Resource Minimization Job Scheduling*

Julia Chuzhoy † Paolo Codenotti ‡

July 17, 2009

Abstract

Given a set J of jobs, where each job j is associated with release date r_j, deadline d_j and processing time p_j, our goal is to schedule all jobs using the minimum possible number of machines. Scheduling a job j requires selecting an interval of length p_j between its release date and deadline, and assigning it to a machine, with the restriction that each machine executes at most one job at any given time. This is one of the basic settings in the resource-minimization job scheduling, and the classical randomized rounding technique of Raghavan and Thompson provides an $O(\log n/ \log\log n)$-approximation for it. This result has been recently improved to an $O(\sqrt{\log n})$-approximation, and moreover an efficient algorithm for scheduling all jobs on $O((\mathsf{OPT})^2)$ machines has been shown. We build on this prior work to obtain a constant factor approximation algorithm for the problem.

1 Introduction

In one of the basic scheduling frameworks, the input consists of a set J of jobs, and each job $j \in J$ is associated with a subset $\mathcal{I}(j)$ of time intervals, during which it can be executed. The sets $\mathcal{I}(j)$ of intervals can either be given explicitly (in this case we say we have a *discrete* input), or implicitly by specifying the release date r_j, the deadline d_j and the processing time p_j of each job j (*continuous* input). In the latter case, $\mathcal{I}(j)$ is the set of all time intervals of length p_j contained in the time window $[r_j, d_j]$. A schedule of a subset $J' \subseteq J$ of jobs assigns each job $j \in J'$ to one of the time intervals $I \in \mathcal{I}(j)$, during which j is executed. In addition to selecting a time interval, each job is also assigned to a machine, with the restriction that all jobs assigned to a single machine must be executed on non-overlapping time intervals.

In this paper we focus on the Machine Minimization problem, where the goal is to schedule all the jobs, while minimizing the total number of machines used. We refer to the discrete and the continuous versions of the problem as Discrete and Continuous Machine Minimization, respectively. Both versions admit an $O(\log n/ \log\log n)$-approximation via the Randomized LP-Rounding technique of Raghavan and Thompson [8], and this is the best currently known approximation for Discrete Machine Minimization. Chuzhoy and Naor [7] have shown that the discrete version is $\Omega(\log\log n)$-hard to approximate. Better approximation algorithms are known for Continuous Machine Minimization: an $O(\sqrt{\log n})$-approximation algorithm was shown by Chuzhoy et. al. [6], who also obtain better performance guarantees when the optimal solution cost is small. Specifically, they give an efficient

*A conference version of this paper is to appear in APPROX 2009

†Toyota Technological Institute, Chicago, IL 60637. Email: cjulia@tti-c.org. Supported in part by NSF CAREER award CCF-0844872

‡Department of Computer Science, University of Chicago, Chicago, IL 60637. Email: paoloc@uchicago.edu

algorithm for scheduling all jobs on $O(k^2)$ machines, where k is the number of machines used by the optimal solution. In this paper we improve their result by showing a constant factor approximation algorithm for Continuous Machine Minimization. Combined with the lower bound of [6], our result proves a separation between the discrete and the continuous versions of Machine Minimization.

Related Work A problem that can be seen as dual to Machine Minimization is Throughput Maximization, where the goal is to maximize the number of jobs scheduled on a single machine. This problem has an $\left(\frac{e}{e-1}+\epsilon\right)$-approximation for any constant ϵ, in both the discrete and the continuous settings [5]. The discrete version is MAX-SNP hard even when each job has only two intervals [9] (i.e., $|\mathcal{I}(j)| = 2$ for all j), while no hardness of approximation results are known for the continuous version. In the more general weighted setting of Throughput Maximization, each job j is associated with weight w_j, and the goal is to maximize the total weight of scheduled jobs. The best current approximation factor for this problem is 2 for both the discrete and the continuous versions [2].

A natural generalization of Throughput Maximization is the Resource Allocation problem, where each job j is also associated with height (or bandwidth) h_j. The goal is again to maximize the total weight of scheduled jobs, but now the jobs are allowed to overlap in time, as long as the total height of all jobs executed at each time point does not exceed 1. For the weighted variant of this problem, Bar-Noy et. al. [3] show a factor 5-approximation, while the unweighted version can be approximated up to factor $(2e-1)/(e-1)+\epsilon$ for any constant ϵ [5]. For the special case of Resource Allocation where each job has exactly one time interval (i.e., $|\mathcal{I}(j)| = 1$ for all j), Calinescu et. al. [4] show a factor $(2+\epsilon)$-approximation for any ϵ, and Bansal et. al. [1] give a Quasi-PTAS.

Our Results and Techniques We show a constant factor approximation algorithm for Continuous Machine Minimization. Our algorithm builds on the work of Chuzhoy et. al. [6]. Since the basic linear programming relaxation for the problem is known to have an $\Omega(\log n/\log\log n)$ integrality gap, [6] design a stronger recursive linear programming relaxation for the problem. The solution of this LP involves dynamic programming, where each entry of the dynamic programming table is computed by solving the LP relaxation on the corresponding sub-instance. Using the LP solution, [6] then partition the input set J of jobs into $k = \lceil \mathsf{OPT} \rceil$ subsets, $J^1, \ldots, J^k$. They show that each subset J^i can be scheduled on $O(k_i)$ machines, where k_i is the total number of machines used to schedule all jobs in J^i by the fractional solution. Since in the worst case k_i can be as large as k for all i, they eventually use $O(k^2)$ machines to schedule all jobs.

We perform a similar partition of jobs into subsets. One of our main ideas is to define, for each job class J^i, a function $f_i(t)$, whose value is the total fractional weight of intervals of jobs in J^i containing time point t. We then find a schedule for each job class J^i, that schedules at most $O(\lceil f_i(t) \rceil)$ jobs at each time point t. The algorithm for finding the schedule itself is similar to that of [6], but more work is needed to adapt their algorithm to this new setting.

2 Preliminaries

In the Continuous Machine Minimization problem the input consists of a set J of jobs, and each job $j \in J$ is associated with a release date r_j, a deadline d_j and a processing time p_j. The goal is to schedule all jobs, while minimizing the number of machines used. In order to schedule a job j, we need to choose a time interval $I \subseteq [r_j, d_j]$ of length p_j during which job j will be executed, and to assign the job to one of the machines. For each machine M, let $\mathcal{J}(M)$ be the set of jobs assigned to M and let $\mathcal{I}(M)$ be the set of intervals chosen for the jobs in $\mathcal{J}(M)$. Then every pair of intervals in $\mathcal{I}(M)$ must be disjoint.

We denote by $\mathcal{I}(j)$ the set of all time intervals of job j, so $\mathcal{I}(j)$ contains all intervals of length p_j

contained in the time window $[r_j, d_j]$. For convenience we will assume that these intervals are open. If $I \in \mathcal{I}(j)$, then we say that interval I *belongs* to job j. Notice that $|\mathcal{I}(j)|$ may be exponential in the input length. Given any solution, if interval I is chosen for job j, we say that j is scheduled on interval I, and for each $t \in I$ we say that j is scheduled at time t. We denote by $\mathcal{T}$ the smallest time interval containing all the input job intervals, and denote by OPT both the optimal solution and its cost. We refer to the time interval $[r_j, d_j]$ as the *time window* of job j. We will use the following simple observation.

Claim 1 *Let $\mathcal{S}$ be a set of intervals containing exactly one interval $I \in \mathcal{I}(j)$ for each job $j \in J$. Moreover, assume that for each time point $t \in \mathcal{T}$, the total number of intervals in $\mathcal{S}$ containing t is at most k. Then all jobs in J can be scheduled on k machines, and moreover, given S, such a schedule can be found efficiently.*

Proof: Consider the interval graph defined by the set $\mathcal{S}$. The size of the maximum clique in this graph is at most k, and therefore it can be efficiently colored by k colors. Each color will correspond to a distinct machine. □

Our goal is therefore to select a time interval $I \in \mathcal{I}(j)$ for each job j, while minimizing the maximum number of jobs scheduled at any time point t.

