Approximation Algorithms for Machine Minimization

Mohsen Mohammadi
Department of Computing Science, University of Alberta, Edmonton, Canada

Mohammad R. Salavatipour
Department of Computing Science, University of Alberta, Edmonton, Canada

Abstract

In this paper we consider a classic scheduling problem known as Machine Minimization (MM). The input to MM is a set J of n jobs, where each job j has processing time p_j, release time r_j, and deadline d_j. The goal is to schedule the jobs to run non-preemptively on minimum number of machines such that each job is fully scheduled within its $[r_j, d_j]$ interval and each machine runs at most one job at a time. This problem generalizes several NP-hard problems (e.g. the case of identical release time and deadlines reduces to the bin packing problem). Using Randomized Rounding [24] one can get an $O(\frac{\log n}{\log \log n})$-approximation. Chuzhoy et al. [11] presented an algorithm that uses $O(\text{OPT}^2)$ machines (i.e. $O(\text{OPT})$-approximation). Combined with the earlier work this yields an $O(\sqrt{\frac{\log n}{\log \log n}})$-approximation and this remains the best known result for over 20 years. Even for when the ratio of largest to smallest processing time $p_{\max}/p_{\min}$ is bounded, or the number of distinct processing times are bounded, there is no (universal) constant approximation. In this paper, we present a number of results. When $p_{\max}/p_{\min} = \rho$ we present an algorithm that yields an asymptotic $(2+\epsilon)$-approximation in time $n^{O(\rho^4/\epsilon^4)}$. When the number of distinct processing times is c, we present a 2-approximation with run time $n^{O(c^2 \log^3 n)}$. If we have c distinct processing times and $p_{\max}/p_{\min} = \rho$ we present an asymptotic $(1+\epsilon)$-approximation that runs in time $n^{O(c^2 \cdot \rho^2 \cdot \log^3 n/\epsilon^2)}$.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Machine minimization, approximation algorithms, scheduling

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.136

Funding *Mohammad R. Salavatipour*: Supported by NSERC.

1 Introduction

Scheduling remains a cornerstone of combinatorial optimization and operations research and problems in this field have been studied over the last several decades due to their wide applications. Despite this, there are still several fundamental problems in this area that are not resolved or fully understood. One such class of scheduling problems are those for scheduling of jobs with release times and deadlines in order to optimize some objective function. These problems include Machine Minimization, Throughput Maximization, Flow time, and the weighted variants of them [12, 23, 22, 21]. In this paper we consider the Machine Minimization (MM) which is a classic scheduling problem and present several approximation algorithms for it. In MM we are given a set J of n jobs, where each job j has a release time r_j, deadline d_j, and processing time p_j. We would like to schedule these jobs non-preemptively on minimum number of machines such that each machine runs at most one job at any given time and each job is completely run between its release time and deadline. Note that this problem generalizes many classic problems. For instance, when all release time and deadlines are identical, this reduces to the bin packing problem. So the problem is NP-hard and in fact NP-hard to approximate within a factor better than $\frac{3}{2}$. The MM problem defined above is also referred to as the *continuous* version as there is a *discrete* version of it defined below. In the *discrete* version of MM each job j is given with a collection

of intervals $\mathcal{I}(j)$ (instead of just an interval $[r_j, d_j]$) and in any feasible schedule j can be scheduled in any of these intervals. The discrete version of MM seems to be significantly more difficult than the continuous version. Chuzhoy and Naor [13] proved that discrete MM is $\Omega(\log\log n)$-hard to approximate. From now on by MM we mean the continuous version and will explicitly use the term discrete MM to refer to the discrete version of it.

The classic randomized rounding approach of Raghavan and Thompson [24] implies an $O(\log n/\log\log n)$-approximation for MM (for both the continuous and discrete version). In fact, one can see that it generates a solution of value at most $\text{OPT} + O(\log n/\log\log n)$, where OPT is the value of the optimum solution. This can be improved to $\lceil(1+\epsilon)\text{OPT}\rceil$ if $\text{OPT} = \Omega(\log n/(\epsilon\log\log n))$, which yields an asymptotic PTAS. Chuzhoy et al. [12] presented an $O(\text{OPT})$-approximation for MM. This, combined with the above randomized rounding algorithm, yields an $O(\sqrt{\log n/\log\log n})$-approximation and this has remained the best approximation so far. In [11] Chuzhoy and Codenotti presented an algorithm with a claimed approximation ratio of $O(1)$ for MM but unfortunately, the proof is wrong [10]. Im et al. [23] presented a DP based framework for several scheduling problems. Their results include a quasi-polynomial $(1+\epsilon, 2)$ bicriteria approximation for MM, i.e. an algorithm with time $2^{\text{polylog}(n, \frac{1}{\epsilon})}$ that finds a 2-approximate solution using machines that have $(1+\epsilon)$ speed up. They also presented a $(2+\epsilon, 1)$ bicriteria approximation with the same time complexity, i.e. an algorithm that finds a solution using machines that are $2+\epsilon$ faster but the solution uses OPT machines where OPT is the optimum value for the solution using normal speed machines. In this paper, we present three results for when the number of distinct processing times and/or the ratio of the largest to smallest processing times are bounded. Although these results seem more restricted, they in fact imply the above mentioned bicriteria approximations as explained below. Our first (and perhaps the main) result is an asymptotic $(2+\epsilon)$-approximation for when the ratio of the largest to smallest processing time, i.e. $p_{\max}/p_{\min}$ is bounded.

▶ **Theorem 1.** *Given an instance I for machine minimization with $p_{\max}/p_{\min} \leq \rho$ and an $\epsilon > 0$, there is an algorithm with time $n^{O(\rho^4/\epsilon^4)}$ that can produce a solution using $\lceil 2(1+\epsilon)\text{OPT}\rceil$ machines where OPT is the value of optimum.*

Note that the previous best known approximation for such case is a $6 \cdot (p_{\max}/p_{\min})$-approximation [31]. So we give first the (universal) constant factor approximation for when $p_{\max}/p_{\min}$ is bounded. This result is obtained by presenting an asymptotic PTAS for two separate cases: when all jobs are large (compared to their span) or when all jobs are small. For the first case we show there is a near optimum solution with certain structures and how one can find an optimum structured solution using Dynamic Programming. For small jobs we show existence of near optimum solutions with other structures. Then we show how to find an approximation for a structured optimum solution using LP rounding.

Our next two theorems are for when the number of distinct processing times (and $p_{\max}/p_{\min}$) are bounded.

▶ **Theorem 2.** *Given an instance I for machine minimization with c distinct processing times, we can produce a 2-approximate solution for it in time $n^{O(c^2\log^3 n)}$.*

An immediate corollary is the following, which recovers the bicriteria approximations for MM obtained by Im et al. [23].

▶ **Corollary 3.** *There is a $(1+\epsilon, 2)$ bicriteria approximation for MM and there is a $(2+\epsilon, 1)$ bicriteria approximation for MM, both running in time $n^{O(\log(1/\epsilon)\cdot\log^4 n)}$. These guarantees apply to arbitrary instances; no constant bound on the number of distinct processing times is assumed.*

▶ **Theorem 4.** *Given an instance I for machine minimization with c distinct processing times and $p_{\max}/p_{\min} \leq \rho$, for any $\epsilon > 0$ there is an asymptotic $(1+\epsilon)$-approximation algorithm with running time $n^{O(c^2 \cdot \rho^2 \cdot \log^3 n/\epsilon^2)}$.*

Note that using Theorem 4 when $p_{\max}/p_{\min} \in \text{polylog}(n)$, using $(1+\epsilon)$-speed up machines we get bicriteria $(1+\epsilon, 1+\epsilon)$ approximation. These two theorems use hierarchical Dynamic Programming similar to the bicriteria approximations developed in [23]. For Theorem 2, we show that any solution on M machines can be converted into a *pseudo-feasible* solution, in which a machine may run two jobs of the same processing time in parallel, for a modified instance whose release times and deadlines are trimmed by a recursive (hierarchical) procedure; an optimum pseudo-feasible solution can be found by a DP over this hierarchical decomposition and turned into a feasible schedule on $2M$ machines. For Theorem 4, we use a similar DP with a refined notion of pseudo-feasibility, where each machine may additionally run a small fraction of one other job of the same processing time in parallel; these fractional pieces are then moved to $\lceil 2\epsilon \cdot M \rceil$ extra machines.

1.1 Related Works

For MM, Garey and Johnson showed that deciding whether all the jobs can be scheduled on one machine or not is NP-hard [20, 19]. For restricted cases on one machine, Elffers et al. showed that even with only two processing times greater than one, the problem remains NP-hard [18]. On the positive side, when all jobs have the same processing time, the problem is solvable in polynomial time [27, 28].

For approximation results (excluding [12, 10]), several works studied the problem under additional assumptions. Cieliebak et al. [15] gave multiple approximation guarantees, including $9 \cdot (p_{\max}/p_{\min})$ for bounded aspect ratio, 4.62-approximation for common release time, and 14.9-approximation when all spans contain a common point. Later, [31] presented greedy approaches yielding a $6 \cdot (p_{\max}/p_{\min})$ approximation for bounded processing-time ratio and a 2-approximation for common release time/deadline.

