BREAKING $1-1/e$ BARRIER FOR NON-PREEMPTIVE THROUGHPUT MAXIMIZATION[*]

SUNGJIN IM[†], SHI LI[‡], AND BENJAMIN MOSELEY [§]

Abstract. In this paper we consider one of the most basic scheduling problems where jobs have their respective arrival times and deadlines. The goal is to schedule as many jobs as possible *non-preemptively* by their respective deadlines on m identical parallel machines. For the last decade, the best approximation ratio known for the single-machine case ($m = 1$) has been $1 - 1/e - \epsilon \approx 0.632$ due to [Chuzhoy-Ostrovsky-Rabani, FOCS 2001 and MOR 2006]. We break this barrier and give an improved 0.644-approximation. For the multiple-machine case, we give an algorithm whose approximation guarantee becomes arbitrarily close to 1 as the number of machines increases. This improves upon the previous best $1-1/(1+1/m)^m$ approximation due to [Bar-Noy et al., STOC 1999 and SICOMP 2009], which converges to $1 - 1/e$ as m goes to infinity. Our result for the multiple-machine case extends to the weighted throughput objective where jobs have different weights, and the goal is to schedule jobs with the maximum total weight. Our results show that the $1 - 1/e$ approximation factor widely observed in various coverage problems is not tight for the non-preemptive maximum throughput scheduling problem.

1. Introduction.

Scheduling jobs with arrival times and deadlines is a fundamental problem in operations research and computer science. Due to this, there has been a large amount of research focusing on the topic. When jobs must be scheduled *non-preemptively* the complexity of several problems has yet to be well-understood. Nonetheless, non-preemptive job scheduling occurs frequently in practice. For instance, when jobs cannot be stopped during execution due to practical constraints or because overhead costs are prohibitively large.

A central scheduling problem is determining how to schedule jobs by their deadline. In many cases, not all jobs can be scheduled by their deadline due to insufficient job processing resources. In these situations an alternative goal is to complete as many jobs as possible by their deadline. This paper considers this problem formally known as *throughput maximization*. There are m identical machines and n jobs. Each job j has size p_j, arrival/release time r_j, and deadline d_j; all these quantities are assumed to be integers in $(0, T]$. The goal is to schedule as many jobs as possible by their deadline non-preemptively on the m machines. Non-preemptive scheduling means that once a job starts being processed at time s_j on a machine, then the job must be scheduled until time $s_j + p_j$ on the machine. A machine can process at most one job at a time. To highlight the non-preemptive aspect of the problem, in this paper we will call this problem Job Interval Scheduling (JIS). Not surprisingly, JIS has various applications in practice. For examples, see [10, 6, 8, 13] for pointers.

Throughput maximization was shown by Garey and Johnson to be NP-Hard [9]. Bar-Noy et al. [4] showed that there is an algorithm achieving an approximation ratio $1 - 1/(1 + \frac{1}{m})^m$. The approximation ratio gets better when m becomes larger. In particular, the ratio is 1/2 if $m = 1$ and converges to $1 - 1/e$ as m tends to infinity.

Later, Chuzhoy et al. [7] gave a $(1 - 1/e - \epsilon)$-approximation algorithm for a

[*]Submitted to the editors September 2017. A preliminary version of this paper appeared in IPCO 2017.

Funding: This work was funded in part by in part by NSF grants CCF-1409130, CCF-1617653, CCF-1566356, CCF-1717134, CCF-1844890, CCF-1844939, CCF-1617724, CCF-1733873, CCF-1824303, CCF-1845146 and CMMI-1938909, a Yahoo Research award, a Google Faculty Award, and a Carnegie-Bosch junior faculty chair.

[†]University of California Merced (sim3@ucmerced.edu).
[‡]University at Buffalo (shil@buffalo.edu).
[§]Carnegie Mellon University, (moseleyb@andrew.cmu.edu).

	m and T	previous results	our results
unweighted	$m = 1$ and $T = \text{poly}(n)$	$1 - 1/e - \epsilon$ [7]	0.6448
	$m = O(1)$ and arbitrary T	$1 - 1/(1 + 1/m)^m$[4]	0.6448
	arbitrary m and T	$1 - 1/(1 + 1/m)^m$ [4] $(= 1 - 1/e$ as $m \to \infty)$	$1 - \epsilon - \tilde{O}(1/\sqrt{m})$ $(= 1 - \epsilon$ as $m \to \infty)$
weighted	$T = \text{poly}(n)$	1/2 [4, 5]	$1 - \epsilon - \tilde{O}(1/\sqrt{m})$
	arbitrary T	1/2 [4, 5]	-

Table 1.1
Summary of our results compared to the previous results. Here, m is the number of machines and T is the length of the time horizon. The $\tilde{O}(\cdot)$-notation in the table is used to suppress a poly-logarithmic factor in m.

discrete version of this problem. In this version, the input is a set of intervals $\mathcal{I}_j$ in $(0, T]$[1] (which may have different lengths) for each job j. To schedule the job, the scheduler needs to select an interval from the set $\mathcal{I}_j$. A schedule is valid if the intervals selected for all the scheduled jobs are disjoint. In this problem, adding more machines does not add more generality to this problem.[2] In this paper, we will refer to the problem we consider as the *continuous* variant to distinguish it from this work. It seems that the discrete version generalizes the continuous version of the problem we consider: for each job j with arrival time r_j, deadline d_j and processing time p_j, the set of intervals for j is all sub-intervals of $(r_j, d_j]$ of length p_j with integer end-points. However, there is a small caveat: the number of intervals can be exponential in n. It was not known how to handle this tricky issue using the algorithm of [7]. Thus, when T is not polynomially bounded by n, the $\big(1 - 1/(1 + \frac{1}{m})^m\big)$-approximation due to [4] remains the state-of-art for this problem; in particular, a 1/2-approximation is the best known when $m = 1$ [4, 1, 16].

1.1. Our Results. This paper improves upon the state-of-the-art approximations for JIS for both the single-machine and multiple-machine cases. First, we show that for constant m, there is an approximation algorithm for JIS whose factor is at least $0.6448 > (1 - 1/e)$.

Theorem 1.1. *For some $\alpha_0 > 0.6448 > 1 - 1/e$ and any $\epsilon > 0$, there exists an $(\alpha_0 - \epsilon)$-approximation algorithm for the (unweighted) Job Interval Scheduling* (JIS) *problem with running time $n^{O(m/\epsilon^5)}$.*

To complement this result, we give a second algorithm whose approximation ratio approaches 1 as the number m of machines goes to infinity, improving upon the previous $1 - 1/e$ limit. Thus, we can make our approximation ratio better than $1 - 1/e$ for any m: we run the first algorithm if m is a small constant; we run the second algorithm if m is large. Indeed, our second algorithm works for the more

[1]Throughout the paper, all time intervals have integer starting and ending time points and are left-open-right-closed. This is to avoid possible confusions when we say two intervals are disjoint or intersecting each other, while at the same time guaranteeing that the number of integer times in a time interval is exactly its length, a property that makes our description simple.

[2]Suppose there are m machines. Then, in our new instance, the time horizon is $(0, mT]$, which can be viewed as the concatenation of m horizons of length T. If a job can be scheduled in $(A, B] \subseteq (0, T]$ in the original instance, it can be scheduled in $(iT + A, iT + B]$ for every $i = 0, 1, \cdots, m - 1$ in the new instance.

general weighted version of the problem, provided that T is polynomially bounded by n. In this version, each job i has some positive weight w_i and the goal is to maximize the total weight of the jobs completed by their deadline. We remark that for the unweighted version, we do not require T to be polynomially bounded.

THEOREM 1.2. *For any* $\epsilon > 0$, *there exists a* $\left(1 - O\left(\sqrt{(\log m)/m}\right) - \epsilon\right)$-*approximation for unweighted* JIS *on* m *machines. If* $T = \text{poly}(n)$, *there exists a* $\left(1 - O\left(\sqrt{(\log m)/m}\right)\right)$-*approximation for weighted* JIS *on* m *machines.*

1.2. Our Techniques. Our result in Theorem 1.2 will follow from a simple rounding procedure based on the naive LP relaxation for the problem. The algorithm scales down a naive LP solution by $(1-\epsilon)$, and applies a standard rounding technique to obtain a tentative schedule. Then the algorithm converts the tentative schedule to one that is feasible by removing jobs in a greedy manner. It is shown that the probability that a job is removed from the tentative schedule is exponentially small in m. An interesting technical contribution from this result is a method to solve the naive LP for unweighted JIS when T is not bounded, that only sacrifices a $(1-\epsilon)$-factor in the LP value. This was not known previously.

Our main technical contribution is in obtaining an $\alpha_0 - \epsilon \approx 0.6448$-approximation stated in Theorem 1.1. The algorithm is based on a slightly different variation of the configuration LP used in [7]. We highlight our algorithmic ideas as follows assuming $m = 1$.

Chuzhoy et al. considered a configuration LP to obtain an approximation ratio $1 - 1/e - \epsilon \approx 0.632$ [7]; it is known that a naive LP has an integrality gap of 2 when $m = 1$ [4, 16]. The configuration LP considered in [7] is fairly natural and builds on "blocks" of jobs: a block is a window (a time interval) together with k jobs scheduled in it, for some fixed k; all the block windows are required to be disjoint. It is straightforward to construct the set of blocks from an integral schedule: take the window in which the first k jobs are scheduled, take the window in which the second k jobs are scheduled, and so on. Chuzhoy et al. [7] used an involved preprocessing step to guess the block windows corresponding to a $(1-\epsilon)$-approximately optimal schedule. Then, for each guessed block window there are variables encoding which k jobs are scheduled within the block.

