Machine Minimization for Scheduling Jobs with Interval Constraints

Julia Chuzhoy* Sudipto Guha† Sanjeev Khanna‡ Joseph (Seffi) Naor§

Abstract

The problem of scheduling jobs with interval constraints is a well-studied classical scheduling problem. The input to the problem is a collection of n jobs where each job has a set of intervals on which it can be scheduled. The goal is to minimize the total number of machines needed to schedule all jobs subject to these interval constraints. In the continuous version, the allowed intervals associated with a job form a continuous time segment, described by a release date and a deadline. In the discrete version of the problem, the set of allowed intervals for a job is given explicitly. So far, only an $O(\frac{\log n}{\log\log n})$-approximation is known for either version of the problem, obtained by a randomized rounding of a natural linear programming relaxation of the problem. In fact, we show here that this analysis is tight for both versions of the problem by providing a matching lower bound on the integrality gap of the linear program. Moreover, even when all jobs can be scheduled on a single machine, the discrete case has recently been shown to be $\Omega(\log\log n)$-hard to approximate.

In this paper we provide improved approximation factors for the number of machines needed to schedule all jobs in the continuous version of the problem. Our main result is an $O(1)$-approximation algorithm when the optimal number of machines needed is bounded by a fixed constant. Thus, our results separate the approximability of the continuous and the discrete cases of the problem. For general instances, we strengthen the natural linear programming relaxation in a recursive manner by forbidding certain configurations which cannot arise in an integral feasible solution. This yields an $O(\texttt{OPT})$-approximation, where OPT *denotes the number of machines needed by an optimal solution. Combined with earlier results, our work implies an $O(\sqrt{\frac{\log n}{\log\log n}})$-approximation for any value of* OPT.

1 Introduction

We consider the problem of scheduling a set of jobs with specified time intervals during which they must be scheduled. Specifically, we are given a set $J = \{1, \ldots, n\}$ of *jobs*, and for each job $j \in J$, there is a set $\mathcal{I}(j)$ of time intervals on the time line, called *job intervals*. Scheduling a job j means choosing one of its associated time intervals from $\mathcal{I}(j)$. The goal is to schedule all the jobs on a minimum number of machines, such that no two jobs assigned to the same machine overlap in time. This means that the maximum number of chosen job intervals at any point of time must not exceed the number of machines.

There are two variations of the problem. In the *discrete* version, the sets of job intervals $\mathcal{I}(j)$ are given explicitly. In the *continuous version*, each job j has a release date r_j, a deadline d_j, and a processing time p_j. The time interval $[r_j, d_j]$ is called the *job window*. The set of job intervals $\mathcal{I}(j)$ is implied by these parameters, and it consists of all the time intervals of length p_j which are contained inside the window $[r_j, d_j]$. Garey and Johnson [6] (see also [7]) show that for this case, deciding whether all the jobs can be scheduled on a single machine is already NP-hard.

The currently best known approximation factor for the machine minimization problem is $O(\frac{\log n}{\log\log n})$. This factor is achieved through a linear programming relaxation of the problem and then applying Chernoff bounds to analyze the randomized rounding procedure [8]. The approximation factor achieved by the randomized rounding in fact improves as the value of the optimal solution increases. Let OPT denote the number of machines used by an optimal solution to the machine minimization problem. For example, using the standard Chernoff bounds, it is easy to show that

*Computer Science Dept., Technion, Haifa, Israel. Email: cjulia@cs.technion.ac.il

†Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19103. Email:sudipto@cis.upenn.edu

‡Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19103. Email:sanjeev@cis.upenn.edu. Supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Career Award CCR-0093117.

§Computer Science Dept., Technion, Haifa, Israel. Email: naor@cs.technion.ac.il. Research supported in part by US-Israel BSF Grant 2002276 and by EU contract IST-1999-14084 (APPOL II).

the randomized rounding procedure guarantees a solution of value $O(\texttt{OPT} \cdot \sqrt{(\log n / \log\log n)})$ machines whenever $\texttt{OPT} = \Omega(\sqrt{(\log n / \log\log n)}$. Once OPT is $\Omega(\log n)$, the approximation factor further improves to $O(1)$. Thus an interesting question is whether the approximation factor can be improved for small values of OPT. For example, does there exist a constant approximation factor in the case OPT is a constant? For the discrete version, the answer to this question is negative. Recently, Chuzhoy and Naor [4] showed that it is $\Omega(\log\log n)$-hard to approximate the machine minimization problem, even if $\texttt{OPT} = 1$.

Results and Techniques: In this paper we consider the continuous version of the machine minimization problem. We show that the "natural" linear programming formulation for the problem has an integrality gap of $\Theta(\frac{\log n}{\log\log n})$ even when $\texttt{OPT} = 1$. However, in the continuous version, the linear programming formulation can be strengthened by adding constraints that explicitly forbid certain configurations which can not arise in any integral solution. We design a rounding scheme that allows us to transform the resulting fractional schedule into an integral one by using a constant number of machines, for this case. We build on this idea to design an approximation algorithm that achieves an $O(\texttt{OPT})$-approximation for any value of OPT. Extending the idea of forbidden configurations to instances where an optimal schedule itself requires multiple machines, is technically difficult since the configurations that need to be forbidden have complex nested structure and we need to discover them by recursively solving linear programs on smaller instances. Specifically, the strengthened linear programming solution for a given time interval is computed via a dynamic program that uses a linear programming subroutine to compose recursively computed solutions on smaller time intervals. We believe this novel idea for strengthening a linear programming relaxation is of independent interest. Combined with earlier results, our results imply an $O(\sqrt{\frac{\log n}{\log\log n}})$-approximation algorithm for any value of OPT.

