Approximations for Throughput Maximization

Dylan Hyatt-Denesik[1]
Department of Combinatorcs and Optimization, University of Waterloo, Canada
dhyattdenesik@uwaterloo.ca

Mirmahdi Rahgoshay
Department of Computing Science, University of Alberta, Edmonton, Canada
rahgosha@ualberta.ca

Mohammad R. Salavatipour
Department of Computing Science, University of Alberta, Edmonton, Canada
mrs@ualberta.ca

Abstract

In this paper we study the classical problem of throughput maximization. In this problem we have a collection J of n jobs, each having a release time r_j, deadline d_j, and processing time p_j. They have to be scheduled non-preemptively on m identical parallel machines. The goal is to find a schedule which maximizes the number of jobs scheduled entirely in their $[r_j, d_j]$ window. This problem has been studied extensively (even for the case of $m = 1$). Several special cases of the problem remain open. Bar-Noy et al. [STOC1999] presented an algorithm with ratio $1 - 1/(1 + 1/m)^m$ for m machines, which approaches $1 - 1/e$ as m increases. For $m = 1$, Chuzhoy-Ostrovsky-Rabani [FOCS2001] presented an algorithm with approximation with ratio $1 - \frac{1}{e} - \varepsilon$ (for any $\varepsilon > 0$). Recently Im-Li-Moseley [IPCO2017] presented an algorithm with ratio $1 - 1/e + \varepsilon_0$ for some absolute constant $\varepsilon_0 > 0$ for any fixed m. They also presented an algorithm with ratio $1 - O(\sqrt{\log m/m}) - \varepsilon$ for general m which approaches 1 as m grows. The approximability of the problem for $m = O(1)$ remains a major open question. Even for the case of $m = 1$ and $c = O(1)$ distinct processing times the problem is open (Sgall [ESA2012]). In this paper we study the case of $m = O(1)$ and show that if there are c distinct processing times, i.e. p_j's come from a set of size c, then there is a randomized $(1-\varepsilon)$-approximation that runs in time $O(n^{mc^7\varepsilon^{-6}} \log T)$, where T is the largest deadline. Therefore, for constant m and constant c this yields a PTAS. Our algorithm is based on proving structural properties for a near optimum solution that allows one to use a dynamic programming with pruning.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Scheduling, Approximation Algorithms, Throughput Maximization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2020.11

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

1 Introduction

Scheduling problems have been studied in various fields, including Operations Research and Computer Science over the past several decades. However, there are still several fundamental problems that are not resolved. In particular, for problems of scheduling of jobs with release times and deadlines in order to optimize some objective functions there are several problems left open (e.g. see [29, 26, 30]). In this paper we consider the classical problem of throughput maximization. In this problem, we are given 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. The jobs are to be scheduled

[1] Most of this work was done when the author was a graduate student in Department of Computing Science at U. of Alberta

non-preemptively on a single (or more generally on m identical) machine(s), which can process only one job at a time. The value of a schedule, also called its throughput, is the number of jobs that are scheduled entirely within their release time and deadline interval. Our goal is to find a schedule with maximum throughput.

Throughput maximization is a central problem in scheduling that has been studied extensively in various settings (even special cases of it are interesting open problems). They have numerous applications in practice [16, 1, 25, 19, 32]. The problem is known to be NP-hard (one of the list of problems in the classic book by Garey and Johnson [17]). In fact, even special cases of throughput maximization have attracted considerable attention. For the case of all p_j's being equal in the weighted setting (where each job has a weight and we want to maximize the total weight of scheduled jobs), the problem can be solved in polynomial time only when $m = O(1)$ (running time is exponential in m) [4, 13]. The complexity of the problem is open for general m. For the case where all processing times are bounded by a constant the complexity of the problem is listed as an open question [30]. It was shown in [14] that even for $m = 1$ and $p_j \in \{p, q\}$ where p and q are strictly greater than 1 the problem is NP-Complete.

1.1 Related Works

It appears the first approximation algorithms for this problem were given by Spieksma [31] where a simple greedy algorithm has shown to have approximation ratio 1/2. This algorithm will simply run the job with the least processing time between all the available jobs whenever a machine completes a job. He also showed that the integrality gap of a natural Linear Program relaxation is 2. Later on, Bar-Noy et al. [6] analyzed greedy algorithms for various settings and showed that for the case of m identical machines greedy algorithm has ratio $1 - 1/(1 + 1/m)^m$. This ratio is 1/2 for $m = 1$ and approaches $1 - 1/e$ as m grows.

In a subsequent work, Chuzhoy et al. [12] looked at a slightly different version, call it *discrete* version, where for each job j, we are explicitly given a collection $\mathcal{I}_j$ of intervals (possibly of different lengths) in which job j can be scheduled. A schedule is feasible if for each job j in the schedule, j is placed within one of the intervals of $\mathcal{I}_j$. This version (vs. the version defined earlier, which we call the "continuous" version) have similarities but none implies the other. In particular, the discrete version can model the continuous version if one defines each interval of size p_j of $[r_j, d_j]$ as an interval in $\mathcal{I}_j$. However, the number of intervals in $\mathcal{I}_j$ defined this way can be as big as $d_j - r_j + p_j$ which is not necessarily polynomial in the input size. Chuzhoy et al. [12] presented a $(1 - 1/e - \epsilon)$-approximation for the discrete version of the problem. Spieksma [31] showed that the discrete version of the problem is $MAX\text{-}SNP$ hard using a reduction to a version of $MAX\text{-}3SAT$. No such approximation hardness result has been proved for the continuous version.

Berman and DasGupta [8] provided a better than 2 approximation for the case when all the jobs are relatively big compared to their window size. A pseudo-polynomial time exact algorithm for this case is presented by Chuzhoy et al. [12] with running time $O(n^{poly(k)}T^4)$, where $k = max_j(d_j - r_j)/p_j$ and $T = max_j d_j$.

For the weighted version of the problem, [3] showed that when we have uniform processing time $p_j = p$, the problem is solvable in polynomial time for $m = 1$. For $m = O(1)$ and with uniform processing time [4, 13] presented polynomial time algorithms. For general processing time 2-approximation algorithms are provided in [8, 5] and this ratio has been the best known bound for the weighted version of the problem. More recently, Im et al. [20] presented better approximations for throughput maximization for all values of m. For the unweighted case, for some absolute $\alpha_0 > 1 - 1/e$, for any $m = O(1)$ and for any $\epsilon > 0$ they

presented an $(\alpha_0 - \epsilon)$-approximation in time $n^{O(m/\epsilon^5)}$. They also showed another algorithm with ratio $1 - O(\sqrt{(\log m)/m} - \epsilon)$ (for any $\epsilon > 0$) on m machines. This ratio approaches 1 as m grows. Furthermore, their $1 - O(\sqrt{(\log m)/m} - \epsilon)$ ratio extends to the weighted case if $T = \text{Poly}(n)$.

Bansal et al. [2] looked at various scheduling problems and presented approximation algorithms with resource augmentation (a survey of the many resource augmentation results in scheduling is presented in [27]). An α-approximation with β-speed augmentation means a schedule in which the machines are β-times faster and the total profit is α times the profit of an optimum solution on original speed machines. In particular, for throughput maximization they presented a 24-speed 1-approximation, i.e. a schedule with optimum throughput however the schedule needs to be run on machines that are 24-times faster in order to meet the deadlines. This was later improved by Im et al. [21], where they developed a dynamic programming framework for non-preemptive scheduling problems. In particular for throughput maximization (in weighted setting) they present a quasi-polynomial time $(1 - \epsilon, 1 + \epsilon)$-bi-criteria approximation (i.e. an algorithm that finds a $(1 - \epsilon)$-approximate solution using $(1 + \epsilon)$ speed up in quasi-polynomial time). We should point out that the PTAS we present for c distinct processing time implies (as an easy corollary) a bi-criteria QPTAS as well, i.e. a $(1 - \epsilon)$-approximation using $(1 + \epsilon)$-speed up. However, they use the $(1 + \epsilon)$-speed up of machines crucially in several places in their algorithms and it does not seem that even adding the assumption of having c many distinct processing times to their algorithm would help to get a polynomial time approximation with ratio $(1 - \epsilon)$.