In contrast, our algorithm does not guess the block windows of an approximate solution. Rather, our configuration LP allows the blocks to be scheduled fractionally, ensuring that at most one fractional block covers every time point. Instead of partitioning time based on guessing where blocks are in an optimum solution, the time horizon $(0, T]$ is partitioned using the fractional blocks obtained from the configuration LP, into a set $\mathcal{W} = \{(t_1, t_2], (t_2, t_3], (t_3, t_4], \cdots\}$ of windows. By rounding each window in $\mathcal{W}$ randomly and independently, our first rounding recovers the $(1-1/e)$-approximation ratio of [7]. We remark that this novelty is not essential in obtaining the improved approximation ratio; the improved approximation ratio could be obtained using the involved preprocessing step and the configuration LP in [7]. However, our configuration LP yields the following byproducts: (1) our configuration LP can handle the case when T is super polynomial; (2) we can reduce the dependence of running time on ϵ from double exponential in [7] to single exponential; (3) we can obtain the improved $(\alpha_0 - \epsilon)$-approximation for any constant m.[3]

[3]As mentioned before, the work [7] focuses on the discrete version of JIS while our work does on the continuous version. The approach in [7] does not seem to easily extend to give a better than

To improve the $(1-1/e)$-approximation, we use a second rounding procedure, which works only for the continuous version of JIS. Suppose each $(r_j, d_j]$ is exactly the union of some windows in $\mathcal{W}$; in other words, $r_j = t_i$ and $d_j = t_{i'}$ for some $i < i'$. The rounding procedure is based on individual jobs as opposed to individual windows as in the first rounding procedure. We assign each job to one of the windows according to how much the job is assigned to each individual window in the LP solution. Here a crucial observation is that the job can be scheduled anywhere in such windows – the only constraint we have to ensure is that we do not assign too much volume of jobs to the same window. With an additional preprocessing step of removing "big" jobs, we can show that such a bad overflow event rarely occurs, and this leads to a $(1-\epsilon)$-approximation. Since each $(r_j, d_j]$ may not be aligned with the partition $\mathcal{W}$, we do not get this $(1-\epsilon)$-approximation in general. Among all the windows in $\mathcal{W}$ that intersect $(r_j, d_j]$, the first and the last ones are special, since we can not schedule j anywhere inside these two windows. However, if the fraction of the job j assigned to these two windows is large, then we observe that, in fact, the first rounding algorithm can give a factor better than $1-1/e$ for the probability we schedule job j. Thus, taking the better solution between those given by these two rounding procedures will lead to an approximation ratio better than $1-1/e$.

Removing Dependency on T: As mentioned above, for the continuous version of JIS, the $\frac{1}{2}$-approximation was the best known polynomial time algorithm for the single-machine case [4]. Interestingly, we also use the configuration LP to remove the dependency on T. This is somewhat counter-intuitive since the configuration LP is more complicated than the standard LP which is a special case of the configuration LP where each block has only one job. Thus, it may seem that using the configuration LP is in the opposite direction to reduce the number of LP variables to obtain a true polynomial time algorithm. One of our key observations is that if a set of k jobs are very flexible, that is, can be scheduled seamlessly in "many" places, then such a block can be added later. Then we show one job can be kicked out to schedule k additional jobs. A similar configuration LP is used in the $\left(1 - O\left(\sqrt{\frac{\log m}{m}}\right)\right)$-approximation to reduce the dependence on T, although only the naive LP is needed when $T = \text{poly}(n)$.

1.3. Related Work.

A simple greedy algorithm that schedules a job with the earliest deadline is known to be a $\frac{1}{2}$-approximation for the single-machine case [1, 16]. There are $\frac{1}{2}$-approximations known for the weighted throughput objective in the multiple-machine setting [4, 5]. Bansal et al. [2] considered JIS when the algorithm is given resource augmentation and gave an $O(1)$-speed 1-approximation. If preemption is allowed, it is known that if $m = 1$ then there exists a polynomial time optimal algorithm [3]. When $m \geq 2$ then the problem becomes NP-Hard [11]. To see why the problem is hard, note that if all jobs have the same release time and deadline then finding the minimum number of machines to schedule the jobs on is effectively the bin packing problem. The problem has also been considered in the online setting [14, 12].

1.4. Organization.

We first prove our theorems under certain simplifying assumptions and then later give the full proof. Specifically, in Section 2, we show Theorem 1.1 when $T = \text{poly}(n)$; recall that $(0, T]$ is the time horizon we are considering. To illustrate how to generalize this proof, we consider the case where $m = 1$ and T is large in Section 3. Finally, the proof is extended to consider large T and any m in Section 4. We continue to show Theorem 1.2 in Section 5 under the assumption

$1-1/e$-approximation for multiple machines.

that T is polynomially bounded. The proof when T is large can be found in Section 6. In the appendix, we prove a technical lemma that is useful in the analysis.

2. Proof of Theorem 1.1 when $T = \text{poly}(n)$.

The algorithm is based on a configuration LP relaxation for the problem. The algorithm will use this relaxation to partition the time horizon into disjoint windows. We remark that this step can replace the involved preprocessing step of [7]. With the definition of windows in place, the algorithm runs the rounding procedure of [7]; this will give a $(1 - 1/e - \epsilon)$-approximation for the unweighted case. To obtain the improved $(\alpha_0 - \epsilon)$ approximation ratio, a different rounding procedure is used and the final solution is obtained by choosing the better solution from the two procedures. This will establish Theorem 1.1 when $T = \text{poly}(n)$.

2.1. Linear Programming Relaxation for Job Interval Scheduling.

An important element we use in the LP for JIS is a *block*, defined below.

DEFINITION 2.1. *A block B is a triple $B = (L_B, R_B, \mathcal{J}_B)$ where L_B and R_B are two integer time points such that $0 \le L_B < R_B \le T$, and $\mathcal{J}_B$ is a subset of jobs that can be scheduled in the interval $(L_B, R_B]$ non-preemptively on m machines. The* size *of a block B, which is denoted as w_B, is defined as the number of jobs in $\mathcal{J}_B$. The* block window *of B is defined as $(L_B, R_B]$.*

We assume B is associated with a specific schedule of $\mathcal{J}_B$ in $(L_B, R_B]$; this can be done, for example, by defining a lexicographic order over all schedules, and associate B with the first valid schedule of $\mathcal{J}_B$ in $(L_B, R_B]$ according to this order.

We define $\mathcal{B}$ to be the set of all blocks, and $\mathcal{B}^* \subseteq \mathcal{B}$ to be the set blocks B with either $w_B = \Delta$, or $R_B = T$ and $w_B < \Delta$. In the integer programming (IP) for JIS we are defining now, we only consider blocks in $\mathcal{B}^*$. Let $k = \lceil 3/\epsilon \rceil$ and $\Delta = 2mk^5$; recall that ϵ is a parameter that stands for the proximity to the desired approximation factor. The IP for JIS is defined as follows.

$$(\mathsf{IP}_{\mathsf{conf}}) \qquad \max \sum_{B \in \mathcal{B}^*} w_B \cdot x_B$$

$$(2.1) \qquad \sum_{B \in \mathcal{B}^* : L_B < t \le R_B} x_B \le 1 \qquad \forall t \in [T]$$

$$(2.2) \qquad \sum_{B \in \mathcal{B}^* : j \in \mathcal{J}_B} x_B \le 1 \qquad \forall j \in \mathcal{J}$$

$$x_B \in \{0, 1\} \qquad \forall B \in \mathcal{B}^*$$

In the above IP, Constraint (2.1) ensures that block windows are disjoint, and Constraint (2.2) requires each job to be scheduled at most once. Note that the number of constraints in (2.1) is polynomially bounded when $T = \text{poly}(n)$ — as mentioned earlier, we discuss how to handle large T in later sections.

It is easy to see that any solution to the IP gives a valid schedule. On the other hand, not every schedule can be converted to a feasible IP solution when $m > 1$: Consider a schedule on $m = 2$ machines where the job scheduling intervals on the two machines are $(0, 2], (2, 4], (4, 6], \cdots, (T - 2, T]$, and $(1, 3], (3, 5], (5, 7], \cdots, (T - 1, T]$ respectively, for a large even integer T; then we have to throw away many jobs when breaking the time horizon into blocks. Thus, the IP may not give the optimum throughput. However, we show that the loss is small in the following lemma.

Lemma 2.2. *The value of* ($\mathsf{IP}_{\mathsf{conf}}$) *is at least* $\frac{\Delta}{\Delta+m-1}$ *times the optimum throughput.*

Proof. Fix an optimal schedule. Given the optimum schedule, sort all the jobs according to their completion time. Let $L = 0$ initially. In each iteration, take the first Δ jobs $\mathcal{J}'$ from the sequence and let R be the completion time of the Δ-th job; if there are less than Δ jobs in the sequence, let $\mathcal{J}'$ be all the jobs in the sequence and let $R = T$. Create a block $B = (L, R, \mathcal{J}')$ and set $x_B = 1$; clearly $B \in \mathcal{B}^*$. Then, remove all jobs whose starting time is before R from the sequence. Then let $L = R$ and start a new iteration. The process ends when the sequence becomes empty. It is easy to see that the blocks created have disjoint windows. Moreover, if $|\mathcal{J}'| = \Delta$, we remove at most $\Delta + m - 1$ jobs from the sequence: other than the Δ jobs in $\mathcal{J}'$, we may remove at most $m - 1$ extra jobs that are scheduled intersecting the interval $(R - 1, R]$. If $|\mathcal{J}'| < \Delta$ in the last iteration, we only remove $|\mathcal{J}'|$ jobs from the sequence. Thus, the value of the IP is at least $\frac{\Delta}{\Delta+m-1}$ times the optimum throughput.[4] ☐

The LP is obtained by relaxing the constraints $x_B \in \{0, 1\}$ to $x_B \geq 0$; we denote the LP by ($\mathsf{LP}_{\mathsf{conf}}$). Note that the running time of solving the LP is $n^{O(\Delta)} = n^{O(m/\epsilon^5)}$. Let $x^* \in [0, 1]^{\mathcal{B}^*}$ denote the optimal solution to the above LP. Let $\mathrm{OPT}_{\mathrm{LP}} = \sum_{B\in\mathcal{B}^*} w_B x^*_B$ denote the optimal LP objective.

2.2. Preprocessing. In the preprocessing step, the time horizon $(0, T]$ is broken into a set $\mathcal{W}$ of disjoint intervals, which we call *base* windows to distinguish them from job windows and block windows. This preprocessing step constructs a new solution $x' \in [0, 1]^{\mathcal{B}}$. The primary goals are:

- to make each block window completely contained in a base window — this makes the rounding procedures easier to apply;
- to ensure that each block contains no "big" jobs compared to its block window size — intuitively, big jobs are less flexible to schedule; and
- to achieve the above while almost preserving the LP objective.