The Linear Programming Relaxation. We now describe the linear programming relaxation of [6], which is also used by our approximation algorithm. We start with the following basic linear programming relaxation for the problem. For each job $j \in J$, for each interval $I \in \mathcal{I}(j)$, we have an indicator variable $x(I, j)$ for scheduling job j on interval I. We require that each job is scheduled on at least one interval, and that the total number of jobs scheduled at each time point $t \in \mathcal{T}$ is at most z, the value of the objective function. The linear programming relaxation is then as follows.

$$
\begin{array}{llcl}
\text{(LP1)} & \min & z & \\
 & \text{s.t.} & \sum_{I \in \mathcal{I}(j)} x(I, j) = 1 & \forall j \in J \\
 & & \sum_{j \in J} \sum_{\substack{I \in \mathcal{I}(j): \\ t \in I}} x(I, j) \le z & \forall t \in \mathcal{T} \\
 & & x(I, j) \ge 0 & \forall j \in J, \forall I \in \mathcal{I}(j)
\end{array}
$$

It is well-known however that the integrality gap of (LP1) is $\Omega\left(\frac{\log n}{\log \log n}\right)$ (e.g. see [6]). To overcome this barrier, Chuzhoy et. al. [6] propose a stronger relaxation for the problem. Consider first the special case where the optimal solution uses only one machine, that is, $\mathsf{OPT} = 1$. Let $I \in \mathcal{I}(j)$ be some job interval, and suppose there is another job $j' \neq j$, whose entire time window $[r_{j'}, d_{j'}]$ is contained in I. Then interval I is called *forbidden interval* for job j. Since $\mathsf{OPT} = 1$, job j cannot be scheduled on interval I. Therefore, we can add the valid constraint $x(I, j) = 0$ to the LP for all jobs j and intervals I, where I is a forbidden interval for job j. Chuzhoy et. al. show an LP-rounding algorithm for this stronger LP relaxation that schedules all jobs on a constant number of machines for this special case of the problem.

When the optimal solution uses more than one machine, constraints of the form $x(I, j) = 0$, where I is a forbidden interval for job j, are no longer valid. Instead, [6] define a function $m(T)$ for each time interval $T \subseteq \mathcal{T}$, whose intuitive meaning is as follows. Let $J(T)$ be the set of jobs whose time window is completely contained in T. Then $m(T)$ is the minimum number of machines needed to schedule jobs in $J(T)$. Formally, $m(T) = \lceil z \rceil$, where z is the optimal solution of the following linear program:

$$
\begin{array}{llll}
\text{(LP(T))} & \min & z & \\
& \text{s.t.} & \sum_{I\in\mathcal{I}(j)} x(I,j) = 1 & \forall j \in J(T) \\
& & \sum_{j\in J(T)} \sum_{\substack{I\in\mathcal{I}(j):\\ t\in I}} x(I,j) \le z & \forall t \in T \qquad (1)\\
& & \sum_{j\in J(T)} \sum_{\substack{I\in\mathcal{I}(j):\\ T'\subseteq I}} x(I,j) \le z - m(T') & \forall T' \subseteq T \qquad (2)\\
& & x(I,j) \ge 0 & \forall j \in J(T), \forall I \in \mathcal{I}(j)
\end{array}
$$

Observe that for integral solutions, where $x(I,j) \in \{0,1\}$ for all $j \in J, I \in \mathcal{I}(j)$, the value $m(T)$ is precisely the number of machines needed to schedule all jobs in $J(T)$. Constraint (2) requires that for each time interval $T' \subseteq T$, the total number of jobs scheduled on intervals containing T' is at most $m(T) - m(T')$. This is a valid constraint, since at least $m(T')$ machines are needed to schedule all jobs in $J(T')$. Therefore, $\lceil \mathsf{OPT}(\mathcal{T}) \rceil \le \mathsf{OPT}$. Notice that the number of constraints in $LP(T)$ may be exponential in the input size. This difficulty is overcome in [6] as follows. First they define, for each job $j \in J$ a new discrete subset $\mathcal{I}'(j)$ of time intervals, with $|\mathcal{I}'(j)| = \text{poly}(n)$. Sets $\mathcal{I}'(j)$ of intervals for $j \in J$ define a new instance of Discrete Machine Minimization, whose optimal solution cost is at most $3\,\mathsf{OPT}$. Moreover, any solution for the new instance implies a feasible solution for the original instance of the same cost. Next they define the set $D \subseteq \mathcal{T}$ of time points, consisting of all release dates and deadlines of jobs in J, and all endpoints of intervals in $\{\mathcal{I}'(j)\}_{j\in J}$. Clearly, the size of D is polynomially bounded. Finally they modify $LP(T)$, so that Constraint (1) is only defined for $t \in D$ and Constraint (2) is only applied to time intervals T with both endpoints in D. The new LP relaxation can be solved in polynomial time and its solution cost is denoted by OPT'. We are guaranteed that $\lceil \mathsf{OPT}' \rceil \le 3\,\mathsf{OPT}$. Moreover, any feasible solution to the new LP implies a feasible solution to the original LP. From now on we will denote by x this near-optimal fractional solution, and by $\mathsf{OPT}'(\mathcal{T})$ its value, $\lceil \mathsf{OPT}'(\mathcal{T}) \rceil \le 3\,\mathsf{OPT}$. For each job $j \in J$, let $\mathcal{I}^*(j) \subseteq \mathcal{I}(j)$ be the subset of intervals I for which $x(I,j) > 0$. For any interval $I \in \mathcal{I}^*(j)$, we call $x(I,j)$ the *LP-weight* of I.

3 The Algorithm

Our algorithm starts by defining a recursive partition of the time line into blocks. This recursive partition in turn defines a partition of the jobs into *job classes* $J^1, J^2, \ldots$ Our algorithm then defines, for each job class J^i, a function $f_i : \mathcal{T} \to \mathbb{R}$, where $f_i(t)$ is the summation of values $x(I,j)$ over all jobs $j \in J^i$ and intervals $I \in \mathcal{I}(j)$ containing t. We then consider each of the job classes J^i separately, and show an efficient algorithm for scheduling jobs in J^i so that at most $O(\lceil f_i(t) \rceil)$ jobs of J^i are executed at each time point $t \in \mathcal{T}$.

3.1 Partition into Blocks and Job Classes

Let T be any time interval, and let $\mathcal{B}$ be any set of disjoint sub-intervals of T. Then we say that $\mathcal{B}$ defines a partition of T into blocks, and each interval $B \in \mathcal{B}$ is referred to as a *block*. Notice that we do not require that the union of the intervals in $\mathcal{B}$ is T.

Let $k = \lceil m(\mathcal{T}) \rceil$ be the cost of the near-optimal fractional solution. We define a recursive partition of the time interval $\mathcal{T}$ into blocks. We use a partitioning sub-routine that receives as input a time interval T and a set $J(T)$ of jobs whose time windows are contained in T. The output of the procedure is a partition $\mathcal{B}$ of T into blocks. This partition in turn defines a partition of the set $J(T)$ of jobs, as follows. For each $B \in \mathcal{B}$, we have a set $J_B \subseteq J(T)$ of jobs whose time window is contained in B, so $J_B = \{j \in J(T) \mid [r_j, d_j] \subseteq B\}$. Let $J'' = \cup_{B \in \mathcal{B}} J_B$, and let $J' = J(T) \setminus J''$. Notice that $J' \dot\cup (\dot\bigcup_{B \in \mathcal{B}} J_B)$ is indeed a partition of $J(T)$, and that for each $j \in J'$, r_j and d_j lie in distinct blocks. The partitioning procedure will also guarantee the following properties: (i) For each job $j \in J'$, each interval $I \in \mathcal{I}^*(j)$ has a non-empty intersection with at most two blocks; and (ii) For each $B \in \mathcal{B}$, there is a job $j \in J'$ and a job interval $I \in \mathcal{I}^*(j)$, with $B \subseteq I$.