For the problem of machine minimization, where we have to find the minimum number of machines with which we can schedule all the jobs, the algorithm provided in [28] has approximation ratio $O(\sqrt{\log n / \log\log n})$ only when $OPT = \Omega(\sqrt{\log n / \log\log n})$, and ratio $O(1)$ when $OPT = \Omega(\log n)$. Later Chuzhoy et al. [10] presented an $O(OPT)$-approximation which is good for the instances with relatively small OPT. Combining this with the earlier works implies an $O(\sqrt{\log n / \log\log n})$-approximation. Chuzhoy and Naor [11] showed a hardness of $\Omega(\log\log n)$ for the machine minimization problem.

Another interesting generalization of the problem is when we assign a height to each job as well and allow them to share the machine as long as the total height of all the jobs running on a machine at the same time is no more than 1. The first approximation algorithm for this generalization is provided by [5] which has ratio 5. Chuzhoy et al. [12] improved it by providing an $(e - 1)/(2e - 1) > 0.3873$-approximation algorithm which is only working for the unweighted and discrete version of the problem. The problem has also been considered in the online setting [7, 15, 23, 24].

1.2 Our Results

Our main result is the following. Suppose that there are c distinct processing times.

▶ **Theorem 1.** *For the throughput maximization problem with m identical machines and c distinct processing times for jobs, for any $\varepsilon > 0$, there is a randomized algorithm which finds a $(1 - \varepsilon)$-approximate solution with high probability runs in time $n^{O(mc^7\varepsilon^{-6})} \log T$, where T is the largest deadline.*

So for $m = O(1)$ and $c = O(1)$ we get a Polynomial Time Approximation Scheme (PTAS). Note that even for the case of $m = 1$ and $c = 2$, the complexity of the problem has been listed as an open problem in [30], however, it has been shown in [14] that even for $m = 1$ and $p_j \in \{p, q\}$ where p and q are strictly greater than 1 the problem is NP-Complete. Our algorithm for Theorem 1 is obtained by proving some structural properties for near optimum

solutions and by describing a randomized hierarchical decomposition which allows us to do a dynamic programming. In order to prove this we prove (and use at the base of our DP) the following (easier) special case:

▶ **Theorem 2.** *Suppose we are given B intervals over the time-line where the machines are pre-occupied and cannot be used to run any jobs, there are R distinct release times, D distinct deadlines, and m machines, where $R, D, B, m \in O(1)$. Then there is a PTAS for throughput maximization with time $2^{\varepsilon^{-1}\log^{-4}(1/\varepsilon)} + \text{Poly}(n)$.*

An easy corollary of Theorem 1 is the following. If the largest processing time $p_{max} = \text{Poly(n)}$ then we get a quasi-polynomial time $(1-\epsilon)$-approximation using $(1+\epsilon)$-speed up of machines. This result of course was already obtained in [21]. We should mention that the framework of [21] heavily relies on machine speed up and it is not clear if that approach can be adapted to give an improved approximation for the original (non-augmented) machine speeds.

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, we assume all these are integers in the range $[0, T]$ (we can think of T as the largest deadline). The jobs are to be scheduled non-preemptively on m machines which can process only one job at a time. We point out that we do not require T to be poly-bounded in n. For each job $j \in J$ we refer to $[r_j, d_j]$ as span of job j, denoted by $span_j$. We use OPT to denote an optimum schedule and opt the value of it. In the weighted case, each job j has a weight/profit w_j which we receive if we schedule the job within its span. The goal in throughput maximization is to find a feasible schedule with maximum weight of jobs. Like most of the previous works, we focus on the unit weight setting (so our goal is to find a schedule with maximum number of jobs scheduled).

We also assume that for each $p \in P$, all the jobs with processing time p in an optimum solution are scheduled based on earliest deadline first rule; which says that at any time when there are two jobs with the same processing time available the one with the earliest deadline would be scheduled. This is known as Jackson rules and we critically use it in our algorithms.

Outline. We start by presenting the proof of Theorem 1 in section 3. We defer the proof of the main Lemma 9 to the journal version of the paper. Finally we present the proof of Theorem 2 in section 4.

3 Proof of Theorem 1

In this section we prove Theorem 1. For ease of exposition, we present the proof for the case of $m = 1$ machine only and then extend it to the setting of multiple machines.

3.1 Overview of the Algorithm

At a high level, the algorithm removes a number of jobs so that there is a structured near optimum solution. We show that the new instance has some structural properties that is amenable to a dynamic programming. At the lowest level of dynamic programming we have disjoint instances of the problem, each of which has a set of jobs with only a constant size set of release times and deadlines, with possibly a constant number of intervals of time being blocked from being used. For this setting we use the algorithm of Theorem 2. We start (at

level zero) by breaking the interval $[0, T]$ into a constant q (where q will be dependent on ε) number of (almost) equal size intervals, with a random offset. Let us call these intervals $a_{0,1}, a_{0,2}, \ldots, a_{0,q}$. Assume each interval has size exactly T/q, except possibly the first and last (and for simplicity assume T is a power of q). For jobs whose span is relatively large, i.e. spans at least λ (where $\frac{1}{\varepsilon} \leq \lambda \leq \varepsilon q$) intervals, while their processing time is relatively small (much smaller than T/q), based on the random choice of break points for the intervals, we can assume the probability that the jobs position in the optimum solution is intersecting two intervals is very small. Hence, ignoring those jobs (at a small loss of optimum), we can assume that each of those jobs are scheduled (in a near optimum solution) entirely within one interval. For each of them we "guess" which of the λ intervals is the interval in which they are scheduled and pass down the job to an instance defined on that interval. For jobs whose span is very small (fits entirely within one interval), the random choice of the q intervals, implies that the probability of their span being "cut" by these intervals is very small (and again we can ignore those that have been cut by these break down). For medium size spans, we have to defer the decision making for a few iterations. We then try to solve each of the q instances, independently and recursively; i.e. we break the intervals again into roughly q equal size intervals and so on. If and when an instance generated has only $O(1)$ release times or deadlines we stop the recursion and use the algorithm of Theorem 2 to find a near optimum solution. So considering the hierarchical structure of this recursion, we have a tree with at most $O(\log_q T)$ depth and at most $O(n)$ leaves, which is polynomial in the input size. There are several technical details that one needs to overcome in this paradigm. One particular technical difficulty is for some jobs we decide to re-define their span to be a smaller subset of their original span by increasing their release time a little and decreasing their deadline a little. We call this procedure, cutting their "head" and "tail". This will be a key property in making our algorithm work. We will show (Lemma 9) that under some moderate conditions, the resulting instance still has a near optimum solution. This allows us to reduce the number of guesses we have to make in our dynamic program table and hence obtain Theorem 1. We should point out that the idea of changing the span or start/finish of a job was done in earlier works. However, using speed-up of machines one could "catch up" in a modified schedule with a near optimum one. The difficulty in our case is we do not have machine speed up.

3.2 Structure of a Near Optimum Solution

Consider an optimum solution OPT. One observation we use frequently is that such a solution is left-shifted, meaning that the start time of any job is either its release time or the finish time of another job. Therefore, we can partition the jobs in schedule OPT 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 the set of possible rightmost points of these segments "slack times".

▶ **Definition 3** (Slack times). *Let slack times Ψ be the set of points t such that there is a release time r_i and a (possibly empty) subset of jobs $J' \subseteq J$, such that $t = r_i + \sum_{j \in J'} p_j$*

So the start time and finish time of each job in an optimum solution is a slack time. The following (simple) lemma bounds the size of Ψ

▶ **Lemma 4.** *There are at most n^{c+1} different possible slack times, where c is the number of distinct processing times.*

Proof. We upper bound number of distinct $r_i + \sum_{j \in J'} p_j$ values. First note that there are only 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. ◀

Given error parameter $\varepsilon > 0$ we set $q = 1/\varepsilon^2$, $k = \log_q T$ and for simplicity of presentation suppose T is a power of q. We define a hierarchical set of partitions on interval $[0, T]$. For each $0 \leq i \leq k$, I_i is a partition of $[0, T]$ into $q^{i+1} + 1$ many intervals such that, except the first and the last intervals, all have length $\ell_i = T/q^{i+1}$, and the sum of the sizes of the first and last interval is equal to ℓ_i as well. We choose a universal random offset for the start point of the first interval. More precisely, we pick a random number $r_0 \in [0, \frac{T}{q}]$ and interval $[0, T]$ is partitioned into $q + 1$ intervals $I_0 = \{a_{0,0}, a_{0,1}, ..., a_{0,q}\}$, where $a_{0,0} = [0, r_0]$, and $a_{0,t} = [(t-1)\frac{T}{q} + r_0, t\frac{T}{q} + r_0]$ for $1 \leq t \leq q-1$ and $a_{0,q} = [T - \frac{T}{q} + r_0, T]$. Note that the length of all intervals in I_0 is $\frac{T}{q}$, except the first and the last which have their length randomly chosen and the sum of their lengths is $\frac{T}{q}$.