Our results provide an interesting separation between the approximability of discrete and continuous versions, especially as no such separation is known for the packing version of the problem, commonly referred to as the throughput maximization problem. In throughput maximization, the goal is to schedule a maximum weight subset of the jobs using only a fixed number of machines. Spieksma [9] showed that the discrete version of throughput maximization is MAX SNP-hard, but no such hardness result for the continuous version is known. However, the best approximation results currently known for both discrete and continuous versions are the same: both problems have a 2-approximation for weighted input [2, 1, 3], and $\frac{e}{e-1} \approx 1.582$ for unweighted input [5].

Organization: In Section 2, we describe the strengthened linear programming (LP) relaxation and the dynamic program approach for recursively computing it. Section 3 provides the description of our rounding procedure, which, given a fractional schedule for the strengthened LP on k machines, produces an integral schedule on $O(k^2)$ machines. Finally, in Section 4 we establish an integrality gap of $\Omega(\frac{\log n}{\log\log n})$ for the natural LP.

2 Linear Programming Formulations

In this section we define the linear programs, starting with the "natural" time-indexed LP-formulation for the problem. This formulation can be used for both discrete and continuous inputs.

Let L denote the set of all the left endpoints of the job intervals. For each job $j \in J$, for each job interval $I \in \mathcal{I}(j)$, we define a variable $x(I, j)$ indicating whether j is scheduled on interval I. Our constraints guarantee that every job is scheduled and that the number of jobs scheduled at each point of time does not exceed the number of available machines. The linear programming formulation (LP1) is as follows.

$$\min z$$

s.t.

$$\sum_{I \in \mathcal{I}(j)} x(I,j) = 1 \qquad \forall j \in J \tag{1}$$

$$\sum_{j \in J} \sum_{I \in \mathcal{I}(j): t \in I} x(I,j) \le z \qquad \forall t \in L \tag{2}$$

$$x(I,j) \ge 0 \qquad \forall j \in J, I \in \mathcal{I}(j)$$

The number of machines we need for the fractional schedule is $k = \lceil z \rceil$.

Note: In the case of continuous input, the number of variables in (LP1) is unbounded. The same problem arises in other linear programs presented in this section. In the Appendix we show how to resolve this issue, making the running time of our algorithms polynomial at the cost of losing only a constant in the approximation factor.

Unfortunately, the integrality gap of (LP1) is $\Omega(\frac{\log n}{\log\log n})$, as shown in Section 4, even for $k = 1$ and discrete input (with polynomially many intervals). Our next step is strengthening (LP1) for the case where $\texttt{OPT} = 1$ by adding valid inequalities.

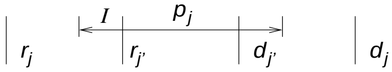

Figure 1. Interval I is a forbidden interval of job j

Consider the case where the input is continuous and $\texttt{OPT} = 1$. Let $j \in J$ be any job and let $I \in \mathcal{I}(j)$ be one of its intervals. Suppose there is some other job $j' \in J$, whose time window is completely contained in I, i.e., $[r_{j'}, d_{j'}] \subseteq I$. Since the optimal solution can use only one machine and all the jobs are scheduled, job j cannot be scheduled on interval I in the optimal solution. We call such an interval I a *forbidden interval of job j* (see Figure 1). All the other job intervals of job j are called *allowed intervals*.

In the linear program, we can either set apriori the values $x(I,j)$, where I is a forbidden interval of job j, to 0, or, alternatively, add the following set of valid inequalities:

$$x(I,j) + \sum_{I' \in \mathcal{I}(j'):\ I' \subseteq I} x(I', j') \leq 1 \qquad \forall j, j' \in J, \forall I \in \mathcal{I}(j) \tag{3}$$

Note that if I is a forbidden interval of job j, then for some job j', $\sum_{I' \in \mathcal{I}(j'):\ I' \subseteq I} x(I', j') = 1$, and thus the value of $x(I,j)$ is going to be 0 in the LP-solution.

We now turn our attention to the scenario where the optimal solution uses multiple machines. Obviously, Inequality (3) is not valid anymore. Indeed, suppose interval I is a forbidden interval of some job j, $I \in \mathcal{I}(j)$, and let j' be some job whose window is contained in I. Suppose now the optimal solution uses two machines. Then, job j can be scheduled on interval I on one machine and job j' can be scheduled inside its window on the other machine, thus (3) does not hold anymore. Now let T be any time interval containing the window of job $j' \in J$. We know that we need at least one machine to accommodate jobs whose windows are contained in T. Therefore, we can schedule at most one job on an interval that contains T. So for the case of two machines, we can add the corresponding inequality for every interval T that contains some job window. This idea extends to arbitrary number of machines.

For each time interval T (not necessarily belonging to any job), we define a function $m(T)$ which is, intuitively, a lower bound on the number of machines needed to accommodate all the jobs whose window is contained in T. We compute the value of $m(T)$ recursively by the means of dynamic programming, going from smallest to largest intervals.