A partitioning procedure with the above properties is provided in [6]. For the sake of completeness we briefly sketch it here. Let $T = [L, R]$. We start with $t = L$ and $\mathcal{B} = \emptyset$. Given a current time point t, the next block $B = (\ell, r)$ is defined as follows. If there is any job $j \in J(T)$ with a time interval $I \in \mathcal{I}^*(j)$ containing t, we set the left endpoint of our block to be $\ell = t$. Otherwise, we set it to be the first (i.e., the leftmost) time point to the right of t for which such a job and such an interval exist. To define the right endpoint of the block, we consider the set S of all job intervals with non-zero LP-weight containing ℓ, so $S = \{I \mid \ell \in I \text{ and } \exists j \in J(T) : I \in \mathcal{I}^*(j)\}$. Among all intervals in S, let I^* be the interval with rightmost right endpoint. We then set r to be the right endpoint of I^*. Block $B = (\ell, r)$ is then added to $\mathcal{B}$, we set $t = r$ and continue.

We are now ready to describe our recursive partitioning procedure. We have k iterations (recall that $k = \lceil m(\mathcal{T}) \rceil$ is the cost of the near-optimal fractional solution). Iteration h, for $1 \leq h \leq k$, produces a partition $\mathcal{B}^h$ of $\mathcal{T}$ into blocks, refining the partition $\mathcal{B}^{h-1}$. Additionally, we produce a partition of the set J of jobs into k classes $J^1, \ldots, J^k$. In the first iteration, we apply the partitioning procedure to time interval $\mathcal{T}$ and the set J of jobs. We set $\mathcal{B}^1$ to be the partition into blocks produced by the procedure. We denote the corresponding partition of the jobs as follows: $J^1 = J'$, and for all $B \in \mathcal{B}^1$, we denote J_B by J^1_B. In general, to obtain partition $\mathcal{B}^h$, we run the partitioning algorithm on each one of the blocks $B \in \mathcal{B}^{h-1}$, together with the associated subset J^{h-1}_B of jobs. For each block $B \in \mathcal{B}^{h-1}$, we denote by $\mathcal{B}_B$ the new block partition and by $J^{h-1}_B = (J'_B, J''_B)$ the new job partition computed by the partitioning procedure. We then set $\mathcal{B}^h = \bigcup_{B \in \mathcal{B}^{h-1}} \mathcal{B}_B$, $J^h = \bigcup_{B \in \mathcal{B}^{h-1}} J'_B$, and for each block $B' \in \mathcal{B}^h$, let $J^h_{B'}$ denote the subset of jobs in J^{h-1}, whose time windows are contained in B'. This finishes the description of the recursive partitioning procedure. An important property, established in the next claim, is that every job is assigned to one of the k classes $J^1, \ldots, J^k$.

Claim 2 $J = J^1 \cup \cdots \cup J^k$.

Proof: It is enough to prove that for each $B \in \mathcal{B}^k$, $J^k_B = \emptyset$. Assume otherwise, and let $B \in \mathcal{B}^k$ be some block with $j \in J^k_B$. Consider the nested set of blocks $B_1, B_2, \ldots, B_k = B$, where for each $h : 1 \leq h < k$, $B_h \in \mathcal{B}^h$ and $B_{h+1} \subseteq B_h$, so B_{h+1} has been created when the partitioning procedure was applied to B_h. Let $j_h \in J^{h-1}_{B_{h-1}}$ be the job whose interval $I_h \in \mathcal{I}^*(j)$ has defined the right endpoint of B_h. We then have a set of nested intervals: $[r_j, d_j] \subseteq B_k \subseteq I_k \subseteq B_{k-1} \subseteq \cdots \subseteq B_1 \subseteq I_1$. We claim that for each $h : 1 \leq h \leq k$, $m(B_h) \geq k - h + 1$. The proof is by induction. We start with $h = k$. Since $[r_j, d_j] \subseteq B_k$, $m(B_k) \geq 1$. Assume now the claim for some h, and we will prove it for $h - 1$. Consider block B_{h-1}. Since interval I_h was used to define the right endpoint of B_h, the time window of j_h is contained in B_{h-1}, while $B_h \subseteq I_h$ and $x(j_h, I_h) > 0$. Constraint (2) then ensures that $m(B_{h-1}) \geq m(B_h) + 1 \geq k - h$. It follows that $m(B_1) = k$. But $B_1 \subseteq I_1$ and $x(j_1, I_1) > 0$, contradicting Constraint (2) of $LP(\mathcal{T})$. □

We have thus obtained a recursive partition $\mathcal{B}^1, \ldots, \mathcal{B}^k$ of $\mathcal{T}$ into blocks, and a partition $J = \bigcup_{h=1}^k J^h$ of jobs into classes. For simplicity we denote $\mathcal{B}^0 = \{\mathcal{T}\}$.

The algorithm of [6] can now be described as follows. Consider the set J^h of jobs, for $1 \leq h \leq k$, together with the partition $\mathcal{B}^{h-1}$ of $\mathcal{T}$ into blocks. Recall that for each block $B \in \mathcal{B}^{h-1}$, J^{h-1}_B is the subset of jobs whose time windows are contained in B, and $J^h \subseteq \bigcup_{B \in \mathcal{B}^{h-1}} J_B^{h-1}$. Consider now some block $B \in \mathcal{B}^{h-1}$ and the corresponding subset $\tilde{J} = J^h \cap J_B^{h-1}$. Let $\mathcal{B}' = \mathcal{B}_B$ be the partition of B into blocks returned by the partitioning procedure when computing $\mathcal{B}^h$. This partition has the property that each interval $I \in \mathcal{I}^*(j)$ of each job $j \in \tilde{J}$ has a non-empty intersection with at most two blocks in $\mathcal{B}'$, and furthermore for each $j \in \tilde{J}$, the window of j is not contained in any single block $B \in \mathcal{B}'$. These two properties are used in [6] to extend a simpler algorithm for the special case where $\mathsf{OPT} = 1$ to the more general setting, where an arbitrary number of machines is used. In particular, if k_h is the fractional number of machines used to schedule jobs in J^h (i.e., k_h is the maximum value, over time points t, of $\sum_{j \in J^h} \sum_{I \in \mathcal{I}(j): t \in I} x(I, j)$), then all jobs in J^h can be efficiently scheduled on $O(\lceil k_h \rceil)$ machines. In the worst case, $\lceil k_h \rceil$ can be as large as $\lceil k \rceil$ for all $h : 1 \leq h \leq k$, and so overall $O(k^2)$ machines are used in the algorithm of [6].

In this paper, we refine this algorithm and its analysis as follows. For each $h : 1 \leq h \leq k$, we define a function $f_h : \mathcal{T} \to \mathbb{R}$, where $f_h(t)$ is the total fractional weight of intervals containing t that belong to jobs in J^h. Clearly, for all t, $\sum_h f_h(t) \leq k$. We then consider each one of the job classes J^h separately. For each job class J^h we find a schedule for jobs in J^h, such that for each time point $t \in \mathcal{T}$, at most $O(\lceil f_h(t) \rceil)$ jobs are scheduled on intervals containing t. The algorithm for scheduling jobs in J^h and its analysis are similar to those in [6]. We partition all jobs in J^h into a constant number of subsets, according to the way the fractional weight is distributed on their intervals. We then schedule each one of the subsets separately. The analysis is similar to that of [6], but does not follow immediately from their work. In particular, more care is needed in the analysis of the subsets of jobs j that have substantial LP-weight on intervals lying inside blocks to which r_j or d_j belong.

