Hierarchy-Based Algorithms for Minimizing Makespan under Precedence and Communication Constraints

Janardhan Kulkarni [*] Shi Li [†] Jakub Tarnawski [‡] Minwei Ye [§]

November 3, 2019

Abstract

We consider the classic problem of scheduling jobs with precedence constraints on a set of identical machines to minimize the makespan objective function. Understanding the exact approximability of the problem when the number of machines is a constant is a well-known question in scheduling theory. Indeed, an outstanding open problem from the classic book of Garey and Johnson [9] asks whether this problem is NP-hard even in the case of 3 machines and unit-length jobs. In a recent breakthrough, Levey and Rothvoss [25] gave a $(1+\epsilon)$-approximation algorithm, which runs in nearly quasi-polynomial time, for the case when job have *unit lengths*. However, a substantially more difficult case where jobs have arbitrary processing lengths has remained open. We make progress on this more general problem. We show that there exists a $(1+\epsilon)$-approximation algorithm (with similar running time as that of [25]) for the *non-migratory setting*: when every job has to be scheduled entirely on *a single machine*, but within a machine the job need not be scheduled during consecutive time steps. Further, we also show that our algorithmic framework generalizes to another classic scenario where, along with the precedence constraints, the jobs also have *communication delay* constraints. Both of these fundamental problems are highly relevant to the practice of datacenter scheduling.

1 Introduction

A classic problem in scheduling theory is as follows: We are given a set J of n jobs, where each job $j \in J$ has a processing length p_j. The jobs have *precedence constraints*, which are given by a partial order "$\prec$". A constraint $j \prec j'$ requires that job j' can only start after job j is completed. The jobs need to be scheduled on a set of m *identical* machines. The goal is to schedule the jobs while respecting the precedence constraints so as to optimize a certain objective function. The most popular objective function is the *makespan* of a schedule, which is the focus of this paper. The makespan of a schedule is defined as the completion time of the last job. In the classic three-field notation[1] [12], the problem is denoted by $P|\text{prec}|C_{\max}$. Since the seminal result of Graham [11], the problem has been studied quite extensively in the

*Microsoft Research, jakul@microsoft.com

†University at Buffalo, shil@buffalo.edu

‡Microsoft Research, jatarnaw@microsoft.com. Part of this work was done while the author was at cole Polytechnique Fdrale de Lausanne (EPFL).

§University at Buffalo, minweiye@buffalo.edu

[1]The first field describes the machine environment; in this paper we only consider the identical machine setting denoted by P. We use Pm to denote $m = O(1)$ machines, $P\infty$ when the number of machines is unbounded. The second field describes constraints that a schedule must satisfy (e.g. prec denotes precedence constraints), as well as assumptions on the input (e.g. $p_j = 1$ denotes unit-length jobs). The third field denotes the objective function, which in this paper is always the makespan ($C_{\max}$).

literature [12, 8, 6, 5, 28, 35, 17, 24, 37, 3, 25, 28, 14, 4, 30]. Despite this, large gaps remain in our understanding of the problem. The influential survey of Schuurman and Woeginger [33] and a more recent one by Bansal [2] list determining the exact approximability of this setting as one of the top ten open problems in scheduling theory (Open Problem 1).

Already in 1966, Graham [11] showed that any greedy non-idling schedule is a $(2 - 1/m)$-approximation to the problem of minimizing makespan with precedence constraints on identical machines. Fifty years later, assuming a variant of the Unique Games Conjecture (UGC) introduced by Bansal and Khot [3], Svensson [37] showed that an approximation factor of $(2 - \epsilon)$ is indeed hard for this problem, even for the unit-length case ($P|\text{prec}, p_j = 1|C_{\max}$). For the unit job length case, the current best approximation factor is $2 - 7/(3m + 1)$, when $m \geq 4$, due to Gangal and Ranade [8].

In most applications, however, the number of machines is typically much smaller than the number of jobs. Thus, a natural question that has attracted much attention is: what is the exact complexity of the problem if m is a constant (setting denoted by Pm)? One of the longest-standing open problems posed in the classic book by Garey and Johnson [9] is whether $Pm|\text{prec}, p_j = 1|C_{\max}$ is NP-hard for any $m \geq 3$. On the other hand, the more general problem where jobs have different processing lengths is strongly NP-hard even if all jobs have lengths 1 or 2 and there are only two machines if no preemption is allowed [39]. On the positive side, in a recent breakthrough Levey and Rothvoss [25] gave a $(1 + \epsilon)$-approximation algorithm for the problem $Pm|\text{prec}, p_j = 1|C_{\max}$ with running time $n^{(\log n)^{O_{m,\epsilon}(\log \log n)}}$, which is nearly quasi-polynomial time. They obtain this result by an elegant, though quite technically involved, rounding of a fractional solution obtained from the Sherali-Adams lift of an LP relaxation of the problem to $(\log n)^{O_{m,\epsilon}(\log \log n)}$ levels, which explains the running time of the algorithm. Later, Garg [10] made the result strictly quasi-polynomial time. However, the more general case, where jobs have arbitrary lengths, has remained open.