Similarly each interval in I_0 will be partitioned into q many intervals to form partition I_1 with each interval in I_1 having length $\frac{T}{q^2}$ except the first interval obtained from breaking $a_{0,0}$ and the last interval in I_1 obtained from breaking $a_{0,q}$, which may be partitioned into less than q many, based on their lengths. All intervals in I_1 have size $\frac{T}{q^2}$ except the very first one and the very last one. We do this iteratively and break intervals of I_i (for each $i \geq 0$) into q equal sized intervals to obtain I_{i+1} (with the exception of the very first and the very last interval of I_{i+1} might have lengths smaller).

We set $\lambda = 1/\varepsilon = \varepsilon q$ and partition the jobs into classes $\mathcal{J}_0, \mathcal{J}_1, ..., \mathcal{J}_k, \mathcal{J}_{k+1}$, based on the size of their span. For each $1 \leq i \leq k$, job $j \in \mathcal{J}_i$ if $\lambda \cdot \ell_i \leq |span_j| < \lambda \cdot \ell_{i-1}$. Also $j \in \mathcal{J}_0$ (and $j \in \mathcal{J}_{k+1}$) if $\lambda \ell_0 \leq |span_j|$ (and $|span_j| < \lambda \cdot \ell_k$). For each interval $a_{i,t}$ in level I_i, we denote the set of jobs whose span is entirely inside $a_{i,t}$ by $J(a_{i,t})$.

Based on our definitions of interval levels and job classes, we can say that for each $0 \leq i \leq k$ if $j \in \mathcal{J}_i$, then $span_j$ would have intersection with at most $\lambda + 1$ (or fully spans at most $\lambda - 1$) many consecutive intervals from I_{i-1} and at least λ many consecutive intervals from I_i. Suppose $j \in I_i$ and $span_j$ has intersection with $a_{i,t_j}, a_{i,t_j+1}, ..., a_{i,t'_j}$ from I_i, then define $span_j \cap a_{i,t_j}$ and $span_j \cap a_{i,t'_j}$ as $head_j$ and $tail_j$, respectively.

We consider two classes of jobs as "bad" jobs and show that there is a near optimum solution without any bad jobs. The first class of bad jobs are those that we call "span-crossing". For each job $j \in J$, we call it "span-crossing" if $j \in \mathcal{J}_i$ for some $2 \leq i \leq k+1$ (so $\lambda \cdot \ell_i \leq |span_j| < \lambda \cdot \ell_{i-1}$), and its span has intersection with more than one interval in I_{i-2}.

▶ **Lemma 5.** *Based on the random choice of r_0 (while defining intervals), the expected number of span-crossing jobs in the optimum solution is at most $\frac{\lambda+1}{q}\text{opt} = \text{O}(\varepsilon\text{opt})$.*

Proof. Observe that because $j \in \mathcal{J}_i$, we have $|span_j| < \lambda \cdot \ell_{i-1}$. This means that the $span_j$ would have intersection with at most $\lambda + 1$ (or fully spans at most $\lambda - 1$) many consecutive intervals from I_{i-1}. Also because of the random offset while defining I_0, and since $\ell_{i-2} = q \cdot \ell_{i-1}$, the probability that job j being "span-crossing" will be at most $\frac{\lambda+1}{q}$. ◀

So, we can assume with sufficiently high probability, that there is a $(1-O(\varepsilon))$-approximate solution with no span-crossing jobs. The second group of bad jobs are defined based on their processing time and their position in the optimum solution. We then prove that by removing these type of jobs, the profit of the optimum solution will be decreased by a small factor.

For each job $j \in J$, we call it "position-crossing" if $\ell_i \leq p_j < \ell_{i-1}$ for some $2 \leq i \leq k+1$, and its position in OPT has intersection with more than one interval in I_{i-2}.

▶ **Lemma 6.** *The expected number of position-crossing jobs in OPT is at most* $\frac{1}{q}\text{opt} = O(\varepsilon^2\text{opt})$.

Proof. Consider OPT and suppose that $j \in J$ is a job with $\ell_i \leq p_j < \ell_{i-1}$. Observe that j can have intersection with at most 2 intervals in I_{i-2} because of its size. Considering our random offset to define interval levels, the probability of job j being a position-crossing (with respect to the random intervals defined) would be at most $\frac{1}{q}$ (since $p_j < \ell_{i-1} = \frac{\ell_{i-2}}{q}$). Thus, the expected number of position-crossing jobs in OPT is at most opt/q. ◀

Hence, using Lemmas 5 and 6, with sufficiently high probability, there is a solution of value at least $(1 - O(\varepsilon))$opt without any span-crossing or position-crossing jobs. We call such a solution a canonical solution.

From now on, we suppose the original instance $\mathcal{I}$ is changed to $\mathcal{I}'$ after we first defined the intervals randomly and removed all the span-crossing jobs. So we focus (from now on) on finding a near optimum feasible solution to $\mathcal{I}'$ that has no position-crossing jobs. By OPT' we mean such a solution of maximum value for $\mathcal{I}'$; we call that a canonical optimum solution. If we find a $(1 - O(\varepsilon))$-approximation to OPT' (that has no position-crossing jobs), then using the above two lemmas we have a $(1 - O(\varepsilon))$-approximate solution to $\mathcal{I}$. So with OPT' being an optimum solution to $\mathcal{I}'$ with no position-crossing jobs we let opt' be its value.

3.3 Finding a Near Optimum Canonical Solution

As a starting point and warm-up, we consider the special case where instance $\mathcal{I}'$ only consists of jobs whose processing time is relatively big compared to their span and show how the problem could be solved. Consider the extreme case where for each $j \in J$, $p_j = |span_j|$. In this case the problem will be equivalent to the problem of finding a maximum independent set in an interval graphs which is solvable in polynomial time [18]. The following theorem shows that if $p_j \geq \frac{|span_j|}{\lambda}$ for each $j \in J$ (which we call them "tight" jobs), then we can find a good approximation as well. Therefore, it is the "loose" jobs (those whose processing time p_j is smaller than $\frac{|span_j|}{\lambda}$) that make the problem difficult. (we should point out that Chuzhoy et al. [12] also considered this special case and presented a DP algorithm with run time $O(n^{\text{Poly}(\lambda)}T^4)$ however, their DP table is indexed by integer points on the time-line and the polynomial dependence on T, which can be exponential in n, is unavoidable). The idea of the dynamic program of the next theorem is the basis of the more general case that we will prove later that handles "loose" and "tight" jobs together but the following theorem is easier to understand and follow and we present it as a warm-up for the main theorem.

▶ **Theorem 7.** *If for all $j \in J$ in $\mathcal{I}'$, $p_j \geq \frac{|span_j|}{\lambda}$ then there is a dynamic programming algorithm that finds a canonical solution for instance $\mathcal{I}'$ with total profit* opt' *in time* $O(\varepsilon^{-1}n^{\varepsilon^{-2}c}\log T)$.

Proof. Recall that $k = \log_q T$ and observe that for each $0 \leq i \leq k-1$ and each $j \in \mathcal{J}_i$: $\lambda \cdot \ell_i \leq |span_j| \leq \lambda p_j$, so $\ell_i \leq p_j$. Now if we somehow know $\text{OPT}' \cap \mathcal{J}_0$ and $\text{OPT}' \cap \mathcal{J}_1$ and remove the rest of jobs in $\mathcal{J}_0$ and $\mathcal{J}_1$, then the remaining jobs (which are all in $\mathcal{J}_{i\geq 2}$) have intersection with exactly one interval in I_0 (recall we have no span-crossing or position-crossing jobs), hence we would have $q+1$ many independent sub-problems (defined on the $q+1$ sub-intervals partitioned in level 0) with jobs from $\mathcal{J}_{i\geq 2}$.