We now proceed to describe our algorithm more formally. For each job class $J^h : 1 \leq h \leq k$, let $f_h : \mathcal{T} \to \mathcal{R}$ be defined as follows. For each $t \in \mathcal{T}$, $f_h(t) = \sum_{j \in J^h} \sum_{\substack{I \in \mathcal{I}(j): \\ t \in I}} x(I, j)$. Our goal is to prove the following theorem:

Theorem 1 *For each job class $J^h : 1 \leq h \leq k$, we can efficiently schedule jobs in J^h so that, for each time point $t \in \mathcal{T}$, at most $O(\lceil f_h(t) \rceil)$ jobs are scheduled on intervals containing t.*

We prove the theorem in the next section. We show here that a constant factor approximation algorithm for Continuous Machine Minimization follows from Theorem 1. For each time point $t \in \mathcal{T}$, the total number of jobs scheduled on intervals containing point t is at most $\sum_h O(\lceil f_h(t) \rceil)$. Since $\sum_h f_h(t) \leq k$, $\sum_{h=1}^k \lceil f_h(t) \rceil \leq 2k$, and so the solution cost is $O(k)$.

3.2 Proof of Theorem 1

Consider a job class J^h and the block partition $\mathcal{B}^{h-1}$. For each block $B \in \mathcal{B}^{h-1}$, let $J_B^* = J_B^{h-1} \cap J^h$ be the set of jobs whose windows are contained in B, and so $J^h = \bigcup_{B \in \mathcal{B}^{h-1}} J_B^*$. Clearly, for blocks $B \neq B'$, the windows of jobs in J_B^* and $J_{B'}^*$ are completely disjoint, and therefore they can be considered separately. From now on we focus on scheduling jobs in J_B^* inside a specific block $B \in \mathcal{B}^{h-1}$. For simplicity, we denote $J^* = J_B^*$, and $\mathcal{B}^*$ is the partition of B into blocks obtained when computing $\mathcal{B}^h$. Recall that we have the following properties: (i) For each job $j \in J^*$, r_j and d_j lie in distinct blocks of $\mathcal{B}^*$; and (ii) For each job $j \in J^*$, each interval $I \in \mathcal{I}^*(j)$ has a non-empty intersection with at most two blocks

For each $t \in B$, let $g(t) = \lceil f_h(t) \rceil$. Observe that $g(t)$ is a step function. Our goal is to schedule all jobs in J^h so that, for each $t \in B$, at most $O(g(t))$ jobs are scheduled on intervals containing t. The rest of the algorithm consists of three steps. In the first step, we partition the area "below" the function $g(t)$ into a set $\mathcal{R}$ of rectangles of height 1. In the second step we assign each job interval $I \in \mathcal{I}^*(j)$

for $j \in J^*$ to one of the rectangles $R \in \mathcal{R}$, such that the total LP-weight of intervals assigned to R at each time point $t \in R$ is at most 5. In the third step, we partition all jobs in J^* into 7 types, and find a schedule for each one of the types separately. The assignment of job intervals to rectangles found in Step 2 will help us find the final schedule.

Step 1: Defining Rectangles. A rectangle R is defined by a time interval $W(R)$, and we think of R as the interval $W(R)$ of height 1. We say that time point t belongs to R iff $t \in W(R)$ and we say that interval I is contained in R iff $I \subseteq W(R)$. We denote by ℓ_R and r_R the left and the right endpoints of $W(R)$ respectively. We find a nested set $\mathcal{R}$ of rectangles, such that for each $t \in T$, the total number of rectangles containing t is exactly $g(t)$.

To compute the set $\mathcal{R}$ of rectangles, we maintain a function $g' : B \to \mathbb{Z}$. Initially $g'(t) = g(t)$ for all $t \in B$ and $\mathcal{R} = \emptyset$. While there is a time point $t \in B$ with $g'(t) > 0$, we perform the following: Let I be the longest consecutive sub-interval of B with $g'(t) \geq 1$ for all $t \in I$. We add a rectangle R of height 1 with $W(R) = I$ to $\mathcal{R}$ and decrease the value $g'(t)$ for all $t \in I$ by 1. Consider the final set $\mathcal{R}$ of rectangles. For each $t \in B$, let $\mathcal{R}(t) \subseteq \mathcal{R}$ be the subset of rectangles containing the point t. Then for each $t \in B$, $|\mathcal{R}(t)| = g(t)$. Furthermore, it is easy to see that $\mathcal{R}$ is a nested set of rectangles, and for every pair $R, R' \in \mathcal{R}$ of rectangles with non-empty intersection, either $W(R) \subseteq W(R')$ or $W(R') \subseteq W(R)$ holds.

Claim 3 *If $R, R' \in \mathcal{R}$ and $W(R) \cap W(R') \neq \emptyset$, then either $W(R) \subseteq W(R')$ or $W(R') \subseteq W(R)$.*

Proof: Assume otherwise, and assume w.l.o.g. that R was added to $\mathcal{R}$ before R'. Consider the interval $I = W(R) \cup W(R')$. Then at the time when R was added to $\mathcal{R}$, for each $t \in I$ $g(t) > 1$, and moreover $W(R) \subset I$. Therefore, we should have added I instead of $W(R)$ to $\mathcal{R}$. □

Notice also that a rectangle $R \in \mathcal{R}$ may contain several blocks or be contained in a block. Its endpoints also do not necessarily coincide with block boundaries.

Step 2: Assigning Job Intervals to Rectangles. We start by partitioning the set $\mathcal{R}$ of rectangles into k layers as follows. The first layer L_1 contains all rectangles $R \in \mathcal{R}$ that are not contained in any other rectangle in $\mathcal{R}$. In general layer L_z contains all rectangles $R \in \mathcal{R} \setminus (L_1 \cup \cdots L_{z-1})$ that are not contained in any other rectangle in $\mathcal{R} \setminus (L_1 \cup \cdots L_{z-1})$ (if we have identical rectangles then at most one of them is added to each layer, breaking ties arbitrarily). Since $\mathcal{R}$ is a nested set of rectangles, each $R \in \mathcal{R}$ belongs to one of the layers $L_1, \ldots, L_k$, and the rectangles in each layer are disjoint.

Let $\mathcal{I} = \{I \in \mathcal{I}^*(j) \mid j \in J^*\}$ be the set of all intervals of jobs in J^* with non-zero weight. For $I \in \mathcal{I}$, we say that I belongs to layer z_I iff z_I is the largest index, for which there is a rectangle $R \in L_z$ containing I. If I belongs to layer L_{z_I}, then for each layer $L_{z'}$, $1 \leq z' \leq z_I$, there is a *unique* rectangle $R(I, z') \in L_{z'}$ containing I. Let $\mathcal{I}_z \subseteq \mathcal{I}$ be the set of intervals belonging to layer z. Then $\mathcal{I} = \bigcup_{z=1}^{k} \mathcal{I}_z$.