Towards achieving the last goal, we will upper bound the number of jobs discarded during the partitioning because their scheduling interval is not fully contained in a base window or their size is big. Here, we make use of the fact that each block contains a large number of jobs, and therefore, we can afford to discard some jobs.

More precisely, we prove the following lemma. Recall that $\mathcal{B} \supseteq \mathcal{B}^*$ is the set of all blocks; so unlike x^*, the vector x' involves blocks outside $\mathcal{B}^*$.

Lemma 2.3. *In time* $n^{O(m/\epsilon^5)}$, *we can construct a vector* $x' \in [0, 1]^{\mathcal{B}}$ *described as a list of* (B, x'_B) *pairs with* $x'_B > 0$, *such that the following holds:*

(2.3a) *For every block* $B \in \mathcal{B}$ *with* $x'_B > 0$, *the block window* $(L_B, R_B]$ *is fully contained in some base window in* $\mathcal{W}$.

(2.3b) *For every block* $B \in \mathcal{B}$ *with* $x'_B > 0$ *and every* $j \in \mathcal{J}_B$, *we have* $p_j < (R_B - L_B)/k^3$.

(2.3c) *If* $\mathrm{OPT}_{\mathrm{LP}} \geq \Delta$, *then* $\sum_{B\in\mathcal{B}} w_B x'_B \geq (1 - \epsilon/3)\mathrm{OPT}_{\mathrm{LP}}$.

(2.3d) *For all base windows* $(L, R] \in \mathcal{W}$, $\sum_{B\in\mathcal{B}:(L_B,R_B]\subseteq(L,R]} x'_B \leq 1 + 1/k$.

(2.3e) *For all jobs* j, $\sum_{B\in\mathcal{B}:j\in\mathcal{J}_B} x'_B \leq 1$.

Proof. We show how to break $(0, T]$ into base windows and obtain $x' \in [0, 1]^{\mathcal{B}}$ from $x^* \in [0, 1]^{\mathcal{B}^*}$. Note that the blocks in the support of x^* can overlap with one another. To create x', we iteratively cut at the time point when an additional $1/k$ fraction of blocks end according to x^*. Formally, we associate each integer time-point

[4] It is worth noting that here we crucially use the assumption that jobs have uniform weights.

$t \in (0, T]$ with a weight $e_t := \sum_{B \in \mathcal{B}^*: R_B = t} x^*_B$, which is the sum of x^*_B over all blocks B ending at time t. Let $L = 0$ and $\mathcal{W} = \emptyset$ initially. Each iteration works as follows. Let R be the first time point such that $\sum_{t=L+1}^{R} e_t \geq 1/k$, or let $R = T$ if no such time point exists — note that the sum is counted from time $L + 1$. Create a base window $(L, R]$ and add it to $\mathcal{W}$. Let $L = R$ and start a new iteration. The procedure terminates when $L = T$. See Figure 2.1 for illustration of the partitioning process.

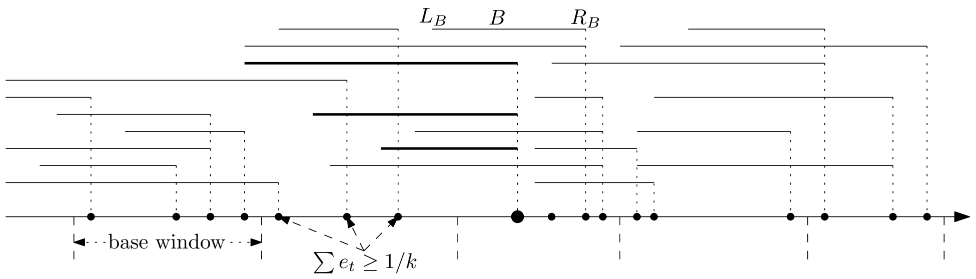

FIG. 2.1. *Partitioning $(0, T]$ into base windows. Each horizontal solid line represents a block. The x-value of a block B, x^*_B, is added to e_{R_B}; for example, the e-value at the time denoted by the big dot is the sum of x^*-values of the 3 blocks denoted by thick lines. The dashed lines below the time horizon give the partitioning of $(0, T]$ into base windows. For each base window $(L, R]$ except for the last one, we have $\sum_{t \in (L,R]} e_t \geq 1/k$.*

Once we defined the base windows $\mathcal{W}$, for every B with $x^*_B > 0$, we cut B into multiple blocks at the boundaries of the base windows. Formally, for every base window $(L, R] \in \mathcal{W}$ that intersects $(L_B, R_B]$, we create a block $B' = (\max\{L, L_B\}, \min\{R, R_B\}, \mathcal{J}')$ where $\mathcal{J}'$ is the set of jobs in $\mathcal{J}_B$ whose scheduling intervals are contained in $(L, R]$. Then we remove *big jobs* from $\mathcal{J}_{B'} = \mathcal{J}'$: a job j in $\mathcal{J}_{B'}$ is said to be big compared to B' if $p_j \geq (R_{B'} - L_{B'})/k^3$.[5] For each block B' created from B, we increase $x'_{B'}$ by x^*_B (initially, all $x'_{B'}$'s are 0). Notice that the jobs across the boundaries of base windows are deleted in this process.

We have constructed a set $\mathcal{W}$ of disjoint base windows and derived a new fractional solution $x' \in [0,1]^{\mathcal{B}}$ from $x^* \in [0,1]^{\mathcal{B}^*}$; the running time of the algorithm is clearly bounded by $n^{O(m/\epsilon^5)}$. It suffices to show that x' satisfies the required properties.

Properties (2.3a), (2.3b) and (2.3e) are immediately true. To see Property (2.3d), consider a base window $(L, R] \in \mathcal{W}$. We know that $\sum_{B \in \mathcal{B}^*: R \in (L_B, R_B]} x^*_B \leq 1$ due to Constraints (2.1). Due to the way we defined R, we have $\sum_{B \in \mathcal{B}^*: R_B \in (L, R-1]} x^*_B = \sum_{t=L+1}^{R-1} e_t < 1/k$; that is, at most $1/k$ fractional block can end during $(L, R - 1]$. Notice that if $(L_B, R_B]$ intersects $(L, R]$, i.e, if $\max\{L_B, L\} < \min\{R_B, R\}$, then either $R \in (L_B, R_B]$ or $R_B \in (L, R - 1]$. So, overall, we have $\sum_{B \in \mathcal{B}^*: (L_B, R_B] \text{ intersects } (L,R]} x^*_B \leq 1 + 1/k$. This implies Property (2.3d).

It now remains to show Property (2.3c). When a block B includes a job j, we say that in the LP solution x^*, the job is (fractionally) scheduled by x^*_B units on its scheduling interval specified by the block. Observe that in x^* there are at most m units of jobs scheduled across the boundary of two adjacent base windows. This is because due to Constraints (2.1), there are at most one unit of blocks whose window includes time R and each block has at most m jobs whose scheduling interval includes the time R. Each time (except for the last one) we built a base window, we collected

[5]As we shall show, the number of big jobs in a block will be small. This is another place where we rely on the assumption that jobs have uniform weights.

at least units of $1/k$ fractional blocks, and thus Δ/k units of fractional jobs within the blocks. So the total number of boundaries is at most $\mathrm{OPT}_{\mathrm{LP}}/(\Delta/k) = k\mathrm{OPT}_{\mathrm{LP}}/\Delta$. Thus we discarded at most $km\mathrm{OPT}_{\mathrm{LP}}/\Delta$ units of fractional jobs because of crossing boundaries. Also at most mk^3 units of fractional big jobs are discarded from each base window. Thus at most $mk^3 \cdot (k\mathrm{OPT}_{\mathrm{LP}}/\Delta + 1)$ units of fractional big jobs are removed. Since $\mathrm{OPT}_{\mathrm{LP}} \geq \Delta$ and $k \geq \frac{3}{\epsilon} \geq 3$, the total unit of fractional jobs removed is at most $2mk^4\mathrm{OPT}_{\mathrm{LP}}/\Delta = \mathrm{OPT}_{\mathrm{LP}}/k \leq \epsilon\mathrm{OPT}_{\mathrm{LP}}/3$. Hence Property (2.3c) follows. ◻

2.3. Rounding. As mentioned before, there are two rounding procedures we use to round $x' \in [0,1]^{\mathcal{B}}$ to obtain an improved approximation. We will first present the two rounding procedures and show how combining the two leads to a better approximation. Throughout this section, we assume $\mathrm{OPT}_{\mathrm{LP}} \geq \Delta$ since otherwise we can easily find an optimal solution by considering every set of Δ jobs in $n^{O(\Delta)}$ time.

2.3.1. The First Rounding Procedure. In this subsection, we describe and analyze the first rounding algorithm. The first rounding utilizes indepdent randomized rounding. The procedure independently samples a block from the set of blocks contained in each base window $\mathcal{W}$. Formally, for each base window $(L, R] \in \mathcal{W}$, we sample 0 or 1 block, so that for each block $B \in \mathcal{B}$ with $(L_B, R_B] \subseteq (L, R]$, B is sampled with probability exactly $\frac{x'_B}{1+1/k}$. This is well defined due to Property (2.3d) — it says that there are only $(1+1/k)$ fractional blocks to be considered for each base window. Then our output schedule is the concatenation of all the sampled blocks; they must have disjoint windows since windows in $\mathcal{W}$ are disjoint. If a job is contained in more than one sampled block, we only keep it in one of these blocks. This completes the description of the first rounding.

A standard analysis of independent rounding can be used to show that each job is scheduled with probability at least $(1-1/e)/(1+1/k)$ times the fraction by which the job is scheduled. To derive an improved approximation better than $1-1/e$, we will perform a more careful analysis.