For intervals T of length 0, set $m(T) = 0$. Given a time interval T, let $J(T)$ be the set of jobs whose time window is completely contained in T. The value of $m(T)$ is defined to be $\lceil z \rceil$, where z is the optimal solution to linear program LP3(T), which is defined as follows:

$$\min z$$

s.t.

$$\sum_{I \in \mathcal{I}(j)} x(I,j) = 1 \qquad \forall j \in J(T) \tag{4}$$

$$\sum_{j \in J(T)} \sum_{I \in \mathcal{I}(j): t \in I} x(I,j) \leq z \qquad \forall t \in T \tag{5}$$

$$\sum_{j \in J(T)} \sum_{I \in \mathcal{I}(j): T' \subseteq I} x(I,j) \leq z - m(T') \qquad \forall T' \subset T \tag{6}$$

$$x(I,j) \geq 0 \qquad \forall j \in J(T), I \in \mathcal{I}(j)$$

The first two sets of constraints are similar to those of (LP1), except they are applied to the time interval T and subset $J(T)$ of jobs. The third set of constraints models Constraint (3) for the case of multiple machines. Suppose we are given some interval $T' \subset T$. As T' is smaller than T, we know the value of $m(T')$ from the dynamic programming table, and this value is a lower bound on the number of machines needed to accommodate jobs whose windows are contained in T'. Therefore, we have at most $z - m(T')$ machines available for scheduling jobs on intervals that contain T'. The third set of constraints ensures that the total number of jobs scheduled on intervals which contain T' does not exceed $z - m(T')$.

Note that Constraints (5) can be omitted, since they are a special case of Constraints (6), for intervals T' of length 0.

3 The Rounding Procedure

In this section we show that, given a fractional solution to (LP3(T)) which uses $k = m(T)$ machines, we can find an integral solution using at most $O(k^2)$ machines. The rounding will proceed iteratively: at each step we will identify a subset of jobs that can be scheduled on $O(k)$ machines, such that the remaining jobs will have a feasible fractional solution on at most $k - 1$ machines. Thus, all jobs will get scheduled on $O(k^2)$ machines.

Suppose we are given a solution to the linear program (LP3(T)) for some time interval T and let $\mathcal{T}$ be some collection of disjoint sub-intervals of T, such that for each $T' \in \mathcal{T}$, $m(T') < m(T)$. We partition the set $J(T)$ of jobs into two subsets, J' and J'', where $j \in J''$ iff its window is completely contained in one of the intervals of $\mathcal{T}$ and $j \in J'$ otherwise. We say that $\mathcal{T}$ is *good* with respect to the LP-solution if for each job $j \in J'$, and each interval $I \in \mathcal{I}(j)$ where $x(I, j) > 0$, I overlaps with at most two intervals belonging to $\mathcal{T}$.

We will show that if the optimal solution cost of (LP3(T)) is z and $\mathcal{T}$ is good with respect to the solution, then we can schedule the jobs J' on at most $O(k)$ machines where $k = \lceil z \rceil$. Before we formalize this argument, let us define PROCEDURE PARTITION, which receives as input an interval T, a set $J(T)$ of jobs, a solution to (LP3(T)), and it produces a collection $\mathcal{T}$ of sub-intervals of T which is good with respect to the LP-solution.

> PROCEDURE PARTITION
>
> **Input:** Time interval T, set of jobs $J(T)$ whose windows are contained in T, and a solution to (LP3(T)).
> Start with $\mathcal{T} = \emptyset$ and set t to be the left endpoint of T. While there are jobs $j \in J(T)$, such that the right endpoint of some interval $I \in \mathcal{I}(j)$ lies to the right of t and $x(I, j) > 0$, do:
>
> 1. If no job j exists such that one of its intervals $I \in \mathcal{I}(j)$ contains t and $x(I, j) > 0$, then move t to the right till the above condition holds.
> 2. Among all the job intervals I that contain time point t, such that for some $j \in J$, $I \in \mathcal{I}(j)$ and $x(I, j) > 0$, choose the interval with rightmost right endpoint and denote this endpoint by t'.
> 3. Add time interval $[t, t']$ to $\mathcal{T}$ and set $t := t'$.
>
> Let $J'' \subset J(T)$ denote the set of jobs whose windows are contained in one of the intervals of $\mathcal{T}$, and let $J' = J(T) \setminus J''$.

Claim 1 *Given time interval T, let $\mathcal{T}$ be the collection of sub-intervals of T produced by* PROCEDURE PARTITION. *Then the intervals in $\mathcal{T}$ are disjoint, and for each $T' \in \mathcal{T}$, $m(T') < m(T)$.*

Proof: The disjointness of the intervals follows easily by the definition of PROCEDURE PARTITION. Suppose now that for some $T' \in \mathcal{T}$, $m(T') = m(T)$. Then there is some job $j \in J(T)$ and its interval $I \in \mathcal{I}(j)$ which defined the interval T', so I contains T' and $x(I, j) > 0$. But this would violate Constraint (6) of (LP3(T)). □

Claim 2 *Let $\mathcal{T}$ be the output of* PROCEDURE PARTITION *applied to interval T and subset $J(T)$ of jobs. Then, $\mathcal{T}$ is good with respect to the LP-solution.*

Proof: Assume for contradiction that there is some job interval $I \in \mathcal{I}(j)$, $j \in J'$, which overlaps with three intervals $T_1, T_2, T_3 \in \mathcal{T}$, and $x(I, j) > 0$. Assume w.l.o.g. that T_2 lies to the right of T_1 and T_3 lies to the right of T_2. Interval I contains the left endpoint of T_2, but its right endpoint lies strictly to the right to that of T_2. This is impossible, because then T_2 would have been defined differently by PROCEDURE PARTITION. □

Theorem 1 *Suppose we are given a feasible solution to (LP3(T)) on $k = m(T)$ machines, a collection $\mathcal{T}$ of disjoint sub-intervals of T, and a corresponding subset $J' \subset J(T)$ of jobs, and assume that $\mathcal{T}$ is good with respect to the LP-solution. Then we can schedule all the jobs in J' on αk machines in polynomial time, for some constant α.*

Before proving the above theorem, we show that it leads to an $O(\texttt{OPT})$-approximation.

