Breaking $1 - 1/e$ Barrier for Non-preemptive Throughput Maximization

Sungjin Im[1], Shi Li[2], and Benjamin Moseley[3]

[1] University of California, Merced, CA 95344. sim3@ucmerced.edu
[2] University at Buffalo, NY 14260. shil@buffalo.edu.
[3] Washington University in St. Louis, MO, 63130. bmoseley@wustl.edu

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 machines 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 numerous areas of computer science and other fields. Due to this, there has been a large amount of research focusing on the topic. However, relatively little is known when jobs must be scheduled *non-preemptively*. Nonetheless, non-preemptive job scheduling occurs frequently in practice for a variety of reasons including because jobs cannot be stopped during execution due to practical constraints or because overhead costs are prohibitively large.

A central problem in the scheduling literature is determining how to schedule jobs by their deadline. In many cases when jobs have deadlines, not all jobs can be scheduled by their deadline. In these situations an alternative goal is to complete as many jobs as possible by their deadline. In this paper, we consider this problem a.k.a. *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, we will call this problem the Job Interval Scheduling (JIS). Not surprisingly, JIS has various applications in practice. For examples, see [10,6,8,13].

It was shown by Garey and Johnson that this problem is 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 discrete version of this problem. In this version, we are explicitly given a set of intervals $\mathcal{I}_j$ in $(0, T]$ (which may have different lengths) for each job j. To schedule the job, we need 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 [4]. 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 with 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].

Our Results: In this paper, we improve upon the state-of-art approximations for JIS for both the single-machine and multiple-machine cases. First, we show that for constant m, there is a $0.6448 > (1 - 1/e)$-approximation for JIS.

Theorem 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 the 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 at most a small constant; we run the second algorithm if m is large. Indeed, our second algorithm works for the more general weighted version of the problem, provided that T is polynomially bounded by n. In this version, each job i has some positive weight

[4] 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 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.

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 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.*

Our Techniques: Our result in Theorem 2 will follow from a simple rounding procedure based on the naive LP relaxation for the problem. We scale down a naive LP solution by $(1 - \epsilon)$, apply a standard rounding technique to obtain a tentative schedule. Then we convert the tentative schedule to one that is feasible by removing jobs in a greedy manner. We show that the probability that a job is removed from the tentative schedule is exponentially small in m. Another 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. 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. 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. [7] used an involved preprocessing step to guess the windows where a block of k jobs are scheduled in some optimum solution. Then for each window corresponding to where a block of k jobs are scheduled in an optimum solution, there are variables encoding which k jobs are scheduled inside it. In our configuration LP, we do not guess where blocks are scheduled in an optimum solution. Rather, we have variables for blocks of k jobs that are scheduled and, when allowing the blocks to be scheduled fractionally, we ensure that at most one fractional block covers every time point. Thus, instead partitioning time based on guessing where blocks are in an optimum solution, we partition the time horizon $(0, T]$ using the fractional blocks obtained from the configuration LP. 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 to single exponential; (3) we can obtain the improved $(\alpha_0 - \epsilon)$-approximation for any constant m [5].

Now we state some intuition on how the configuration LP can help give a better approximation. Suppose we know the time steps when the optimal solution schedules k additional jobs, $t_1, t_2, t_3, \cdots$. (Since we do not know, we need to lose $1 - \epsilon$ factor in the approximation ratio and k needs to be a large constant.) Then, we only need to consider windows $\mathcal{W} = \{(t_1, t_2], (t_2, t_3], (t_3, t_4], \cdots\}$; let's call this the optimal partition. The configuration LP gives a distribution over sets of k jobs scheduled in each window. By randomly choosing one set of k jobs for each window, we can easily show a $(1 - 1/e)$-approximation following a standard analysis for the maximum coverage problem.

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 one and the last one 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 better than a $1 - 1/e$ factor for the probability we schedule job j. Thus, taking the best solution given by these two rounding procedures will lead to an approximation ratio better than $1 - 1/e$.

Removing Dependency on T: As mentioned above, in our problem, 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 will 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

[5] As mentioned before, [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 $1 - 1/e$-approximation for multiple machines.

configuration LP is used in the $\left(1 - O(\sqrt{\frac{\log m}{m}})\right)$-approximation to reduce the dependence on T, although only the naive LP is needed when $T = \text{poly}(n)$.

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 machines setting [4,5]. [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].