We process intervals in $\mathcal{I}_1, \ldots, \mathcal{I}_k$ in this order, while intervals belonging to the same layer are processed in non-increasing order of their lengths, breaking ties arbitrarily. Let $I \in \mathcal{I}_z$ be some interval, and assume that $I \in \mathcal{I}^*(j)$. Consider the rectangles $R(I, 1), \ldots, R(I, z_I)$. For each $z' : 1 \leq z' \leq z_I$, we say that I is *feasible* for $R(I, z')$ iff, for each time point $t \in I$, the total LP-weight of intervals currently assigned to R that contain t is at most $5 - x(I, j)$. We select any rectangle $R(I, z')$, $1 \leq z' \leq z_I$, for which I is feasible and assign I to $R(I, z')$. In order to show that this procedure succeeds, it is enough to prove the following:

Claim 4 *When interval I is processed, there is at least one rectangle $R(I, z')$, with $1 \leq z' \leq z_I$, for which I is feasible.*

Proof: Assume otherwise. Let $I' \in \mathcal{I}$ be any interval that has already been processed. It is easy to see that $I' \not\subseteq I$: If I' and I belong to the same layer, then the length of I' should be greater than or equal to the length of I, so $I' \not\subseteq I$. If I' belongs to some layer z and I belongs to layer $z_I > z$, then by the definition of layers it is impossible that $I' \subseteq I$ (since then any rectangle containing I would also contain I'). Therefore, any job interval that has already been processed and overlaps with I must contain either the right or the left endpoint of I. Let ℓ and r denote the left and the right endpoints of I, respectively.

Let R be any rectangle in $\{R(I,1),\ldots,R(I,z_I)\}$. Let $w_\ell(R)$ denote the total LP-weight of job intervals assigned to R that contain ℓ, and define $w_r(R)$ similarly for r. Since I cannot be assigned to R, $w_\ell(R)+w_r(R) > 4$. Therefore, either $\sum_{z=1}^{z_I} w_\ell(R(I,z)) > 2z_I$ or $\sum_{z=1}^{z_I} w_r(R(I,z)) > 2z_I$. Assume w.l.o.g. that it is the former. So we have a set S of job intervals belonging to layers $1,\ldots,z_I$, all containing point ℓ, whose total LP-weight is greater than $2z_I$. Let t_1, t_2 be the time points closest to ℓ on left and right respectively, such that $g(t_i) < z_I + 1$ for $i \in \{1,2\}$. Then there is a layer-(z_I+1) rectangle $R \in \mathcal{R}$ with $W(R) = [t_1, t_2]$. Let I' be any interval in S. Since I' belongs to one of the layers $1,\ldots,z_I$, it is not contained in $W(R)$, and so either $t_1 \in I'$ or $t_2 \in I'$. Therefore, either the total LP-weight of intervals I' in S containing t_1 is more than z_I, or the total LP-weight of intervals I' in S containing t_2 is more than z_I. But this contradicts the fact that $g(t_i) < z_I + 1$. □

Step 3: Scheduling the Jobs Given a rectangle $R \in \mathcal{R}$, let $\mathcal{I}(R) \subseteq \mathcal{I}$ be the set of job intervals assigned to R. For simplicity from now on we denote J^* by J and the block partition $\mathcal{B}^*$ by $\mathcal{B}$. As before, for each time point t, $\mathcal{R}(t) \subseteq \mathcal{R}$ denotes the set of rectangles containing t. We partition the jobs into 7 types $Q_1,\ldots,Q_7$. We then schedule each of the types separately. Each job $j \in J$ will be scheduled on one of its time intervals $I \in \mathcal{I}(j)$. If $I \in \mathcal{I}(R)$, then we say that j is scheduled *inside* R. Given a subset S of jobs scheduled inside a rectangle R, we say that the schedule *uses α machines* iff for each time point $t \in R$, the total number of jobs of S scheduled on intervals in $\mathcal{I}(R)$ containing t is at most α. We will ensure that for each job type Q_i, for each rectangle $R \in \mathcal{R}$, all jobs of Q_i scheduled inside R use a constant number of machines. Since $|\mathcal{R}(t)| = g(t)$ for all $t \in B$, overall we obtain a schedule where the number of jobs scheduled at time t is at most $O(g(t))$ for all $t \in B$, as desired. We start with a high level overview. The set Q_1 contains jobs with a large LP-weight on intervals intersecting block boundaries. The set Q_2 contains all jobs with large LP-weight on intervals I whose length is more than half the length of $R(I)$. These two job types are taken care of similarly to type 1 and 2 jobs in [6]. The sets Q_3 and Q_5 contain jobs j with large LP-weight on intervals belonging to rectangles that contain d_j. These sets corresponds to jobs of type 3 in [6]. However, in our more general setting, we need to consider many different rectangles contained in a block simultaneously, and so these job types require more care and the algorithm for scheduling them and its analysis are more complex. Job types 4 and 6 are similar to types 3 and 5, except that we use release dates instead of deadlines. Finally, type 7 contains all remaining jobs, and we treat them similarly to jobs of type 5 in [6]. We now proceed to define the partition of jobs into 7 types, and show how to schedule jobs of each type.

Type 1: Let P be the set of time points that serve as endpoints of blocks in $\mathcal{B}$. We say that $I \in \mathcal{I}$ is a type-1 interval, and denote $I \in \mathcal{I}_1$, iff it contains a point in P. We define the set of jobs of type 1: $Q_1 = \left\{ j \in J \mid \sum_{I \in \mathcal{I}(j) \cap \mathcal{I}_1} x(I,j) \geq 1/7 \right\}$. These jobs are treated similarly to type-1 jobs in [6]. For the sake of completeness, we sketch the algorithm below.

To schedule type-1 jobs, we construct a directed flow network $G = (V,E)$, as follows. Let $V_1 = Q_1$, and $V_2 = \{(p,R) \mid p \in P, R \in \mathcal{R}(p)\}$, i.e., V_2 contains pairs (p,R) where p is a block boundary and R is a rectangle containing p). Additionally, we have a source s and a sink t, so the final set of vertices is $V = V_1 \cup V_2 \cup \{s,t\}$. The edges are defined as follows. There is an edge of capacity 1 from the source s to every vertex in V_1, and an edge of capacity 35 from every vertex in V_2 to t. Additionally, for each $j \in V_1$ and $(p,R) \in V_2$, there is an edge of capacity 1 from j to (p,R) iff there is an interval $I \in \mathcal{I}^*(j)$

containing p that is assigned to R. The solution to the linear program defines a feasible flow in this graph of value $|V_1|$ as follows. We send one flow unit from the source s to each vertex in V_1. For each $j \in V_1$ and $(b, R) \in V_2$, with $(j, (b, R)) \in E$, the amount of flow sent on e is proportional to the sum of values $x(I, j)$ for intervals $I \in \mathcal{I}^*(j) \cap \mathcal{I}(R)$ containing p. Formally, we send $\frac{\sum_{\substack{I \in \mathcal{I}^*(j) \cap \mathcal{I}(R): \\ p \in I}} x(I,j)}{\sum_{I \in \mathcal{I}^*(j) \cap \mathcal{I}1} x(I,j)}$ flow units on edge e.

By the definition of Q_1, the total amount of flow leaving j is exactly 1, and the value of the flow on edge $(j, (b, R))$ is at most $7 \sum_{\substack{I \in \mathcal{I}^*(j) \cap \mathcal{I}(R): \\ p \in I}} x(I, j)$. Finally, for each $(b, R) \in V_2$, the amount of flow sent on edge $((b, R), t)$ is the same as the total flow entering (b, R). Notice that since the total sum of weights $x(I, j)$ of intervals $I \in \mathcal{I}(R)$ containing any point $t \in R$ is at most 5, the total flow on edge $((b, R), t)$ is at most 35. By the integrality of flow, there is an integral flow of the same value. We will use this integral flow to schedule jobs of type Q_1, as follows. If an edge $(j, (p, R))$ carries one flow unit, then we schedule j on any interval $I \in \mathcal{I}^*(j) \cap \mathcal{I}(R)$ that contains point p. We then say that j is scheduled *inside* R. Consider now some rectangle $R \in \mathcal{R}$, and let $p \in P \cap R$. Then at most 35 jobs of Q_1 are scheduled inside R on intervals containing p. Moreover, since each interval $I \in \mathcal{I}$ has a non-empty intersection with at most two blocks, for each time point $t \in R$, at most 70 jobs are scheduled inside R on intervals containing t. This finishes the algorithm for scheduling jobs of Q_1.