The online variant has also been studied, where jobs arrive over time and must be scheduled before their deadline. Saha [25] introduced this model and gave a $\log(p_{\max}/p_{\min})$ approximation (competitive ratio), and also showed some lower bounds. (see also [16]).

Related generalizations have also been studied. In the rent minimization variant, instead of minimizing the number of machines, the objective is to minimize the number of machine rentals of duration T. This problem generalizes machine minimization, and the first approximation result is due to Saha [25], who gave an $o(\alpha)$-approximation, where α is the approximation factor for machine minimization. For the unit-processing-time case, Bender et al. [6] gave a 2-approximation, later improved to a PTAS by Chen et al. [8]. Whether there exist any polynomial-time algorithms for this variant is still an open question. Saha [25] also gave a tight $\log(p_{\max}/p_{\min})$ approximation algorithm for the online rent minimization problem. For when p_j's are 1, [9, 30] gave $3e+7$ and 6 approximations, respectively.

Throughput maximization is a related objective: with a fixed number of machines, maximize the number of scheduled jobs. Spieksma [29] gave early approximation results, including a greedy 1/2-approximation (schedule the shortest available job when a machine becomes idle) and an LP gap of 2. Bar-Noy et al. [5] further analyzed greedy rules and obtained ratio $1 - 1/(1+1/m)^m$ for m identical machines. Chuzhoy et al. [14] considered the discrete variant, where each job j has an explicit set $\mathcal{I}_j$ of feasible intervals and must be assigned to one of them. Although related to the continuous model, neither formulation directly reduces to the other in polynomial time, since enumerating all length-p_j subintervals

of $[r_j, d_j]$ can be pseudo-polynomial. They gave a $(1-1/e-\epsilon)$-approximation for the discrete case [14]. Chuzhoy et al. [14] also gave a pseudo-polynomial exact algorithm with running time $O(n^{\text{poly}(k)}T^4)$, where $k = \max_j (d_j - r_j)/p_j$ and $T = \max_j d_j$. For the weighted version, [2] showed polynomial-time solvability for $m = 1$ when all jobs have equal processing time. For $m = O(1)$ and uniform processing times, polynomial-time algorithms were given in [3, 17]. For general processing times, [7, 4] gave 2-approximation algorithms, which remained the best known for this weighted setting. Im et al. [22] improved throughput approximations for all m: for unweighted throughput they gave an $(\alpha_0 - \epsilon)$-approximation (for some absolute $\alpha_0 > 1-1/e$) in time $n^{O(m/\epsilon^5)}$ for $m = O(1)$, and also a $1-O(\sqrt{(\log m)/m}-\epsilon)$ approximation that tends to 1 as m grows. More recently, Armbruster et al. [1] improved this to $1.551 + \epsilon$. Hyat et al. [21] gave a PTAS for the throughput maximization problem when number of different processing times is constant.

Organization

We start with some preliminaries in the next section. We prove Theorem 1 in Section 3. Some parts of proof of Theorem 2 is given in Appendix A, and the full details and the proof of Theorem 4 are deferred to the full version of the paper.

2 Preliminaries

Recall that we have a set J of n jobs where each job $j \in J$ has a processing time p_j, a release time r_j as well as a deadline d_j, which we assume all integers in the range $[0, T]$ (we can think of T as the largest deadline, or $d_{\max}$). For each job $j \in J$ we refer to $[r_j, d_j]$ as span of job j, denoted by span_j and $|\text{span}_j| = d_j - r_j$. The jobs are to be scheduled non-preemptively on minimum number of machines which can process only one job at a time within their span. We use OPT to denote an optimum schedule and M^* the value of it (i.e. minimum number of machines on which we can have a feasible schedule of all the jobs). For an instance J, let $r_{\min} = \min_{j\in J} r_j$, $d_{\max} = \max_{j\in J} d_j$, $p_{\min} = \min_{j\in J} p_j$, and $p_{\max} = \max_{j\in J} p_j$.

Note that in any schedule, we can shift-left the jobs so that each job either starts at its release time or immediately after a job has been finished on the same machine. Therefore, we can partition the jobs in any schedule into continuous segments of jobs being run, whose leftmost points are release times and the jobs in each segment are being run back to back. We call such a maximal continuous segment of jobs a *block* in a schedule. So if there are c different processing times among the jobs then there are at most n^{c+1} different possible start times for the jobs in a left-shifted solution. To see this, we upper bound the number of distinct $r_i + \sum_{j\in J'} p_j$ values for any subset $J' \subseteq J$. First note that there are at most n different r_i values. Also, for each set $J' \subseteq J$, the sum $\sum_{j\in J'} p_j$ can have at most n^c possible values as the number of jobs in J' with a specific processing time can be at most n and we assumed there are only c distinct processing times.

We can assume $T \le 2n^2 \cdot p_{\max}$ by the following argument. First we can assume any time t belongs to at least one span or else the timeline can be divided into independent instances. Given any instnace J, if $j_{\max}$ is the job with largest span we can assume $\text{span}_{j_{\max}} \le 2n \cdot p_{\max}$. Else in any feasible solution to $J - j_{\max}$, there is at most $n \cdot p_{\max}$ busy time on any machine in $\text{span}_{j_{\max}}$, so there is enough idle time and we can repeat this argument. Thus $T \le n \cdot \text{span}_{j_{\max}}$.

For any time point $t \in [0, T]$, we use J_t to denote the set of jobs whose spans cross t, i.e. $J_t := \{j \in J : t \in [r_j, d_j]\}$. For any two times $t_1 \le t_2$, we use $J[t_1, t_2]$ to denote the set of jobs whose spans are fully contained in $[t_1, t_2]$, i.e. $J[t_1, t_2] := \{j \in J : r_j \ge t_1,\ d_j \le t_2\}$.

We often use M^* to denote the value of an optimum solution (i.e. minimum number of machines in a feasible solution). Many of our approximation algorithms take a parameter M and either produce a solution on M machines or say there is no solution on $\leq M$ machines. We can use a binary search and find the smallest value of $M \geq M^*$ for which such algorithm returns a feasible solution. As mentioned earlier, careful analysis of the known results [24] imply that if $M^* \in \Omega(\frac{1}{\epsilon}\log n/\log\log n)$ is sufficiently large, then a simple randomized rounding yields a PTAS for MM. So from now on we assume that $M^* \in O(\frac{1}{\epsilon}\log n/\log\log n)$. Under this assumption, for any fixed time point t, we can guess the set of jobs running at time t in time n^{M^*} (i.e. quasi-polynomial).

3 Asymptotic $(2+\epsilon)$-Approximation for Bounded $p_{\max}/p_{\min}$

In this section, we prove Theorem 1. Our objective is to compute, in polynomial time (for a sufficiently small fixed $\epsilon > 0$), a feasible schedule that uses at most $\lceil(2+\epsilon)M^*\rceil$ machines. As mentioned earlier we assume $M^* \in O(\frac{\log n}{\log\log n})$. Given $\delta \in (0,1)$, we classify the jobs as small (or δ-small) and large (or δ-large) as follows. We say a job j is δ-small if $p_j \leq \delta \cdot |\text{span}_j|$, otherwise it is called δ-large. Let J_S, J_L be the subsets of δ-small and δ-large jobs, respectively (so $J = J_S \cup J_L$). We present different asymptotic PTASs for each of J_S and J_L, separately. The union of the schedules for these two sets gives the desired approximation for J. In fact we can ensure that the total number of extra machines used for both J_S and J_L is at most $\lceil\epsilon \cdot M^*\rceil$. We will use $\delta = \epsilon^2/(100\rho)$, where $\rho = p_{\max}/p_{\min}$, in this section and we simply call the jobs small and large (instead δ-small or δ-large). To get an asymptotic PTAS for large jobs we first show there is a structured near optimum solution. Then we show how to find an optimum structured solution using a DP. For small jobs we prove the existence of a near optimum solution with a different structure. We then show how we can find an approximately optimum structured solution using LP rounding.

3.1 Asymptotic PTAS for Large Jobs

In this section we show that for J_L if the optimum solution uses M_L^* machines, then we can give an algorithm that produces a schedule using $\lceil(1+\delta)M_L^*\rceil$ machines (recall that $\delta = \epsilon^2/(100\rho)$). In order to get this asymptotic PTAS for the large jobs we first show the existence of a near optimum solution that has certain structures. The near optimum structured solution has the property that the jobs in the schedule appear in sequences of back-to-back running jobs with no idle time in between jobs within a sequence, where the length of each sequence is at most $p_{\max}/\delta^2$, and the start of each of these sequences is either the release time of a job, or is a multiple of $p_{\min}$. In order to find an optimum solution with such structure, we use dynamic programming. Our approach relies on the fact that, when all the jobs are large and $p_{\max}/p_{\min} = \rho$, for any time point t, the set of jobs whose span contains time t has size $O(\rho \cdot M_L^*/\delta)$.