Organization: To deliver the main ideas of our $(\alpha_0 - \epsilon) \approx 0.6448$-approximation for JIS stated in Theorem 1, in Section 2, we first present our result assuming that the number of machines, m is a constant and $T = \text{poly}(n)$; recall that $(0, T]$ is the time horizon we are considering. In Appendices A and B, we prove Theorem 1 for the general case. Specifically, in Appendix A we first extend the $(\alpha_0 - \epsilon)$-approximation to the case when T is super-polynomial in n and $m = 1$, and then in Appendix B to the case when T is super-polynomial in n and $m \geq 2$. We prove Theorem 2 in Section 3 under the assumtion that T is polynomially bounded and remove the assumption in Appendix C.

2 Proof of Theorem 1 when $m = O(1)$ and $T = \text{poly}(n)$

Our algorithm is based on a configuration LP relaxation for the problem. We will then 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, we can run the rounding procedure of [7]; this will give us $(1 - 1/e - \epsilon)$-approximation for the unweighted case. To obtain the improved $(\alpha_0 - \epsilon)$ approximation ratio, we run a different rounding procedure and choose the better solution from the two procedures. This will give us Theorem 1 when $T = \text{poly}(n)$.

2.1 Linear Programming

We define a block as a triple $B = (L_B, R_B, \mathcal{J}_B)$ where L_B and R_B are two integer time points such that $0 \leq L_B < R_B \leq T$, 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. We assume that B is associated with a specific schedule where jobs in $\mathcal{J}_B$ are scheduled in $(L_B, R_B]$ on m machines. The size of block B is defined as the number of jobs in $\mathcal{J}_B$, which is denoted as w_B. We say that B has *block window* $(L_B, R_B]$.

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 integer programming for

JIS is defined as follows. We only consider the blocks B with either $w_B = \Delta$, or $R_B = T$ and $w_B < \Delta$ in the IP; for simplicity we omit this constraint.

$$\begin{aligned}
\max \quad & \sum_B w_B \cdot x_B && && (\mathsf{LP}_{\mathsf{conf}})\\
& \sum_{B: L_B < t \le R_B} x_B \le 1 && \forall t \in [T] && (1)\\
& \sum_{B: j \in \mathcal{J}_B} x_B \le 1 && \forall j \in \mathcal{J} && (2)\\
& x_B \in \{0,1\} && \forall B
\end{aligned}$$

In the above IP, Constraint (1) ensures that block windows are disjoint, and Constraint (2) requires each job to be scheduled at most once. Note that the number of constraints in (1) is polynomially bounded when $T = \text{poly}(n)$ – as mentioned earlier, we discuss how to handle non-polynomially bounded 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 $m > 1$. Thus the IP may not give the optimum throughput. However we show that the loss is small. To see this, fix an optimal schedule. Given the optimum schedule, we sort all the jobs according to their completion time. Let $L = 0$ initially. In each iteration, we 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, we let $\mathcal{J}'$ be all the jobs in the sequence and let $R = T$. We create a block $B = (L, R, \mathcal{J}')$ and set $x_B = 1$. Then, we 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 we 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 who 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.[6]

The LP is obtained by relaxing the constraints $x_B \in \{0,1\}$ to $x_B \ge 0$. Note that the running time of solving the LP is $n^{O(\Delta)} = n^{O(m/\epsilon^5)}$. Let $\{x^*_B\}$ denote the optimal solution to the above LP. Let $\text{OPT}_{\text{LP}} = \sum_B w_B x^*_B$ denote the optimal LP objective.

2.2 Preprocessing

In the preprocessing step, we break the time horizon $(0, T]$ into a set $\mathcal{W}$ of disjoint intervals, which we call *base* windows to distinguish them from jobs windows and block windows. We also construct a new solution $\{x'_B\}$. The main

[6] It is worth noting that this is where we crucially use the assumption that jobs have uniform weights.

goal is two-fold: (i) to preserve most of the LP objective and (ii) to make each block window completely contained in a base window; this makes the rounding procedures more applicable.

We now formally show how to break $(0, T]$ into base windows and obtain $\{x'_B\}$ from $\{x^*_B\}$. Note that the blocks in the support of $\{x^*_B\}$ can overlap with one another. To create $\{x'_B\}$, we iteratively cut at the time point when an additional $1/k$ fraction of blocks end in $\{x^*_B\}$. Formally, we associate each integer time-point $t \in (0, T]$ with a weight $e_t = \sum_{B:R_B=t} x^*_B$, 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 $L = T$.