Theorem 2 *Suppose we are given a solution to (LP3(T)) on job set $J(T)$ on k machines. Then we can schedule all the jobs on $\alpha k^2 = \alpha m^2(T)$ machines for some constant α.*

Proof: The proof is by induction on $m(T)$. The base case is where $m(T) = 1$. We perform PROCEDURE PARTITION on the input and obtain subsets J', J'' of jobs and a collection of time intervals $\mathcal{T}$. We claim that $J'' = \emptyset$ and thus $J' = J$. Assume otherwise. Then there is some job $j \in J''$, such that for some interval $T' \in \mathcal{T}$, the window of j is contained in T'. Then $m(T') = 1$. But there must be some interval I of some job j', $I \in \mathcal{I}(j')$, which defined T', and such that $x(I, j') > 0$. But then Constraint (6) of (LP3(T)) does not hold for T'. Since $J = J'$, and $\mathcal{T}$ is good with respect to the LP solution, by Theorem 1, we can schedule all the jobs on α machines.

Now consider the case where $m(T) > 1$. After performing PROCEDURE PARTITION, we obtain subsets J', J'' of jobs and collection $\mathcal{T}$ of intervals. As $\mathcal{T}$ is good with respect to the LP solution, by Theorem 1, we can schedule all the jobs in J' on $\alpha m(T)$ machines. We schedule the jobs in J'' separately, as follows. For each interval $T' \in \mathcal{T}$, we solve the problem recursively for T' and the jobs in J'' whose windows are contained in T' (recall that $m(T') < m(T)$)). By the inductive hypothesis, for each such sub-problem, we obtain a schedule of all the jobs whose window is contained in T' on at most $\alpha(m(T) - 1)^2$ machines. As the intervals in $\mathcal{T}$ are disjoint, we can schedule all the jobs in J'' on $\alpha(m(T) - 1)^2$ machines. In total we use $\alpha m(T) + \alpha(m(T) - 1)^2 \leq \alpha(m(T))^2$ machines. □

Corollary 1 *There is an $O(\sqrt{\log n / \log\log n})$-approximation algorithm for machine scheduling.*

Proof: Let z be the value of the optimal fractional solution. If $\lceil z \rceil = k \leq \sqrt{\log n / \log\log n}$, then following Theorem 2, we have an $O(k)$-approximation. If $k \geq \sqrt{\log n / \log\log n}$, then a randomized rounding of the fractional solution can be applied. The expected number of machines used by the randomized rounding procedure is $O(k \cdot \sqrt{\log n / \log\log n})$, yielding the desired result. $\square$

3.1 Proof of Theorem 1

Suppose we have a feasible solution to (LP3(T)) that uses $k = \lceil z \rceil$ machines, a collection $\mathcal{T}$ of disjoint sub-intervals of T, a subset $J' \subset J(T)$ of jobs, and assume also that $\mathcal{T}$ is good with respect to the LP-solution. We show how to schedule all the jobs in J' on αk machines, for some constant α. Recall that J' consists only of such jobs whose window is not contained in any one of the intervals $T' \in \mathcal{T}$.

We divide all the jobs into a number of types and schedule each type separately on $O(k)$ machines. We first describe the job types and give some intuition as to how the jobs corresponding to these types are scheduled. We provide a formal description of the algorithms later.

Given a time interval I and a time point t, we say that I *crosses* t, if the left endpoint of I lies strictly to the left of t and the right endpoint of I lies strictly to the right of t. Recall that $k = m(T)$.

Definition: The job types are defined as follows:

Type 1: Denote by $\mathcal{I}^C$ the intervals which cross the boundaries of the intervals in $\mathcal{T}$. A job j is defined as a job of type 1 if

$$\sum_{I \in \mathcal{I}(j) \bigcap \mathcal{I}^C} x(I, j) \geq 0.2.$$

The idea of scheduling jobs of this type on $O(k)$ machines is to find a matching between the jobs and the boundaries of the intervals in $\mathcal{T}$. The LP-solution gives a fractional matching where for each job of type 1 at least a fraction 0.2 of the job is scheduled. Therefore, the integral matching gives a schedule of jobs of type 1, where on each boundary of an interval in $\mathcal{T}$, at most $5k$ jobs are scheduled. As the intervals of the jobs that have non-zero values in the LP-solution overlap with at most two intervals in $\mathcal{T}$, the maximum number of jobs running at any time t is at most $10k$.