▶ **Lemma 5.** *Consider an instance J_L of large jobs that is scheduled on M machines. For any time point t, the number of jobs whose span contains t is at most $\frac{2\rho}{\delta}M$.*

Proof. Fix any time point t, and recall $J_t := \{j \in I_L : t \in [r_j, d_j]\}$. Let $r = \min_{j\in J_t} r_j$ and $d = \max_{j\in J_t} d_j$. Since each $j \in J_t$ is δ-large, we have $\delta \cdot |\text{span}_j| < p_j \leq p_{\max}$, hence $|\text{span}_j| \leq p_{\max}/\delta$. Together with $t \in [r_j, d_j]$, this implies $r_j \geq t - p_{\max}/\delta$ and $d_j \leq t + p_{\max}/\delta$, so $d - r \leq \frac{2p_{\max}}{\delta}$. All the jobs in J_t must be processed inside $[r, d]$, therefore

$$\sum_{j\in J_t} p_j \leq M \cdot (d-r) \leq \frac{2p_{\max}}{\delta}M.$$

Also $p_j \geq p_{\min}$ for all the jobs, so

$$p_{\min} \cdot |J_t| \leq \sum_{j \in J_t} p_j \leq \frac{2p_{\max}}{\delta} \cdot M \implies |J_t| \leq \frac{2}{\delta} \cdot \frac{p_{\max}}{p_{\min}} \cdot M \leq \frac{2\rho}{\delta} M,$$

where we used $p_{\max}/p_{\min} \leq \rho$. ◀

3.1.1 Near Optimal Well Structured Solution for Large Jobs

For any feasible schedule for J_L, without loss of generality, we can assume it is left-shifted. Therefore, for each job, if we know the first job in its block and all the jobs before it in the block, then we can compute its exact position.

▶ **Definition 6** (Position-Fixed Job). *A job j is called* position fixed *if its start time is one of the following: 1) its release time r_j; or 2) $d \cdot p_{\min}$ for some integer d.*

We say a schedule S is *well structured* if each job is either position fixed, or is scheduled back-to-back after a position-fixed job whose start time differs by at most $\hat{P} = p_{\max}/\delta^2$. In other words, in the left-shifted solution, we want the start of each block be either a release time, or a multiple of $p_{\min}$ and the length of a block is at most $p_{\max}/\delta^2$.

▶ **Theorem 7.** *Consider any feasible solution for δ-large jobs on M machines. Then there exists a well structured solution on $\lceil (1+\delta)M \rceil$ machines.*

Proof. Consider any feasible solution S on M machines indexed 1 to M. Let $\sigma = \lfloor 1/\delta \rfloor$ and we group the machines into bundles of size σ: bundle i contains machines with index $(i-1) \cdot \sigma + \alpha$, for $\alpha \in [1, \sigma]$. We show how for each bundle i, we can schedule the jobs currently scheduled on those machines in S on $\sigma + 1$ machines s.t. the schedule is well structured. So we use 1 extra machine for each σ machines and this implies the total extra machine bound of $\delta \cdot M$. So let's focus on one bundle, say bundle 1.

Now let the time horizon be divided into consecutive windows of length $\tilde{P} = \lfloor p_{\max}/\delta \rfloor$, where window W_i starts at time $(i-1) \cdot \tilde{P}$ to $i \cdot \tilde{P}$; let these windows be $W_1, W_2, \ldots, W_h$, where $h = \lceil |T|/\tilde{P} \rceil$. Our goal is to remove (at most) one job from each machine in every σ consecutive windows (and shift-left the subsequent jobs) so that at least one job becomes position fixed (or else the window has no job that has started in that window). We do the following for $1 \leq i \leq \sigma$. We look at the windows with indices $W_i, W_{\sigma+i}, W_{2\sigma+i}, W_{3\sigma+i}, \ldots$, remove the first job that is started in these windows on machine i (if there are any), and schedule each of those jobs in the same window and same time on the auxiliary machine, and then left-shift it to be position fixed. Note that this removal of a job (if exists) ensures that any block of back to back jobs of length at least $p_{\max}/\delta^2$ on any machine is "chopped" (by removing a job). Also, in the left-shift operation, these jobs that are removed (and scheduled on the auxiliary machine) move at most $p_{\min}$ and because the size of a window is sufficiently large (compared to $p_{\min}$) these jobs will not overlap on the auxiliary machine. Observe that by these "chopping" operation, we cannot have a block of back to back jobs of length longer than $p_{\max}/\delta^2$ on any machine. We also left-shift all the jobs in all the windows (after removing the first job) on machine i so that the first job in that window is now position fixed. So for any scheduled job at a time t on a machine, within the time of $[t - \hat{P}, t]$ there is at least one position fixed job on that machine. This means we have well-structured solution. ◀

3.1.2 DP For Finding A Well Structured Solution

Next we explain how to find a well structured solution on $\lceil (1+\delta)M_L^* \rceil$ machines for J_L. Recall that although we don't know M_L^*, we can use binary search to find the smallest value $M \geq M_L^*$ for which we can find a well structured solution on M machines. So in the remaining we describe a DP to check if we can find a well structured solution on M machines.

Note that in a well-structured solution, for any job scheduled at a time t on a machine, within the time of $[t-\hat{P}, t]$ there is at least one position fixed job on that machine. Such a window can have at most $\hat{P}/p_{\min} = \rho/\delta^2$ jobs in it. The idea of our DP is a sweeping line that goes left-to-right on timeline (at increments of $p_{\min}$) and for any time point t keeps track of the last block (if any) on machine i (for each $1 \leq i \leq M$) where the block starts before t and extends beyond t. More specifically, let V_i $(1 \leq i \leq M)$ be an ordered list of size at most $\Delta = \rho/\delta^2$ of jobs which describes the list of jobs of the last block scheduled on machine i, assuming that this block crosses time t (if no such block exists, $V_i = \emptyset$). Note that by the above arguments, the first job of each block (V_i) is position fixed. So we also have value f_i $(1 \leq i \leq M)$ for those V_i's that are non-empty, which describes the start time of the head of the block: this is either the release time of the head of the block or the multiple of $p_{\min}$ that it starts at. Recall that J_t is the set of jobs whose span contains t. We keep a set $U \subseteq J_t$ as the set of jobs which are not scheduled yet. Note that by Lemma 5: $|J_t| \leq 2\rho \cdot M/\delta$. So the number of choices for U is at most $2^{|J_t|} \in O(2^{\rho/\epsilon^2}\text{poly}(n))$ since we assumed $M \in O(\log n/\log\log n)$.

Our DP table is a True/False table which has entries of the form $A[t, V_1, \ldots, V_M, f_1, \ldots, f_M, U]$ and this is true if and only if the following holds: There is a feasible well-structured schedule of a subset of jobs such that all the jobs whose deadline is at most t are scheduled and the description of the last block on each machine that crosses time t is given by the lists $V_1, \ldots, V_M$ (these are the sequences of the jobs in these last blocks) and $f_1, \ldots, f_M$ (start time of the head of the blocks), and U is the set of jobs from J_t that are not scheduled yet. First, let us show that the table size is polynomial. Note that the length of a block is at most $\hat{P}$, therefore the start of each block is in the interval $[t-\hat{P}, t]$ and that first job is position fixed. This means its start time is either its release time or has the form $d \cdot p_{\min}$ which can be indexed based on t; i.e. we keep which multiple of $p_{\min}$ before t it starts at. Therefore the number of choices for value d is at most $\hat{P}/p_{\min} \leq \Delta$. So the information needed to specify each f_i is $O(\Delta)$: a flag which indicates its start time is its release time or a value for d indicating which multiple of $p_{\min}$ before t it starts at. Now we bound the complexity of vectors V_i. Note that each job in V_i must be a job whose span crosses some point in $[t-\hat{P}, t]$. So if we define $T_{t,\hat{P}}$ to be the set of time points that are multiple of $p_{\min}$ in this interval $[t-\hat{P}, t]$, then the span of each job in V_i must cross at least one time $t' \in T_{t,\hat{P}}$. Since $|J_t| \leq 2\rho \cdot M/\delta$ and $|T_{t,\hat{P}}| \leq \Delta$, the number of choices for each job in V_i is at most $(2\rho \cdot M/\delta) \cdot \Delta \in O(\rho^2 \cdot M/\delta^2)$. Since $|V_i| \leq \Delta$, so the total number of choices for all V_i's and ordering of them is $O((\Delta \cdot M)^{O(\rho^2 \cdot M/\delta^2)})$. Noting that $\Delta = \rho/\delta^2 = \rho^2/\epsilon^4$, and $\delta = \epsilon^2/100\rho$, and $M = O(\log n/\log\log n)$, the total runtime is $n^{O(\rho^4/\epsilon^4)}$.