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 $(L, R]$ that intersect $(L_B, R_B]$, 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 $(L, R]$. For all these created blocks B', we let $x'_{B'} = x^*_B$. Notice that the jobs across the boundaries of base windows are deleted in this process. After that, we delete big jobs from each created block B': a job j in $\mathcal{J}_{B'}$ is said to be big compared to B' if $p_j \geq (R_{B'} - L_{B'})/k^3$.[7]

We have constructed a set $\mathcal{W}$ of disjoint base windows and derived a new fractional solution $\{x'_B\}$ from $\{x^*_B\}$ that satisfies the following properties. All the blocks B in the description are restricted to the ones with $x'_B > 0$.

Properties of $\{x'_B\}$:

1. For every block B, the block window $(L_B, R_B]$ is fully contained in some base window in $\mathcal{W}$.
2. No job j in $\mathcal{J}_B$ is big compared to B. That is, for all $k \in \mathcal{J}_B$ it is the case that $p_j < (R_{B'} - L_{B'})/k^3$.
3. If $\mathrm{OPT}_{\mathrm{LP}} \geq \Delta$, then $\sum_{B'} w_B x'_B \geq (1 - \epsilon/3)\mathrm{OPT}_{\mathrm{LP}}$.
4. For all windows $(L, R] \in \mathcal{W}$, $\sum_{B:(L_B,R_B]\subseteq(L,R]} x'_B \leq 1 + 1/k$.
5. For all jobs j, $\sum_{B:j\in\mathcal{J}_B} x'_B \leq 1$.

Properties (1), (2) and (5) are very easy to check. To see Property (4) holds, consider a base window $(L, R] \in \mathcal{W}$. We know that at most 1 fractional block intersects R due to the Constraints (1). Due to the way we defined base windows, at most $1/k$ fractional block can end during $(L, R-1]$.

Property (3) is the most non-trivial one. Observe that there are at most m fractional jobs across the boundary of two adjacent base windows. Each time (except for the last one) we build a base window, we collected at least $1/k$ fractional blocks, and thus Δ/k fractional jobs. So the total number of boundaries is at most $\mathrm{OPT}_{\mathrm{LP}}/(\Delta/k) = k\mathrm{OPT}_{\mathrm{LP}}/\Delta$. Thus the total jobs we discarded due to

[7] This is another place where we rely on the assumption that jobs have uniform weights.

"boundary crossing" is at most $km\mathrm{OPT}_{\mathrm{LP}}/\Delta$. Also at most mk^3 fractional big jobs are discarded from each base window. Thus at most $mk^3 \times (k\mathrm{OPT}_{\mathrm{LP}}/\Delta+1)$ fractional big jobs are removed. If $\mathsf{opt} \geq \Delta$ and $k \geq 3$, the total number of fractional big 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 (3) follows. If the condition $\mathrm{OPT}_{\mathrm{LP}} \geq \Delta$ is not satisfied, we can simply guess the optimal solution using enumeration.

2.3 The First Rounding Procedure

In this subsection, we show how to round $\{x'_B\}$ to obtain an improved approximation. As mentioned before, we have two rounding procedures. The first rounding is an independent rounding that 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 a block B with $(L_B, R_B] \subseteq (L, R]$ with probability $\frac{x'_B}{1+1/k}$. This is well defined due to Property (4) – it says that there are only $(1 + 1/k)$ fractional blocks to be considered for each base window. If a job is scheduled more than once, we keep only one scheduling of the job. This completes the description of the first rounding.

A standard analysis for independent rounding can only show that each job can be 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 need to do a more careful analysis. Let's focus on each job j. Consider the set of base windows that intersect $(r_j, d_j]$. We call the first and the last of these base windows the boundary base windows. All these base windows except the boundary ones are completely contained in $(r_j, d_j]$. Job j may appear in multiple base windows, more precisely in blocks contained in multiple base windows. Let a_j be the fraction by which job j is scheduled in boundary base windows, scaled down by $1 + 1/k$. There may be only one boundary window for j, but it only helps the approximation ratio, hence for simplicity, let's proceed with our analysis assuming that there are two boundary base windows for every job. Now we turn our attention to non-boundary windows. Observe that in every non-boundary base window in which j is scheduled, we can schedule j anywhere inside it. Let b_j be the fraction by which job j is scheduled in non-boundary base windows, scaled down by $1 + 1/k$.

We show that the first rounding schedules j with probability at least:

$$1 - (1 - a_j/2)^2 e^{-b_j} \tag{3}$$

Let's take a close look at why this is the case. Let a, a' be the fractions by which job j is scheduled on the two boundary base windows, scaled down by $1 + 1/k$. Likewise, let $b(u)$ be the fraction by which job j is scheduled on a non-boundary base window u, scaled down by $1 + 1/k$. Note that $\sum_u b(u) = b_j$. Then, j is scheduled with probability at least $1 - (1 - a)(1 - a') \prod_u (1 - b(u)) \geq 1 - (1 - a)(1 - a') \prod_u e^{-b(u)} = 1 - (1 - a)(1 - a')e^{-b_j} \geq 1 - (1 - a_j/2)^2 e^{-b_j}$ where we use the well-known inequality $e^x \geq 1 + x$.