One of the tantalizing questions posed in [25] is whether LP hierarchies can also give a $(1 + \epsilon)$-approximation for the significantly harder case of jobs with arbitrary processing times. In this paper, we show that the answer to this question may depend on the type of schedule we are looking for. When jobs have arbitrary processing lengths, the optimal schedule can be of three types:

1. *Fully preemptive* ($Pm|\text{prec}, \text{pmtn}, \text{migration}|C_{\max}$): In this case, a job j can be scheduled on multiple machines (respecting the precedence constraints). However, at any time step t only one machine can be processing the job j.
2. *Preemptive but non-migratory* ($Pm|\text{prec}, \text{pmtn}|C_{\max}$): In this case, a job j must be scheduled completely on a single machine. However, within a machine the job can be preempted and need not be processed during consecutive time steps.
3. *Non-preemptive* ($Pm|\text{prec}|C_{\max}$): Here we require that a job j must be scheduled on a single machine during p_j consecutive time steps.

It is interesting to note that the optimal solutions for the above three cases can be quite different.[2] See Appendix A for examples where the above three schedules are a constant factor away from each other. Hence, there cannot be black-box reductions among these problems if our goal is to achieve $(1 + \epsilon)$-approximation. Observe that the fully preemptive case ($Pm|\text{prec}, \text{pmtn}, \text{migration}|C_{\max}$) is equivalent to the unit-length case, assuming all sizes are polynomially bounded: one can break a job j of length p_j into a chain of p_j unit-length jobs

[2]We are not aware of any other setting for makespan minimization where there is a gap between the optimal schedules 2 and 3. However, this situation is quite common for flow-time objective functions, where *preemptive but non-migratory* is the standard assumption.

(see Appendix B for how to handle the case when this assumption does not hold). Thus, the $(1+\epsilon)$-approximation algorithm due to [25] readily extends to this case.

In this paper we first consider the preemptive but non-migratory setting ($Pm|\text{prec},\text{pmtn}|C_{\max}$). We prove that there is a Sherali-Adams hierarchy based algorithm that gives a $(1+\epsilon)$-approximation, which generalizes the framework of Levey and Rothvoss [25] for the unit-length case to the general job length case, thus positively answering the question posed by the authors in our setting.

The first main result of this paper is the following.

Theorem 1.1. *For any $\epsilon > 0$, there is a $(1+\epsilon)$-approximation algorithm for the problem $Pm|prec,pmtn|C_{\max}$ that runs in time $n^{(\log n)^{O((m^2/\epsilon^2)\log\log n))}}$.*

We give a detailed explanation of our algorithm and techniques in Section 2.3, preceded by an introduction to Levey and Rothvoss' framework in Section 2.2. In Section 4, we give a complete proof of Theorem 1.1.

Next, we turn our attention to the non-preemptive case ($Pm|\text{prec}|C_{\max}$). Here we give some evidence to show that the algorithmic framework based on the Sherali-Adams hierarchy may not be sufficient to get a polynomial-time $(1+\epsilon)$-approximation, even when there are only 2 machines. Our reasoning behind this claim is as follows. The rounding algorithm used in the proof of Theorem 1.1 reduces our problem to the case of a deadline scheduling problem with the objective of maximizing throughput; that is, maximizing the number of job units completed, where a job may be processed partially. For this problem, somewhat surprisingly, we prove that any $o(\log n)$-level Sherali-Adams lift of the basic LP has at least a constant integrality gap. Since the throughput problem is equivalent to a special case of non-preemptive scheduling on 2 machines ($P2|\text{prec}|C_{\max}$), we believe that $P2|\text{prec}|C_{\max}$ also has a constant integrality gap with an $o(\log n)$-level Sherali-Adams lift. This is in contrast to the unit-length case, where a 2-level Sherali-Adams hierarchy can solve the problem $P2|\text{prec},p_j = 1|C_{\max}$ exactly [32], and many experts believe that $O(m)$ levels should give either an exact solution or a $(1+\epsilon)$-approximation. We give more details about the hard instances for $P2|\text{prec}|C_{\max}$ in Section 6.

The second main result of the paper concerns another classic problem: scheduling jobs with precedence and *communication delay constraints*. This model was introduced by Papadimitriou and Yannakakis [29] and Veltman et al. [40] to capture the fact that in multiprocessor systems, when jobs have dependencies $j \prec j'$, it takes time to transfer the output of a job j to another machine, where it will become the input of job j'. MapReduce systems and multi-core processors are modern examples of such systems. The formal setting of this problem is similar to that of makespan minimization with precedence constraints. However, if two jobs with $j \prec j'$ are executed on *different machines*, the second job j' is allowed to start only $c_{j,j'} \geq 0$ time units after the completion time of job j. Here $c_{j,j'}$ is the *communication delay* between j and j'. On the other hand, if j and j' are executed on the same machine, then j' can start right after j completes.