We will now focus on the set $\mathcal{I}' = \mathcal{I} \setminus \mathcal{I}_1$ of intervals that do not cross block boundaries. We can now refine our definition of rectangles to intersections of blocks and rectangles. More formally, for each $R \in \mathcal{R}$, the partition $\mathcal{B}$ of B into blocks also defines a partition of R into a collection $\mathcal{C}(R, \mathcal{B})$ of rectangles. We then define a new set $\mathcal{R}' = \bigcup_{R \in \mathcal{R}} \mathcal{C}(R, \mathcal{B})$ of rectangles. The set $\mathcal{I}(R')$ of intervals assigned to $R' \in \mathcal{C}(R, \mathcal{B})$ is the set of intervals in $\mathcal{I}(R)$ that are contained in R'. We will schedule the remaining jobs inside the rectangles of $\mathcal{R}'$, such that the schedule inside each $R \in \mathcal{R}'$ uses a constant number of machines. Recall that for each $R, R' \in \mathcal{R}$, if $R \cap R' \neq \emptyset$, then either $W(R) \subseteq W(R')$ or $W(R') \subseteq W(R)$. It is easy to see that the same property holds for rectangles in $\mathcal{R}'$.

Type 2 An interval $I \in \mathcal{I}'$ is called *large* iff the length of the rectangle $R \in \mathcal{R}'$, where $I \in \mathcal{I}(R)$, is at most twice the length of I. Let $\mathcal{I}_2$ denote the set of all large intervals. We define $Q_2 = \left\{ j \in J \setminus Q_1 \mid \sum_{I \in \mathcal{I}(j) \cap \mathcal{I}_2} x(I, j) \geq 1/7 \right\}$. These jobs are scheduled similarly to type-2 jobs in [6], as follows.

Note that for every rectangle $R \in \mathcal{R}'$, $\sum_{I \in \mathcal{I}(R) \cap \mathcal{I}_2} x(I, j) \leq 10$. We can now find a matching between rectangles in $\mathcal{R}'$ and jobs in Q_2, that will define the assignment of the jobs to the rectangles. We construct a graph whose vertices are jobs in Q_2 and rectangles in $\mathcal{R}'$, and there is an edge between $j \in Q_2$ and $R \in \mathcal{R}'$ iff there is a large interval $I \in \mathcal{I}^*(j) \cap \mathcal{I}_2 \cap \mathcal{I}(R)$. As in jobs of type 1, the fractional solution gives us a fractional matching, where each job is fractionally assigned to one rectangle, and each rectangle is assigned at most 70 jobs. We can therefore find an integral assignment, where each rectangle R is assigned at most 70 jobs, and job j is assigned to R only if there is an interval $I \in \mathcal{I}(j) \cap \mathcal{I}(R)$.

Type 3 Consider an interval $I \in \mathcal{I}(j)$ for some job $j \in J \setminus (Q_1 \cup Q_2)$, and assume that $I \in \mathcal{I}(R)$ for $R \in \mathcal{R}'$. We say that I is *deadline large* iff $d_j \in R$ and $p_j > \frac{1}{2}(d_j - \ell_R)$. Let $\mathcal{I}_3$ be the set of all deadline large intervals. We define the set Q_3 of jobs of type 3 as follows: $Q_3 = \left\{ j \in J \setminus (Q_1 \cup Q_2) \mid \sum_{I \in \mathcal{I}(j) \cap \mathcal{I}_3} x(I, j) \geq 1/7 \right\}$.

For each job $j \in Q_3$, define the interval $\Gamma_j = (d_j - p_j, d_j)$. Notice that Γ_j is the right-most interval in $\mathcal{I}(j)$. We simply schedule each job $j \in Q_3$ on interval Γ_j.

Claim 5 *The total number of jobs of Q_3 scheduled at any time t is at most $70g(t)$.*

Proof: For each job $j \in Q_3$, for each rectangle $R \in \mathcal{R}'$, with $\mathcal{I}(j) \cap \mathcal{I}(R) \cap \mathcal{I}_3 \neq \emptyset$, we define a fractional value $x''_R(\Gamma_j, j)$. We will ensure that for each $j \in Q_3$, $\sum_{R \in \mathcal{R}'} x''_R(\Gamma_j, j) = 1$, and for each rectangle $R \in \mathcal{R}'$, for each $t \in \mathcal{R}'$, $\sum_{j: t \in \Gamma_j} x''_R(\Gamma_j, j) \leq 70$. Since for each point t, $|\mathcal{R}(t)| = g(t)$, the claim follows.

Consider now some fixed rectangle $R \in \mathcal{R}'$. We change the fractional schedule of intervals inside R in two steps. In the first step, for each $j \in Q_3$, we set $x'(I,j) = x(I,j) / \sum_{I \in \mathcal{I}_3 \cap \mathcal{I}(j)} x(I,j)$ for each $I \in \mathcal{I}(j) \cap \mathcal{I}(R) \cap \mathcal{I}_3$. By the definition of jobs of type 3, we now have that

$$\forall t \in R \qquad \sum_{j \in Q_3} \sum_{\substack{I \in \mathcal{I}(j) \cap \mathcal{I}(R): \\ t \in I}} x'(I,j) \leq 35 \tag{3}$$

Next, for each job $j \in Q_3$ with $\Gamma_j \subseteq R$, we set $x''_R(\Gamma_j, j) = \sum_{I \in \mathcal{I}(R)} x'(I,j)$. Notice that since $j \in Q_3$, $\sum_{R \in \mathcal{R}'} x''_R(\Gamma_j, j) = 1$. It is now enough to prove that for each time point $t \in R$, $\sum_{j \in Q_3: t \in \Gamma_j} x''_R(\Gamma_j, j) \leq 70$.

Assume otherwise. Let t be some time point, such that $\sum_{j \in Q_3: t \in \Gamma_j} x''_R(\Gamma_j, j) > 70$. Let S_t be the set of jobs $j \in Q_3$ with $t \in \Gamma_j$ and $x''_R(\Gamma_j, j) > 0$, and let $j' \in S_t$ be the job with smallest processing time. Consider the time point $t' = d_{j'} - p_{j'}$. We claim that for each $j \in S_t$, for each interval $I \in \mathcal{I}(j) \cap \mathcal{I}(R)$, either $t' \in I$ or $t \in I$. If this is true then we have that either $\sum_{j \in Q_3} \sum_{\substack{I \in \mathcal{I}(j) \cap \mathcal{I}(R): \\ t \in I}} x'(I,j) > 35$ or $\sum_{j \in Q_3} \sum_{\substack{I \in \mathcal{I}(j) \cap \mathcal{I}(R): \\ t' \in I}} x'(I,j) > 35$, contradicting (3).

Consider some job $j \in S_t$ and assume for contradiction that there is some time interval $I \in \mathcal{I}(j) \cap \mathcal{I}(R)$ that contains neither t nor t'. Then I must lie completely to the left of t' and hence to the left of $\Gamma_{j'}$. But since $p_j \geq p_{j'}$, we have that $t' - \ell_R \geq p_j \geq p_{j'}$, and so $d_{j'} - \ell_R \geq 2p_{j'}$, contradicting the fact that $j' \in S_t$. □

Type 4 Same as type 3, but for release date instead of deadline. Is treated similarly to Type 3. The set of type 4 jobs is denoted by Q_4.