2.4 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. The fractional solution x'_B 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. We show that each job j is successfully scheduled with probability at least

$$(1 - \epsilon/3)b_j \tag{4}$$

Formally, the second rounding algorithm is as follows. Let $f_{j,W}$ be the amount by which job j is assigned to a base window W, i.e. $f_{j,W} := \sum_{B:(L_B,R_B]\subseteq W, j\in\mathcal{J}_B} x'_B$. Then b_j is $\sum_W f_{j,W}/(1+1/k)$, where W is over all non-boundary base windows of j. Consider each job j. We assign job j to one of its non-boundary base windows W with probability $f_{j,W}/(1+1/k)$. Let $\mathcal{J}(W)$ be the set of jobs selected to be scheduled in the base window W. We schedule these jobs greedily on m machines within W. Since 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, the greedy packing is pretty good.

Lemma 1. *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. The proof immediately follows from the fact that all jobs in $\mathcal{J}(W)$ have sizes no greater than $(R - L)/k^3$ (Property (2)), and all jobs can in $\mathcal{J}(W)$ can be scheduled everywhere inside W. □

If the total size of jobs in $\mathcal{J}(W)$ is greater than $(1-k^3)m$, we simply discard all jobs in $\mathcal{J}(W)$. Our goal is to show that this bad event happens with low probability. The following observation is immediate.

Lemma 2. *The total size of jobs in $\mathcal{J}(W)$ is at most $m(R - L)/(1 + 1/k)$ in expectation.*

Proof. Notice that $\sum_{B:t\in(L_B,R_B]} x'_B \leq 1$ for every $t \in (L, R]$. Thus, $\sum_{B:(L_B,R_B]\subseteq(L,R]}(R_B - L_B)x'_B = \sum_{t\in(L,R]}\sum_{B:t\in(L_B,R_B]} x'_B \leq (R - L)$. Since we can schedule at most $m(R_B - L_B)x'_B$ volume of jobs in B, the total volume of jobs scheduled in B is at most $m(R - L)$; here the volume of a job j refers to p_j times the fraction by which the job is scheduled. The claim follows since we scaled down the assignment probability down by a factor of $1 + 1/k$. □

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 3. *For any window $W = (L, R)$, the total size of jobs in $\mathcal{J}(W)$ is at most $(1 - 1/2k)(R - L)$ 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, 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 $\Pr[Z \geq (1 - 1/2k)m(R - L)]$. By Theorem 3 in Appendix E, we have

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

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

If the total size of jobs assigned to the base window $(L, R]$ is at most $(1 - 1/2k)m(R - L) \leq (1 - 1/k^3)m(R - L)$, then all these jobs can be scheduled in $(L, R]$ on m machines. Further, this happens with probability at least $(1 - \epsilon/3)$ due to Lemma 3. Since a job is assigned to one of a non-boundary window with probability b_j, the probability that job j is scheduled due to the second rounding is at least $(1 - \epsilon/3)b_j$. This shows the probability claimed in (4).

2.5 Combining the Two Rounding Procedures

Finally, we take the better between the two rounding solutions. That is, we take the maximum of the two lower bounds, (3) and (4). The following lemma lower bounds (3) by a linear combination of a_j and b_j, which follows by approximating e^x by a piecewise linear function and performing some case analysis. The proof is deferred to Appendix D.

Lemma 4. *For all a_j, b_j such that $0 \leq a_j + b_j \leq 1$, $\big(1 - (1 - a_j/2)^2 e^{-b_j}\big) \geq \lambda_1 a_j + \lambda_2 b_j$ where $\lambda_1 = 0.69$ and $\lambda_2 = 0.62$.*

Then the expected number of jobs we schedule is at least

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

Let $\alpha_0 = \frac{\lambda_1}{\lambda_1 - \lambda_2 + 1} \geq 0.6448$. Notice that $\sum_j a_j + \sum_j b_j$ is the total number of jobs scheduled by the solution $\{x'_B\}$, scaled down by $1 + 1/k$, which is at least $(1 - 1/k)(1 - \epsilon/3)\mathrm{OPT}_{\mathrm{LP}}$, due to the Property (3). 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.