So our first task is to "guess" the jobs in $\mathrm{OPT}' \cap (\mathcal{J}_0 \cup \mathcal{J}_1)$ (as well as their positions) and then remove the rest of the jobs in $\mathcal{J}_0 \cup \mathcal{J}_1$ from J as well as the jobs whose span is crossing any of the intervals in I_0; then recursively solve the problem on independent sub-problems obtained for each interval in I_0 together with the jobs whose spans are entirely within such interval. In order to guess the positions of jobs in $\mathrm{OPT}' \cap (\mathcal{J}_0 \cup \mathcal{J}_1)$ we use the fact that each job can start at a slack time. Since jobs in $\mathcal{J}_0 \cup \mathcal{J}_1$ have size at least $\ell_1 = T/q^2$, we can have at most q^2 of them in a solution. We guess a set S of size at most q^2 of such jobs and a schedule for them; there are at most $|\Psi|^{q^2} = n^{O(q^2 c)}$ choices for the schedule of S. Then we remove the rest of $\mathcal{J}_0$ and $\mathcal{J}_1$ from J for the rest of our dynamic programming. The guessed schedule of S defines a vector $\vec{v}$ of blocked spaces (those that are occupied by the jobs from S) and for each interval $a_{0,t}$, the projection of vector $\vec{v}$ in interval $a_{i,t}$, denote it by $\vec{v}_t$, has dimension at most q ($a_{0,t}$ has length $\ell_0 = T/q$ and each job in S has length at least $\ell_1 = T/q^2$). We pass each such vector $\vec{v}$ to the corresponding sub-problem.

Consider an interval $a_{i,t} \in I_i$ for some $0 \leq i \leq k$ and $0 \leq t \leq \frac{T}{\ell_i}$. Recall that the set of jobs $j \in J$ whose span is completely inside $a_{i,t}$ is $J(a_{i,t})$. Because of the assumption of no span-crossing jobs, for each job $j \in J \setminus J(a_{i,t})$, if its span has intersection with $a_{i,t}$, then it would be in $\mathcal{J}_{i'}$ for some $i' \leq i+1$ (jobs from $\mathcal{J}_{i+2}$ are entirely within one interval of level I_i) and $|span_j|$ would be at least $\lambda \ell_{i+1}$, and hence $p_j \geq \ell_{i+1}$. Thus we can have at most $\ell_i/\ell_{i+1} = q$ such jobs. Assume we have a guessed vector $\vec{v}$ of length q where each entry of the vector denotes the start time as well as the end time of one of such jobs. This vector describes the sections of $a_{i,t}$ that are blocked for running such jobs from $J \setminus J(a_{i,t})$. The number of guesses for such vectors $\vec{v}$ is at most $n^{2q(c+1)}$ based on the bounds on the number of slack times. Given $\vec{v}$ and $J(a_{i,t})$ we want to schedule the jobs of $J(a_{i,t})$ in the free (unblocked by $\vec{v}$) sections of $a_{i,t}$.

Now we are ready to precisely define our dynamic programming table. For each $a_{i,t}$ and for each q-dimensional vector $\vec{v}$, we have an entry in our DP table A. This entry, denoted by $A[a_{i,t}, \vec{v}]$, will store the maximum throughput for an schedule of jobs running during interval $a_{i,t}$, using jobs in $J(a_{i,t})$ by considering the free slots defined by $\vec{v}$. The final solution would be $\max_S\{\sum_t A[a_{0,t}, \vec{v}_t] + |S|\}$, where the max is taken over all guesses S of jobs from $\mathcal{J}_0 \cup \mathcal{J}_1$ and $\vec{v}_t$ is the blocked area of $a_{i,t}$ based on S.

The base case is when $a_{i,t}$ has only constantly many release/deadline times. Given that we have also only constantly many processing times and $\vec{v}$ defines at most q many sections of blocked (used by bigger jobs) areas, then using Theorem 2 we can find a $(1 - O(\varepsilon))$-approximation in time Γ, where Γ is the running time of the PTAS for Theorem 2.

We can bound the size of the table as follows. First note that we do not really need to continue partitioning an interval $a_{i,t}$ if there are at most $O(1)$ many distinct release times and deadlines within that interval, since this will be a base case of our dynamic program. So the hierarchical decomposition of intervals $I_0, I_1, \ldots, I_k$ will actually stop at such an interval $a_{i,t}$ when there are at most $O(1)$ release times and deadlines. Therefore, at each level I_i of the random hierarchical decomposition, there are at most $O(n)$ intervals in I_i that will be decomposed into q more intervals in I_{i+1} (namely those that have at least a constant number of release times and deadlines within them). Thus the number of intervals at each level I_i is at most $O(nq)$ and the number of levels is at most $k = \log_q T$. Therefore, the total number of intervals in all partitions is bounded by $O(knq)$. To bound the size of the table A, each $\vec{v}$ has $n^{2q(c+1)}$ many options, based on the fact that we have at most n^{c+1} many choices of start time and end time (from the set Ψ of slacks) for each of the q dimensions of $\vec{v}$. Also as argued above, there are $O(knq)$ many intervals $a_{i,t}$ overall. So the size of table is at most $kqn^{O(qc)}$.

Now we describe how to fill the entries of the table. To fill $A[a_{i,t}, \vec{v}]$ for each $0 \leq i \leq k-1$ and $0 \leq t \leq \frac{T}{\ell_i}$, suppose $a_{i,t}$ is divided into q many equal size intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ in I_{i+1}. We first guess a subset $\tilde{J_{i,t}}$ of jobs from $\mathcal{J}_{i+2} \cap J(a_{i,t})$, to be processed during interval $a_{i,t}$ consistent with free slots defined by $\vec{v}$. This defines a new vector $\vec{v}'$ that describes the areas blocked by jobs guessed recently as well as those blocked by $\vec{v}$. Projection of $\vec{v}'$ onto the q intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ defines q new vectors $\vec{v}'_1, ..., \vec{v}'_q$. Now we check the sum of

$$A[a_{i+1,t'+1}, \vec{v}'_1] + A[a_{i+1,t'+2}, \vec{v}'_2] + ... + A[a_{i+1,t'+q}, \vec{v}'_q] + |\tilde{J_{i,t}}|$$

We would choose the $\tilde{J_{i,t}}$ which maximizes the above sum. Observe that jobs in $J(a_{i,t}) \setminus \mathcal{J}_{i+2}$ have length at most ℓ_{i+3} and because we have no position-crossing jobs, each of them is inside one of intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ and would be considered in sub-problems.

Note that to fill each entry $A[a_{i,t}, \vec{v}]$ the number of jobs from $\mathcal{J}_{i+2}$ possible to be processed in $a_{i,t}$ would be at most q^2, because of their lengths. So the total number of guesses would be at most $n^{O(q^2 c)}$. This means that we can fill the whole table in time at most $kqn^{O(q^2 c)}$, where $q = 1/\varepsilon^2$ and $k = \log_q T$. ◀

Considering Theorem 7, we next show how to handle "loose" jobs, i.e. those for which $p_j < \frac{|span_j|}{\lambda}$. Recall that for each $0 \leq i \leq k$ and for each $j \in \mathcal{J}_i$, if $span_j$ has intersection with intervals $a_{i,t_j}, a_{i,t_j+1}, ..., a_{i,t'_j}$ of I_i, then we denote $span_j \cap a_{i,t_j}$ and $span_j \cap a_{i,t'_j}$ as the head and tail of (span of) j, respectively. Our next (technical) lemma states that if we reduce the span of each loose job by removing its head and tail then there is still a near optimum solution for $\mathcal{I}'$. More specifically, for each job loose $j \in \mathcal{J}_i$ ($p_j \leq \frac{|span_j|}{\lambda}$), whose span has intersection with intervals $a_{i,t_j}, a_{i,t_j+1}, ..., a_{i,t'_j}$ of I_i, we replace its release time to start at the beginning of a_{i,t_j+1} and its deadline to be end of a_{i,t'_j-1}; so $span_j$ will be replaced with with $span_j \setminus (a_{i,t_j} \cup a_{i,t'_j})$. Let this new instance be called $\mathcal{I}''$. Note that a feasible solution for instance $\mathcal{I}''$ would be still a valid solution for $\mathcal{I}'$ as well.

