computing the schedule for all cover intervals in parallel takes $O(\log n)$ time and $O(n)$ processors. $\square$.

We summarize the performance of the algorithm below.

Theorem 4.1 *Given n unit-time jobs with arbitrary release times and deadlines, there is a CREW PRAM algorithm that determines if there is a schedule of the jobs on a single processor, and if so, it produces a schedule. The algorithm uses $O((\log n)^2)$ time and $O(n^4/(\log n))$ processors.*

Proof: By Theorem 2.1, if there is a schedule, then there is a canonical schedule. The discussion accompanying the description of the algorithm in this section, plus the correctness within Lemmas 4.2 through 4.7, establish that the algorithm computes a canonical schedule whenever one exists. The time and processor complexities follow from Lemmas 4.2 through 4.7. $\square$.

Suppose k is the number of distinct fractional parts of the release times. The algorithm uses fewer than $\Theta(n^4/(\log n))$ processors if k is $o(n)$. The number of starting

positions for anchored gaps is thus reduced to k . If there are fewer distinct fractional parts of the deadlines, then breakpoints can be based on deadlines, as discussed at the end of section 2. Let k be the minimum of the number of distinct fractional parts of release times and the number of distinct fractional parts of deadlines.

Corollary 4.1.1 *Given n unit-time jobs with arbitrary release times and deadlines, there is a CREW PRAM algorithm that determines if there is a schedule of the jobs on a single processor, and if so, it produces a schedule. If k is the minimum of the number of distinct fractional parts of release times and the number of distinct fractional parts of deadlines, then the algorithm uses $O((\log n)^2)$ time and $O(k^3n/(\log n) + n^2/(\log n)^2)$ processors.*

Proof: Finding a cover can be done in $O((\log n)^2)$ time using just $O(n^2/(\log n)^2)$ processors, by having each of the processors simulate $\log n$ processors in the algorithm stated earlier. Similarly, computing best r-sets and best d-sets for anchored gaps can be done in $O((\log n)^2)$ time using $O(kn/\log n)$ processors. Computing best r-sets and d-sets for anchored multiple gaps takes $O((\log n)^2)$ time and uses $O(k^3n/(\log n))$ processors. Determining selected gaps and selected multiple gaps uses $O(kn/\log n)$ processors and $O((\log n)^2)$ time. Scheduling the gap jobs and the cover jobs uses $O(\log n)$ time and $O(n)$ processors. $\square$.

5 Discussion

In [SW], a sequential algorithm is presented for scheduling unit-time jobs with release times and deadlines to run on m machines. Their algorithm runs in $O(mn^2)$ time. A natural question to ask is whether this problem is in NC. It is not immediately apparent how to modify our approach to give an NC algorithm for this problem.

Acknowledgement

We thank the referees for their careful reading of the manuscript and their many helpful comments.

References

- [AGK] M. J. Atallah, M. T. Goodrich, and S. R. Kosaraju. On the parallel complexity of evaluating some sequences for set manipulation operations. In *Lecture Notes in Computer Science, 319: VLSI Algorithms and Architectures, 3rd Aegean Workshop on Computing, AWOC 88*, pages 1–10. Springer-Verlag, New York, 1988.
- [B] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21:201–206, 1974.
- [C] J. Carlier. Probleme a une machine dans le cas ou les taches ont des durees egales. Technical report, Institut de Programmation, Universite Paris VI, 1979.
- [DS] E. Dekel and S. Sahni. Binary trees and parallel scheduling algorithms. *IEEE Transactions on Computers*, 32(3):307–315, March 1983.
- [DUW] D. Dolev, E. Upfal, and M. K. Warmuth. The parallel complexity of scheduling with precedence constraints. *Journal of Parallel and Distributed Computing*, 3:553–576, 1986.
- [GJ] M. R. Garey and D. S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM Journal of Computing*, 6:416–426, 1977.
- [GJST] M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal of Computing*, 10(2):256–269, May 1981.
- [HIM1] D. Helmbold and E. Mayr. Fast scheduling algorithms on parallel computers. In F. Preparata, editor, *Advances in Computing Research*. Jai Press, Inc., 1987.

- [HM2] D. Helmbold and E. Mayr. Two processor scheduling is in NC. *SIAM Journal of Computing*, 16:747–759, 1987.
- [H] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.
- [KR] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 869–942. Elsevier Science Publishers, 1990.
- [R] S. H. Rodger. An optimal parallel algorithm for preemptive job scheduling that minimizes the maximum lateness. In *Proceedings of the Twenty-sixth Annual Allerton Conference on Communication, Control, and Computing*, pages 293–302, 1988.
- [S] B. B. Simons. A fast algorithm for single processor scheduling. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, pages 246–252, 1978.
- [SW] B. B. Simons and M. K. Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM Journal of Computing*, 18(4):690–710, 1989.