This model has been studied quite extensively in the literature, and yet our understanding of it is very limited. The surveys by Schuurman and Woeginger [33] and Bansal [2] list the approximability status of problems in this model as a top-ten open problem in scheduling theory. Most known results are for the special case where all the jobs have unit lengths and the communication delays $c_{j,j'}$ are also identically 1. In the classic notation, this special case is denoted by $P|\text{prec},p_j = 1, c = 1|C_{\max}$. For this problem, Hanen and Munier [15] gave a polynomial-time 7/3-approximation algorithm, while on the hardness side Hoogeveen et al. [18] showed the problem does not admit a better than 5/4-approximation algorithm unless $P = NP$. Another important case that has gained a lot of interest is when the number of machines is unbounded; the setting is non-trivial in presence of communication delays. The problem, denoted as $P\infty|\text{prec}, p_j = 1, c = 1|C_{\max}$, admits a 4/3-approximation due to Munier and Konig [27], and it is NP-hard to do better than 7/6 [18]. Papadimitriou and Yannakakis claim that there is a lower bound of 2 for the $P\infty|\text{prec}, p_j = 1, c|C_{\max}$ problem (where the communication delays are uniform but arbitrary), yet there has been no proof of this claim as far as we know; see Open Problem 3 in the survey by Schuurman and Woeginger [33] for more details.

As communication delay constraints with $c_{j,j'} \geq 0$ strictly generalize scheduling with precedence constraints[3], Svensson's hardness result [37] for $P|\text{prec}, p_j = 1|C_{\max}$, assuming UGC, also applies to our problem with the communication delay even for unit job lengths; that is, ($P|\text{prec}, p_j = 1, c_{j,j'}|C_{\max}$). Hence, we initiate the study of this problem when the number of machines is a constant. To the best of our knowledge, we are the first to consider the communication delay problem in the $m = O(1)$ setting. Our second main result is a generalization of our first result Theorem 1.1 to this setting.

Theorem 1.2. *For any $\epsilon > 0$, there is a $(1+\epsilon)$-approximation algorithm for the problem $Pm|prec, pmtn, c_{j,j'}|C_{\max}$ that runs in time $n^{(\log n)^{O((m^2/\epsilon^2)\log\log n)}}$ if $\max_{j \prec j'} c_{j,j'} = O(1)$.*

Observe also that $\max_{j \prec j'} c_{j,j'} = O(1)$ is a weaker assumption than $c = 1$, which is the only setting where previously known results hold. We obtain the above result by extending the Sherali-Adams hierarchy framework introduced for the problem without communication delay constraints, i.e., the setting of Theorem 1.1. This shows the versatility of the LP-hierarchy based approaches to the study of these problems. We anticipate that such approaches should help in resolving open problems in this model, which is one of the poorest-understood in scheduling. We introduce our techniques behind this result in Section 2.4 and give the complete analysis in Section 5.

1.1 Outline

The proofs of Theorems 1.1 and 1.2 are quite involved and build on [25], and hence need a fair amount of background. Therefore, our paper is organized as follows. In Section 2, we give a detailed but informal description of our two algorithms for Theorems 1.1 and 1.2, and discuss the new ideas of this paper. We formally define the properties of Sherali-Adams solutions that we use in our algorithms in Section 3. In Section 4, we first give a formal description of our algorithm for the first problem along with all the necessary lemmas, and show how these lemmas come together for the proof of Theorem 1.1; then we give complete proofs for these lemmas. In Section 5, we show how to extend the algorithm to the setting with communication delays, thus proving Theorem 1.2. Finally, we give our integrality gap result in Section 6.

2 High-Level Description of Our Algorithms and Techniques

Our proofs for Theorems 1.1 and 1.2 are obtained by rounding LP-hierarchy solutions of natural LPs for the problems. First we focus on the makespan minimization problem with precedence constraints for general job lengths when no migrations are allowed. Later we explain the new ideas needed to extend this result to the communication delay setting. We often refer to the former setting as the "no-delay" setting, and to the latter as the "delay setting".

As our work generalizes the framework introduced by Levey and Rothvoss [25], we begin by describing their algorithm for minimizing makespan when jobs have unit lengths. Beforehand, it will be worthwhile to give an intuitive explanation of LP hierarchies and the "conditioning" operation of LP-hierarchy solutions used in the design of our algorithms.

[3]This statement is not true if $c_{j,j'} > 0$ instead of $c_{j,j'} \geq 0$. Thus, Svensson's hardness result does not immediately apply to the $c = 1$ case, for example.