Type 2: An interval I belonging to job j is called *large*, if it is completely contained in an interval $T' \in \mathcal{T}$ whose size is at most $2p_j$. Let $\mathcal{I}^L$ denote the set of the large intervals. Jobs of type 2 are all the jobs j which are not of type 1, and for which the following property holds:

$$\sum_{I \in \mathcal{I}(j) \bigcap \mathcal{I}^L} x(I, j) \geq 0.2.$$

In order to schedule jobs of type 2, observe that in the LP-solution, for each interval $T \in \mathcal{T}$, the sum of $x(I, j)$ where I is a large sub-interval of T and $j \in J'$, is at most $2k$. We perform a matching between jobs of type 2 and the intervals in T to determine the schedule of these jobs on $10k$ machines.

Type 3: For each job j, let $T^d(j)$ denote the interval in $\mathcal{T}$ that contains its deadline d_j. Job j is of type 3, if it does not belong to any of the previous types, and

$$\sum_{I \in \mathcal{I}(j), I \subseteq T^d(j)} x(I, j) \geq 0.2.$$

Each job j of type 3 is going to be scheduled inside the interval $T^d(j)$. Consider some interval $T' \in \mathcal{T}$ and the subset of type 3 jobs j whose deadline belongs to T'. As the release dates of these jobs are outside T', this can be viewed as scheduling jobs with identical release dates. We solve this problem approximately and use the LP-solution to bound the number of machines we use.

Type 4: For each job j, let $T^r(j)$ denote the interval in $\mathcal{T}$ that contains its release time r_j. Job j is of type 4, if it does not belong to any of the previous types, and

$$\sum_{I \in \mathcal{I}(j), I \subseteq T^r(j)} x(I, j) \geq 0.2.$$

The scheduling is performed similarly to the scheduling of the jobs belonging to type 3.

Type 5: This type contains all the other jobs. Note that for each job j of this type, the sum of fractions $x(I, j)$ for intervals I, such that I is not large, does not cross any boundary of intervals in $\mathcal{T}$, and the interval $T' \in \mathcal{T}$ which contains I does not contain the release date or the deadline of j, is at least 0.2. The LP-solution ensures that all the jobs of this type can be (fractionally) scheduled inside intervals $T \in \mathcal{T}$ (i.e., without crossing their boundaries), even if we shrink the job windows so that their release dates and deadlines coincide with boundaries of intervals in $\mathcal{T}$, using $5k$ machines. This allows us to schedule all the jobs of type 5 on $O(k)$ machines.

3.1.1 The Schedule

In the final schedule, jobs of each type are scheduled separately. We now provide a formal description of the schedules of each type, and prove that $O(k)$ machines suffice for scheduling each type.

Type 1: Recall that job j belongs to type 1 iff

$$\sum_{I \in \mathcal{I}(j) \bigcap \mathcal{I}^C} x(I,j) \geq 0.2$$

where $\mathcal{I}^C$ is the set of intervals crossing an endpoint of an interval in $\mathcal{T}$. We claim that we can schedule all the jobs of type 1 on at most $10k$ machines.

Construct a directed bipartite graph $G = (V, U, E)$, where V is the set of jobs of type 1 and U is the set of boundaries of the intervals of $\mathcal{T}$. There is an edge (j, b) of capacity 1 from $j \in V$ to $b \in U$ if and only if there is an interval $I \in \mathcal{I}(j)$ that crosses the boundary b and $x(I,j) > 0$. Add a source vertex s and an edge (s, j) of capacity 1 for each $j \in V$. Add a sink vertex t and an edge (b, t) of capacity $5k$ for each $b \in U$.

The solution to the linear program defines a feasible flow in this graph of value at least $|V|$, as follows. For each $j \in V$ and $b \in U$, let $\mathcal{I}(j, b)$ be the subset of intervals $\mathcal{I}(j) \bigcap \mathcal{I}^C$ that cross a boundary b. Set the flow value on the edge (j, b) to be:

$$\frac{\sum_{I \in \mathcal{I}(j,b)} x(I,j)}{\sum_{I \in \mathcal{I}(j) \bigcap \mathcal{I}^C} x(I,j)}.$$

Note that by the definition of type 1, the value of the flow on edge (j, b) is at most

$$5 \cdot \sum_{I \in I(j,b)} x(I,j),$$

and the total amount of flow leaving j is exactly 1. Since the total fraction of intervals scheduled at each time point is at most k, and the flow on edge (j, b) is at most

$$5 \cdot \sum_{I \in I(j,b)} x(I,j)$$

the total flow entering each $b \in U$ is at most $5k$.

Therefore, there is an integral flow of value $|V|$. This flow defines a schedule of jobs of type 1 as follows: for each such job j, there is a unique boundary b, such that the flow on edge (j, b) is 1. By the construction of the network, there is an interval $I \in \mathcal{I}(j, b) \subseteq \mathcal{I}(j)$, such that $x(I, j) > 0$. We say that j is scheduled on boundary b.

The number of machines used in such a schedule is at most $10k$. Since each job interval $I \in \mathcal{I}(j)$ with $x(I, j) > 0$ overlaps with at most two intervals in $\mathcal{T}$, at any point t inside some interval $T' \in \mathcal{T}$, the jobs that run at time t are either scheduled on the left or on the right boundary of T'. Therefore, the number number of jobs scheduled at any such point is at most $10k$.

Type 2: Recall that type 2 jobs are all the jobs j which are not of type 1, and for which the following property holds:

$$\sum_{I \in \mathcal{I}(j) \bigcap \mathcal{I}^L} x(I,j) \geq 0.2.$$

Note that in the fractional solution, for each interval $T' \in \mathcal{T}$, the sum of $x(I, j)$ where I is a large sub-interval T' is at most $2k$.