▶ **Lemma 8.** *Starting from $\mathcal{I}'$, let $\mathcal{I}''$ be the instance obtained from removing the head and tail part of $span_j$ for each job $j \in J$ with $p_j \leq \frac{|span_j|}{\lambda}$. Then there is a canonical solution for $\mathcal{I}''$ with throughput at least $(1 - 120\varepsilon c)\text{opt}'$.*

Proof. The proof to the following important key lemma is deferred to the journal version of the paper:

▶ **Lemma 9** (Head and tail cutting)**.** *Consider any fixed processing time $p \in P$. Start with instance $\mathcal{I}'$ and remove only the head (or only the tail) part of $span_j$ for all jobs $j \in J$ with $p_j = p \leq \frac{|span_j|}{\lambda}$. Then there is a solution for the remaining instance with profit at least $(1 - \frac{60}{\lambda})\text{opt}'$.*

Considering Lemma 9, the proof of Lemma 8 would be easy. We just need to apply Lemma 9 for all c many distinct processing times $p \in P$ and for both "head" and "tail". Then the total loss for removing all head and tail parts would be $\frac{60}{\lambda} \cdot 2c = 120\varepsilon$ fraction:

$$\text{opt}(\mathcal{I}'') \geq (1 - \frac{60 \times 2\text{c}}{\lambda})\text{opt}' \geq (1 - 120\varepsilon\text{c})\text{opt}'.$$ ◀

The next theorem together with Lemmas 5, 6, and 9 will help us to complete the proof.

▶ **Theorem 10.** *There is a dynamic programming algorithm that finds an optimum solution for instance $\mathcal{I}''$ in time $\varepsilon^{-3} n^{O(\varepsilon^{-6} c)} \log T$.*

Before presenting the proof of this theorem we show how this can be used to prove Theorem 1 for $m = 1$.

Proof of Theorem 1. Starting from instance $\mathcal{I}$ we first reduced it to instance $\mathcal{I}'$ at a loss of $1 - O(\varepsilon)$. Then remove the head and tail part of the span for all the loose jobs to obtain instance $\mathcal{I}''$. Based on Lemma 9, we only loose a factor of $(1 - O(\varepsilon c))$ compared to optimum of $\mathcal{I}'$. Theorem 10 shows we can actually find an optimum canonical solution to instance $\mathcal{I}''$. This solution will have value at least $(1 - O(\varepsilon c))$opt using Lemmas 5, 6, and 9. To get a $(1 - \varepsilon')$-approximation we set $\varepsilon' = \varepsilon/c$ in Theorem 10. The run time will be $c^3\varepsilon^{-3}n^{O(\varepsilon'^{-6}c^7)}\log T$. ◀

Now we prove Theorem 10.

Proof. The idea of the proof is similar to that of Theorem 7. However, the presence of "loose" jobs needs to be handled too. Suppose $j \in \mathcal{J}_i$ is a loose job, so $\lambda\ell_i \leq |span_j| < \lambda\ell_{i-1}$ and $p_j \leq \frac{span_j}{\lambda} < \ell_{i-1}$. We break these loose jobs into two categories. For the loose jobs that $p_j < \ell_{i+1}$, because they are not position-crossing, their position in the final solution will have intersection with at most one interval of I_i (and so we can pass them down to lower sub-problems). But for loose jobs where $\ell_{i+1} \leq p_j < \ell_{i-1}$ we need to guess them (similar to the tight jobs) and we can do the guessing since their size (relative to ℓ_i) is big. In order to handle these guesses, we add one more vector to the DP table, and we do the guess for two consecutive levels of our decomposition as we go down the DP.

Suppose $P = \{p_1, p_2, ..., p_c\}$. For each interval $a_{i,t}$ $(0 \leq i \leq k, 0 \leq t \leq \frac{T}{\ell_i})$, q^2-dimensional vector $\vec{v}$ (where $0 \leq v_i \leq n$), (qc)-dimensional vector $\vec{u} = (u_{1,1}, \ldots, u_{q,c})$, where each $u_{\gamma,\sigma}$, $0 \leq u_{\gamma,\sigma} \leq n$, we have an entry in our DP table A. Suppose $a_{i,t}$ is partitioned into intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ in I_{i+1}. Entry $A[a_{i,t}, \vec{v}, \vec{u}]$, will store the maximum throughput of a schedule in interval $a_{i,t}$ by selecting subsets of jobs from the following two collections of jobs:

- $J(a_{i,t}) \cap \mathcal{J}_{\geq(i+2)}$
- $u_{\gamma,\sigma}$ many jobs with processing time p_σ where $p_\sigma < \ell_{i+2}$ whose span is the entire interval $a_{i+1,t'+\gamma}$, for each $1 \leq \gamma \leq q$, and $1 \leq \sigma \leq c$.

by considering the free slots defined by vector $\vec{v}$ (that describes blocked spaces by jobs of higher levels).

Vector $\vec{u}$ is defining the sets of jobs from loose jobs (from higher levels of DP table) whose span was initially much larger than ℓ_{i+1}, the guesses we made requires them to be scheduled in interval $a_{i+1,t'+\gamma}$ (of length ℓ_{i+1}) and hence their span is the entire interval $a_{i+1,t'+\gamma}$. Like before, $\vec{v}$ is defining the portions of the interval which are already used by bigger jobs (that are guessed at the higher levels), and for similar reasons as in Theorem 7, we only need to consider $\vec{v}$'s of size at most q^2 and each job listed in $\vec{v}$ will be denoted by its start position and end position (so there is $O(|\Psi|^{2q^2}) = n^{O(q^2c)}$ possible values for $\vec{v}$).

Similar to Theorem 7, suppose we start at I_0. We guess a subset of tight jobs from $\mathcal{J}_0$ to decide on their schedule. Note that tight jobs will have $p_j \geq \ell_0$. We also need to guess (and decide on their schedule) those "loose" jobs $j \in \mathcal{J}_0$ where $p_j \geq \ell_2 = T/q^3$ (since their position may cross more than one I_1 intervals in the final solution). So we guess a set $S_0 \subseteq \mathcal{J}_0$ with $|S_0| \leq q^3$ of jobs j where $p_j \geq \ell_2$ and a feasible schedule for them. This will take care of guessing tight and those loose jobs of $\mathcal{J}_0$ with $p_j \geq \ell_2$. We need to do similarly for jobs from $\mathcal{J}_1$, i.e. we need to guess a set of tight jobs j from $\mathcal{J}_1$ (note that for them $p_j \geq \ell_1$) and also guess (and decide on their schedule) those "loose" jobs $j \in \mathcal{J}_1$ with $p_j \geq \ell_2$. To do so, we guess a set $S_1 \subseteq \mathcal{J}_1$ of jobs j where $p_j \geq \ell_2 = T/q^3$ and a feasible schedule for them (given the guesses for S_0); note that $|S_0 \cup S_1| \leq q^3$ (since all of $S_0 \cup S_1$ must fit in $[0, T]$). For each such guess, their schedule projects a vector of blocked spaces (occupied time of machine). This will be vector $\vec{v}$. The projection of $\vec{v}$ to each interval $a_{0,t}$ will be $\vec{v}_t$ which is the blocked area of $a_{0,t}$. Note that although $\vec{v}$ has up to q^3 blocks, each $a_{0,t}$ can have at most q^2 blocks since each block has size at least $\ell_2 = T/q^3$ and each $a_{0,t}$ has size $\ell_0 = T/q$.

For all the other jobs in $\mathcal{J}_0 \cup \mathcal{J}_1$ that have $p_j < \ell_2$, because they are not position-crossing, we can assume their position (in the final solution) has intersection with only one interval of I_1. For all these jobs of $\mathcal{J}_0 \cup \mathcal{J}_1$, we use the assumption that there is a near optimum solution in which they are not scheduled in their head or tail. So for the jobs in $\mathcal{J}_0 \cup \mathcal{J}_1$ with processing time less than ℓ_2 we can re-define their span to a guessed interval of I_1; these guesses define the qc-dimensional vectors $\vec{u}_t$ for each of the q sub-intervals of $a_{0,t}$ at level I_1 (how many loose jobs from $\mathcal{J}_0 \cup \mathcal{J}_1$ with $p_j < \ell_2$ have their span redefined to be one of sub-intervals of $a_{0,t}$). The final solution will be $\max_{S_0,S_1}\{\sum_t A[a_{0,t}, \vec{v}_t, \vec{u}_t] + |S_0 \cup S_1|\}$, where the max is taken over all guesses $S_0 \subseteq \mathcal{J}_0$, $S_1 \subseteq \mathcal{J}_1$ and $\vec{u}_t$ as described above.