2.1 Intuitions behind Sherali-Adams Hierarchy and "Conditioning" Operation

We give a brief explanation of how we use the Sherali-Adams hierarchy in our algorithms; see Section 3 for more details. Let us start with an ideal situation. Assume we have a set $\mathcal{X} \subseteq \{0,1\}^n$ corresponding to the set of valid integral solutions for some instance of a problem. Further suppose that we are given a vector $x \in \text{convex-hull}(\mathcal{X})$, the convex hull of $\mathcal{X}$. That is, there is an implicit distribution π over $\mathcal{X}$ such that $x = \mathbb{E}_{\widetilde{x} \sim \pi} \widetilde{x}$. Then, we hope there is an oracle that, for an index $i \in [n]$ with $x_i > 0$, can return the vector $x' = \mathbb{E}_{\widetilde{x} \sim \pi : \widetilde{x}_i = 1} \widetilde{x}$; that is, the solution corresponding to the distribution $\widetilde{x} \sim \pi$ conditioned on the event $\widetilde{x}_i = 1$. This is called the "conditioning" or "inducing" operation. We use the word conditioning, as it is same as the conditioning operation known from probability theory.

In an intuitive sense, the Sherali-Adams hierarchy (and other LP/SDP hierarchies) provides a weaker form of the oracle that can support the conditioning operations. Assume there is a polytope $\mathcal{P} \supseteq \text{convex-hull}(\mathcal{X})$ that corresponds to the feasible solutions for some LP relaxation for the instance. Then the SA-hierarchy can be applied to $\mathcal{P}$, giving the oracle with the following restrictions. First, every vector x given to the oracle is associated with a *level* $\ell \in \mathbb{Z}_{>0}$. If we give a level-ℓ vector x to the oracle, the vector x' returned by the oracle will only have level $\ell - 1$. Second, the vector x given to the oracle does not need to be in $\text{convex-hull}(\mathcal{X})$. Indeed, for each level ℓ, the SA-hierarchy gives a polytope $\text{SA}(\mathcal{P}, \ell) \subseteq [0,1]^n$ such that

$$\mathcal{P} = \text{SA}(\mathcal{P}, 1) \supseteq \text{SA}(\mathcal{P}, 2) \supseteq \cdots \supseteq \text{convex-hull}(\mathcal{X}) .$$

The oracle only needs a level-ℓ vector x to be in $\text{SA}(\mathcal{P}, \ell)$ to perform the conditioning operation; the returned vector x' will be in $\text{SA}(\mathcal{P}, \ell - 1)$. So for this reason, the vectors in $\text{SA}(\mathcal{P}, \ell)$ on which the conditioning operation is performed are sometimes called "pseudo-distributions". Finally, to construct a vector $x \in \mathcal{P}_\ell$, in general we need a running time of $n^{\Theta(\ell)}$. This means that ℓ needs to be small, which places a limit on the number $\ell - 1$ of conditioning operations that we can apply to a vector x "sequentially".

Certain key properties will be satisfied by the (pseudo-)conditioning operation, as in the ideal case. For example, if we condition x on the event $\widetilde{x}_i = 1$, then the returned vector x' will have $x'_i = 1$. Also, conditioning can only shrink the support of vectors: if x' is obtained from x by conditioning, then $x_{i'} = 0$ implies $x'_{i'} = 0$ for every i'. Section 3 contains formal statements. See also the following references [23, 34, 22, 26, 31]; in particular, [31] is an excellent introduction to the use of hierarchies in approximation algorithms.

2.2 Overview of Levey-Rothvoss Algorithm

At the heart of the analysis of the Levey-Rothvoss algorithm [25] is the following simple observation: if the maximum chain length of jobs in the set J is at most ϵT, where T is a guess of the optimal makespan, then Graham's algorithm already gives a $(1 + \epsilon)$-approximation. At a high level, the algorithm in [25] uses the above observation in the following way: it partitions the input instance J into three sets $J_{\text{top}}, J_{\text{mid}}$ and J_{bot}. It guarantees that the size of J_{mid} is small, so for the moment we can ignore jobs in J_{mid}. The partitioning is done, guided by an LP-hierarchy solution, satisfying the following three properties: 1) The maximum chain length among J_{top} is small. 2) The precedence constraints between the jobs in the sets J_{top} and J_{bot} are *loose*, and can be easily satisfied. 3) There is a partition of the entire time horizon into small-length intervals such that each job in J_{bot} has fractional support completely contained in one interval in the partition. Then one can recursively schedule the jobs in J_{bot} by considering sub-instances defined by the small intervals, and such a schedule can be easily extended to include J_{top} due to Properties 1) and 2).

Now we give more details about the algorithm. The Levey-Rothvoss algorithm first makes a guess on the makespan T of the optimum schedule, which we can assume is a power of two. The time horizon $[T]$ is partitioned into a binary laminar family $\mathcal{I}$ of intervals, where the root interval is $[T]$ and each subsequent level is constructed by dividing each interval of the previous level into two equal-sized intervals. Each leaf interval contains a single slot. To obtain a schedule of J with makespan at most $(1+\epsilon)T$, it suffices to obtain a schedule with makespan T, but with up to ϵT jobs discarded. This new goal is indeed more convenient for the description and analysis of the algorithm. At the end, the discarded jobs are re-inserted into the final schedule, each on a time slot of its own.