Now it is good time for us to introduce some new notations that will be used for analyzing both rounding procedures. Let's focus on each job j. Let $\mathcal{W}_j$ be the set of base windows that intersect $(r_j, d_j]$. We call the first and the last of the base windows the *boundary base windows* for j and the other base windows in $\mathcal{W}_j$ the *non-boundary base windows* for j (see Figure 2.2). Let $\mathcal{W}_j^{\mathsf{nb}}$ and $\mathcal{W}_j^{\mathsf{b}}$ be the set of non-boundary and boundary base window jobs for j respectively. Thus $\mathcal{W}_j = \mathcal{W}_j^{\mathsf{nb}} \uplus \mathcal{W}_j^{\mathsf{b}}$. In the case where $\mathcal{W}_j$ contains only one base window, we assume the window is a boundary base window. Notice that all non-boundary base windows for j are are completely contained in $(r_j, d_j]$. Define

$$f_{j,W} := \frac{1}{1+1/k} \sum_{B \in \mathcal{B}: (L_B, R_B] \subseteq W, j \in \mathcal{J}_B} x'_B, \quad \forall W \in \mathcal{W}_j, \tag{2.3}$$

$$a_j := \sum_{W \in \mathcal{W}_j^{\mathsf{nb}}} f_{j,W}, \quad \text{and} \tag{2.4}$$

$$b_j := \sum_{W \in \mathcal{W}_j^{\mathsf{b}}} f_{j,W}. \tag{2.5}$$

That is, $f_{j,W}$ is the fraction by which job j is scheduled in W, scaled down by $1+1/k$, and a_j and b_j are respectively the fraction by which job j is scheduled in its non-boundary and boundary base windows, scaled down by $1+1/k$. Notice that $a_j + b_j \leq \frac{1}{1+1/k} < 1$.

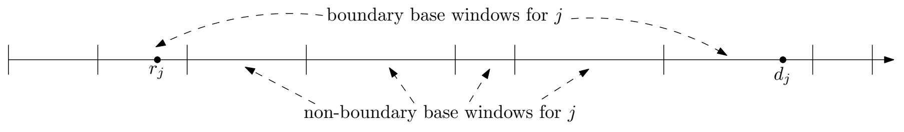
boundary base windows for j
r_j
d_j
non-boundary base windows for j

FIG. 2.2. *Boundary and non-boundary base windows for job j.*

Job j may appear in multiple base windows, more precisely in blocks contained in multiple base windows. We give a refined analysis of the probability that each job is scheduled by the first rounding by distinguishing how much the job is scheduled in its boundary and non-boundary base windows.

LEMMA 2.4. *The first rounding schedules each job j with probability at least $1 - (1 - b_j/2)^2 e^{-a_j}$.*

Proof. Consider any fixed job j. Job j has only one boundary base window only if $(r_j, d_j]$ is fully contained in the boundary base window. In this case, job j is scheduled with probability exactly $b_j \geq 1-(1-b_j/2)^2$ and the lemma immediately follows since $a_j = 0$. Suppose j has two boundary base windows and call them W_j^1 and W_j^2. Let $b_j^1 = f_{j,W_j^1}$ and $b_j^2 = f_{j,W_j^2}$ be the contribution to b_j made by each of the two boundary base windows; so $b_j = b_j^1 + b_j^2$. Then, j is scheduled with probability

$$\begin{aligned} 1 - (1-b_j^1)(1-b_j^2) \prod_{W \in \mathcal{W}_j^{\mathsf{nb}}} (1 - f_{j,W}) &\geq 1 - (1-b_j^1)(1-b_j^2) \prod_{W \in \mathcal{W}_j^{\mathsf{nb}}} e^{-f_{j,W}} \\ = 1 - (1-b_j^1)(1-b_j^2)e^{-a_j} &\geq 1 - (1-b_j/2)^2 e^{-a_j}, \end{aligned}$$

where we used the well-known inequality $e^x \geq 1 + x$. ∎

2.3.2. The Second Rounding Procedure. The second rounding makes use of the flexibility of non-boundary base windows. This rounding procedure completely ignores the boundary base windows and assigns jobs *individually.* Consider each job j together with its non-boundary base windows $\mathcal{W}_j^{\mathsf{nb}}$. The fractional solution x' tells us how much job j can be scheduled in each of its base windows, and we randomly assign the job to one of them exactly as the fractional solution suggests. Then, what is the probability that job j cannot be scheduled since a lot of jobs are assigned to the same base window? We can show such a probability is tiny by scaling down the assignment probability slightly and using the fact that all jobs are small compared to base windows.

Formally, the second rounding algorithm is as follows. Consider each job j. We assign job j, independent of other jobs, to one of its non-boundary base windows $W \in \mathcal{W}_j^{\mathsf{nb}}$ with probability $f_{j,W}$. Notice that $a_j = \sum_{W \in \mathcal{W}_j^{\mathsf{nb}}} f_{j,W} \leq \frac{1}{1+1/k}$. So, the probability that j is assigned to some base window is exactly a_j. For every $W \in \mathcal{W}$, let $\mathcal{J}(W)$ be the set of jobs assigned to the base window W. Notice that for every job $j \in \mathcal{J}(W)$, we have $(r_j, d_j] \supseteq W$ since W is a non-boundary base window for j. So, j can be scheduled anywhere inside W. Then we try to assign $\mathcal{J}(W)$ to the m machines using the following simple greedy packing procedure. Initially, all jobs in $\mathcal{J}(W)$ are not assigned. For every job $j \in \mathcal{J}(W)$ in any order we do the following: If there is a machine i such that assigning j to i will not make the total size of jobs assigned to i exceed the length of W, then we assign j to i; otherwise we declare failure. If the greedy packing failed, then we discard all jobs in $\mathcal{J}(W)$. Otherwise, all jobs in $\mathcal{J}(W)$ are assigned to machines and the total size of jobs assigned to each machine is at most the length of W. Then we can schedule $\mathcal{J}(W)$ on the m machines within the window W.

The remainder of this subsection is devoted to proving the following:

LEMMA 2.5. *The second rounding successfully schedules each job j with probability at least $(1-\epsilon/3)a_j$.*

To establish Lemma 2.5, we first show that the greedy packing is quite effective using the facts that each job j in $\mathcal{J}(W)$ can be scheduled anywhere within the window and all jobs in $\mathcal{J}(W)$ are small compared to W.

LEMMA 2.6. *Consider a base window $W = (L, R]$. If the total size of jobs in $\mathcal{J}(W)$ is no greater than $(1-1/k^3)m(R-L)$, then all jobs in $\mathcal{J}(W)$ can be scheduled on m machines within the window W.*

Proof. Properties (2.3a) and (2.3b) imply that all jobs in $\mathcal{J}(W)$ have sizes no greater than $(R-L)/k^3$. If the greedy packing procedure for W failed, then it must be the case that each of the m machines got assigned jobs of total size greater than $R-L-(R-L)/k^3 = (1-1/k^3)(R-L)$ before the failure. But this contradicts the assumption that the total size of jobs in $\mathcal{J}(W)$ is at most $(1-1/k^3)m(R-L)$. ∎

We show that the bad event that job j is discarded after randomly assigned to W — as the greedy packing fails to schedule all jobs in $\mathcal{J}(W)$ — happens with a low probability. Towards this end, we make following observation.

LEMMA 2.7. *For any $W \in \mathcal{W}$, the total size of jobs in $\mathcal{J}(W)$ is at most $m(R-L)/(1+1/k)$ in expectation.*

Proof. We fix the base window $W = (L, R] \in \mathcal{W}$. It is the case that $\sum_{B\in\mathcal{B}:t\in(L_B,R_B]} x'_B \le 1$ for every $t \in (L, R]$; this holds due to Constraint (2.1) and the way we obtain x' from x^*. Thus, $\sum_{B\in\mathcal{B}:(L_B,R_B]\subseteq W}(R_B - L_B)x'_B = \sum_{t\in W}\sum_{B\in\mathcal{B}:t\in(L_B,R_B]} x'_B \le (R-L)$. The probability that a job j with $W \in \mathcal{W}_j^{\mathsf{nb}}$ is assigned to W is exactly $f_{j,W}$. So the expected total size of jobs in $\mathcal{J}(W)$ is

$$\begin{aligned}
\sum_{j:W\in\mathcal{W}_j^{\mathsf{nb}}} f_{j,W}p_j &= \frac{1}{1+1/k}\sum_{j:W\in\mathcal{W}_j^{\mathsf{nb}}}\ \sum_{B\in\mathcal{B}:(L_B,R_B]\subseteq W, j\in\mathcal{J}_B} x'_B p_j \\
&\le \frac{1}{1+1/k}\sum_{B\in\mathcal{B}:(L_B,R_B]\subseteq W} x'_B \sum_{j\in\mathcal{J}_B} p_j \\
&\le \frac{1}{1+1/k}\sum_{B\in\mathcal{B}:(L_B,R_B]\subseteq W} x'_B m(R_B - L_B) \\
&\le \frac{m}{1+1/k}(R-L).
\end{aligned}$$

The equality follows from the definition of $f_{j,W}$, and the second inequality from the fact that the total size of jobs in $\mathcal{J}_B$ is at most $m(R_B - L_B)$. ∎

This claim, together with the fact that all jobs are small compared to the base window, will allow us to show that the bad event happens with a low probability. To show this, fix a base window $W = (L, R]$. The upper bound in the following lemma easily follows from a well known concentration inequality.

LEMMA 2.8. *For any window $W = (L, R]$, the total size of jobs in $\mathcal{J}(W)$ is at most $(1-1/2k)m(R-L)$ with probability at least $1-\epsilon/3$.*

Proof. Let X_j be p_j if job j is in $\mathcal{J}(W)$, and otherwise 0. Note that $X_j \leq (R-L)/k^3$. Let $Z = \sum_j X_j$. By Lemma 2.7, we know that $\mu := \mathbf{E}[Z] \leq m(R-L)/(1+1/k) \leq (1-0.9/k)m(R-L)$ when k is large enough. By adding enough dummy random variables, we may assume $\mu = (1-0.9/k)m(R-L)$; this only increases $\Pr[Z \geq (1-1/2k)m(R-L)]$. We use the following concentration inequality (see Theorem 2.3 in [15]).