We show how to schedule all the jobs of type 2 on $10k$ machines. This is done in a similar fashion to type 1. We build a directed bipartite graph $G = (V, \mathcal{T}, E)$, where V is the set of all the jobs of type 2. There is an edge (j, T') of capacity 1 if and only if job j has a large interval in $T' \in \mathcal{T}$. Add a source vertex s and an edge (s, j) of capacity 1 for each $j \in V$. Add a sink vertex t and an edge (T', t) of capacity $10k$ for each $T' \in \mathcal{T}$. The value of the maximum flow in this network is exactly $|V|$. A fractional flow of this value is obtained as follows: for each $j \in V, T' \in \mathcal{T}$, set the flow on edge (j, T') to be

$$\frac{\sum_{I \in \mathcal{I}(j) \bigcap \mathcal{I}^L, I \subseteq T'} x(I,j)}{\sum_{I \in \mathcal{I}(j) \bigcap \mathcal{I}^L} x(I,j)}.$$

Note that by the definition of type 2, this value is at most

$$5 \cdot \sum_{I \in \mathcal{I}(j) \bigcap \mathcal{I}^L, I \subseteq T'} x(I,j)$$

and the total flow leaving j is 1. Since in the solution to the linear program, for each interval $T' \in \mathcal{T}$, the total fraction of all the large intervals inside T' is at most $2k$, the value of flow entering T' is at most $10k$.

Therefore, there is an integral flow of value $|V|$. For each job j of type 2, there is exactly one $T' \in \mathcal{T}$ such that there is a flow on edge (j, T'). We schedule j in this interval. In each interval T', we thus schedule at most $10k$ jobs (note that each such job has at least one interval in T'). We schedule these $10k$ jobs on $10k$ different machines. Observe that the intervals belonging to $T', T'' \in \mathcal{T}$ are disjoint, and thus we need $10k$ machines in all.

Type 3: Recall that a job j is of type 3, if it does not belong to any of the previous types, and

$$\sum_{I \in \mathcal{I}(j), I \subseteq \mathcal{T}^d(j)} x(I, j) \geq 0.2$$

where $\mathcal{T}^d(j)$ denotes the interval in $\mathcal{T}$ that contains d_j.

Consider some $T' \in \mathcal{T}$ that starts at time $S_{T'}$ and finishes at time $F_{T'}$. Let $J^d(T')$ be the set of all jobs of type 3 whose deadline is inside T'. We show how to schedule all these jobs on $10k$ machines. Once again since the intervals $T' \in \mathcal{T}$ are disjoint, we can use the same $10k$ machines for all jobs of type 3.

For each job j, define $t_j = d_j - p_j$. We build a new fractional schedule x' of jobs in $J^d(T')$ in T'. For each job $j \in J^d(T')$, for each $I \in \mathcal{I}(j), I \subseteq T'$, define

$$x'(I, j) = \frac{x(I, j)}{\sum_{I' \in \mathcal{I}(j), I' \subseteq T'} x(I', j)}.$$

Note that by the definition of type 3 jobs, $x'(I, j) \leq 5x(I, j)$. This new fractional solution has the following properties:

- The sum of fractions of intervals that cross any time point t is at most $5k$.
- Each $j \in J^d(T')$ is scheduled completely at subintervals of T' that start before or at t_j.

We now show a greedy schedule of job set $J^d(T')$ on $10k$ machines, in interval T'. We proceed from left to right on all the $10k$ machines simultaneously, using the following greedy rule: whenever any machine becomes idle, schedule any available job $j \in J^d(T')$ that has minimal t_j.

We claim that all the jobs in $J^d(T')$ are scheduled in the end of this procedure. Suppose this is not the case, and let j be the job with minimal t_j that the procedure does not schedule. Let B be the set of all the jobs that are scheduled by the greedy procedure at intervals that start before or at time t_j. By the definition of greedy algorithm,

- All the $10k$ machines are busy during the time interval $[S_{T'}, t_j]$.
- The set of jobs that are scheduled at intervals that start before or at time t_j is exactly B (because j is the first job that we were unable to schedule).

Let Z_{5k} denote the sum of lengths of $5k$ longest jobs from B. Then

$$\sum_{j' \in B} p_{j'} > Z_{5k} + 5k(t_j - S_{T'}).$$

We show, using the fractional solution, that this is not true, thus getting a contradiction.

We now bound

$$\sum_{j' \in B} p_{j'}$$

using the new fractional solution. As already mentioned, all the jobs in B must be completely scheduled at intervals starting before or at time t_j in the fractional solution. Given an interval $I \in \mathcal{I}(j')$, the volume of the interval is defined to be $x'(I, j')p_{j'}$. The sum of lengths of jobs in B equals exactly the total volume of their intervals, which is at most the total volume of intervals belonging to jobs from B that finish before time t_j plus the total volume of intervals crossing the time point t_j. The former is bounded by $5k(t_j - S_{T'})$, and the latter is at most Z_{5k}. Therefore,

$$\sum_{j' \in B} p_{j'} \leq Z_{5k} + 5k(t_j - S_{T'})$$

which is a contradiction.

Thus, all the jobs in $J^d(T')$ are scheduled by the greedy procedure inside T' on $10k$ machines.

Type 4: This type is defined exactly like type 3, only with respect to the release dates of jobs. As in type 3, all jobs of this type can be scheduled on $10k$ machines.