We compute the entries in increasing values of t. For small values of t (base cases) the table is easy to compute as the number of jobs to consider is bounded by $O(\rho \cdot M/\delta)$. Now we describe the recurrence to compute the entries of this table. For each table entry $A[t, V_1, \ldots, V_M, f_1, \ldots, f_M, U]$ we consider all the entries of the form $A[t - p_{\min}, V_1', \ldots, V_M', f_1', \ldots, f_M', U']$ such that $V_1', \ldots, V_M', f_1', \ldots, f_M', U'$ is consistent (defined below) with $V_1, \ldots, V_M, f_1, \ldots, f_M, U$ and we set $A[t, V_1, \ldots, V_M, f_1, \ldots, f_M, U] = True$ if at least one of the consistent ones is true. We say $V_1', \ldots, V_M', f_1', \ldots, f_M', U'$ is consistent with $V_1, \ldots, V_M, f_1, \ldots, f_M, U$ if:

1. **Consistency of V_i, V_i', f_i, f_i' (for each $1 \leq i \leq M$):** there are three cases.
   a. If V_i is empty then the last job of V_i' must have ended at t or V_i' is also empty. If V_i' is empty (but not V_i) then f_i must be in $[t - p_{\min}, t]$ and V_i consist of only one job that starts at f_i (and is finishing before its deadline).
   b. If $f_i = f_i'$, then the block on machine i is continued from time $t - p_{\min}$ to time t. Then either $V_i' = V_i$ (when the last job continues from time $t - p_{\min}$ to t) or either V_i contains exactly one extra job appended at the end of V_i' and it starts in $[t - p_{\min}, t]$ and finishes before its deadline. Also the block-size bound must hold, i.e. $|V_i| \leq \Delta$.
   c. If $f_i \neq f_i'$, then a new block starts at time f_i. In this case V_i must consist of exactly one job, and that job must start in $[t - p_{\min}, t]$ and finishes before its deadline.
2. **Consistency of U and U':** first, every job in U must have deadline larger than t; otherwise the current entry is False. For transitions between U' and U:
   a. if $j \in U \setminus U'$, then $r_j \geq t - p_{\min}$ (otherwise j should already be in U'), and
   b. if $j \in U' \setminus U$, then job j must start in $[t - p_{\min}, t]$ (and finishes before its deadline), which can be checked from the chosen lists $V_1, \ldots, V_M$.
3. **Recurrence condition:** $A[t, V_1, \ldots, V_M, f_1, \ldots, f_M, U]$ is True if and only if there exists at least one True entry of the form $A[t - p_{\min}, V_1', \ldots, V_M', f_1', \ldots, f_M', U']$ that is consistent with it under items 1 and 2 above. Since by definition all jobs with deadline at most $t - p_{\min}$ are already scheduled in the previous entry, and the consistency checks enforce correct updates from U' to U, no job with deadline at most t can be missed.

So for each entry of the table we have to consider all other entries that are consistent which are at most $n^{O(\rho^4/\epsilon^4)}$ others. So the total time to compute all the entries has the same time complexity.

3.2 Asymptotic PTAS for Small Jobs

Now we describe an asymptotic PTAS for J_S, i.e. small jobs. Let M_S^* be the value of optimum solution for J_S. We show there is a near optimum solution where the jobs appear in windows or blocks of $O(M_S^*)$ jobs each where each window is specified by a start and end time and all the jobs of a window are scheduled between its start and end. To do this, we first show the existence of a near optimum window-based structured solution. We then show we can modify the instance by trimming the spans of jobs to be a smaller subset of their original span while loosing only an $O(\epsilon)$ factor in the optimum. We then show how we can find an asymptotic PTAS for this new instance using LP rounding. This solution will give us an asymptotic PTAS for J_S as well.

For this section we define $\tilde{P} = 16p_{\max}/\epsilon$ and consider the timeline partitioned into windows of size $\tilde{P}$ each. So window W_i ($i \geq 1$) starts from time $(i-1) \cdot \tilde{P}$ to time $i \cdot \tilde{P}$ and we will have $h = \lceil T/\tilde{P} \rceil$ windows (note that $h = \text{poly}(n)$ since $T \leq 2n^2 \cdot p_{\max}$). Note that since all the jobs are δ-small (where $\delta = \epsilon^2/(100\rho)$), the span of each job j intersects at least $100/(16\epsilon)$ windows. Also, the window sizes are much greater than any processing time. So in any feasible schedule a job position cannot be intersecting more than two windows. For each job j, we use $\mathcal{W}(j)$ to denote the indices of the windows that intersect the span of job j, i.e. $\mathcal{W}(j) := \{i \in [h] : [r_j, d_j] \cap W_i \neq \emptyset\}$. The *head* window of j, denoted by $h(j)$ is the index of the window that contains r_j and the *tail* window of j, denoted by $t(j)$, is the (index of) window that contains d_j. We define *core* windows of j to be the set of all those windows that intersect span of j but are not head or tail windows, i.e. $\mathcal{C}(j) = \mathcal{W}(j) - h(j) - t(j)$. Note that all windows in $\mathcal{C}(j)$ are fully contained in span_j. In a schedule, we say a job j is *position-crossing* if it starts in a window W_i and finishes in W_{i+1} (i.e. its position crosses two windows).

3.2.1 Near-Optimal Structured Solution

Next, we show that there exists a near-optimal solution with no position-crossing jobs. Given a feasible solution S on, say M, machines we build a solution with no position-crossing jobs on $M + \lceil \epsilon \cdot M \rceil$ machines. The idea is that for any time t that is start/end of a window we remove all the jobs whose position (in S) is crossing t and schedule them all on $\lceil \epsilon \cdot M \rceil$ auxiliary machines. The intuition is that these jobs (crossing t) have a lot of space to be scheduled either to the left or right of t (at least half the span of them). So these (at most) M jobs removed (for time t) can be scheduled in the interval $[t - \tilde{P}, t + \tilde{P}]$ on $\lceil \epsilon \cdot M \rceil$ machines and we can ensure that the schedule for "removed" jobs for time t and $t + \tilde{P}$ (i.e. the two ends of a window) are not overlapping on the auxiliary machines as they differ by at least $\tilde{P}$. In fact, we will use only at most $\frac{1}{4}$ of available time on each auxiliary machine (in each window) and the reserve space will be used later for other jobs we reschedule. The following lemma is easy to see since the span of each job j intersects at least $100/16\epsilon$ windows:

▶ **Lemma 8.** *For every δ-small job j, we have $|\mathcal{C}(j)| \geq \frac{100}{16\epsilon} - 2$.*

▶ **Theorem 9.** *Given a feasible solution S on M machines we can build a solution S' with no position-crossing jobs on $M + \lceil \epsilon \cdot M \rceil$ machines and at most the first $\frac{1}{4}$ of time of the extra $\lceil \epsilon \cdot M \rceil$ machines are used in each window.*

Proof. For each $i < h$, define X_i to be the set of jobs whose position in S intersects the end of W_i. Note that $|X_i| \leq M$. We partition each X_i into two parts X_i^l, X_i^r. For each job $j \in X_i$ if $r_j \in W_i$ then we place j in X_i^r otherwise we place it in X_i^l. Note the span of each job in X_i^l contains W_i and the span of each job in X_i^r contains W_{i+1} using the fact that span of each job intersects at least $1/\epsilon$ (consecutive) windows. We will show that for each i, all the jobs in X_i^l and X_{i-1}^r can be scheduled on $\lceil \epsilon \cdot M \rceil$ auxiliary machine in window W_i such that each auxiliary machine is busy in at most the first $1/4$ fraction of each window and is free the rest of W_i. This space will be used later. Hence, there exists a near-optimal solution in which no position-crossing job exists on $M + \lceil \epsilon \cdot M \rceil$ machines and each auxiliary machine still have $3/4$ of their time free in each window.

Fix some window W_i and all the position-crossing jobs in $A_i = X_i^l \cup X_{i-1}^r$ that need to be scheduled in window W_i on the auxiliary machines. Recall that for each job j in A_i: $W_i \subseteq \text{span}_j$. Also $|W_i| = \tilde{P} = 16p_{\max}/\epsilon$ and $p_j \leq p_{\max}$ for $j \in A_i$. Note that we want to use no more than the first $1/4$ of each window W_i. Looking at each job as an item of size p_j and the quarter window as a bin of size $\tilde{P}/4$, we have a total of at most $2M$ items in A_i, each of which is small compared to the bin size $\tilde{P}/4$ since $p_{\max} = \epsilon \cdot \tilde{P}/16$. So greedy packing the items into these bins, using $\lceil \epsilon M \rceil$ bins and using only the first $1/4$ of their capacity we can pack all the items; since we pack at least $\tilde{P}/4 - p_{\max}$ in each quarter bin and the total sizes of the items is at most $2M \cdot \epsilon \cdot \tilde{P}/16 = \epsilon \cdot \tilde{P} \cdot M/8$, which means we use at most $\lceil \frac{\epsilon \cdot \tilde{P} \cdot M}{8(\tilde{P}/4 - p_{\max})} \rceil = \lceil \frac{\epsilon}{8-4\epsilon} \cdot M \rceil \leq \lceil \epsilon \cdot M \rceil$ quarter bins. ◀

Given a feasible schedule, a job j is called *head-scheduled* (*tail scheduled*) if it is scheduled in its head (tail) window (we sometimes call it a head job or a tail job as well). If j is scheduled in one of its core windows then it is called a *core scheduled* (or simply a core) job. We next show that given a solution with no position crossing as generated by Theorem 9, we can reschedule all the head or tail jobs by using the empty space ($3/4$ of each window W_i) on the auxiliary machines so that no job is scheduled in its head or tail window. Before proving that we state a simple lemma that we will use in our proof.