Theorem 2.9. *Let Z be the sum of n independent random variables where each random variable takes value in $[0,K]$. Let $\mu = E[Z]$. Then for any $\lambda \in [0,1]$, it is the case that*

$$\Pr\Big[Z \geq (1+\lambda)\mu\Big] \leq e^{-\lambda^2\mu/3K}.$$

Then, we have

$$\begin{aligned}
&\Pr[Z \geq (1-1/2k)m(R-L)]\\
&\leq \exp\Big(-\frac{(0.4/k)^2/(1-0.9/k)^2 \times (1-0.9/k)m(R-L)}{3(R-L)/k^3}\Big)\\
&= \exp\Big(-\frac{0.16km}{3(1-0.9/k)}\Big)\\
&\leq \exp(-k/20),
\end{aligned}$$

which is at most $\epsilon/3$ when ϵ is small enough. ∎

Recall that each job j is assigned to one of its non-boundary windows with probability a_j. Conditioned on j being assigned to a fixed base window W, the total size of jobs in $\mathcal{J}(W) \setminus \{j\}$ doesn't exceeds $(1-1/2k)m(R-L)$ with probability at least $(1-\epsilon/3)$ due to Lemma 2.8. Then, as observed in Lemma 2.6, all jobs in $\mathcal{J}(W)$ can be scheduled within W via greedy packing as $(1-1/2k)m(R-L)+p_j \leq (1-1/2k)m(R-L)+(1/k^3)m(R-L) \leq (1-1/k^3)m(R-L)$. Thus, the probability that job j is scheduled due to the second rounding is at least $(1-\epsilon/3)a_j$. This completes the proof of Lemma 2.5.

2.3.3. Combining the Two Rounding Procedures. We complete our rounding by taking the better between the two rounding solutions. For the sake of analysis, we lower bound the probability stated in Lemma 2.4, by a linear combination of a_j and b_j. The proof follows from a simple algebra and is given in Appendix A.

Lemma 2.10. *For all a_j, b_j such that $0 \leq a_j + b_j \leq 1$, $\big(1-(1-b_j/2)^2e^{-a_j}\big) \geq \lambda_1 a_j + \lambda_2 b_j$ where $\lambda_1 = 0.62$ and $\lambda_2 = 0.69$.*

From Lemmas 2.4 and 2.5, the expected number of jobs we schedule is at least

$$\begin{aligned}
&(1-\epsilon/3)\max\Big\{\sum_j \big(1-(1-b_j/2)^2e^{-a_j}\big), \sum_j a_j\Big\}\\
&\geq (1-\epsilon/3)\max\Big\{\lambda_1\sum_j a_j + \lambda_2\sum_j b_j, \sum_j a_j\Big\}\\
&\geq (1-\epsilon/3)\frac{\lambda_2}{\lambda_2-\lambda_1+1}\Big(\sum_j a_j + \sum_j b_j\Big).
\end{aligned}$$

Let $\alpha_0 = \frac{\lambda_2}{\lambda_2-\lambda_1+1} > 0.64485$. Notice that $\sum_j a_j + \sum_j b_j$ is the total number of jobs scheduled by the solution x', scaled down by $1+1/k$, which is at least $(1-1/k)(1-$

$\epsilon/3)\mathrm{OPT}_{\mathrm{LP}}$, due to Property (2.3c). Noticing that $\mathrm{OPT}_{\mathrm{LP}}$ is at least $\frac{\Delta}{\Delta+m-1} \geq (1-\epsilon/3)$ times the optimum throughput, our approximation ratio is at least $(1-\epsilon/3)(1-1/k)(1-\epsilon/3)(1-\epsilon/3)\alpha_0 \geq (1-4\epsilon/3)\alpha_0 \geq \alpha_0 - \epsilon$. This proves Theorem 1.1.

3. Proof of Theorem 1.1 when $m = 1$ and T is large. In this section, we remove the dependency of running time on T for the $(\alpha_0-\epsilon)$-approximation algorithm in Section 2. Since the case $m = 1$ is simpler yet uses the key ideas, we consider this case first to illustrate them. The case where $m \geq 2$ is considered in Section 4. Throughout this section, it is assumed that $\Delta := 2mk^5 = 2k^5, k = \lceil 3/\epsilon \rceil$.

We first give an overview of our approach. The main issue with $\mathsf{LP}_{\mathsf{conf}}$ is that the number of its variables could become super-polynomial in n when T is large because there is a variable x_B corresponding to each block $B = (L_B, R_B, \mathcal{J}_B)$, where $0 \leq L_B < R_B \leq T$. To keep the number of variables polynomially bounded, we will identify polynomially many blocks while ensuring the optimum value of the LP defined over the corresponding variables is almost as large as the optimum throughput.

Towards this end, we slightly modify the definition of a block by replacing $\mathcal{J}_B$ with a sequence of jobs. Since we will only consider blocks of size Δ, which is a constant for any fixed $\epsilon > 0$, specifying the order of the jobs will have no effect on the asymptotic running time. Note that in this section the following definition overrides Definition 2.1.

Definition 3.1. *[Modification of Definition 2.1] A block B is a triple $B = (L_B, R_B, \sigma_B)$ where L_B and R_B are two integer time points such that $0 \leq L_B < R_B \leq T$, and σ_B is a sequence of some distinct jobs in $\mathcal{J}$ that can be non-preemptively scheduled inside $(L_B, R_B]$ on $m = 1$ machine in the order they appear in σ_B. The* size *of a block B, which is denoted as w_B, is defined as the number of jobs in $\mathcal{J}_B$. The* block window *of B is defined as $(L_B, R_B]$.*

For notational convenience, we will allow σ_B to denote the set of jobs in the sequence. Note that one can check if the jobs in σ_B can be actually scheduled inside $(L_B, R_B]$ in polynomial time. We let $\sigma_B(h)$ denote the h-th job in the ordering σ_B.

The first observation we make to decrease the number of blocks to consider is that we only need to consider minimal blocks in terms of their window (except some "boundary" blocks whose $R_B = T$). Also, we will distinguish two types of blocks depending on whether the jobs in the sequence are tightly scheduled in their window — for each type, we will use a different approach.

Definition 3.2. *We say a block $B = (L_B, R_B, \sigma_B)$ is minimal if both $(L_B + 1, R_B, \sigma_B)$ and $(L_B, R_B - 1, \sigma_B)$ are not blocks. A minimal block B is said to be tight if $R_B - L_B = \sum_{j \in \sigma_B} p_j$, otherwise loose.*

We can show that the number of loose blocks is polynomial in n (Lemma 3.5). Intuitively, the jobs in a loose block cannot be scheduled in a sub-interval of its window, which includes some idle times of the machine, because compressing the block window would force some jobs out of their window. Thus, such jobs will determine the pair of the block window's starting and ending times.

However, the number of tight blocks can be super-polynomially large. Even if we limit the potential starting and/or ending times of tight blocks based on the total size of jobs in the block, there still can be too many tight blocks to consider. However, we observe that such jobs must be very flexible in terms of where to be scheduled, and therefore, we can take away such jobs via preprocessing. Then, we can bound the number of tight blocks to consider. Using this, we obtain a compact LP. After solving the compact LP and using the same rounding method (the running time of the rounding does not depend on T), we bring back the jobs removed in the preprocessing, where we lose only a few jobs.

3.1. Preprocessing. The goal of this preprocessing step is to ensure that there is no job sequence of length $\Delta + 1$ that is very flexible in terms of where the jobs can be scheduled.

DEFINITION 3.3. *Fix a sequence σ of jobs and consider the set of integers L such that $(L, L + \sum_{j \in \sigma} p_j, \sigma)$ is a tight block (it is easy to see that the set is an interval). Let L_1 be the smallest such L and L_2 be the largest such L.*[6] *We say the sequence σ is flexible if $L_2 - L_1 \geq 2n \sum_{j \in \sigma} p_j$.*

The preprocessing step is done by a simple greedy procedure: whenever there is a flexible sequence σ of length $\Delta + 1$, we remove all the $\Delta + 1$ jobs in σ from $\mathcal{J}$. Let $\mathcal{F}$ denote the set of removed flexible sequences.

We now show that the removed flexible sequences can be added back into *any* schedule. Intuitively, this is because the sequence is so flexible and therefore there must be times where it can be scheduled.

LEMMA 3.4. *Let σ be a flexible sequence of length $\Delta + 1$ and $\mathcal{J}' = \mathcal{J} \setminus \sigma$. Given any schedule of a subset of jobs, $A \subseteq \mathcal{J}'$, we can find a schedule of all jobs in $A \cup \sigma$ except one job.*

Proof. Given a schedule of A, we try to insert the sequence σ to the scheduling with the least overlap. Let $P = \sum_{j \in \sigma} p_j$ and L_1 and L_2 be the smallest and largest L such that $(L, L + P, \sigma)$ is a tight block. Since σ is flexible, we have $L_2 - L_1 \geq 2nP$. Focus on the window $[L_1, L_2 + P]$ of the schedule; σ can be scheduled in any sub-window of $[L_1, L_2 + P]$ of length P. If some job takes at least P time slots in this window, we can then remove the job and schedule σ in the P time slots. Since we removed one job from A to schedule jobs in σ, the lemma follows. Otherwise, since there are less than n jobs in $\mathcal{J}'$, there must be an idle interval of length P during $[L_1, L_2 + P]$. In this case, we can insert all jobs in σ without removing any job from A. ◻

Once we find a schedule for the remaining jobs, we can repeatedly add flexible sequences $\mathcal{F}$ to the schedule using the above lemma. Let opt' be the optimum throughput for the remaining jobs and suppose we have an α-approximate solution for them; recall that our target approximation ratio is less than 0.7. The resulting approximation ratio is at least $\frac{\alpha \mathsf{opt}' + \Delta |\mathcal{F}|}{\mathsf{opt}' + (\Delta+1)|\mathcal{F}|}$ for the original instance defined by $\mathcal{J}$. This approximation guarantee is at least α when $\alpha \leq \frac{\Delta}{\Delta+1}$, which holds true when ϵ is small enough. From now on, we assume w.l.o.g. there are no flexible sequences of length $\Delta + 1$ in $\mathcal{J}$.

3.2. Building a Compact LP by Creating Blocks Parsimoniously. We will build a set $\mathcal{B}'$ whose size is polynomially bounded by n so that $\mathsf{LP}_{\mathsf{conf}}$ restricted to the variables corresponding to blocks in $\mathcal{B}'$ has the optimum objective that is almost as large as the optimum throughput. As discussed, we will only consider minimal blocks (except the boundary blocks, which will be discussed below). We first show there are not so many loose (minimal) blocks to consider.