Type 5 Consider some interval $I \in \mathcal{I}(j)$ for $j \in J \setminus (Q_1 \cup \cdots \cup Q_4)$, and assume that $I \in \mathcal{I}(R)$ for $R \in \mathcal{R}'$. We say that I is of type 5 ($I \in \mathcal{I}_5$) iff $d_j \in R$ and $I \notin \mathcal{I}_3$ (so $d_j - \ell_R \geq 2p_j$). We define the set Q_5 of jobs of type 5 as follows: $Q_5 = \left\{ j \in J \setminus (Q_1 \cup \cdots \cup Q_4) \mid \sum_{I \in \mathcal{I}(j) \cap \mathcal{I}_5} x(I,j) \geq 1/7 \right\}$.

For a job $j \in Q_5$ and a rectangle $R \in \mathcal{R}'$, we say that R is *admissible* for j iff $d_j \in R$ and $d_j - \ell_r \geq 2p_j$. We say that an interval $I \in \mathcal{I}(j)$ is admissible for j iff $I \in \mathcal{I}_5$. Notice that if $j \in Q_5$ then the sum of values $x(I,j)$ where I is admissible for j is at least $1/7$. Let $R \in \mathcal{R}'$ be any rectangle, and let $S \subseteq Q_5$ be any subset of jobs of type 5. We say that set S is *feasible* for R iff R is admissible for each $j \in S$, and, for each time point $t \in R$, $\sum_{j \in S: d_j \leq t} p_j < 70(t - \ell_R)$. We now proceed as follows. First we show that if S is feasible for R, then we can schedule all jobs of S inside R on at most 140 machines. After that we show how to assign all jobs of Q_5 to rectangles such that each rectangle is assigned a feasible subset. We start with the following lemma.

Lemma 1 *If $S \subseteq Q_5$ is a feasible subset of jobs for R then all jobs in S can be scheduled inside R on at most 140 machines.*

Proof: We will schedule all jobs of S on 140 machines inside the time interval $W(R)$. We scan all 140 machines simultaneously from left to right starting from time point ℓ_R. Whenever any machine becomes idle, we schedule on it the job with earliest deadline among all available jobs of S. It is easy to see that all jobs are scheduled: Assume otherwise, and let j be the first job that we are unable to schedule. Consider the time point $t = d_j - p_j$. All the machines are occupied at time t, and they only contain jobs whose deadline is before d_j. Therefore, $\sum_{j' \in S: d_{j'} < d_j} p_{j'} \geq 140(t - \ell_R)$. But since $d_j - \ell_R \geq 2p_j$, we have that $t - \ell_R = d_j - p_j - \ell_R \geq \frac{1}{2}(d_j - \ell_R)$, and so $\sum_{j' \in S: d_{j'} < d_j} p_{j'} \geq 70(d_j - \ell_R)$, contradicting the fact that S is feasible for R. $\square$

We now show how to assign jobs of Q_5 to rectangles, such that each rectangle is assigned a feasible subset. Consider some block $B' \in \mathcal{B}$. Let $\mathcal{R}(B') \subseteq \mathcal{R}'$ be the set of rectangles contained in B', and let $H(B') \subseteq Q_5$ be the subset of jobs of type 5 whose deadline is inside B'. We will assign jobs in $H(B')$ to rectangles in $\mathcal{R}(B')$. Recall the partition of the set $\mathcal{R}(B')$ of rectangles into layers. Layer i, denoted by L_i, consists of all rectangles that are not contained in any other rectangle of $\mathcal{R}(B') \setminus (L_1 \cup \cdots \cup L_{i-1})$ (if we have identical rectangles then at most one of them is assigned to each layer and we break the ties arbitrarily). Consider some job $j \in H(B')$. Let $z(j)$ be the maximum index i, such that some rectangle $R \in L_i$ is admissible for j. Then for each $z : 1 \leq z \leq z(j)$, there is a unique layer-z rectangle $R_z(j)$ that is admissible for j.

We will assign a subset $A(R)$ of jobs to each rectangle $R \in \mathcal{R}(B')$. We start with $A(R) = \emptyset$ for all R. We process jobs of $H(B')$ in non-decreasing order of their deadlines. When job j is processed, it is assigned to $R_z(j)$, where z is the maximum index, $1 \leq z \leq z(j)$, such that $A(R) \cup \{j\}$ is feasible for R. It now only remains to prove is that every job j can be assigned to a rectangle. The next lemma will finish the analysis of the algorithm for type-5 jobs.

Lemma 2 *For each job $j \in H(B')$, when j is processed, there is a rectangle $R_z(j)$, $1 \leq z \leq z(j)$, such that j can be assigned to $R_z(j)$.*

Proof: Assume otherwise, and let j be the first job that cannot be assigned to any such rectangle. We now proceed as follows. We construct a subset $\tilde{\mathcal{R}} \subseteq \mathcal{R}(B')$ of rectangles, and for each $R \in \tilde{\mathcal{R}}$ we define a time point $t_R \in R$. For each $R \in \tilde{\mathcal{R}}$, we define a subset $\tilde{J}(R) \subseteq A(R)$ of jobs whose deadline is before t_R and show that the total processing time of jobs in $\tilde{J}(R)$ is more than $35(t_R - \ell_R)$. On the other hand, we ensure that for each $j \in \bigcup_{R \in \tilde{\mathcal{R}}} \tilde{J}(R)$, for each admissible interval I for j, if $I \in \mathcal{I}(R)$, then $R \in \tilde{\mathcal{R}}$ and $I \subseteq [\ell_R, t_R]$. This leads to a contradiction, since for each $j \in \tilde{J}$, at least 1/7 of the LP weight is on admissible intervals, and all such intervals are contained in the intervals $[\ell_R, t_R]$ for $R \in \tilde{\mathcal{R}}$. On the other hand, for each rectangle $R \in \tilde{\mathcal{R}}$, for each time point $t \in R$, the total LP-weight of intervals of R containing t is at most 5.

Let $R \in \mathcal{R}(B')$ be any rectangle, and let $t \in R$. We say that R is *overpacked for t* iff $\sum_{j' \in A(R): d_{j'} \leq t} p_{j'} > 35(t - \ell_R)$. We process the rectangles layer-by-layer. At the beginning, we set $\tilde{\mathcal{R}} = \emptyset$ and $\tilde{J} = \emptyset$. In the first iteration, we consider the rectangles of layer L_1. Let $R = R_1(j)$. We add R to $\tilde{\mathcal{R}}$ and set $t_R = d_j$. Note that since j could not be assigned to R, rectangle R must be overpacked for t_R. We add to $\tilde{J}$ all jobs in $A(R) \cup \{j\}$.

In iteration i, we consider rectangles $R \in L_i$. Consider the set $Y(R)$ of jobs j' for which $z(j') \geq i$ and $R_i(j') = R$. If $\tilde{J} \cap Y(R)$ is non-empty, we add R to $\tilde{\mathcal{R}}$, and set t_R to be the maximum deadline of any job $j' \in \tilde{J} \cap Y(R)$. Notice that since j' was not assigned to R, rectangle R is overpacked for t_R. Let $\tilde{J}(R)$ be the set of all jobs $j'' \in A(R)$ with $d_{j''} \leq t_R$. We add jobs in $\tilde{J}(R)$ to $\tilde{J}$.