To bound the size of the table, as argued before, we would have at most $O(knq)$ many intervals in all of $I_0, I_1, \ldots, I_k$. For each of them we consider a table entry for at most $n^{O(q^2c)}$ many vectors $\vec{v}$, $n^{O(qc)}$ many vectors $\vec{u}$. So the total size of the table would be $(kq)n^{O(q^2c)}$.

Like before, the base case is when interval $a_{i,t}$ has $O(1)$ many release times and deadlines. These base cases $A[a_{i,t}, \vec{v}, \vec{u}]$ can be solved using Theorem 2 for each vector $\vec{v}$ and $\vec{u}$.

To fill $A[a_{i,t}, \vec{v}, \vec{u}]$ in general (when $0 \le i \le k$ and $0 \le t \le \frac{T}{\ell_i}$ and there are more than $O(1)$ many release times and deadlines in $a_{i,t}$), suppose $a_{i,t}$ is divided into q many equal size intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ in I_{i+1}. What we decide at this level is:

- make a decision for all the jobs $j \in \mathcal{J}_{i+2} \cap J(a_{i,t})$; those that are bigger than ℓ_{i+3} will be scheduled or dropped by making a guess; the rest we narrow down their span (guess) to be one of the lower level sub-intervals of $a_{i,t}$ and will be passed down as $\vec{u}'$ to sub-problems below $a_{i,t}$;
- make a decision for jobs in $\vec{u}$: those that are bigger than ℓ_{i+3} will be scheduled or dropped; the rest we narrow down their span (by a guess) to be one of the lower level sub-intervals of $a_{i,t}$

As in the case of I_0, we need to guess a set of tight jobs from $\mathcal{J}_{i+2} \cap J(a_{i,t})$ and some loose jobs j with $p_j \ge \ell_{i+3}$ and their positions to be processed in $a_{i,t}$ (considering the blocked areas defined by $\vec{v}$). Let S_0 with $s_0 = |S_0|$ be this guessed set. Note that $s_0 \le q^3$ since $p_j \ge \ell_{i+3} = \ell_i/q^3$. Also for each non-zero $u_{\gamma,\sigma}$ where $p_\sigma \ge \ell_{i+3}$ we guess how many of those $u_{\gamma,\sigma}$ many jobs should be scheduled and where exactly in $a_{i+1,t'+\gamma}$ (consistent with $\vec{v}$ and S_0); let S_1 be this guessed subset and $|S_1| = s_1$. Note that $s_0 + s_1 \le q^3$ and there are at most $|\Psi|^{2q^3}$ possible guesses for S_0 and S_1 together with their positions; thus a total of $n^{O(q^3c)}$ possible ways to guess $S_0 \cup S_1$ and guess their locations in the schedule. Then for each possible pair of such guessed sets S_0, S_1 we compute the resulting $\vec{v}'$; this defines the space available for the rest of the jobs in $J(a_{i,t}) \cap \mathcal{J}_{\ge i+3}$, and those defined by $\vec{u}$ where $p_j < \ell_{i+3}$ after blocking the space defined by $\vec{v}$ and the space occupied by the pair of guessed sets S_0, S_1 above. We divide $\vec{v}'$ into q many vectors $\vec{v}'_1, ..., \vec{v}'_q$, (as we divided $a_{i,t}$ into q intervals).

We also change $\vec{u}$ to $\vec{u}'$ by setting all the entries of $u_{\gamma,\sigma}$ with $p_\sigma \ge \ell_{i+3}$ to zero and guess how to distribute $\vec{u}'$ into q many (qc)-dimensional vectors $\vec{u}'_1, ..., \vec{u}'_q$ such that $\vec{u}'_1 + \vec{u}'_2 + ... + \vec{u}'_q = \vec{u}'$, where $\vec{u}'_\gamma$ is describing the number of jobs of different sizes whose span is re-defined to be one of the sub-intervals of $a_{i+1,t'+\gamma}$ at level I_{i+2}. The number of ways to break $\vec{u}'$ into $\vec{u}'_1, \ldots, \vec{u}'_q$ is bounded by $n^{O(q^2c)}$.

For all the other jobs in $\mathcal{J}_{i+2} \cap J(a_{i,t})$ that have $p_j < \ell_{i+3}$, because they are not position-crossing, we can assume their position (in the final solution) has intersection with only one interval of I_{i+2}. We also use the assumption that there is a near optimum solution in which they are not scheduled in their head or tail. So for the jobs in $\mathcal{J}_{i+2} \cap J(a_{i,t})$ with processing time less than ℓ_{i+3} we can re-define their span to a guessed sub-interval of $a_{i+1,t'+\gamma}$ at level I_{i+2}; these guesses define the qc-dimensional vectors $\vec{w}_\gamma$ for each interval $a_{i+1,t'+\gamma}$ (how many loose jobs from $\mathcal{J}_{i+2} \cap J(a_{i,t})$ with $p_j < \ell_{i+3}$ have their span redefined to be one of the q

sub-intervals of $a_{i+1,t'+\gamma}$ at level $i+2$). Observe that, by only knowing how many of w_σ many jobs with processing times p_σ are scheduled in each interval $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$ in the optimum solution, we would be able to detect which job is in which interval. The reason is that we know for each $p_\sigma \in P$, all jobs with processing time p_σ are scheduled based on earliest deadline first rule, which basically says that at any time when there are two jobs with the same processing time available the one with earliest deadline would be scheduled first.

Note that the jobs in $J(a_{i,t}) \cap \mathcal{J}_{\geq(i+3)}$ all have processing time at most ℓ_{i+3} and their spans are completely inside one of intervals $a_{i+1,t'+1}, ..., a_{i+1,t'+q}$. These jobs will be passed down to the corresponding smaller sub-problems. So for each given $\vec{v}$ and $\vec{u}$, we consider all guesses S_0, S_1 and consider the resulting $\vec{u}', \vec{v}'$ and any possible way of breaking $\vec{u}'$, and $\vec{w}$ into q parts, we check:

$$A[a_{i+1,t'+1}, \vec{v}'_1, \vec{u}'_1 + \vec{w}_1] + A[a_{i+1,t'+2}, \vec{v}'_2, \vec{u}'_2 + \vec{w}_2] + \ldots + A[a_{i+1,t'+q}, \vec{v}'_q, \vec{u}'_q + \vec{w}_q] + s_0 + s_1,$$

where s_0, s_1 are the sizes of the subsets S_0, S_1 of jobs with processing time $p_j \geq \ell_{i+3}$ guessed from $J(a_{i,t}) \cap \mathcal{J}_{i+2}$ and those from $\vec{u}$ with processing time $p_j \geq \ell_{i+3}$. We would choose the maximum over all guesses $S_0 \subseteq \mathcal{J}_{i+2} \cap J(a_{i,t})$, S_1, and all possible ways to distribute jobs with $p_j < \ell_{i+3}$ to create $\vec{u}'_\gamma$ and $\vec{w}'_\gamma$ as described above.

Note that to fill each entry $A[a_{i,t}, \vec{v}, \vec{u}]$ the number of jobs from $\mathcal{J}_{i+2} \cap J(a_{i,t})$ plus jobs from $\vec{u}$ with processing time bigger than ℓ_{i+3} possible to be processed in $a_{i,t}$ would be at most q^3, because of their lengths. So we could have at most $n^{O(q^3 c)}$ many different $\vec{v}'$ to consider. For $\vec{u}'$ and $\vec{w}$ we would have at most $n^{O(q^2 c)}$ many ways to distribute each of them into q many qc-dimensional vectors. This means that we can fill the whole table in time at most $\Gamma k q n^{O(q^3 c)} = n^{O(\varepsilon^{-6} c)} \log_q T$, where Γ is the running time of the PTAS for Theorem 2, which is at most $2^{\varepsilon^{-1} \log^{-4}(1/\varepsilon)} + \text{Poly}(n)$. So the total time will be $n^{O(\varepsilon^{-6} c)} \log T$. ◀

For extension to a constant number of machines please refer to the journal version of the paper.