[6]One can find L_1 (if it exists) in polynomial time. This is because it must be the case that $L_1 = r_{\sigma_B(h)} - (p_{\sigma_B(1)} + p_{\sigma_B(2)} + \cdots + p_{\sigma_B(h)})$ for some h since L_1 is the smallest such L. Finding L_2 can be done similarly.

LEMMA 3.5. *The number of loose blocks of size s is at most $s^2 n^s$, for any positive integer s.*

Proof. Focus on a loose block B and a sequence σ_B in $(L_B, R_B]$. Since B is minimal, $\sigma_B(1)$ starts at L and $\sigma_B(s)$ completes at R. Since the block is loose, there is an idle time slot in $(L_B, R_B]$, i.e, no jobs are executed in the time slot. Consider the first idle slot and the set of jobs in σ_B scheduled before this slot. We then try to shift the scheduling intervals for these jobs to right by one slot. Shifting will make the scheduling invalid since B is a minimal block. This is because some job j in the set has completion time equal to its deadline. Similarly, find the last idle slot in $(L_B, R_B]$ and try to shift the scheduling intervals for jobs scheduled after this time slot to left. There must be a job j' whose starting time equals its arrival time. For fixed σ_B, j and j', L_B and R_B are determined. There are at most n^s different sequences σ_B and s^2 different (j, j') pairs. Thus, there are at most $s^2 n^s$ different non-tight minimal blocks. ∎

We now formally describe how to construct $\mathcal{B}'$. As before, we only consider blocks B with either $w_B = \Delta := 2mk^5 = 2k^5, k = \lceil 3/\epsilon \rceil$, or $R_B = T$. Thanks to Lemma 3.5, we can afford to add all loose (minimal) blocks of size Δ to $\mathcal{B}'$. Then, we add "boundary" blocks: For every σ of length at most Δ, we add a block (L, T, σ) to $\mathcal{B}'$, where L is the largest L such that (L, T, σ) is a block[7](if no such L exists, nothing is added for this σ). These boundary blocks are not necessarily minimal.

Finally, we need to add some tight (minimal) blocks to $\mathcal{B}'$. Consider each sequence σ of length $\Delta + 1$. Let $P = \sum_{j \in \sigma} p_j$ be the total size of jobs in σ. Let L_1 (L_2, resp.) be the smallest (largest, resp.) integer L such that $(L, L+P, \sigma)$ is a tight block (in case L_1 and L_2 are not well-defined, we do not do anything for this σ). It is an easy observation that the set of such L forms an interval, i.e., $[L_1, L_2]$. Let σ' be the sequence obtained from σ by removing the largest job. For every $L \in [L_1, L_2 + P]$ that is a multiple of $Q := \left\lceil \frac{P}{2(\Delta+1)} \right\rceil$, let R be the smallest integer such that σ' can be scheduled in $[L, R]$; if such R exists, then add a block (L, R, σ') to $\mathcal{B}'$. For every $R \in [L_1, L_2 + P]$ that is a multiple of Q, let L be largest integer such that σ' can be scheduled in $[L, R]$; if such L exists, we add the block (L, R, σ') to $\mathcal{B}'$. This is where the preprocessing helps — as it is assumed w.l.o.g. that there are no flexible sequences of length $\Delta + 1$ in $\mathcal{J}$, we can bound the number of tight blocks we added to $\mathcal{B}'$.

The following lemma formally bounds the number of blocks in $\mathcal{B}'$.

LEMMA 3.6. *The number of blocks in $\mathcal{B}'$ is at most $10\Delta n^{\Delta+2}$.*

Proof. By Lemma 3.5, the initial size of $\mathcal{B}'$ is at most $\Delta^2 n^\Delta$. For each σ of length at most Δ, we added at most one block (L, T, σ) to $\mathcal{B}'$. Therefore, we added at most n^Δ "boundary" blocks. Now for each σ of length $\Delta + 1$, we inserted at most $2\left\lceil \frac{L_2+1+P-L_1}{Q} \right\rceil \le 2\left(\frac{(L_2+1+P-L_1)\cdot 2(\Delta+1)}{P} + 1\right) \le 2(4n(\Delta+1)+1) \le 9\Delta n$ blocks for large enough Δ and n. The second inequality used the fact that σ is not flexible. Thus, the size of $\mathcal{B}'$ is at most $\Delta^2 n^\Delta + n^\Delta + n^{\Delta+1} \times 9\Delta n \le 10\Delta n^{\Delta+2}$ for large enough n. ∎

After changing $\mathcal{J}_B$ to σ_B in Constraint (2.2), we solve the $\mathsf{LP}_{\mathsf{conf}}$, with the restriction that only blocks in $\mathcal{B}'$ can take positive x value. We refer this LP as the compact LP. Since $|\mathcal{B}'|$ is at most $10\Delta n^{\Delta+2}$, the compact LP can be solved in $n^{O(\Delta)}$

[7]We can find L in polynomial time by considering the jobs in σ in the reverse order.

running time. Once the LP is solved, we can use the same rounding algorithm that was used to solve the case when T is polynomial in n.

To complete the proof, we show that the set $\mathcal{B}'$ suffices to give a good relaxation. The following lemma shows that this process of removing the dependency on T has a negligible effect on the approximation ratio.

LEMMA 3.7. *The optimum objective of the compact LP is at least* $\frac{\Delta}{\Delta+1}\mathsf{opt} = (1 - O(\epsilon^5))\mathsf{opt}$.

Proof. To prove the lemma, from a fixed optimum solution we will construct a feasible LP solution whose objective is at least $\frac{\Delta}{\Delta+1}$ times the optimum throughput. The challenge is to use only blocks in $\mathcal{B}'$ to construct such an LP solution. Towards this end, divide the set of scheduled jobs in the optimum solution into blocks of size $\Delta + 1$. If the last block (L, R, σ) has size at most Δ, we can replace it with the block $(L', T, \sigma) \in \mathcal{B}'$ (Notice that $L' \geq L$) — this is a "boundary" block we added to $\mathcal{B}'$. Now consider each block (L, R, σ) of size $\Delta + 1$ constructed from the optimum solution. Let σ' be the sequence of Δ jobs obtained from σ by removing the largest job. We will find a block $(L', R', \sigma') \in \mathcal{B}'$ such that $(L', R') \subseteq (L, R)$. This suffices to prove the lemma. Intuitively, removing the largest job will allow us to shift the block's starting or ending time. Therefore, even if we only added blocks to $\mathcal{B}'$ whose starting or ending time are a multiple of Q, we will be able to find such a block (L', R', σ').

We find an arbitrary minimal block (L'', R'', σ') such that $(L'', R'') \subseteq (L, R)$. If (L'', R'', σ') is loose, then $(L' = L'', R' = R'', \sigma')$ is in $\mathcal{B}'$ since we added all loose blocks to $\mathcal{B}'$. It remains to consider the case that (L'', R'', σ') is tight. Then $R'' - L'' \leq \frac{\Delta}{\Delta+1}(R - L)$ since $R'' - L''$ is the total size of jobs in σ', $R - L$ is at least the total size of jobs in σ, and σ' is obtained from σ by removing the largest job. Then $L'' - L + R - R'' \geq \frac{R-L}{\Delta+1}$, implying either $L'' - L \geq Q := \left\lceil \frac{R-L}{2(\Delta+1)} \right\rceil$ or $R - R'' \geq Q$. We consider the case $L'' - L \geq Q$ (the other case is analogous). There is a $L' \in [L, L'']$ such that L' is a multiple of Q. Then (L', R', σ') is a block where R' is the smallest time such that σ' can be scheduled during (L', R'). Regardless of whether the block is tight or loose, by our construction, we have added a block (L', R', σ') with $R' \leq R'' \leq R$ to $\mathcal{B}'$.

To summarize, for each block B resulted from partitioning the optimum solution, we found a block $B' \in \mathcal{B}'$ whose window is fully contained in B's window and has an equal number of jobs or one less. In particular, when B' had one less job, B had $\Delta + 1$ jobs. Thus, we have shown the lemma. ∎

4. $(\alpha_0 - \epsilon)$-approximation for Unweighted JIS when $m \geq 2$ and T is Super-polynomial in n.

In this section, we describe our $(\alpha_0 - \epsilon)$-approximation for JIS on constant $m \geq 2$ machines when T is super-polynomial in n. The algorithm works by combining ideas similar to those we developed in Section 2 and 3. Since the analysis is almost identical, we will only formally state the new definitions and lemmas explaining how their proof should be modified. Throughout this section, let $k = \lceil 3/\epsilon \rceil$ and $\Delta = 2mk^5$.

We start by extending the definition of blocks to multiple machines.

DEFINITION 4.1. *[Extension of Definition 3.1] A block B is a triple $B = (L_B, R_B, \sigma_B)$ where L_B and R_B are two integer time points such that $0 \leq L_B < R_B \leq T$, and σ_B, called a sequence tuple, is a set of m sequences of some distinct jobs in $\mathcal{J}$ such that the i-th sequence σ_B^i in σ_B can non-preemptively scheduled inside $(L_B, R_B]$ on one machine in the order they appear in σ_B^i. The size of a block B, which is denoted as w_B, is defined as the total number of jobs in the m sequences.*

The block window of B is defined as $(L_B, R_B]$.

Note that the jobs in the m sequences are all distinct, and the sequences may have different number of jobs. We let $\sigma^i_B(h)$ denote the h-th job in σ^i_B.

DEFINITION 4.2. *[Extension of Definition 3.2] We say a block $B = (L_B, R_B, \sigma_B)$ is minimal if both $(L_B + 1, R_B, \sigma_B)$ and $(L_B, R_B - 1, \sigma_B)$ are not blocks. A minimal block B is said to be tight if $R_B - L_B = \max_{i\in[m]} \sum_{j\in\sigma^i_B} p_j$, otherwise loose.*

As before, we first take a preprocessing step and construct a set $\mathcal{B}'$ of a polynomially many blocks.