We start by solving a standard LP for the problem lifted to r rounds of the Sherali-Adams hierarchy, for some $r = (\log n)^{O_{m,\epsilon}(\log\log n)}$, to obtain a solution x. It takes $n^{O(r)}$ time to solve a r-round Sherali-Adams lift of an LP with poly(n) variables; this is the reason for the running time of this algorithm. Based on the solution x, each job j is assigned to the smallest interval I in the binary decomposition $\mathcal{I}$ that fully contains the fractional schedule of the job. We then say j is *owned* by I.

The recursive algorithm begins by considering the jobs assigned to the first k^2 levels of the binary decomposition $\mathcal{I}$, for $k = \Theta(\log\log n)$. In the first step, we condition on some variables of the LP solution to reduce the maximum chain length among these jobs. The number of conditioning operations can be bounded due to the following reason: If there is a long chain of jobs in the set, we can condition on some variable so that the support of many jobs in the chain will shrink a lot; thus a few conditioning operations will be sufficient. In the second step, we partition the intervals in $\mathcal{I}$ into top, middle and bottom intervals. Let $\ell^* \in [k+1, k^2]$ be selected satisfying some property. The top intervals are those from level 0 to $\ell^* - k - 1$ in the laminar tree (the root has level 0), the middle intervals are those from level $\ell^* - k$ to $\ell^* - 1$ and the bottom intervals are those at level ℓ^* and below. Then, we define J_{top}, J_{mid} and J_{bot} as the set of jobs owned by top, middle, and bottom intervals respectively. In the third step, the rounding algorithm recursively and separately solves each of the bottom instances: such an instance is defined by a bottom interval $I \in \mathcal{I}$ at the level ℓ^* and the set of bottom jobs owned by sub-intervals of I, along with the solution x restricted to the interval I. This gives a schedule of the jobs inside interval I. In the fourth step, we insert (a large subset of) the top jobs into the constructed schedule to obtain our tentative schedule for the current instance.

Notice that the algorithm will discard all middle jobs and some top jobs. The number of middle jobs discarded can be bounded by choosing the parameter ℓ^* carefully, while the number of top jobs discarded across the entire algorithm can be bounded using Properties 1) and 2) mentioned above. Property 1) (the chain lengths among top jobs are small) is satisfied because of our conditioning in the first step. Property 2) (the precedence constraints between top and bottom jobs are "loose") is easy to satisfy using the fact that a top interval is at least 2^k times longer than the bottom intervals. We can "shrink" the fractional support of a top job by a small amount in order to remove the dependence between top and bottom jobs; since a top interval is so long, the shrinking operations only incur a small number of discarded top jobs. Finally, we can put together all these pieces to show that the total number of discarded jobs is at most ϵT, which gives a $(1+\epsilon)$-approximation to the problem.

2.3 Our Algorithm for General Job Lengths (Theorem 1.1)

A simple way to extend [25] to the case when jobs have arbitrary processing lengths would be the following: Replace each job j of length p_j by a chain $a_{1,j} \prec a_{2,j} \prec \ldots \prec a_{p_j,j}$ of length p_j, where each $a_{i,j}$ is a "task" of unit length. Now, treat these tasks as separate entities, and schedule them using the algorithm in [25]. However, as the subproblems in [25] are solved independently, this approach does not guarantee that the schedule is non-migratory; that is, tasks belonging to a single job are scheduled on the same machine. A natural idea to get around this issue is to assign a job to an interval in the binary laminar decomposition only if *all the tasks of the job* belong to that interval. This will make sure that the tasks belonging to a single job are handled in a single recursive call, and are not scheduled independently of each other. This is the approach we wish to take. However, one difficulty that arises is: how to reduce the maximum chain length among the top jobs? To understand this issue, consider the case when a long chain consists entirely of a single top job. In this case, no matter how we condition, the support of the job will not move down to the lower intervals. In general, it seems unavoidable that we have to allow tasks of a same job to go to different branches of recursive calls to keep the chain lengths among top jobs small. This brings us to the first main technical hurdle an algorithm for general job lengths has to deal with: How to keep track of tasks of a single job going into different recursive calls, and still ensure that they get scheduled on the same machine?

A crucial observation about the algorithm in [25] helps us in mitigating the tricky situation: the total number of conditionings performed to reduce the chain length of the top jobs is small in any recursive call of the algorithm. Let us call the jobs that the algorithm conditions on *special jobs*. There cannot be more than $O(\log T \cdot \gamma)$ such special jobs at any level of the recursion, where γ is the total number of conditionings performed by our algorithm in a single recursive call. As we argued earlier, this number is not too large. Further, for every special job, the entire support of the job becomes concentrated on a single machine. This is guaranteed by the non-migratory constraints of our machine-indexed LP (see Eq. (5)).