▶ **Lemma 10.** *Given a feasible schedule S with no no position-crossing jobs, we can assume that for each window W_i and each machine, the jobs in W_i appear in the following order: all the tail scheduled jobs, then the core scheduled jobs, and then the head scheduled jobs.*

Proof. Suppose that two jobs j, j' are scheduled consecutively in this order (with possible idle time between them) on a machine where j is a core job while j' is a tail job. Suppose that j starts at s_j and j' starts at $s_{j'}$. We can simply swap the order of the jobs and let j' start at s_j followed immediately by j. It's easy to see that this remains a feasible schedule. We can do similar swaps if a head job j appears immediately before a core job. ◀

▶ **Theorem 11.** *Given a feasible schedule generated by Theorem 9 on $M + \lceil \epsilon \cdot M \rceil$ machines with no position-crossing jobs, we can build a feasible schedule with no position-crossing jobs and no head-scheduled or tail-scheduled job on the same machines.*

Proof. Consider the feasible solution S on $M + \lceil \epsilon \cdot M \rceil$ machines as generated by Theorem 9. Note that for each auxiliary machine, only the first $1/4$ of each window the machine is busy. Also, the jobs that were moved to be scheduled on these machines in Theorem 9 are scheduled in their core (so they do not need to move). We show how to build a new solution by modifying S by moving those jobs on the first M machines that are head or tail scheduled. To do this, in the first phase we reschedule the head-scheduled jobs to be scheduled in a different window (other than their heads). We do the same for tail scheduled jobs, separately. So let's focus on the first phase for now.

Starting from $i = 1$ and for increasing values of i, we consider each window W_i and remove all the jobs that are scheduled in W_i and this is their head window; let's say $\lambda_{i,j}$ is the total freed up space (which by Lemma 10 we can assume is one continuous block) on machine j in this window. When considering W_i, we try to schedule as many of the jobs removed from the previous windows in this window on the freed up spaces $\lambda_{i,1}, \ldots, \lambda_{i,M}$ plus using half of the remaining empty space in window W_i (i.e. the next $\frac{3}{8}|W_i|$ empty space) on the auxiliary machines (we will use the last $\frac{3}{8}|W_i|$ space on auxiliary machines to reschedule the tail scheduled jobs). We use the deadline-first order to schedule these jobs. We will show that this generates the desired schedule of all the jobs and all the head-scheduled jobs are rescheduled in their core windows.

Algorithm 1 Rescheduling head-scheduled jobs.

1: Initialize queue $\mathcal{Q} \leftarrow \emptyset$; $Q' \leftarrow \emptyset$
2: **for** $i = 1$ to h **do**
3: Remove the head scheduled jobs of W_i (and place them in Q'); say the total freed up space in window W_i on machine $1 \le j \le M$ has length $\lambda_{i,j}$
4: Schedule as many jobs from $\mathcal{Q}$ as possible by earliest deadline first ordering into the free spaces in W_i on machines $1, \ldots, M$ and on the $\frac{3}{8}|W_i|$ of $\lceil \epsilon M \rceil$ auxiliary machines and remove them from $\mathcal{Q}$.
5: Add all jobs of Q' to end of queue $\mathcal{Q}$ and reset $Q' = \emptyset$
6: **end for**

▶ **Lemma 12.** *Consider the iteration i and let $\mathcal{Q}_i$ be the value of $\mathcal{Q}$ at the start of this iteration and $P(\mathcal{Q}_i) = \sum_{j \in \mathcal{Q}} p_j$. If $\lambda_{i,1}, \ldots, \lambda_{i,M}$ are the freed up spaces in W_i by removing head scheduled jobs in Step 3 then the total processing time of the jobs we schedule in Step 4 is at least:* $\min\left\{\sum_{k=1}^{M} \lambda_{i,k} + 4p_{\max}M,\ P(\mathcal{Q}_i)\right\}$.

Proof. Note that in iteration i the space we use to schedule jobs in Step 4 consists of two parts: $\sum_{k=1}^{M} \lambda_{i,k}$ created by removing head scheduled jobs in Step 3; and the half of remaining $\frac{3|W_i|}{4}$ free space of each window W_i on the auxiliary machines. For each free space $\lambda_{i,k}$ we can schedule a total of at least $\max\{\lambda_{i,k} - p_{\max}, 0\}$ by greedy scheduling. So the total processing time of the jobs scheduled in the freed up spaces is at least

$$\sum_{k=1}^{M} \max\{\lambda_{i,k} - p_{\max}, 0\} \geq \sum_{k=1}^{M} (\lambda_{i,k} - p_{\max}) = \sum_{k=1}^{M} \lambda_{i,k} - M \cdot p_{\max}.$$

For the space in W_i on auxiliary machines, since $|W_i| = 16p_{\max}/\epsilon$, and $\frac{3}{4}$ of this remains free (after using the at most 1/4 in Theorem 9) and we use 1/2 of that (which is $\frac{3}{8}|W_i|$) for rescheduling head jobs, we can schedule a total of at least $\frac{3}{8}|W_i| - p_{\max} = \frac{6p_{\max}}{\epsilon} - p_{\max}$ processing time on each auxiliary machine. Therefore, the total processing time of the jobs scheduled (from $\mathcal{Q}_i$) on the auxiliary machines on W_i is at least $\epsilon M \left(\frac{6p_{\max}}{\epsilon} - p_{\max}\right) = 6M \cdot p_{\max} - \epsilon M \cdot p_{\max} \geq 5M \cdot p_{\max}$, where we use $\epsilon \leq \frac{1}{6}$. Combining the two, the total processing time of the jobs from $\mathcal{Q}_i$ scheduled in Step 4 is at least $\left(\sum_{k=1}^{M} \lambda_{i,k} - M \cdot p_{\max}\right) + 5M \cdot p_{\max} = \sum_{k=1}^{M} \lambda_{i,k} + 4M \cdot p_{\max}$. ◀

Note that this lemma shows that the total amount of processing time scheduled in Step 4 is at least $\sum_{k=1}^{M} \lambda_{i,k} + 4M \cdot p_{\max}$ (if $\mathcal{Q}$ is not entirely rescheduled in this iteration).

▶ **Lemma 13.** *Algorithm 1 reschedules all the head-scheduled jobs within their core windows and they use only $\frac{3}{8}$ of each window W_i on each auxiliary machine.*

Proof. We show that Algorithm 1 will successfully schedule all the head-scheduled jobs that it removes (and places in $\mathcal{Q}$) into one of its core windows. To prove this, we need to show that each job j added to $\mathcal{Q}$ is scheduled in one of the windows before it reaches its tail window. By way of contradiction, suppose that some j is added to $\mathcal{Q}$ at iteration i (so W_i was its head window) but is never scheduled before iteration i', which is its tail window and this is the first job for which this happens. So j has remained in $\mathcal{Q}$ in iterations $i+1, i+2, \ldots, i'-1$, i.e. for a total of for $k = i' - i - 2$ iterations. Since j is δ-small, by Lemma 8, we have $k \geq \frac{100}{16\epsilon} - 2$. Note that if at some iteration $\mathcal{Q}$ becomes empty, then all the previously removed head-scheduled jobs are rescheduled and therefore we can reset the analysis from that window onward (i.e. the next window be window 1). So without loss of generality, we assume from window 1 until $i'-1$, $\mathcal{Q}$ has never become empty. Note that using Lemma 12 (and the observation after this lemma) over iterations $2, \ldots, i'-1$, the total processing time of jobs from $\mathcal{Q}$ that are rescheduled in windows $W_2, \ldots, W_{i'-1}$ on the M machines, plus the space on auxiliary machines is at least: $\Lambda = \sum_{\ell=2}^{i'-1} \sum_{r=1}^{M} \lambda_{\ell,r} + 4p_{\max} M \cdot (i'-2)$.