4 Proof of Theorem 2

In this section we prove Theorem 2. We start by presenting a $(1-\varepsilon)$-approximation algorithm for the case of $m = 1$ that runs in time $\text{Poly}(\text{n}, \text{p}_{\max})$ where p_{max} is the largest processing time, and then show how to extend it to a PTAS. We assume that r_j's comes from a set of size R, d_j's from a set of size D where $R, D \in O(1)$. Also, we are given a vector $\vec{v}$ with $|\vec{v}| = B \in O(1)$ where each $\vec{v}_i$ is a pair $(\vec{v}_i(s), \vec{v}_i(f))$ that specifies the start and end of a blocked interval over time in which the machine cannot be used.

Our approach will be to find windows in the time-line where jobs can feasibly be scheduled in any order; these will be windows that do not contain any release time or deadline nor any blocked space. Each of these windows will be contained entirely between a pair of release times or deadlines or blocks defined by $\vec{v}$, so we can schedule jobs in a window in any order. We call the pair of release time and deadline of a job its *type*

▶ **Definition 11** (Types). *We say a job $j \in J$ is of type $t = (u, v)$ if u is the release time of job j, r_j, and if v is the deadline of job j, d_j. We let $\mathcal{T}$ denote the set of all job types.*

Since we assume $R, D \in O(1)$, therefore $|\mathcal{T}| \leq RD \in O(1)$. With these classifications, before scheduling individual jobs, we first guess how much processing time each job type t has in an optimal solution and use this guess as a budget for job processing times and maximize the number of jobs of type t scheduled given this budget. The number of such guesses will be at most $O((np_{max})^{|\mathcal{T}|}) \in O((np_{max})^{RD})$.

If a release time r_j is within a blocked interval $(\vec{v}_i(s), \vec{v}_i(f))$ we change r_j to $\vec{v}_i(f)$. Similarly if a deadline d_j is within a blocked interval $(\vec{v}_i(s), \vec{v}_i(f))$ we change d_j to $\vec{v}_i(s)$. We call the union of these release times and deadlines *straddle points*, which we denote by $\mathcal{S}$. Note that $|\mathcal{S}| \leq R + D \in O(1)$. We say a job j in a schedule straddles a straddle point if it starts before the straddle point and finishes after the straddle point (hence at the time of the straddle point the machine is busy with job j).

Let $\mathcal{S}'$ be the union of $\vec{v}_i(s)$'s and $\vec{v}_i(f)$'s (i.e. start and end points of the blocked windows defined by $\vec{v}$). For each point $\vec{v}_i(s) \in \mathcal{S}'$ we assume there is a dummy job of size $\vec{v}_i(f) - \vec{v}_i(s)$ that is being run exactly at start point $\vec{v}_i(s)$ until point $\vec{v}_i(f)$ and its position is fixed. We enumerate the points in $\mathcal{S}'' = \mathcal{S} \cup \mathcal{S}'$ so that $s_i \in \mathcal{S}''$ is the i^{th} point in increasing order.

If the number of jobs in an optimum solution is smaller than $\mathcal{S}''/\varepsilon = O((R + D + B)/\varepsilon)$ then we guess all these $O(1)$ jobs and a permutation/schedule for them in optimum and this can be done in time $n^{\mathcal{S}''/\varepsilon}(\mathcal{S}/\varepsilon)!$. So let's assume otherwise. If we remove all the jobs in optimum that straddle a straddle point (i.e. span a release time or deadline), we incur a loss of at most $|\mathcal{S}''|$ and we are left with a solution of value at least $(1 - \varepsilon)$opt. So there is a near optimum solution with no straddle job. Let us call such a near optimum solution $\mathcal{O}$. Our goal is to find such a solution.

We define *windows*, which will denote the intervals where we schedule non-straddle jobs. The free interval between two consecutive points in $\mathcal{S}''$ define a window, i.e. the free intervals between consecutive straddle points or between a dummy job and a straddle point. Let these windows be $\mathcal{W}$. Note that there are at most $R + D + B$ many windows. Before describing the algorithm, we will take the near optimal schedule $\mathcal{O}$ with no straddle jobs, and reschedule its jobs to nicely adhere to the definitions of straddle jobs and dummy jobs and allotments (total processing time allocated for each job type). We will also note that any feasible schedule can be left-shifted, meaning that the start time of any job is its release time or the end time of another job, or the start time of the interval right after a dummy job. This will then define *canonical* schedules that we can enumerate over in our algorithm. We will look at the schedule $\mathcal{O}$ and shift-left the jobs until either: (1) they hit their release time, or (2) hit the finish time of another job (dummy or not), or (3) hit another release time/deadline point. Let $\vec{a}^*$ be the allotment of jobs in each window. Lastly, we have the following observation that will be important for finding optimal canonical schedules.

▶ **Observation 12.** *Given the allotments $\vec{a}^*$, the problem of scheduling jobs of type t is independent of every other job type.*

This last observation is important as it allows our algorithm to deal with each job type independently. This is clearly true since each job type has a specified allotment that jobs of that type can be scheduled in, and the allotments of two job types do not overlap. Given ℓ windows and allotments $\vec{a}_{i,t}$ for each type t and window i we have to see what is the maximum number of jobs of type t that we can pack into these ℓ windows given the allotments for them in each window. This is a multiple knapsack problem.

4.1 Algorithm

The algorithm here is a sweep across all canonical schedules by iterating through the windows and allotments, combined with a Multiple Knapsack dynamic program to schedule jobs of each type in their corresponding allotments. For each window $W_i \in \mathcal{W}$ we guess an optimal choice of allotments in W_i, denoted $\vec{a}_i$, where $a_{i,\sigma} \in [0, np_{\max}]$ is the allotment in the ith window for jobs of type σ. We check that this choice of allotments corresponds to a canonical schedule in W_i by checking if the allotments can be scheduled feasibly as if they were jobs

(as explained below). More specifically, we let window W_1 begin from the first straddle point s_1 and check that the point $s_1 + \sum_{\sigma=1}^{|\mathcal{T}|} a_{1,\sigma}$ is at most I) the next straddle point or II) start of a dummy job (whichever comes first), if not then the check fails as the allotments are too large to fit in the window. We then repeat this process from start of window W_2 and so on. We also check that for any $a_{i,\sigma} \neq 0$, that the release time of type σ is before start of window i, and the deadline of type σ is at least end of this window, this ensures that when the jobs are scheduled in their allotments they are scheduled feasibly. We repeat this procedure for each window to get a choice of allotments $\vec{a} = \{\vec{a}_i\}_{i\in[\ell]}$. If the checks succeed for each window then the allotments can correspond to a canonical schedule.

Note that for any fixed job type, the size of an allotment for that type in a given window is in $[0, np_{\max}]$, so there are $O(np_{\max})$ many guesses for each job type in this window. There are at most $(R + D + B)$ many windows and and RD job types, so there are at most $(np_{\max})^{(R+D+B)^3}$ many allotment choices. With a choice of allotments that correspond to a canonical schedule, we apply Observation 12 to reduce the problem to solving an instance of the Multiple Knapsack problem for each job type. For the problem corresponding to jobs of type σ, say there is a knapsack m_i corresponding to every window i, of size $a_{i,\sigma}$, and for each job j of type σ there is a corresponding item, x_j in the Multiple Knapsack problem, with weight equal to p_j and profit of 1. Using a standard DP for the Multiple Knapsack problem with $R + D + B$ many knapsacks, we can solve this problem in time $O((np_{max})^{(R+B+D)^3})$. This establishes the following lemma.