4.1. Preprocessing.

DEFINITION 4.3. *[Extension of Definition 3.3] Fix a sequence tuple σ. Let $P = \max_{i\in[m]} \sum_{j\in\sigma^i} p_j$, and L_1 (L_2, resp.) be the smallest (largest, resp.) L such that $(L, L+P, \sigma)$ is a tight block. We say the sequence tuple σ is flexible if $L_2 - L_1 \geq 2nP$.*

LEMMA 4.4. *[Extension of Lemma 3.4] Let σ be a flexible sequence tuple of $\Delta + m$ jobs and $\mathcal{J}' = \mathcal{J} \setminus \sigma$. Given any schedule of a subset of jobs, $A \subseteq \mathcal{J}'$, we can find a schedule of all jobs in $A \cup \sigma$ except m jobs.*

Proof. As in the proof of Lemma 3.4, we can add all jobs in σ^i to the i-th machine while removing at most one job. ∎

Thanks to Lemma 4.4, we can repeatedly remove a flexible sequence tuple σ of $\Delta + m$ jobs if such a tuple exists. Let $\mathcal{F}$ denote the set of removed flexible sequence tuples.

We first show that this preprocessing has no effect on our target approximation ratio. Once we find a schedule for the remaining jobs, we can repeatedly add flexible sequence tuples $\mathcal{F}$ to the schedule using the above lemma. Let opt' be the optimum throughput for the remaining jobs and suppose we have an α-approximate solution for them. The resulting approximation ratio is then at least $\frac{\alpha\mathsf{opt}' + \Delta|\mathcal{F}|}{\mathsf{opt}' + (\Delta+m)|\mathcal{F}|}$ for the original instance defined by $\mathcal{J}$. This approximation guarantee is at least α when $\alpha \leq \frac{\Delta}{\Delta+m}$, which holds true when ϵ is small enough.

We briefly discuss the running time of finding a flexible sequence tuple σ of size $\Delta + m$. Firstly, we enumerate all sequence tuple σ of size $\Delta + m$, which can be obviously done in $n^{O(\Delta)}$ time. Let $P = \max_{i\in[m]} \sum_{j\in\sigma^i} p_j$. Let's focus on each σ^i. We find an interval $[L^i_1, L^i_2]$ such that all jobs in σ^i can be schedulable in $(t, t + P]$ on a single machine for any time $L^i_1 \leq t \leq L^i_2$. Then, we will set $(L_1, L_2) = (\max_{i\in[m]} L^i_1, \min_{i\in[m]} L^i_2)$. Thus, we can find a flexible sequence tuple σ of size $\Delta + m$ in $n^{O(\Delta)}$ time.

From now on, we assume w.l.o.g. that there are no flexible sequence tuples of $\Delta + m$ jobs in $\mathcal{J}$.

4.2. Building a Compact LP by Creating Blocks Parsimoniously. In this section we discuss how we construct a small set $\mathcal{B}'$ of blocks. Using essentially the same argument as we used in the proof of Lemma 3.5, we can prove that the number of loose blocks B of size s is at most $s^2 n^s$. Thus, we can afford to add all loose blocks of size Δ to $\mathcal{B}'$. Then for every sequence tuple σ of size at most $\Delta + m - 1$, let L be the largest number such that (L, T, σ) is a block if such L exists, and add the block to $\mathcal{B}'$. So far the size of $\mathcal{B}'$ is $n^{O(\Delta)}$.

Next, we add some tight blocks to $\mathcal{B}'$. Consider each sequence tuple σ of size $\Delta + m$. Let $P = \max_{i\in[m]} \sum_{j\in\sigma^i} p_j$ be the maximum total size of jobs in any sequence. Let L_1 (L_2, resp.) be the smallest (largest, resp.) integer L such that $(L, L + P, \sigma)$ is a tight block (in case L_1 and L_2 is not well-defined, we do not do anything for this σ). For each non-empty sequence σ^i, we remove the largest job; if we removed less than m jobs, we remove more jobs arbitrarily until we removed exactly m jobs. Let σ' be the new sequence tuple (it has size Δ). For every $L \in [L_1, L_2 + P]$ that is a multiple of $Q := \left\lceil \frac{P}{2(\Delta+m)} \right\rceil$, let R be the smallest integer such that (L, R, σ') is a block; if such R exists, then add (L, R, σ') to $\mathcal{B}'$. For every $R \in [L_1, L_2 + P]$ that is a multiple of Q, let L be largest integer such that (L, R, σ') is a block; if such L exists, we add (L, R, σ') to $\mathcal{B}'$.

Using essentially the same argument as we used in the proof of Lemma 3.6, we can show that $|\mathcal{B}'| = 10\Delta n^{\Delta+2} = n^{O(\Delta)}$.

We solve the same $\mathsf{LP}_{\mathsf{conf}}$, except $\mathcal{J}_B$ in Constraint (2.2) being changed to $\bigcup_{i\in[m]} \sigma^i_B$ and $\mathcal{B}$ being replaced with a smaller subset $\mathcal{B}'$ we constructed. We refer to this LP as the compact LP. It is clear that we can solve the compact LP in $n^{O(\Delta)}$ time. As before, we can use the same rounding procedure we developed in Section 2.

It now remains to show that solving this compact LP loses only little compared to the optimum.

LEMMA 4.5. *The optimum objective of the compact LP is at least* $\frac{\Delta}{\Delta+2m-1}\mathsf{opt} = (1 - O(\epsilon^5))\mathsf{opt}$.

Proof. The proof of this lemma is very similar to that of Lemma 3.7. Consider the optimum schedule for the instance and divide the set of scheduled jobs into blocks of size $\Delta + m$ (some jobs may be dropped, but the number is at most $m - 1$). If the last block (L, R, σ) has size at most $\Delta + m - 1$, we can replace it with the block $(L', T, \sigma) \in \mathcal{B}'$; notice that $L' \geq L$. Now consider each block (L, R, σ) of size $\Delta + m$ constructed from the optimum solution. We remove the largest job from each non-empty sequence in σ; remove more jobs if needed so that the number of removed jobs is exactly m. Let σ' be the new sequence tuple of size Δ. It is sufficient to find a block $(L', R', \sigma') \in \mathcal{B}'$ such that $(L', R') \subseteq (L, R)$.

We find an arbitrary minimal block (L'', R'', σ') such that $(L'', R'') \subseteq (L, R)$. If (L'', R'', σ') is loose, then $(L' = L'', R' = R'', \sigma')$ is in $\mathcal{B}'$. It remains to consider the case that (L'', R'', σ') is tight. Then $R''-L'' \leq \frac{\Delta+m-1}{\Delta+m}(R-L)$. Then $L''-L+R-R'' \geq \frac{R-L}{\Delta+m}$, implying either $L'' - L \geq Q := \left\lceil \frac{R-L}{2(\Delta+m)} \right\rceil$ or $R - R'' \geq Q$. We consider the case $L'' - L \geq Q$ (the other case is analogous). There is a $L' \in [L, L'']$ such that L' is a multiple of Q. Then (L', R', σ') is a block where R' is the smallest time such that all sequences in σ' can be scheduled during $(L', R']$. By our construction, we have added a block (L', R', σ') with $R' \leq R'' \leq R$ to $\mathcal{B}'$.

To summarize, when we obtained a block B of size $\Delta + m$ from partitioning the optimum solution, we had to drop at most $m - 1$ jobs; and then, we found a block $B' \in \mathcal{B}'$ of size Δ whose window is fully contained in B's window. If B was of smaller size, we didn't lose any job. Thus, we have shown the lemma. ∎

5. $1 - O\left(\sqrt{(1/m)\ln m}\right)$-approximation for JIS. In this section our goal is to prove Theorem 1.2 assuming that $T = \mathrm{poly}(n)$.

We start by describing our algorithm which works by rounding the naive LP relaxation for the problem. The relaxation is the following. Let $x_{j,t}$ denote whether job j is started at time t. This variable is defined if $r_j \le t \le d_j - p_j$.

$$(\mathsf{LP}_{\mathsf{naive}}) \qquad \max \quad \sum_j \sum_t w_j x_{j,t}$$

$$\sum_j \sum_{t'=\max\{r_j, t-p_j\}}^{\min\{d_j-p_j, t-1\}} x_{j,t'} \le m \qquad \forall t \in [T]; \tag{5.1}$$

$$\sum_t x_{j,t} \le 1 \qquad \forall j \in \mathcal{J}. \tag{5.2}$$

where $x_{j,t} \ge 0$ for all $j \in \mathcal{J}, t \in [r_j, d_j - p_j]$. The first constraint ensures that at most m jobs are scheduled at any point in time. The second constraints ensure that each job is scheduled at most once.

Our algorithm solves the LP and then rounds the fractional solution. For each job, we do randomized rounding. Each job j selects a starting time t to be scheduled with probability $(1-\epsilon)x_{j,t}$, and j is not scheduled with probability $1-(1-\epsilon)\sum_t x_{j,t}$ where $\epsilon < 1$ is a parameter depending on m which will be fixed later. This is the tentative schedule, which could be infeasible. Order the jobs by their *starting* times in the tentative schedule. Consider a job j, whose starting time is t in the tentative schedule. Schedule job j at time t whenever there is a machine free at time t. A job j can be feasibly scheduled if and only if the time slot $(t, t+1]$ is covered by less than m already-assigned jobs.

To bound the quality of the solution, the goal is to bound the probability that a job is scheduled. The proof idea is quite simple. Consider any fixed job j. Condition on the event that the algorithm chooses to tentatively schedule j at time t; this happens with probability $(1-\epsilon)x_{j,t}$. We bound the probability that j is removed from the tentative schedule. We apply a concentration inequality (Theorem 2.9) since each job is rounded independently of other jobs.

LEMMA 5.1. *Each job j is scheduled with probability at least* $(1-\epsilon)\left(1-\exp\left(-m\frac{\epsilon^2}{3(1-\epsilon)}\right)\right)\sum_t x_{j,t}$.

Proof. Consider any fixed job j. We condition on the event that we chose to tentatively schedule j at time t; this happens with probability $(1-\epsilon)x_{j,t}$. We then bound the probability that j is removed from the tentative schedule. For this to happen, there must be at least m jobs $i \neq j$ with tentative scheduling intervals covering $(t, t+1]$. Let X_i be an indicator random variable that is 1 if the tentative interval for job i covers $(t, t+1]$ and 0 otherwise. Then $Z := \sum_{i\neq j} X_i \ge m$ if job j is removed from the tentative schedule. Also notice that $\mu := \mathbf{E}[\sum_{i=1}^n X_i] \le (1-\epsilon)m$ because at most m jobs are fractionally scheduled at time $(t, t+1]$, and we schedule i at t' with probability $(1-\epsilon)x_{i,t'}$. By adding dummy random variables, we assume $\mu := (1-\epsilon)m$; this only increases $\Pr[Z \ge m]$.