Our algorithm exploits the above two properties in tandem. For the special jobs, we allow tasks of a single job to go to different branches of the recursion. On the other hand, for a job that is not special, we ensure that all the tasks belonging to the job are considered by the same recursive call. For each special job, our algorithm needs to know exactly which tasks belonging to it go to each branch of the recursion. Moreover, we also require such tasks to be completely scheduled in that interval. This is needed to argue that there will be enough free slots to insert the top jobs after the tasks belonging to bottom jobs are recursively scheduled. Our algorithm accomplishes this by introducing a new type of conditioning operation called *splitting*, which precisely guarantees the above two invariants.

Using the splitting operation, our algorithm propagates all the tasks of special jobs down to the lowest level of the recursion. In particular, our algorithm maintains the invariant that the special jobs (more precisely, the tasks belonging to special jobs) will *never be a part of the set of top or middle jobs.* At the lowest level, the tasks of special jobs eventually get scheduled by conditioning on the LP solution. Since all the tasks of a special job have their support entirely on a single machine due to the LP constraints, this implies that they are all scheduled on the same machine.

Thus, the key ideas of special jobs and splitting conditioning operations help us to construct a non-migratory schedule of bottom jobs. Now, we need to extend this schedule to include top jobs, which brings us to the second major technical issue an algorithm for the general lengths case has to solve: How can we schedule top jobs in a non-migratory fashion in the slots left open by the bottom jobs? This is tricky because: a) Scheduling the bottom jobs is done independently of the top jobs and it is not guaranteed that every job has p_j units of empty slots on some machine. b) The LP solution for top jobs only says that there is enough space to schedule jobs if migration were allowed. Moreover, we also need to satisfy the precedence relationships among the top jobs, and between the top and bottom jobs.

The second key contribution of this paper is a new algorithm to insert the top jobs. Notice that the algorithm of Levey and Rothvoss [25] does give such a procedure, but only for the unit-length case. In contrast, our algorithm needs to deal with the substantially harder case of arbitrary job sizes and the non-migratory constraints. Our method proceeds in two stages. In the first stage, we build a *tentative schedule* which allows migration of jobs, but guarantees that the precedence constraints between the top and the bottom jobs are satisfied. To that end, we identify an interval $[r_j, d_j]$ for every job j such that if j is scheduled within this interval, then the precedence constraints between the top and the bottom jobs are satisfied. Here, $[r_j, d_j]$ may be shorter than the interval defined by the support of the LP solution for j. This leads to discarding some tasks; however, a simple argument based on an extension of Hall's theorem proved in [25] shows that the number of discarded tasks is small. This stage is similar to the procedure in [25, Section 5.1].

The more difficult question is how to convert the migratory schedule from the first stage into a non-migratory schedule respecting the precedence constraints among the top jobs; this is another technical contribution of our paper. To solve this problem, our algorithm considers the jobs in the Earliest Deadline First (EDF) order of their d_j values. Suppose $B_j \geq r_j$ is the first time slot where the job j can be scheduled respecting *all* the precedence constraints. Then our algorithm assigns j to the machine i on which it will have *the Earliest Completion Time* (ECT). Suppose C_j is its completion time. Now we make the following crucial observation about the ECT policy: If one looks at the interval $[B_j, C_j]$ in which j was scheduled, there can be at most p_j empty slots on any other machine $i' \neq i$. If this were not true, then it implies that there is another machine $i' \neq i$ and time slot $t' < C_j$ such that job j could have been feasibly scheduled in the interval $[B_j, t']$ on the machine i'. This leads to a contradiction with our policy. This observation, combined with the invariant maintained by our algorithm – that the chain lengths are small among the top jobs – guarantees that the number of slots we waste because of the precedence and non-migratory constraints is not too large.

Finally, it is possible that there is no machine i on which job j can be scheduled completely within its deadline d_j. In this case, we find the machine with the maximum number of empty slots, and schedule the job partially. We discard the tasks that we could not schedule. This *fragmentation* also leads to some more slots being wasted. However, we argue that the EDF+ECT policy guarantees that the number of slots wasted due to fragmentation is also small. In the end, somewhat surprisingly, we prove that the total number of tasks our algorithm discards is asymptotically the same as in [25].

2.4 Our Algorithm for Problem with Communication Delays (Theorem 1.2)

Now we give a high-level overview of how the LP hierarchy framework easily extends to the problem with communication delays where $\max_{j,j'} c_{j,j'} = O(1)$, i.e, $Pm|\text{prec}, p_j = 1, \text{pmtn}, c_{j,j'}|C_{\max}$, which we call the "delay problem". Recall that in the delay problem, along with precedence constraints, the algorithm also needs to enforce communication delay constraints; that is, if $j \prec j'$ and j and j' are scheduled on two different machines, then j' cannot start earlier than $C_j + c_{j,j'}$, where C_j is the completion time of job j. For simplicity we assume that all communication delays are 1. The arguments directly extend to the more general case $c_{j,j'} = O(1)$.