▶ **Lemma 13.** *This algorithm gives an $(1-\varepsilon)$-approximation solution to Throughput Maximization with a constant number of release times and deadlines and blocked intervals and runs in time $(np_{max})^{(R+B+D)^3} + n^{R+D/\varepsilon}(R + D/\varepsilon)!$.*

4.2 A PTAS

If job sizes are not assumed to be bounded by a polynomial in n then the run-time of our algorithm has two problems. The first is that we make $O(np_{\max})$ many guesses for each allotment. The second one, we exactly solve the Multiple Knapsack problem using an algorithm with run-time that is polynomial with respect to both n and $p_{\max}$. To deal with the first problem we use a rounding and buckeing procedure with 2ϵ loss in the objective to reduce the number of guesses. To deal with the second problem, we use a PTAS for the Multiple Knapsack problem to find a schedule (e.g. [9, 22]). To deal with the first problem we will use the following lemma, which states that given a $(1-\varepsilon)$-optimal canonical schedule $\mathcal{O}$ with no straddle jobs, for each allotment $a_{i,t}$, if the allotment has at least $\lceil 1/\varepsilon^2 \rceil$ jobs then we can reduce the size of the allotment to the nearest power of $(1+\varepsilon)$ and drop jobs in order from largest to smallest until the remaining jobs can be scheduled entirely in this reduced allotment, at a loss of factor at most $1 - 2\varepsilon$.

▶ **Lemma 14.** *Given a canonical schedule $\mathcal{O}$, if we apply the above rounding procedure then the throughput of this new schedule is a $(1-2\varepsilon)$-approximation of the throughput of $\mathcal{O}$.*

Proof. Take a canonical schedule $\mathcal{O}$. For a fixed window, if an allotment has at least $\alpha = \lceil 1/\varepsilon^2 \rceil$ jobs then we round down the size of the allotment to the nearest power of $(1+\varepsilon)$. We drop jobs in order of largest to smallest until the remaining jobs fit in the allotment.

We want to show that the fraction of jobs remaining after this rounding is at least $\frac{1}{1+\varepsilon}$. The worst case for this fraction is when the jobs in this allotment is exactly α many jobs. Rounding the allotment size down to the nearest $(1+\varepsilon)$ power means that there will be at least $\lfloor \frac{\alpha}{1+\varepsilon} \rfloor$ jobs. If we let $\alpha = \frac{1}{\varepsilon^2}$, the fraction of jobs remaining will be at least $(1-2\varepsilon)$. ◀

So the number of guesses we have to make for allotment of each job type in each window will reduce from $O(np_{max})$ to $O(\log(np_{max}))$. The algorithm we use will be similar to the pseudo-polynomial time algorithm. We will sweep across the windows as before, checking that they correspond to canonical schedules. To sweep across allotments, we will guess from both allotment sizes that are powers of $(1+\varepsilon)$ and that are equal to combinations of up to $\lceil 1/\varepsilon^2 \rceil$ many job sizes. This reduces the number of guesses from $(np_{max})^{(R+D+B)^3}$ to $(\log(np_{max}))^{(R+D+B)^3}$. The reduction to the Multiple Knapsack problem is the same but instead of the pseudo-polynomial time solution, we use the PTAS due to [22] which runs in time $2^{\varepsilon^{-1}\log^{-4}(1/\varepsilon)} + \text{Poly(n)}$. The proof of the following is immediate.

▶ **Lemma 15.** *This algorithm runs in polynomial time.*

▶ **Theorem 16.** *This algorithm is a PTAS for the Throughput Maximization problem with a constant number of release times and deadlines and blocked intervals.*

Proof. We know we restrict our choices of allotments to be either the case that the size of the allotment is some rounded value, or that are combinations of up $\lceil 1/\varepsilon^2 \rceil$ many jobs. As we have shown in Lemma 14 this will give an allotment whose optimal packing is within $1-2\varepsilon$ of the optimal value for that job type and window.

Given this choice of allotments, a solution to Multiple Knapsack problems with constant many knapsacks with unit weighted jobs of arbitrary size can be solved using a PTAS due to [22]. Therefore, we find a solution that is at least a $(1-\varepsilon)(1-2\varepsilon)(1-\varepsilon) = 1-O(\varepsilon)$ factor of the optimal solution where one $1-\varepsilon$ factor is to assume there are no straddle jobs, on $1-\varepsilon$ factor is due to use of a PTAS for the Multiple Knapsack problem, and the $1-2\varepsilon$ factor is due to the rounding up the guessed sizes of allotments to powers of $1+\varepsilon$. Total time will be $(2^{\varepsilon^{-1}\log^{-4}(1/\varepsilon)} + \text{Poly}(n))(\log(np_{max}))^{(R+D+B)^3} = O(2^{\varepsilon^{-1}\log^{-4}(1/\varepsilon)} + \text{Poly}(n))$. ◀

For extension to a constant number of machines please refer to the journal version of the paper.

References

1 Micah Adler, Arnold L. Rosenberg, Ramesh K. Sitaraman, and Walter Unger. Scheduling time-constrained communication in linear networks. *Theory Comput. Syst.*, 35(6):599–623, 2002. doi:10.1007/s00224-002-1001-6.

2 Nikhil Bansal, Ho-Leung Chan, Rohit Khandekar, Kirk Pruhs, Clifford Stein, and Baruch Schieber. Non-preemptive min-sum scheduling with resource augmentation. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 614–624, 2007. doi:10.1109/FOCS.2007.46.

3 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.

4 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.

5 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.

6 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.

7 Sanjoy K. Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis E. Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992. doi:10.1007/BF00365406.

8 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.

9 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005. doi:10.1137/S0097539700382820.

10 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.

11 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.

12 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.

13 Mitre Dourado, Rosiane Rodrigues, and Jayme 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. URL: https://EconPapers.repec.org/RePEc:spr:annopr:v:169:y:2009:i:1:p:81-91:10.1007/s10479-008-0479-y.

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

15 Ulrich Faigle and Willem M. Nawijn. Note on scheduling intervals on-line. *Discrete Applied Mathematics*, 58(1):13–17, 1995. doi:10.1016/0166-218X(95)00112-5.

16 Matteo Fischetti, Silvano Martello, and Paolo Toth. The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3):395–403, 1989. doi:10.1287/opre.37.3.395.

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

18 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. North-Holland Publishing Co. Amsterdam, The Netherlands, 2004.

19 Roshdy H. M. Hafez and G. R. Rajugopal. Adaptive rate controlled, robust video communication over packet wireless networks. *MONET*, 3(1):33–47, 1998. doi:10.1023/A:1019156211458.

20 Sungjin Im, Shi Li, and Benjamin Moseley. Breaking 1 - 1/e barrier for non-preemptive throughput maximization. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 292–304, 2017. doi:10.1007/978-3-319-59250-3_24.

21 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.

22 Klaus Jansen. A fast approximation scheme for the multiple knapsack problem. In *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21-27, 2012. Proceedings*, pages 313–324, 2012. doi:10.1007/978-3-642-27660-6_26.

23 Gilad Koren and Dennis E. Shasha. D^{over}; an optimal on-line scheduling algorithm for overloaded real-time systems. In *Proceedings of the Real-Time Systems Symposium - 1992, Phoenix, Arizona, USA, December 1992*, pages 290–299, 1992. doi:10.1109/REAL.1992.242650.

24 Richard J. Lipton and Andrew Tomkins. Online interval scheduling. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA.*, pages 302–311, 1994. URL: `http://dl.acm.org/citation.cfm?id=314464.314506`.

25 Hang Liu and Magda El Zarki. Adaptive source rate control for real-time wireless video transmission. *MONET*, 3(1):49–60, 1998. `doi:10.1023/A:1019108328296`.

26 Chris N. Potts and Vitaly A. Strusevich. Fifty years of scheduling: a survey of milestones. *JORS*, 60(S1), 2009. `doi:10.1057/jors.2009.2`.

27 Kirk Pruhs, Jirí Sgall, and Eric Torng. Online scheduling. In *Handbook of Scheduling – Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004. URL: `http://www.crcnetbase.com/doi/abs/10.1201/9780203489802.ch15`.

28 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`.

29 Petra Schuurman and Gerhard J. Woeginger. Polynomial time approximation algorithms for machine scheduling: ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999.

30 Jirí Sgall. Open problems in throughput scheduling. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 2–11, 2012. `doi:10.1007/978-3-642-33090-2_2`.

31 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`.

32 David K. Y. Yau and Simon S. Lam. Adaptive rate-controlled scheduling for multimedia applications. *IEEE/ACM Trans. Netw.*, 5(4):475–488, 1997. `doi:10.1109/90.649461`.