Consider the final set $\tilde{\mathcal{R}}$ of rectangles and the set $\tilde{J}$ of jobs. Clearly, the set $\tilde{J}$ of jobs is the disjoint union of sets $\tilde{J}(R)$ for $R \in \tilde{\mathcal{R}}$. Recall that $\tilde{J}(R)$ contains all jobs $j' \in A(R)$ with $d_{j'} \leq t_R$. Since each rectangle $R \in \tilde{\mathcal{R}}$ is overpacked for t_R, we have that $\sum_{j \in \tilde{J}} p_j > 35 \sum_{R \in \tilde{\mathcal{R}}} (t_R - \ell_R)$. On the other hand, the next claim shows that for each job $j \in \tilde{J}$, for each admissible interval I of j, if $I \in \mathcal{I}(R)$, then $R \in \tilde{\mathcal{R}}$ and I lies to the left of t_R.

Claim 6 *Let $j \in \tilde{J}$, let I be any admissible interval for j, and assume that $I \in \mathcal{I}(R)$. Then $R \in \tilde{R}$, and $I \subseteq [\ell_R, t_R]$.*

We now obtain a contradiction as follows. We have shown that $\sum_{j\in\tilde{J}} p_j > 35\sum_{R\in\tilde{\mathcal{R}}}(t_R - \ell_R)$. On the other hand, for each job $j \in \tilde{J}$, at least $1/7$ of its LP-weight lies on admissible intervals. Since all these admissible intervals are contained inside intervals $[\ell_R, t_R]$ for $R \in \tilde{\mathcal{R}}$, we have that $\sum_{R\in\tilde{\mathcal{R}}}\sum_j \sum_{\substack{I\in\mathcal{I}(j)\cap\mathcal{I}(R):\\ I\subseteq[\ell_R,t_R]}} x(I,j) \geq \frac{1}{7}\sum_{j\in\tilde{J}} p_j > 5\sum_{R\in\tilde{\mathcal{R}}}(t_R - \ell_R)$. This contradicts the fact that for every rectangle $R \in \mathcal{R}'$, for each $t \in R$, $\sum_j \sum_{I\in\mathcal{I}(j)\cap\mathcal{I}(R):t\in I} x(I,j) \leq 5$. It now only remains to prove Claim 6.

Proof: [Of Claim 6] Consider some job $j' \in \tilde{J}$, and suppose it was added to $\tilde{J}$ in iteration i. Let I be any admissible interval of j'. Then there must be an index $z : 1 \leq z \leq z(j')$ such that $I \in \mathcal{I}(R_z(j))$. We now consider three cases. First, if $z = i$, then let $R = R_i(j')$. Then, since j' was added to $\tilde{J}$ in iteration i, $j' \in \tilde{J}(R)$ and so $I \subseteq [\ell_R, t_R]$. Clearly, $R \in \tilde{\mathcal{R}}$. Assume now that $z > i$ and let $R = R_z(j')$. Then $j' \in \tilde{J}$ in iteration z, and so when R was considered, $j' \in Y(R) \cap \tilde{J}$. So R has been added to $\tilde{\mathcal{R}}$ and t_R has been set to be at least $d_{j'}$. Finally, assume that $z < i$. Let $R = R_i(j')$ and $R' = R_z(j')$. Then $R \subseteq R'$. It is then enough to prove the following claim:

Claim 7 *Let $R \in L_i$ and $R' \in L_{i-1}$, with $R \subseteq R'$. Assume that $R \in \tilde{R}$. Then $R' \in \tilde{R}$, and moreover $t_{R'} \geq t_R$.*

Proof: Consider the iteration i when R was added to $\tilde{\mathcal{R}}$, and let $j'' \in Y(R)$ be the job which determined t_R, so $t_R = d_{j''}$. Two cases are possible. If $j'' \in A(R')$, then j'' has been added to $\tilde{J}$ in iteration $i-1$ when R' was processed. So $R' \in \tilde{\mathcal{R}}$ and $t_{R'} \geq d_{j''} = t_R$. Otherwise, j'' was in $\tilde{J}$ when R' was processed. Since $R \subseteq R'$ and R is admissible for j'', so is R'. Therefore, $j'' \in Y(R') \cap \tilde{J}$ and so $R' \in \tilde{\mathcal{R}}$ and $t_{R'} \geq d_{j''} = t_R$. □

□

□

Type 6 Like type 5, but for release date.

Type 7 All other jobs. The algorithm for these jobs is the same as the one used in [6], substituting rectangles for blocks. For completeness, we go over the algorithm.

Let $\mathcal{I}_7 = \mathcal{I} \setminus (\mathcal{I}_1 \cup \cdots \mathcal{I}_6)$. Notice that if $I \in \mathcal{I}_7$, it does not cross any block boundaries. Moreover if $I \in \mathcal{I}(j) \cap \mathcal{I}(R)$ for some $j \in J$ and $R \in \mathcal{R}'$, then R does not contain d_j or r_j, and the length of rectangle R is at least $2p_j$. Let $Q_7 = J \setminus (Q_1 \cup \cdots \cup Q_6)$. Then, we have:

$$Q_7 = \left\{ j \mid \sum_{I\in\mathcal{I}(j)\cap\mathcal{I}_7} x(I,j) \geq 1/7 \right\}.$$

We divide Q_7 into classes based on size. Let H_i be the set of jobs $j \in Q_7$ with $2^{i+1} < p_j \leq 2^i$. For each rectangle R, let $X(R,i) = \sum_{j\in H_i}\sum_{I\in\mathcal{I}(j)\cap\mathcal{I}(R)\cap\mathcal{I}_7} x(I,j)$. For each $R \in \mathcal{R}'$, we will schedule at most $\lceil X(R,i)\rceil$ jobs from H_i (simultaneously, for all i). Notice that for $j \in Q_7$, if $\mathcal{I}(j)\cap\mathcal{I}(R)\cap\mathcal{I}_7 \neq \emptyset$, then j can be scheduled anywhere inside rectangle R, since R does not contain d_j or r_j.

Claim 8 *For each rectangle $R \in \mathcal{R}'$, we can schedule $\lceil X(R,i)\rceil$ jobs of H_i inside R on 142 machines, simultaneously for all i.*

Proof: First note that we can schedule $\lfloor X(R,i)\rfloor$ jobs on 70 machines, allowing to break a job and schedule parts of it on different machines. Next, note that because jobs are small, we can schedule

$\lceil X(R,i) \rceil$ jobs on 71 machines. We need to double the number of machines to schedule the jobs that were broken. □

Now for each size class H_i, we need to decide which job is scheduled in which rectangle, so that each rectangle gets assigned at most $\lceil X(R,i) \rceil$ jobs. The fractional LP solution implies a feasible assignment exists, and earliest deadline greedy assignment will give an integral solution.

4 Conclusion

We have shown a polynomial time constant factor approximation algorithm for the Continuous Machine Minimization problem. This improves upon the best previously known bound of $O(\sqrt{\log n})$ [6], while a lower bound of $\Omega(\log \log n)$ is known for the discrete version of the problem [7]. Hence our result proves a separation between the discrete and the continuous versions. For the discrete version, the best known approximation factor is $O(\log n / \log \log n)$ [8]. Closing this gap remains an interesting open problem.

References

[1] Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-ptas for unsplittable flow on line graphs. In *STOC '06: Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing*, pages 721–729, New York, NY, USA, 2006. ACM.

[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*, pages 622–631, 1999.

[3] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

[4] Gruia Calinescu, Amit Chakrabarti, Howard Karloff, and Yuval Rabani. Improved approximation algorithms for resource allocation. In *In 9th International Integer Programming and Combinatorial Optimization Conference, volume 2337 of LNCS*, pages 401–414. Springer-Verlag, 2002.

[5] J. Chuzhoy, R. Ostrovsky, and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Math. Oper. Res.* 31, 4, pp. 730–738, 2006.

[6] Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *FOCS*, pages 81–90, 2004.

[7] Julia Chuzhoy and Joseph (Seffi) Naor. New hardness results for congestion minimization and machine scheduling. *J. ACM*, 53(5):707–721, 2006.

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

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