It should be clear from the overview of our algorithm for $Pm|\text{prec}, \text{pmtn}|C_{\max}$, which we call the "no-delay problem", that it is almost impossible to control how jobs are scheduled in the different branches of recursive calls. Hence, there is no easy way our algorithm can make scheduling choices taking into account the delay constraints. Instead, we let our recursive algorithm make choices without considering the communication delay constraints; when the actual assignment of tasks to time slots is carried out, we will enforce the communication delay constraints, assuming the worst possible scenario. Before we explain our strategy to do so, let us first summarize the three points at which our algorithm for the no-delay problem actually commits to the assignment of tasks to time slots.

1. Scheduling tasks by conditioning. At the lowest level of recursive call, our algorithm schedules the tasks by conditioning on the LP hierarchy solution.
2. Scheduling top jobs.
3. Scheduling discarded tasks.

Enforcing the communication delay in the first and third steps is rather easy: Our LP for the problem, which includes communication delay constraints, guarantees that if two jobs are scheduled by conditioning in the same recursive call, then the communication delay constraints are satisfied. To argue that communication delay constraints are also satisfied if they belonged to different branches of recursion needs a bit more work, but is not difficult. On the other hand, as the number of discarded tasks is small, we can afford to create three new private slots for each discarded task, and schedule the task in the middle, leaving the other two slots empty. This will take care of communication delay constraints, no matter how other tasks are scheduled. Thus, it only remains to argue how we guarantee that the communication delay constraints are satisfied while inserting the top jobs.

Here, we go back to a central idea in [25]: If chain lengths are small (bounded by ϵT), then Graham's algorithm already gives a $(1+\epsilon)$-approximation to the no-delay problem. This crucial observation easily extends in the presence of communication delay constraints: It is not hard to argue that Graham's list scheduling algorithm gives a $(1 + 2\epsilon)$-approximation to makespan if the chain lengths are small. Now note that in our algorithm the maximum chain length among the top jobs is small due to conditioning. We make use of this fact, along with several new observations, to give an extension of our algorithm for inserting top jobs for the no-delay problem to the delay problem.

To argue that communication delay constraints are also satisfied even if one job gets scheduled by conditioning and the other job gets scheduled as a top job needs some care and some additional tricks. However, the overall argument still relies mainly on the above three cases.

2.5 Note About Practical Applications

Besides being fundamental problems, the scheduling models studied in this paper have gained much importance recently in the context of datacenter scheduling literature; see a recent workshop on this topic [38] for more pointers. We give some context here. Programming models such as Dryad [7] or SparkSQL [36] compile scripts into job DAGs, which give rise to precedence-constrained scheduling problems. See [13, 1] and references therein. Similarly, communication delay problems arise in workloads from MapReduce systems and in the *model parallelism* paradigm [16] for training complex machine learning models on large clusters; see [16, 41, 20, 42] and references therein. We do not claim that the algorithms in this paper can be used in these applications directly; however, the framework of the algorithms proposed in this paper and in [25] shares many parallels to the heuristic for DAG scheduling developed (independently) in [13], which can be viewed as replacing the Sherali-Adams based "conditioning" step by a brute-force search; see here [21] for an exposition.

3 Basics of the Sherali-Adams Hierarchy

In this section, we formally state some basic facts about the Sherali-Adams hierarchy that we will need in our proofs. We refer the reader to [23, 34, 22, 26, 31] for an extensive introduction to hierarchies. The purpose of this section is to formally state the properties intuitively described in Section 2.1 so that there is no ambiguity in the proofs of Theorems 1.1 and 1.2. Hence, it can be skipped on the first reading. However, the integrality gap result in Section 6 needs to use the details in the definition of the SA hierarchy.

Assume we have a feasibility LP of the form $Ax \le b$, which includes the constraints $0 \le x_i \le 1$ for all $i \in [n]$. The set of feasible integral solutions is defined as $\mathcal{X} = \{x \in \{0,1\}^n : Ax \le b\}$. It is convenient to think of each $i \in [n]$ as an event, and in a solution $x \in \{0,1\}^n$, x_i indicates whether the event i happens or not.

The idea of the Sherali-Adams hierarchy is to strengthen the original LP $Ax \le b$ by adding more variables and constraints. Of course, each $x \in \mathcal{X}$ should still be a feasible solution to the strengthened LP (when extended to a vector in the higher-dimensional space). For some $r \ge 1$, the r-th round of the Sherali-Adams lift of the linear program has variables x_S for every $S \subseteq [n]$ of size at most r. For every solution $x \in \mathcal{X}$, x_S is intended to indicate whether all the events in S happen in the solution x; that is, $x_S = \prod_{i \in S} x_i$. Thus each $x \in \mathcal{X}$ can be naturally extended to a 0/1-vector in the higher-dimensional space defined by all the variables.