On the other hand, the total processing time of the jobs added to $\mathcal{Q}$ in iterations $1, \ldots, i'-2$ is $\Lambda' = \sum_{\ell=1}^{i'-2} \sum_{r=1}^{M} \lambda_{\ell,r}$. Clearly, if $\Lambda \geq \Lambda'$ then job j must have been scheduled before window i'. We derive a contradiction if this inequality doesn't hold (i.e. if $\Lambda' > \Lambda$). Suppose $\Lambda' - \Lambda = \sum_{r=1}^{M} \lambda_{1,r} - \sum_{r=1}^{M} \lambda_{i'-1,r} - 4M \cdot p_{\max} \cdot (i'-2) > 0$. By definition of $\lambda_{1,r}$, the total head length in window W_1 is at most window size times number of machines, so $\sum_{r=1}^{M} \lambda_{1,r} \leq |W_1| \cdot M = \frac{16p_{\max}}{\epsilon} \cdot M$. Thus we must have $\frac{16p_{\max}}{\epsilon} \cdot M - 4M \cdot p_{\max} \cdot (i'-2) > 0$. But this is impossible since $k \geq \frac{100}{16\epsilon} - 2$ (and so $i' - 1 \geq \frac{100}{16\epsilon} - 2$) we have $4M \cdot p_{\max} \cdot (i'-2) \geq 4M \cdot p_{\max} \left(\frac{100}{16\epsilon} - 3\right) > \frac{16p_{\max}}{\epsilon} M$, where the last inequality holds for $\epsilon < \frac{1}{6}$. Therefore no job can be missed. ◀

Now we a ready to complete the proof of Theorem 11. Lemma 13 implies that we can reschedule all the head scheduled jobs on the $\frac{3}{8}|W_i|$ of the free spaces of the extra machines. Similar arguments work for rescheduling the tail scheduled jobs using the rest of the free spaces on auxiliary machines. At the end we have a schedule on $M + \lceil \epsilon \cdot M \rceil$ machines with no position crossing and no head or tail scheduled job. ◀

We should note that given a feasible solution S with M machines, the modifications done in Theorem 9, generates a solution on $M + \lceil \epsilon \cdot M \rceil$ machines with no position crossing jobs and all the jobs on the auxiliary machines are running in their core. Therefore, applications of Theorem 9 and Theorem 11 results in a feasible solution with no position crossing and no head or tail jobs on $M + \lceil \epsilon \cdot M \rceil$ machines.

3.2.2 LP rounding for Finding A Near-Optimal Solution

Theorems 9 and 11 show that there is a solution for J_S in which there is no position-crossing jobs and no job is scheduled in their head or tail window on $M_S^* + \lceil \epsilon \cdot M_S^* \rceil$ machines. In this subsection, we explain how to find a solution on $M_S^* + 2\lceil \epsilon \cdot M_S^* \rceil$ (i.e. using $\lceil \epsilon \cdot M_S^* \rceil$ extra machines) satisfying these properties using LP rounding. The LP relaxation we present is insipred by the LP relaxation for the Generalized Assignment Problem (GAP) [26]. Again, we find the smallest value of $M \geq M_S^*$ for which we find a feasible solution.

Let S be any feasible solution with no position-crossing jobs and no head-scheduled or tail-scheduled jobs on M machines. Recall that the timeline is partitioned into h windows, each of length $\tilde{P} = 16p_{\max}/\delta$. A *machine-slot* is a pair (i, μ) where $i \in [h]$ is a window index and $\mu \in [M]$ is a machine index. Hence the set of all slots is $\mathcal{M} = [h] \times [M]$, and $|\mathcal{M}| = hM$.

We are looking for an assignment of each job to one of its core windows to be scheduled in one of the slots (i.e. on a machine) on that window so that all the jobs assigned to a slot (i.e. a machine in a window) can fit into $\tilde{P}$. For each job $j \in J$, let $\mathcal{M}(j) := \{(i, \mu) \in \mathcal{M} : \ i \in \mathcal{C}(j)\}$ be the set of slots in which j can be placed (i.e. windows fully contained in its allowed core range, on any machine). For each slot $(i, \mu) \in \mathcal{M}$, define $\mathcal{J}(i, \mu) := \{j \in J : \ (i, \mu) \in \mathcal{M}(j)\}$. For each pair $((i, \mu), j)$ with $(i, \mu) \in \mathcal{M}(j)$, let $x_{(i,\mu),j}$ denote whether job j is assigned to slot (i, μ). Our LP relaxation is a feasibility LP described below:

$$\begin{aligned}
\sum_{j \in \mathcal{J}(i,\mu)} p_j \cdot x_{(i,\mu),j} &\leq \tilde{P} \qquad \forall (i,\mu) \in \mathcal{M} \qquad (1)\\
\sum_{(i,\mu) \in \mathcal{M}(j)} x_{(i,\mu),j} &= 1 \qquad \forall j \in J,\\
x_{(i,\mu),j} &\geq 0 \qquad \forall j \in J,\ \forall (i,\mu) \in \mathcal{M}(j).
\end{aligned}$$

Note that the solution S is an integer feasible solution to this LP. Also one can see that this is a special case of the GAP and hence the same LP rounding method as for GAP implies the following lemma:

▶ **Lemma 14.** *Given a fractional solution x to the feasibility LP (1), there exists an integral assignment of jobs to machine-slots such that for every slot $(i, \mu) \in \mathcal{M}$ the total processing time of jobs assigned to it is at most $\tilde{P} + p_{\max}$. More specifically, there is one job from those assigned to (i, μ) s.t. when removed, the total processing time of the remaining jobs is at most $\tilde{P}$.*

Proof. Fix a slot $(i, \mu) \in \mathcal{M}$. Let

$$k_{(i,\mu)} = \left\lceil \sum_{j \in \mathcal{J}(i,\mu)} x_{(i,\mu),j} \right\rceil .$$

Create $k_{(i,\mu)}$ integer sub-slots for slot (i, μ), each of unit capacity. Sort jobs with $x_{(i,\mu),j} > 0$ in non-increasing order of p_j, and distribute their fractional values $x_{(i,\mu),j}$ into these sub-slots by first-fit (fill sub-slot 1 up to 1, then sub-slot 2, etc.).

Now build a bipartite graph between jobs and sub-slots: connect a job j to a sub-slot if a positive fraction of j was placed there in the previous step. This gives a fractional matching between jobs and sub-slots that assigns a total fraction of 1 to each job and uses at most capacity 1 at each sub-slot. By integrality of bipartite matching polytope, there is an integral matching between jobs and sub-slots that matches every job exactly once. For each sub-slot, let its *representative size* be the largest processing time among jobs adjacent to it. The job matched to that sub-slot has processing time at most this representative size. Hence the total load on slot i is bounded by the sum of representative sizes of all of its $k_{(i,\mu)}$ sub-slots. Because we filled sub-slots in non-increasing order of job sizes, for each sub-slot $k \geq 2$ of (i, μ), the size of the representative for it is no more than the size of the smallest job that had been fractionally assigned for the previous sub-slot (i.e. $k - 1$). So we can say, the size of representative of sub-slot k is no more than the *average* size of the jobs fractionally assigned to slot $k - 1$. This implies that the total sizes of the representatives of sub-slots $2, \ldots, k_{(i,\mu)}$ is no more than the average of the jobs in slots $1, \ldots, k_{(i,\mu)} - 1$. Therefore the sum of representative sizes over sub-slots $2, \ldots, k_{(i,\mu)}$ is at most $\sum_{j \in \mathcal{J}(i)} p_j x_{ij} \leq \tilde{P}$. The first sub-slot contributes at most $p_{\max}$. So total load on slot i is at most $\tilde{P} + p_{\max}$. ◀

3.2.2.1 Rescheduling the excess load

We now describe how to change the assignment of jobs assigned using Lemma 14 to a feasible solution, by removing the largest job from each slot and schedule them on some auxiliary machines. Since $\tilde{P} = 16\frac{p_{\max}}{\epsilon}$, the additive load (total processing time) assigned to each slot is at most $p_{\max}$ which is bounded by $\epsilon\tilde{P}$. The next lemma shows this extra load can be moved to a small number of extra machines. The argument here is essentially the same as what we had to move the position-crossing jobs to some extra machines as in Lemma 9.

▶ **Lemma 15.** *An integral assignment with per-slot total processing time at most* $(1 + \epsilon)\tilde{P}$ *can be converted into a feasible schedule (slot capacity* $\tilde{P}$*) using at most* $\lceil \epsilon M/8 \rceil$ *additional machines per window.*

Proof. Fix one window W_i. It has M slots, each with total processing time of jobs assigned to it (called load) $L_i \leq \tilde{P} + p_{\max}$. If $L_i > \tilde{P}$, remove the largest job from those assigned to this slot so the remaining load is at most $\tilde{P}$. Hence total removed processing in window W_i is at most $\sum_{\mu=1}^{M} p_{\max} = M \cdot p_{\max}$. Each auxiliary machine contributes capacity $\tilde{P}$ in that window and when we use greedy to pack the (δ-small) jobs, it will be at least $\tilde{P} - p_{\max}$ full. So the number of auxiliary machines needed is at most $\left\lceil \frac{M \cdot p_{\max}}{\tilde{P} - p_{\max}} \right\rceil = \left\lceil \frac{M \cdot p_{\max}}{16 p_{\max}/\epsilon - p_{\max}} \right\rceil = \lceil \epsilon M/8 \rceil$. Since removed jobs are δ-small and their spans contain the corresponding window, they can be scheduled on these auxiliary machines. ◀

Combining the structural results and the rounding steps, we obtain a feasible schedule on $M_S^* + 2\lceil \epsilon \cdot M_S^* \rceil$ machines. By being a bit more careful in choosing parameters $\delta, \tilde{P}$ we can reserve enough empty space on the $\lceil \epsilon M \rceil$ auxiliary machines generated in Theorem 11, and

then modify the slot sizes in the LP for the auxiliary machines (to be the empty reserved space on the auxiliary machines) so that the reserved space is used in the LP rounding without the need to get another $\lceil \epsilon M \rceil$. This yields the asymptotic $(1+\epsilon)$-approximation for small jobs. By being even more careful in solving the large and small jobs, we can ensure that we use a total of $\lceil \epsilon \cdot M \rceil$ auxiliary machines across both of them; where the large jobs use about 1/2 the space of auxiliary machines and the small jobs use the other half.