3 $1 - O\left(\sqrt{(1/m)\ln m}\right)$-approximation for JIS

In this section our goal is to prove Theorem 2. For convenience, we consider weighted JIS with $T = \text{poly}(n)$. In Appendix C, we show how to handle the unweighted case when T is super-polynomial in 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 $r_j \le t \le d_j - p_j$.

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

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

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 works as follows. After solving the LP, we round the 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. Then we order the jobs by their *starting* times in the tentative schedule. Consider a job j, whose starting time is t in the tentative schedule. Then we schedule job j at time t whenever we can. It is easy to see that we can schedule j 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, our goal is to bound the probability that a job is scheduled.

Lemma 5. *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 $i \ne j$ 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 $Z := \sum_{i \ne j} X_i \ge m$ 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 $(1-\epsilon)\left(1-\exp\left(-\frac{\epsilon^2 m}{3(1-\epsilon)}\right)\right)\sum_t x_{j,t}$. □

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

References

1. Adler, M., Rosenberg, A.L., Sitaraman, R.K., Unger, W.: Scheduling time-constrained communication in linear networks. Theory of Computing Systems 35(6), 599–623 (2002)
2. Bansal, N., Chan, H.L., Khandekar, R., Pruhs, K., Stein, C., Schieber, B.: Non-preemptive min-sum scheduling with resource augmentation. In: FOCS. pp. 614–624 (2007)
3. Baptiste, P.: 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. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating the throughput of multiple machines in real-time scheduling. SIAM Journal on Computing 31(2), 331–352 (2001)
5. Berman, P., DasGupta, B.: Improvements in throughout maximization for real-time scheduling. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing. pp. 680–687. ACM (2000)
6. Błażewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Weglarz, J.: Scheduling computer and manufacturing processes. Springer Science & Business Media (2013)
7. Chuzhoy, J., Ostrovsky, R., Rabani, Y.: Approximation algorithms for the job interval selection problem and related scheduling problems. Math. Oper. Res. 31(4), 730–738 (2006)
8. Fischetti, M., Martello, S., Toth, P.: The fixed job schedule problem with spread-time constraints. Operations Research 35(6), 849–858 (1987)
9. Garey, M.R., Johnson, D.S.: Two-processor scheduling with start-times and deadlines. SIAM J. Comput. 6(3), 416–426 (1977)
10. Hall, N.G., Magazine, M.J.: Maximizing the value of a space mission. European journal of operational research 78(2), 224–241 (1994)
11. Hong, K.S., Leung, J.Y.T.: Preemptive scheduling with release times and deadlines. Real-Time Systems 1(3), 265–281 (1989)
12. Koren, G., Shasha, D.: Dˆover: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. SIAM Journal on Computing 24(2), 318–339 (1995)
13. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A., Shmoys, D.: Sequencing and scheduling: Algorithms and complexity. Hanbooks in Operations Resarch vol. 4 (1993)
14. Lipton, R.J., Tomkins, A.: Online interval scheduling. In: SODA. pp. 302–311 (1994)
15. McDiarmid, C.: Concentration. In: Probabilistic methods for algorithmic discrete mathematics, pp. 195–248. Springer (1998)
16. Spieksma, F.C.: On the approximability of an interval scheduling problem. Journal of Scheduling 2(5), 215–227 (1999)

A Proof of Theorem 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, we consider this case first, and defer the case where $m \geq 2$ to Appendix B.

For some technicality issue, we need to modify the definition of a block slightly. A block B is defined as a triple (L_B, R_B, σ_B), where $L_B < R_B$ are integers and σ_B is a sequence of some distinct jobs in $\mathcal{J}$ (instead of a subset as in the previous definition). Let $\sigma_B(h)$ denote h_{th} job in the ordering σ_B. B is a block if σ_B can be scheduled inside $(L_B, R_B]$ on $m = 1$ machine: this means we can schedule the set of jobs in σ_B in $(L_B, R_B]$ according to the order σ_B. The size of a block, denoted as w_B, is the length of σ_B. By abusing notation, sometimes we also use σ_B to denote the set of jobs in the sequence σ_B. The initial LP we are considering is exactly the same as $\mathsf{LP}_{\mathsf{conf}}$ in Section 2, except that we change $\mathcal{J}_B$ to σ_B in Constraint (2). As before, we only consider blocks B with either $w_B = \Delta := 2mk^5 = 2k^5, k = \lceil 3/\epsilon \rceil$, or $R_B = T$.

It is easy to see that the whole algorithm in Section 2 works for this new LP since our new LP can only be stronger. To remove the dependence on T, we shall decrease the number of interesting blocks. We say a block 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*. For a tight block B, jobs in σ_B must be scheduled in $(L_B, R_B]$ without gaps. We first show that there are not so many loose minimal blocks to consider in the following lemma.