Since $\epsilon < 1$, we have

$$\Pr[Z \ge m] \le \exp\Big(-\left(\frac{\epsilon}{1-\epsilon}\right)^2 (1-\epsilon)m/3\Big) = \exp\Big(-\frac{\epsilon^2 m}{3(1-\epsilon)}\Big).$$

Thus, the probability that j is tentatively scheduled at t and is not removed from the tentative scheduling is at least $(1-\epsilon)\left(1-\exp\left(-\frac{\epsilon^2 m}{3(1-\epsilon)}\right)\right)x_{j,t}$. Adding this over all t, job j is scheduled with probability at least $(1-\epsilon)\left(1-\exp\left(-\frac{\epsilon^2 m}{3(1-\epsilon)}\right)\right)\sum_t x_{j,t}$.□

Set $\epsilon = \sqrt{\frac{2\ln m}{m}}$. Then $\exp\left(-\frac{\epsilon^2 m}{3(1-\epsilon)}\right) \le m^{-2/3}$ and $(1-\epsilon)\left(1-\exp\left(-\frac{\epsilon^2 m}{3(1-\epsilon)}\right)\right) \ge 1-\sqrt{\frac{2\ln m}{m}} - m^{-2/3} = 1 - O\left(\sqrt{\frac{\log m}{m}}\right)$. This implies the second half of Theorem 1.2.

6. Handling Large T in the $\left(1-O\left(\sqrt{\frac{\log m}{m}}\right)\right)$-approximation for Unweighted JIS. In this section, we show how to handle large T in the $\left(1-O\left(\sqrt{\frac{\log m}{m}}\right)\right)$-approximation in Section 5. Note that this is for the *unweighted* case. Though the algorithm in Section 5 is for weighted JIS, we can only handle large T when the jobs are unweighted and we will point out the technical reason below.

Using the preprocessing step described in Section 3, we can assume w.l.o.g. that there are no flexible sequences of length $\Delta+1$ in $\mathcal{J}$; see Definition 3.3 for the definition of flexible sequences. This is because once we find a schedule for the remaining jobs after removing some flexible sequences in the preprocessing step, we can repeatedly add back the deleted flexible sequences to the schedule using Lemma 3.4 on an arbitrary machine. Then, define a block $B = (L_B, R_B, \sigma_B)$ as in Section 3: $L_B < R_B$ are integers in $(0, T]$ and σ_B is a sequence of different jobs that can be scheduled on $(L_B, R_B]$ on one machine. The size w_B of B is the number of jobs in σ_B.

The LP is the following.

$$\begin{aligned}
\max \quad & \sum_B w_B \cdot x_B \\
& \sum_{B: L_B < t \le R_B} x_B \le m && \forall t \in [T] \\
& \sum_{B: j \in \sigma_B} x_B \le 1 && \forall j \in \mathcal{J} \\
& x_B \ge 0 && \forall B
\end{aligned}$$

Notice that the difference between this LP and $\mathsf{LP}_{\mathsf{conf}}$ in Section 2 (other than the difference between sequences and sets): in the above LP, the jobs in a block need to be schedulable on 1 machine, and we require each time point to be covered by at most m blocks. In contrast, in ($\mathsf{LP}_{\mathsf{conf}}$), the jobs in a block need to be schedulable on m machines and we require each time point to be covered by at most 1 block. We note that the more complicated blocks used in ($\mathsf{LP}_{\mathsf{conf}}$) that encode schedules on m machines was only needed to a get a better than $(1-1/e)$-approximation.

The value of the above LP is clearly at least the optimum throughput. Now our goal is to reduce the number of blocks to remove the dependence on T similarly to the procedure in Section 3. Thus, we limit blocks B considered in the LP to the set $\mathcal{B}'$ of blocks we constructed in Section 3. Lemma 3.6 ensures that $|\mathcal{B}|$ has size at most $10\Delta n^{\Delta+2}$, independent of T assuming that $\Delta := 2k^5, k = \lceil 3/\epsilon \rceil$.

We have now restricted the possible blocks, potentially changing the value of the LP objective. The proof of Lemma 3.7 ensures the LP with the restriction that only blocks in $\mathcal{B}$ are considered has value at least $\frac{\Delta}{\Delta+1}$ times the optimum throughput. This is critically where we use that the jobs are unweighted.

With this restriction, the LP has size independent of T. We solve this LP. Now observe that a fractional solution to the above LP yields a solution to the naive LP of the same objective value. We construct the solution to the naive LP. Using this LP solution we run the algorithm in Section 5 to get an integer solution. The analysis in

Section 5 gives the final approximation ratio of $1 - O\left(\sqrt{\frac{\log m}{m}}\right) - \epsilon$. This proves the first part of Theorem 1.2.

REFERENCES

[1] Micah Adler, Arnold L Rosenberg, Ramesh K Sitaraman, and Walter Unger. Scheduling time-constrained communication in linear networks. *Theory of Computing Systems*, 35(6):599–623, 2002.

[2] Nikhil Bansal, Ho-Leung Chan, Rohit Khandekar, Kirk Pruhs, Clifford Stein, and Baruch Schieber. Non-preemptive min-sum scheduling with resource augmentation. In *IEEE Symposium on Foundations of Computer Science*, pages 614–624, 2007.

[3] Philippe Baptiste. An O(n^4) algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Oper. Res. Lett.*, 24(4):175–180, 1999.

[4] Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing*, 31(2):331–352, 2001.

[5] Piotr Berman and Bhaskar DasGupta. Improvements in throughout maximization for real-time scheduling. In *ACM Symposium on Theory of computing*, pages 680–687. ACM, 2000.

[6] Jacek Błażewicz, Klaus H Ecker, Erwin Pesch, Günter Schmidt, and Jan Weglarz. *Scheduling computer and manufacturing processes*. Springer Science & Business Media, 2013.

[7] 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.

[8] Matteo Fischetti, Silvano Martello, and Paolo Toth. The fixed job schedule problem with spread-time constraints. *Operations Research*, 35(6):849–858, 1987.

[9] M. R. Garey and David S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM J. Comput.*, 6(3):416–426, 1977.

[10] Nicholas G Hall and Michael J Magazine. Maximizing the value of a space mission. *European journal of operational research*, 78(2):224–241, 1994.

[11] Kwang Soo Hong and Joseph Y.-T. Leung. Preemptive scheduling with release times and deadlines. *Real-Time Systems*, 1(3):265–281, 1989.

[12] Gilad Koren and Dennis Shasha. Dˆover: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24(2):318–339, 1995.

[13] Eugene L Lawler, Jan Karel Lenstra, AHG Rinnooy Kan, and DB Shmoys. Sequencing and scheduling: Algorithms and complexity. *Hanbooks in Operations Resarch*, vol. 4, 1993.

[14] Richard J. Lipton and Andrew Tomkins. Online interval scheduling. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, 1994.

[15] Colin McDiarmid. Concentration. In *Probabilistic methods for algorithmic discrete mathematics*, pages 195–248. Springer, 1998.

[16] Frits CR Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2(5):215–227, 1999.

Appendix A. Proof of Lemma 2.10.

Proof of Lemma 2.10. Fix j. For notational convenience, drop j from the subscripts. Our goal is to show $1-(1-b/2)^2e^{-a} \geq 0.62a + 0.69b$ for all $a, b \geq 0$ such that $a+b \leq 1$.

We observe that we can assume w.l.o.g. that either $b = 0$ or $a+b = 1$. To see this, fix a. So, we have $0 \leq b \leq 1-a$. Notice that it suffices to show the inequality when $(b/2-1)^2e^{-a}+0.69b$ is maximized, which occurs when either $b = 0$ or $b = 1-a$. Consider the easier case $b = 0$. Then, the inequality simplifies to $1 - 0.62a \geq e^{-a}$ which is true for all $0 \leq a \leq 1.04$.

We now consider the other case when $b = 1-a$. By a simple calculation, the inequality simplifies to

$$e^a(0.28a + 1.24) \geq (a+1)^2. \tag{A.1}$$

Let $g(a) = e^a(0.28a+1.24) - (a+1)^2$ and we need to show that $g(a) \geq 0$ for every $a \in [0,1]$. Let $g'(a) := \frac{\mathsf{d}g}{\mathsf{d}a} = e^a(0.28a+1.52) - 2(a+1)$ and $g''(a) := \frac{\mathsf{d}^2 g}{\mathsf{d}^2 a} = e^a(0.28a+1.8) - 2$ be the first-order and second-order differential function of g respectively. Notice that $g''(a)$ is an increasing function of a, $g''(0) < 0$ and $g''(1) > 0$. Let $a_0 \in (0,1)$ be the number such that $g''(a_0) = 0$. Then $g'(a)$ is decreasing over $a \in [0, a_0]$ and increasing over $a \in [a_0, 1]$. So, the minimum of $g(a)$ for $a \in [0,1]$ is achieved when $a = 0$, $a = 1$, or $a = a_1$, where a_1 is the unique number in $[a_0, 1]$ such that $g'(a_1) = 0$, if this a_1 exists. Notice that $g(0) = 0.24 > 0$ and $g(1) = 1.52e - 4 > 0.1317 > 0$. So it suffices to check if $g(a_1) > 0$ when a_1 exists.

We notice that $g'(0.6715) = e^{0.6715} \times (0.28 \times 0.6715 + 1.52) - 2 \times (0.6715+1) < 0$ and $g'(0.6716) = e^{0.6716} \times (0.28 \times 0.6716 + 1.52) - 2 \times (0.6716 + 1) > 0$. So a_1 exists and we must have $a_1 \in (0.6715, 0.6716)$. But then $g(a_1) > e^{0.6715}(0.28 \times 0.6715 + 1.24) - (0.6716+1)^2 > 0$. Thus, we proved $g(a) > 0$ for every $a \in [0,1]$. ◻