References

1 Alexander Armbruster, Fabrizio Grandoni, Antoine Tinguely, and Andreas Wiese. Improved approximation algorithms for non-preemptive throughput maximization. In *Proceedings of the 58th Annual ACM Symposium on Theory of Computing, 2026*, 2026.

2 Philippe Baptiste. On minimizing the weighted number of late jobs in unit execution time open-shops. *European Journal of Operational Research*, 149(2):344–354, 2003. doi:10.1016/S0377-2217(02)00759-2.

3 Philippe Baptiste, Peter Brucker, Sigrid Knust, and Vadim G. Timkovsky. Ten notes on equal-processing-time scheduling. *4OR*, 2(2):111–127, 2004. doi:10.1007/s10288-003-0024-4.

4 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001. doi:10.1145/502102.502107.

5 Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, 31(2):331–352, 2001. doi:10.1137/S0097539799354138.

6 Michael A. Bender, David P. Bunde, Vitus J. Leung, Samuel McCauley, and Cynthia A. Phillips. Efficient scheduling to minimize calibrations. In Guy E. Blelloch and Berthold Vöcking, editors, *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada – July 23–25, 2013*, pages 280–287. ACM, 2013. doi:10.1145/2486159.2486193.

7 Piotr Berman and Bhaskar DasGupta. Improvements in throughout maximization for real-time scheduling. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 680–687, 2000. doi:10.1145/335305.335401.

8 Lin Chen, Minming Li, Guohui Lin, and Kai Wang. Approximation of scheduling with calibrations on multiple machines (brief announcement). In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 237–239. ACM, 2019. doi:10.1145/3323165.3323173.

9 Zuzhi Chen and Jialin Zhang. Online scheduling of time-critical tasks to minimize the number of calibrations. *Theor. Comput. Sci.*, 914:1–13, 2022. doi:10.1016/J.TCS.2022.01.040.

10 Julia Chuzhoy and Paolo Codenotti. Erratum: Resource minimization job scheduling. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, Lecture Notes in Computer Science. Springer, 2009. doi:10.1007/978-3-642-03685-9_54.

11 Julia Chuzhoy and Paolo Codenotti. Resource minimization job scheduling. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, Lecture Notes in Computer Science, pages 70–83. Springer, 2009. doi:10.1007/978-3-642-03685-9_6.

12 Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 81–90, 2004. doi:10.1109/FOCS.2004.38.

13 Julia Chuzhoy and Joseph Naor. New hardness results for congestion minimization and machine scheduling. *J. ACM*, 53(5):707–721, 2006. doi:10.1145/1183907.1183908.

14 Julia Chuzhoy, Rafail Ostrovsky, and Yuval Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Math. Oper. Res.*, 31(4):730–738, 2006. doi:10.1287/moor.1060.0218.

15 Mark Cieliebak, Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004), 22-27 August 2004, Toulouse, France*, IFIP, pages 209–222. Kluwer/Springer, 2004. doi:10.1007/1-4020-8141-3_18.

16 Nikhil R. Devanur, Konstantin Makarychev, Debmalya Panigrahi, and Grigory Yaroslavtsev. Online algorithms for machine minimization. *CoRR*, abs/1403.0486, 2014. arXiv:1403.0486.

17 Mitre Costa Dourado, Rosiane de Freitas Rodrigues, and Jayme Luiz Szwarcfiter. Scheduling unit time jobs with integer release dates to minimize the weighted number of tardy jobs. *Annals of Operations Research*, 169(1):81–91, 2009. doi:10.1007/s10479-008-0479-y.

18 Jan Elffers and Mathijs de Weerdt. Scheduling with two non-unit task lengths is np-complete. *CoRR*, abs/1412.3095, 2014. arXiv:1412.3095.

19 M. R. Garey and David S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM J. Comput.*, 6(3):416–426, 1977. doi:10.1137/0206029.

20 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

21 Dylan Hyatt-Denesik, Mirmahdi Rahgoshay, and Mohammad R. Salavatipour. Approximations for throughput maximization. *Algorithmica*, 86(5):1545–1577, 2024. doi:10.1007/S00453-023-01201-4.

22 Sungjin Im, Shi Li, and Benjamin Moseley. Breaking $1 - 1/e$ barrier for nonpreemptive throughput maximization. *SIAM J. Discret. Math.*, 34(3):1649–1669, 2020. doi:10.1137/17M1148438.

23 Sungjin Im, Shi Li, Benjamin Moseley, and Eric Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines [extended abstract]. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1070–1086. SIAM, 2015. doi:10.1137/1.9781611973730.72.

24 Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987. doi:10.1007/BF02579324.

25 Barna Saha. Renting a cloud. In Anil Seth and Nisheeth K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, Guwahati, India, December 12-14, 2013*, LIPIcs, pages 437–448. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.437.

26 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993. doi:10.1007/BF01585178.

27 Barbara B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal on Computing*, 12(2):294–299, 1983. doi:10.1137/0212018.

28 Barbara B. Simons and Manfred K. Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM J. Comput.*, 18(4):690–710, 1989. doi:10.1137/0218048.

29 Frits C. R. Spieksma. Approximating an interval scheduling problem. In *Approximation Algorithms for Combinatorial Optimization, International Workshop APPROX'98, Aalborg, Denmark, July 18-19, 1998, Proceedings*, pages 169–180, 1998. doi:10.1007/BFb0053973.

30 Enze Sun, Zonghan Yang, and Yuhao Zhang. Improved algorithms for online rent minimization problem under unit-size jobs. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, Amsterdam, The Netherlands, September 4-6, 2023*, LIPIcs, pages 97:1–97:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.97.

31 Guosong Yu and Guochuan Zhang. Scheduling with a minimum number of machines. *Oper. Res. Lett.*, 37(2):97–101, 2009. doi:10.1016/J.ORL.2009.01.008.

A A 2-Approximation Algorithm for Constant Number of Processing Times

In this section we prove Theorem 2. We assume that in the given instance $\mathcal{I}$, the processing times are drawn from the set $\{p_1, p_2, \ldots, p_c\}$. Suppose we are given a solution S for $\mathcal{I}$ using M machines. We describe a method to modify $\mathcal{I}$ to a new instance $\mathcal{I}'$ with certain properties, while building a *pseudo-feasible* solution S' for $\mathcal{I}'$ in the sense that in S' each machine m might be running two jobs j, j' of the same processing time in parallel at the same time, i.e. they both start and finish at the same time on m. Such a schedule S' can be easily transformed into a feasible schedule S'' on $2M$ machines by making a copy for each machine and let the two jobs that are running in parallel on the same machine to run on the two copies. The instance $\mathcal{I}'$ has the same set of jobs as $\mathcal{I}$ (but with a possibly reduced span) and therefore S' will be a pseudo-feasible solution for $\mathcal{I}$ as well. In building $\mathcal{I}'$ from $\mathcal{I}$ we trim the release time and deadline of jobs in a recursive (heirarchical) procedure such that the number of distinct release time and deadlines of jobs relevant to each step of the recursive procedure is bounded by a polylog(n). This reduced size allows us to compute solution S' in quasi-polynomial time.

Building $\mathcal{I}'$ and S'. We start by $\mathcal{I}' = \mathcal{I}$ and we consider a pseudo-feasible schedule S' where initially the schedule of machines of S' are identical to those of S. We modify the release time and deadline of jobs in $\mathcal{I}'$ in a recursive procedure. We describe a set of cut-points on time horizon recursively.

The first cut point h_1 is chosen such that $\max\{|J(0, h_1)|, |J(h_1, T|\} \leq n/2$, i.e. at most half the jobs have their span entirely to the left or right of h_1. We say h_1 is a balanced cut-point for $0, T$. We focus on $J(h_1)$ and we call them jobs of level 1. For any t, let $S(t)$ be the set of jobs whose schedule in S crosses time t (i.e. the job starts before t in S and finishes after t in S). Note that $S(h_1) \subseteq J(h_1)$ has at most M jobs. We remove all $S(h_1) \subseteq J(h_1)$ from $\mathcal{I}$. Now we partition the remaining jobs of $J(h_1) \cap \mathcal{I}$ into two parts: $J_L(h_1), J_R(h_1)$ where $J_L(h_1)$ consists of those jobs whose position in S is entirely to the left of h_1 (similarly for $J_R(h_1)$). We do the following for each processing time p_c separately.