Lemma 6. *The number of loose minimal blocks of size s is at most $s^2 n^s$, for any positive integer s.*

Proof. Focus on a loose minimal block B and a schedule of σ_B in $(L_B, R_B]$. Since B is minimal, $\sigma_B(1)$ is started at L and $\sigma_B(\Delta)$ is completed 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. □

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. We say the sequence σ is *flexible* if $L_2 - L_1 \geq 2n \sum_{j \in \sigma} p_j$.

In the preprocessing step, we guarantee that there is no flexible sequence of length $\Delta + 1$. This is done greedily: 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.

Lemma 7. *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 in $\mathcal{J}'$, we can find a schedule of $A \cup \sigma$ 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 kicking out any job from A. □

Once we find a schedule for $\mathcal{J}'$, we can repeatedly add flexible sequences to the schedule following the process in the above lemma. 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 $\mathcal{J}$ where opt' is the optimum value for the instance defined by $\mathcal{J}'$. The approximation guarantee is at least α when $\alpha \leq \frac{\Delta}{\Delta+1}$. From now on, we assume there are no flexible sequences of length $\Delta+1$ in $\mathcal{J}$.

We now construct a set $\mathcal{B}$ of blocks of length Δ from $\mathcal{J}$. Initially $\mathcal{B}$ is the set of loose minimal blocks of size Δ. 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 (if no such L exists, nothing is added for this σ).

For any sequence σ of length $\Delta+1$, we shall add some blocks to $\mathcal{B}$ as follows. Let $P = \sum_{j\in\sigma} p_j$ be the total size of jobs in σ. Let L_1 (L_2) be the smallest (largest) 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). 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 $\mathcal{B}$.

Lemma 8. *The number of blocks in $\mathcal{B}$ is at most $10\Delta n^{\Delta+1}$.*

Proof. By Lemma 6, 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 1 block (L, T, σ) to $\mathcal{B}$. Now for each σ of length $\Delta+1$, we inserted at most $2\left\lceil \frac{L_2+1+P-L_1}{Q} \right\rceil \leq 2\left(\frac{(L_2+1+P-L_1)\cdot 2(\Delta+1)}{P} + 1\right) \leq 2(4n(\Delta+1)+1) \leq 9\Delta n$ 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 $\Delta^2 n^\Delta + n^{\Delta+1} + n^\Delta \times 9\Delta n \leq 10\Delta n^{\Delta+1}$ for large enough n. □

We solve the LP, with the restriction that only blocks in $\mathcal{B}$ can take positive x value. Since $|\mathcal{B}|$ is at most $10\Delta n^{\Delta+1}$, the LP can be solved in $n^{O(\Delta)}$ running time. We complete this section by showing that the set $|\mathcal{B}|$ suffices to give a good relaxation. The following lemma shows that this process of removing the dependency on T only loses a $1 - 1/(\Delta+1) = 1 - O(\epsilon^5)$ approximation factor. It is easy to see that allowing a block to have size $\Delta+1$ as well as Δ only helps the analysis. The general m will be handled in Appendix B.

Lemma 9. *The value of the LP is at least* $\frac{\Delta}{\Delta+1}\mathsf{opt}$.

Proof. Consider the optimum schedule for the instance and divide the set of scheduled jobs 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 $L' \geq L$). 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 shall find a block $(L', R', \sigma') \in \mathcal{B}$ such that $(L', R') \subseteq (L, R)$. This suffices to prove the lemma.

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 $R'' - L'' \leq \frac{\Delta}{\Delta+1}(R - L)$ 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'). By our construction, we have added a block (L', R', σ') with $R' \leq R'' \leq R$ to $\mathcal{B}$. □

B $(\alpha_0 - \epsilon)$-approximation for Unweighted JIS when $m \geq 2$ is a Constant 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 A. Since we have m machines, we need m sequences to define a block: a block $B = (L_B, R_B, \sigma_B)$, where $L_B < R_B$ are integers in $[0, T]$ and σ_B is a set of m sequences that can be scheduled in $(L_B, R_B]$. The jobs in the m sequences are all distinct, and the sequences may have different number of jobs. We call such a σ_B a *sequence tuple*. We use σ_B^i to denote the i-th sequence in σ_B and $\sigma_B^i(h)$ to denote the h-th job in the i-th sequence in σ_B. Recall that a sequence σ_B^i can be scheduled in $(L_B, R_B]$ if all jobs in the sequence can be scheduled during $(L_B, R_B]$ on a single machine according to the order σ_B^i. The size of a block is the total number of jobs in the m sequences.