Type 5: All the other jobs. Let G be the set of all the intervals $I \in \mathcal{I}(j), j \in J$, such that I does not cross boundaries of any interval in $\mathcal{T}$, and if $I \subseteq T' \in \mathcal{T}$, then T' does not contain the release date or the deadline of j and further length of T' is more than twice the length of I. Note that each job j of type 5, in the fractional solution, has

$$\sum_{I \in \mathcal{I}(j) \bigcap G} x(I, j) \geq 0.2.$$

Furthermore, if $I \in \mathcal{I}(j)$ and $I \in G$, then job j can be scheduled anywhere inside the interval $T' \in \mathcal{T}$ that contains I.

We divide the jobs of this type into size classes. J_i contains all jobs j of type 5 such that $2^{i-1} < p_j \leq 2^i$. For each interval T' and for each i, let $X(T', i)$ be the total fraction of intervals of size $(2^{i-1}, 2^i]$, that belong to set G and that are contained in T'.

We are going to schedule at most $\lceil 5X(T', i) \rceil$ jobs from J_i inside T'.

Claim 3 *Each interval T' can accommodate, on $22k$ machines, $\lceil 5X(T',i)\rceil$ jobs of size 2^i, simultaneously for all i.*

Proof: In the solution to the linear program, the total volume of intervals scheduled in T' is at least $\sum_i X(T',i)\cdot 2^{i-1}$. Therefore, the length of interval T' is at least $\frac{1}{k}\sum_i X(T',i)\cdot 2^{i-1}$. Interval T' on $10k$ machines has enough space to accommodate $\lfloor 5X(T',i)\rfloor$ jobs of size 2^i simultaneously for all i (we allow here to break a job and schedule parts of it on different machines). In order to accommodate in this fashion $\lceil 5X(T',i)\rceil$ jobs for each i, we need one additional machine. (Note that $X(T',i)>0$ only for such job sizes that are at most half the length of the interval). Finally, in order to schedule jobs that are broken, we need additional $10k+1$ machines. In total, we can schedule all these jobs on $20k+2$ machines inside the interval T'. □

All we have to decide now is for each job size, which job is scheduled in which block. Consider a set of jobs J_i. We want to assign all the jobs in J_i to blocks, such that:

- Job j can only be assigned to a block that is completely contained in j's window.
- At most $\lceil 5X(T',i)\rceil$ jobs are assigned to each each interval $T'\in\mathcal{T}$.

Note that the solution to the linear program implies a feasible fractional solution to this problem: for each j, and T' such that T' is contained in j's window, the fraction of j assigned to T' is 5 times the total fraction of intervals belonging to j inside block T' in the fractional solution. It is easy to see that an integral solution to the problem above can be obtained by the earliest deadline greedy assignment of jobs to blocks.

Finally, each job that is assigned to an interval T' can be scheduled anywhere inside the interval. Therefore, we can use the schedule from Claim 3 to accommodate all the jobs.

4 Integrality Gap

We show in this section that the integrality gap of the "natural" linear program (LP1) presented in Section 2 is $\Omega(\frac{\log n}{\log\log n})$. In fact, the proof of the integrality gap holds for discrete input as well. The set of jobs J is the union of N subsets $J_1,\ldots,J_N$, N to be specified later.

For each i, $1\le i\le N$, for each $j\in J_i$, the processing time is $p_j=(2N)^{i-1}$, and the window size is exactly N times the processing time, i.e., $d_j-r_j=Np_j$. For the sake of convenience, we divide the time window of job j into N non-overlapping intervals of size p_j, denoted by $I_1(j),\ldots,I_N(j)$. Note that for all r, $1\le r\le N$, $I_r(j)\in\mathcal{I}(j)$. We refer to these intervals as *special job intervals*.

The job subsets J_i are defined recursively. The set J_N consists of only a single job whose processing time is $(2N)^{N-1}$, its release date is 0, and its deadline is $N\cdot(2N)^{N-1}$.

In order to define job set J_i, for $1\le i<N$, consider job $j\in J_{i+1}$, and one of its special intervals $I_r(j)$, $1\le r\le N$. The size of the interval is $p_j=(2N)^i$. There are two jobs $j',j''\in J_i$ that correspond to this interval. The release date of j' is at the beginning of interval $I_r(j)$ and its deadline is in the middle of $I_r(j)$. The release date of j'' is in the middle of $I_r(j)$, and its deadline is at the end of $I_r(j)$. Therefore, the window sizes of both jobs are $N\cdot(2N)^{i-1}$. Their processing times are $(2N)^{i-1}$.

The fractional solution to the scheduling problem uses a single machine, and it is defined as follows. For each job $j\in J$, each one of its special intervals receives a fraction $\frac{1}{N}$. Since each job has N special intervals, all the jobs are scheduled. Observe that special intervals of different jobs can overlap only if they belong to different subsets J_i, $J_{i'}$, $i\ne i'$. As the number of subsets is N, this is a feasible fractional schedule on a single machine.

It is easy to see that the construction has the following useful property. For each job $j\in J_i$, for $1<i\le N$, for any interval $I\in\mathcal{I}(j)$ in which job j is scheduled, there is a job $j'\in J_{i-1}$ whose time window is completely contained in I. This property guarantees that any integral solution uses at least N machines. Finally, note that the input size is $n=O((2N)^N)$, and therefore the integrality gap is at least $N=\Omega(\frac{\log n}{\log\log n})$.