For now, fix an arbitrary processing time p_α and $J_L^\alpha(h_1)$ is the subset of $J_L(h_1)$ restricted to this processing time. For each machine $1 \leq m \leq M$, consider the jobs of $J_L^\alpha(h_1)$ scheduled on m, denoted by $J_L^{m,\alpha}(h_1)$ and consider them in the order of their schedule in S from h_1 to the left. We partition them into groups $G_0^m, G_1^m, \ldots, G_{\sigma_m}^m$ ($\sigma_m = \lfloor \log |J_L^{m,\alpha}(h_1)| \rfloor \in O(\log n)$) where G_0^m and G_1^m consist of the first and the 2nd job respectively (so each group has size 1), then G_2^m consists of the next two jobs, G_3^m consists of the next 2^2 jobs, and in general G_k^m consists of jobs with rank $2^{k-1}+1$ to 2^k (i.e. the next 2^{k-1} jobs), where the last group $G_{\sigma_m}^m$ might have less than $2^{\sigma_m - 1}$ jobs. Note that $|G_k^m| = 2|G_{k-1}^m|$ for $k \geq 2$ and also the

number of jobs scheduled after the last job of $G^m_{\sigma_m}$ is at most $2|G^m_{\sigma_m}|$. This allows us to shift the schedule of the jobs (on S') in each group G^m_k to positions of jobs in G^m_{k-1} by allowing two jobs per position to run in parallel. (for each $k \geq 2$). More specifically, starting from G^m_1, we move its single job to be scheduled in the same position as the first job on machine m (i.e. the job of G^m_0); this frees up the space of the job of G^m_1 on machine m. Then we move the jobs in G^m_2 to be scheduled in positions of jobs in G^m_1 on machines m (two jobs per position of a job in G^m_1), and so on. And finally the jobs scheduled after the last job in $G^m_{\sigma_m}$ are all moved to be scheduled on the positions of the jobs in G^m_σ (that are now moved) on machines m.

So in general, the jobs of G^m_k are now scheduled in positions of jobs G^m_{k-1} in S' with two jobs running in parallel on each machine. We remove these shifted jobs from $\mathcal{I}$. Next we change the release time and deadline of the jobs in $J^{m,\alpha}(h_1)$ in $\mathcal{I}'$ accordingly. For each group G^m_k, let s^m_k be the start time of the first job scheduled in S in this group. For the jobs that were in G^m_k ($k \geq 2$) in S and are now moved to be scheduled in positions of jobs G^m_{k-1} in S' we change their release time to be s^m_{k-1} and their deadline to be s^m_{k-2} . For both jobs that are now scheduled in place of the single job in G^m_0, their release time is s^m_0 and deadline is h_1. Now after trimming release times and deadlines of jobs of each G^m_k, all the jobs in G^m_k has same release time and deadline and processing time, so we call all of them as a group with same job profile. Note that with this change, S' is a pseudo-feasible solution for $\mathcal{I}'$ and these new spans for different groups are non-overlapping for each machine and the total number of different spans the jobs of $J_L(h_1)$ belong to is $O(\log n)$ per machine, for a total of $O(M \cdot \log n)$.

Doing the above for each processing time p_α, we see that all the jobs in $J_L(h_1)$ are now scheduled on M machines in S' (while each machine might run two parallel jobs of the same processing time simultaneously) and their release time and deadlines belongs to one of $O(M \cdot \log n)$ groups (one start time and finish time for each group). We can do symmetrically for jobs in $J_R(h_1)$ and move those jobs to be scheduled in S' on M machines (2 jobs per machine) and change the release time and deadline of them to be one of $O(M \cdot \log n)$ possible start/release times (defined by the groups). We also remove $J_R(h_1)$ from $\mathcal{I}$. This completes the description of modifying $\mathcal{I}$ and $\mathcal{I}'$ based on the first cut-point h_1. Note that the jobs that were in $S(h_1) \cap \mathcal{I}$ are at most M and we have not trimmed their span. However, the total number of distinct span intervals (or job profiles) for all the jobs of $J(h_1)$ after the modification is still at most $O(M \cdot \log n)$.

Note that for all the remaining jobs in $\mathcal{I}$, their span is now either entirely to the left of h_1 or entirely to the right of h_1 and these two sets of jobs have at most $n/2$ jobs each. We consider a balanced cut-point for each of these two sub-problems separately and recursively repeat the same procedure as above. More specifically, let J_L, J_R be the set of (remaining) jobs of $\mathcal{I}$ whose span is completely to the left and right of h_1, respectively. Assuming that $J_L \neq \emptyset$, let h' be the balanced cut-point for J_L (where the time horizon is now $[0, h_1]$) and assuming that $J_R \neq \emptyset$ let h'' be the balanced cut-point for J_R. Jobs in $J_L(h'), J_R(h'')$ are level 2 jobs. We consider $J_L(h')$ and first trim the release time and deadline of $S(h')$ (those jobs whose position in S crosses h') and remove them from $\mathcal{I}$. Then for jobs of $J_L(h')$ whose position in S is to the left of h' and those whose position in S is to the right of h' and use the same grouping and shifting as we did for h_1 and remove them from $\mathcal{I}$ and update S' accordingly. We do similarly for $J_R(h'')$. Now the remaining jobs in $\mathcal{I}$ can be partitioned into 4 parts: those whose span is to the left of h', those whose span is between h', h_1, those with span between h_1, h'', and finally those whose span is to the right of h'' and we recurse. We continue this process as long as the set of jobs for the current subproblem (and time horizon) is non-empty.

One can associate (naturally) a rooted binary tree T (with root r) with this procedure where the root node r corresponds to h_1 with time horizon $[0, T]$ and the set of jobs associated $J^r = J$ being all the jobs. The root is level 1 and all the jobs whose span crosses h_1 are level 1 jobs. For each node v with time horizon $[a^v, b^v]$ and a set of jobs $J^v = J[a^v, b^v]$ and balanced cut-point h^v, the left child corresponds to interval $[a^v, h^v]$ and the set of jobs will be all those in $J^v - J^v(h^v)$ whose span is to the left of h^v and the right child corresponds to interval $[h^v, b^v]$ and the set of jobs will be all those in $J^v - J^v(h^v)$ whose span is to the right of h^v. A level i job is a job that is not of level $i' < i$ and whose span is crossing h^v for a level i node v. The following observations are easy to see.

▶ **Observation 16.** *The height of this tree is at most $O(\log n)$ (since we pick balanced cut-points). Also, for each original job j that is scheduled in S on a machine, say m, at most once we move two jobs to be scheduled in place of j on machines m. Once this happens we remove j from $\mathcal{I}$ and this rescheduling in position of j never happens.*

▶ **Observation 17.** *The selection of balanced cut-points is independent of S: given original instance $\mathcal{I}$ we can determine all the balanced cut-points associated with this recursive procedure. Accordingly, we can find all the jobs of level i for each level.*

Note in the described procedure for building S' that moves from top to bottom level of the tree, for each level i, the jobs of level i are partitioned into at most 2^{i-1} parts (corresponding to nodes at level i of the tree) and for each part the spans of the jobs after the change to their release time and deadline belong to a set of size at most $O(\log n)$ non-overlapping intervals on each machine, for a total of $O(M \cdot \log n)$ intervals.

The discussion above implies the following theorem regarding the existence of a pseud-feasible structured solution with each machine running two jobs in parallel.

▶ **Theorem 18** (Structured approximate solution)**.** *Given a set J of jobs and a solution S on M machines, we can modify the release time and deadlines of the jobs in a hierarchical way and build a pseudo-feasible solution with load 2 per machine on M machines such that the resulting instance has the following property:*

1. *For each node v (at level i) of the tree T with balanced cut-point h^v and set of jobs J^v, for the set $J^v(h)$ the number of different spans (job profiles) corresponding to these jobs in the new instance is at most $O(c \cdot M \cdot \log n + M) = O(c \cdot M \cdot \log n)$, and*
2. *For every node v at level i, the number of already trimmed spans or job profiles (from ancestors of v) that cross the interval of v is at most $O((i-1)c \cdot M \cdot \log n)$.*

It is easy to see that the following theorem together with a simple binary search for the smallest value of M imply Theorem 2 by replacing each machine in the pseudo-feasible solution with with 2 machines.

▶ **Theorem 19.** *Given instance $\mathcal{I}$ with a set of jobs J and an integer M, there exists a dynamic programming algorithm that, if $\mathcal{I}$ has a feasible solution S on M machines, it returns a pseudo-feasible solution S' with load 2 per machine on at most M machines in time $O\left(n^{c^2 \cdot M \cdot \log^2 n}\right)$, or returns "No solution".*

We defer the proof of this theorem to the full version of the paper. The idea is to build a hierarchical dynamic program on the binary tree T described above for S' by guessing the jobs of each level as well as guessing the number of jobs of each job profile (trimmed spans), starting from the root interval and the original instance I.