Let $k = \lceil 1/\epsilon \rceil$ and $\Delta = 2mk^5$. Our LP is the same as $\mathsf{LP}_{\mathsf{conf}}$, except $\mathcal{J}_B$ in Constraint (2) is changed to $\bigcup_{i\in[m]} \sigma_B^i$. We only consider blocks B with either

$w_B = \Delta$, or $R_B = T$ and $w_B < \Delta + m$; we will impose additional restrictions on the set of blocks considered in the LP to reduce the running time.

We say a block B is *minimal* if both (L_B+1, R_B, σ_B) and (L_B, R_B-1, σ_B) are not blocks. A block $B = (L_B, R_B, \sigma_B)$ is *tight* if $R_B - L_B = \max_{i\in[m]} \sum_{j\in\sigma^i_B} p_j$, otherwise *loose*. Using essentially the same argument as in Lemma 6, we can prove that the number of loose minimal blocks B of size s is at most $s^2 n^s$.

Then we define the flexibility of a sequence tuple σ in a similar way as in Section A. Let $P = \max_{i\in[m]} \sum_{j\in\sigma^i} p_j$, L_1 (L_2) be the smallest (largest) L such that $(L, L+P, \sigma)$ is a tight block. Then, we say σ is flexible if $L_2 - L_1 \geq 2nP$. We can derive a result similar to Lemma 7: when adding σ to the existing schedule, we do it machine by machine; we can add σ^i to the i-th machine while kicking out at most one job. Since we kick out at most m jobs to schedule $\Delta+m$ new jobs in σ, removing flexible sequences is not an issue if the approximation ratio we are aiming at is at most $\frac{\Delta}{\Delta+m}$. Thus, we can assume there is no flexible sequence tuple of size $\Delta + m$.

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 time $n^{O(\Delta)}$. Let $P = \max_{i\in[m]} \sum_{j\in\sigma^i} p_j$. Let's focus on each σ^i. We will 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 $(L_1, L_2) = (\max_{i\in[m]} L^i_1, \min_{i\in[m]} L^i_2)$. Now we show how to find L^i_1. Since the ordering of jobs in σ^i is fixed, if the first job's starting point τ is given, then it is easy to find $R(\tau)$, the earliest time point we finish all jobs in σ^i by trying to schedule jobs as early as possible. Now imagine that we increase τ by one. Then the following three events can happen: (1) $R(\tau)$ remains the same; (2) $R(\tau)$ increases by 1; or $R(\tau)$ becomes ∞, meaning that there's no feasible schedule for σ^i starting at time τ. Notice that $R(\tau) - \tau$ can only decrease unless (3) happens. We would like to find in polynomial time the earliest τ such that $R(\tau) - \tau \leq P$, and the time τ such that $R(\tau+1) = \infty$. These two τ times will define (L^i_1, L^i_2). To speed up this process, we only need to focus on the first two chunks of jobs in the current schedule of σ^i – here by a chunk we mean a set of jobs scheduled consecutively with no gaps. Let's try to move the first chunk to the right. Then, one of the jobs in the first chunk may hit the deadline, or the first chunk may be merged with the second. The number of these discrete events is linear in the number of jobs in σ^i. Hence we can compute (L^i_1, L^i_2) in polynomial time.

We now discuss how we construct a set $\mathcal{B}$ of blocks. First all loose minimal blocks of size Δ are added 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 $n^{O(\Delta)}$.

Then, focus on sequence tuples 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) be the smallest (largest) 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 $\mathcal{B}$. By the same argument as in Section A, the size of $\mathcal{B}$ is $n^{O(\Delta)}$.

It now remains to show that solving the LP with the blocks in $\mathcal{B}$ loses only little compared to the optimum. 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 small). 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 $L'' - L + R - R'' \geq \frac{R-L}{\Delta+1}$, 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 $\mathcal{B}$.

C 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 3. Though the algorithm in Section 3 is for weighted JIS, we can only handle large T when the jobs are unweighted. Define a block $B = (L_B, R_B, \sigma_B)$ as in Section A: $L_B < R_B$ are integers in $[0, T]$ and σ_B is a sequence of different jobs that can be scheduled on $(L_B, R_B]$ on 1 machine. The size w_B of B is the number of jobs in σ_B. The LP we are considering is the

following. We only consider blocks B with either $w_B = \Delta := \lceil 1/\epsilon \rceil$ or $R_B = T$.

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

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.