We remark that the above construction can be used to prove the same lower bound on the integrality gap in the case of discrete input. For each job $j\in J$, let the interval set $\mathcal{I}(j)$ contain only the special intervals, i.e., $\mathcal{I}(j)=\{I_1(j),\ldots,I_r(j)\}$.

Acknowledgements

We thank the anonymous referees whose valuable comments have helped improve our presentation. In particular, we would like to thank one of the referees for suggesting a more efficient and elegant proof of Theorem 3.

References

[1] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor and B. Schieber, A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48:1069-1090, 2001.

[2] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *Proceedings of the* 31^{st} *Annual ACM Symposium on Theory of Computing*, pp. 622–631, 1999.

[3] P. Berman and B. DasGupta. Improvements in throughout maximization for real-time scheduling. *Proceedings of the* 32^{nd} *Annual ACM Symposium on Theory of Computing*, pp. 680–68, 2000.

[4] J. Chuzhoy and J. Naor. New hardness results for congestion minimization and machine scheduling. *Proceedings of the* 36^{th} *Annual ACM Symposium on Theory of Computing*, pp. 28–34, 2004.

[5] J. Chuzhoy, R. Ostrovsky, and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Proceedings of the* 42^{nd} *Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 348–356, 2001.

[6] M. R. Garey and D. S. Johnson. Two processor scheduling with start times and deadlines, *SIAM Journal on Computing*, 6:416–426, 1977.

[7] M. R. Garey and D. S. Johnson. Computers and Intractability: a Guide to the Theory of NP-Completeness, W. H. Freeman, San Francisco, 1979.

[8] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[9] F.C.R. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2:215–227, 1999.

A Implementing the Algorithms Efficiently

We now focus on efficiently computing an optimal solution to linear program (LP3(T)). In fact we will modify the original problem instance so that the size of the linear program (and of the dynamic programming table used for its solution) becomes polynomial, at the cost of losing a constant factor in its solution value.

Theorem 3 *Given an instance of* (LP3(T))*, we can construct a modified instance in which the total number of job intervals is polynomially bounded, and the number of machines in an optimal solution to the new instance,* $\mathtt{OPT}'$*, is at most* 3 *times the number of machines used in an optimal solution,* $\mathtt{OPT}$*, to the original instance of* (LP3(T)).

Proof: We will prove the above in two steps. In the first step we will identify a set of jobs D such that all of them can be scheduled on a single machine. Simultaneously, we will identify a set of time points P, such that every remaining job in $J \setminus D$, for any feasible schedule, must be scheduled at an interval which contains at least one point of P. In the second step we will show that any feasible solution to the original instance can transformed to a feasible solution of a modified instance with a factor 2 blowup. Further, all feasible solutions of the new instance will be feasible for the original instance as well. Our original proof along the lines of [2] had $|P| = \Theta(n^2)$. The following elegant proof of the first step, suggested by an anonymous referee, gives $|P| = \Theta(n)$.

In the first step, we run a greedy algorithm, that schedules some of the jobs on one machine. We start at time point $t = 0$, and use the following greedy rule: whenever the machine becomes idle, schedule a job that is going to finish first among all the available jobs. Let $D \subseteq J$ denote a subset of jobs scheduled by this algorithm.

Let P be the set of time points containing the endpoints of all the intervals on which jobs in D are scheduled *and* the release dates and deadlines of all the jobs in $J \setminus D$. Notice that for each job $j \in J \setminus D$, for each of its intervals $I \in \mathcal{I}(j)$, there is at least one time point $t \in P$ that belongs to I (otherwise the greedy algorithm would have scheduled j on interval I).

For each job $j \in J \setminus D$, we redefine the set of intervals belonging to j as follows: the left endpoint of the interval becomes the nearest point in P that lies to the left of the original interval and the right endpoint of the interval becomes the nearest point in P that lies to the right of the original interval. Thus, the number of intervals belonging to j becomes polynomial. As a result, if there was a feasible solution with X jobs being scheduled at any point of time for the original instance of (LP3(T)), the total number of jobs from $J \setminus D$ scheduled at a time point (after the intervals grow) can be at most $2X$.

Putting everything together, $2\mathtt{OPT} + 1$ machines suffice to give us a schedule for the new instance. $\square$

We can solve (LP3(T)) on the new instance in polynomial time. Note that in the dynamic programming procedure, we only need to compute the values $m(T')$ of time intervals T' whose endpoints are in P. The values $m(T')$ are defined with respect to the new instance and for time interval T containing all the release dates and deadlines, $m(T) \leq 3\mathtt{OPT}$ holds. This LP-solution implies a feasible fractional LP solution to the original instance, where all the intervals $I \in \mathcal{I}(j)$ are replaced with their original counterparts, and thus the number of such intervals for which $x(I, j) > 0$ is polynomial. We now perform our LP-rounding procedure on this instance. The only subtle point to notice is that in our recursive procedure we will be using solutions of (LP3) for different intervals T' produced by PROCEDURE PARTITION. However, in our dynamic programming table we only have solutions of (LP3) for those intervals T' (and corresponding job sets $J(T')$) whose endpoints belong to P, while the intervals produced by PROCEDURE PARTITION might not be of this type. However, since the release dates and the deadlines of all the jobs belong to P, we can use the solution of (LP3) for the largest sub-interval of T' whose endpoints belong to P, and this solution is a valid solution of (LP3) for T'.