It is not hard to see that the value of the above LP is at least the optimum throughput. To reduce the running time, we apply the same process as in Section A. We first remove flexible sequences of length $\Delta + 1$ and then construct the set $\mathcal{B}$ of blocks. By the same argument, we can prove that, 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. Notice that a fractional solution to the above LP yields a solution to the naive LP. Then, we can run the algorithm in Section 3. The final approximation ratio is $1 - O\left(\sqrt{\frac{\log m}{m}}\right) - \epsilon$. This proves the first part of Theorem 2.

D Proof of Lemma 4

Proof (of Lemma 4). Fix j. For notational convenience, drop j from the subscripts. Our goal is to show $1 - (1 - a/2)^2 e^{-b} \geq 0.69a + 0.62b$ for all $a, b \geq 0$ such that $a + b \leq 1$.

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

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

$$e^b(0.28b + 1.24) \geq (b + 1)^2 \tag{5}$$

To show this, we lower bound e^b by the following piecewise linear function.

$$g(b) = \begin{cases} e^{1/3}(b-1/3) + e^{1/3} \geq 1.3956b + 0.9304 & \text{if } b \in [0.00, 0.45) \\ e^{0.45}(b-0.45) + e^{0.45} \geq 1.5683b + 0.8625 & \text{if } b \in [0.45, 0.55) \\ e^{0.60}(b-0.60) + e^{0.60} \geq 1.8221b + 0.7288 & \text{if } b \in [0.55, 0.60) \\ e^{0.65}(b-0.65) + e^{0.65} \geq 1.9155b + 0.6704 & \text{if } b \in [0.60, 0.67) \\ e^{0.67}(b-0.67) + e^{0.67} \geq 1.9542b + 0.6448 & \text{if } b \in [0.67, 0.70) \\ e^{0.70}(b-0.70) + e^{0.70} \geq 2.0137b + 0.6041 & \text{if } b \in [0.70, 0.85) \\ e^{0.85}(b-0.85) + e^{0.85} \geq 2.3396b + 0.3509 & \text{if } b \in [0.85, 1.00] \end{cases}$$

Plugging in Eq. (5) each linear function in the right-hand-side and rearranging terms, it suffices to show that

$$\begin{cases} -0.609232b^2 - 0.008944b + 0.154192 \geq 0 & \text{if } b \in [0.00, 0.45); b_0 \approx -0.0073403 \\ -0.560876b^2 + 0.186192b + 0.0695 \geq 0 & \text{if } b \in [0.45, 0.55); b_0 \approx 0.165983 \\ -0.489812b^2 + 0.463468b - 0.096288 \geq 0 & \text{if } b \in [0.55, 0.60); b_0 \approx 0.473108 \\ -0.46366b^2 + 0.562932b - 0.168704 \geq 0 & \text{if } b \in [0.60, 0.67); b_0 \approx 0.607052 \\ -0.452824b^2 + 0.603752b - 0.200448 \geq 0 & \text{if } b \in [0.67, 0.70); b_0 \approx 0.666651 \\ -0.436164b^2 + 0.666136b - 0.250916 \geq 0 & \text{if } b \in [0.70, 0.85); b_0 \approx 0.763630 \\ -0.344912b^2 + 0.999356b - 0.564884 \geq 0 & \text{if } b \in [0.85, 1.00]; b_0 \approx 1.448711 \end{cases}$$

where b_0 denotes the value of $b \in (-\infty, \infty)$ that minimizes each of the quadratic functions (the global optimum). Hence the quadratic functions are minimized (from top to bottom) when $b = 0.45, 0.55, 0.6, 0.562932/0.46366/2, 0.7, 0.666136/0.436164/2, 0.85$ over their respective domains. For such values of b, the functions are valued at

$$\begin{aligned} -0.609232 * 0.45^2 - 0.008944 * 0.45 + 0.154192 &= 0.02679772 \\ -0.560876 * 0.55^2 + 0.186192 * 0.55 + 0.0695 &= 0.00224061 \\ -0.489812 * 0.6 + 0.463468 * 0.6 - 0.096288 &= 0.00546048 \\ -0.168704 - \frac{0.562932^2}{-0.46366 * 4} &> 0.0021 \\ -0.452824 * 0.7^2 + 0.603752 * 0.7 - 0.200448 &= 0.00029464 \\ -0.250916 - \frac{0.666136^2}{-0.436164 * 4} &> 0.0034 \\ -0.344912 * 0.85^2 + 0.999356 * 0.85 - 0.564884 &= 0.03536968, \end{aligned}$$

which are all positive, as desired. This completes the proof. □

E Concentration Inequalities

The following concentration inequality is a restatement of Theorem 2.3 in [15].

Theorem 3. *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 $\lambda \in [0, 1]$, we have*

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