To derive the set of constraints, let us focus on the j-th constraint $\sum_{i=1}^n a_{j,i}x_i \le b_j$ in the original linear program. Consider two subsets $S, T \subseteq [n]$ such that $|S| + |T| \le r - 1$. Then the following constraint is valid for all $x \in \mathcal{X}$:

$$\prod_{i \in S} x_i \prod_{i \in T}(1 - x_i) \left(\sum_{i=1}^n a_{j,i}x_i - b_j \right) \le 0.$$

To *linearize* the above constraint, we expand the left side of the above inequality and replace each monomial with the corresponding variable $x_{S'}$. Then, we obtain the following linear constraint:

$$\sum_{T' \subseteq T} (-1)^{|T'|} \left(\sum_{i=1}^n a_{j,i}x_{S \cup T' \cup \{i\}} - b_j x_{S \cup T'} \right) \le 0. \tag{1}$$

The r-th round of the Sherali-Adams lift contains the above constraint for all j, S, T such that $|S| + |T| \le r - 1$, and the trivial constraint that $x_\emptyset = 1$. For a linear program $\mathcal{P}$ and an integer $r \ge 1$, we use $\mathrm{SA}(\mathcal{P}, r)$ to denote the r-th round Sherali-Adams lift of $\mathcal{P}$. We also view $\mathcal{P}$ (resp. $\mathrm{SA}(\mathcal{P}, r)$) as the polytope of feasible solutions to the linear program $\mathcal{P}$ (resp. $\mathrm{SA}(\mathcal{P}, r)$). For every $i \in [n]$, we identify the variable x_i in the original LP and $x_{\{i\}}$ in the lifted LP.

A simple observation is that $x_{S_1} \ge x_{S_2}$ if $S_1 \subseteq S_2$, for a valid solution $x \in \mathrm{SA}(\mathcal{P}, r)$ and $|S_1| \le |S_2| \le r$. For consider the case where $S_2 = S_1 \cup \{i\}$ for some $i \notin S_1$: linearizing the constraint $x_i \le 1$ multiplied by $\sum_{i' \in S_1} x_i$ gives the constraint $x_{S_2} \le x_{S_1}$. This implies that all the variables have values in $[0, 1]$, as $x_\emptyset = 1$.

Conditioning Let $x \in \mathrm{SA}(\mathcal{P}, r)$ for some linear program $\mathcal{P}$ on n variables and $r \ge 2$. Let $i \in [n]$ be an event such that $x_i > 0$; then we can define a solution $x' \in \mathrm{SA}(\mathcal{P}, r - 1)$ obtained from x by "conditioning" on the event i. For every $S \subseteq [n]$ of size at most $r - 1$, x'_S is defined as

$$x'_S := \frac{x_{S \cup \{i\}}}{x_i}.$$

Observation 3.1. *Let x' be obtained from $x \in \mathrm{SA}(\mathcal{P}, r)$ by conditioning on some event i, for some $r \ge 2$. Then $x' \in \mathrm{SA}(\mathcal{P}, r - 1)$ and $x'_i = 1$.*

Proof. $x'_i = \frac{x_{\{i\} \cup \{i\}}}{x_i} = \frac{x_i}{x_i} = 1$, and $x'_\emptyset = \frac{x_{\emptyset \cup \{i\}}}{x_i} = \frac{x_i}{x_i} = 1$. The constraint (1) on x' for j, S and T is implied by (1) on x for $j, S \cup \{i\}$ and T. □

Observation 3.2. *Let $x \in \mathrm{SA}(\mathcal{P}, r)$ for some $r \geq 2$ and $x' \in \mathrm{SA}(\mathcal{P}, r-1)$ be obtained from x by conditioning on some event i. Then, if $x_{i'} \in \{0,1\}$ for some $i' \in [n]$, then $x'_{i'} = x_{i'}$.*

Proof. If $x_{i'} = 0$, then $x'_{i'} = \frac{x_{\{i'\}\cup\{i\}}}{x_i} = 0$ since $x_{\{i'\}\cup\{i\}} \leq x_{i'} = 0$. Consider the case $x_{i'} = 1$. Expanding the constraint $(1-x_i)(1-x_{i'}) \geq 0$ gives the constraint $1 - x_i - x_{i'} + x_{\{i',i\}} \geq 0$. This implies $x_i = x_{\{i',i\}}$. Thus, $x'_{i'} = \frac{x_{\{i,i'\}}}{x_i} = 1$. □

The observation says that once an event i' happens with extension 0 or 1 w.r.t. a lifted solution x, then it will always happen with the same extension (0 or 1) w.r.t any solution x' obtained from x by conditioning. To understand the conditioning operation and the above observations better, it is useful to consider the ideal case where x corresponds to a convex combination of integral solutions in $\mathcal{X}$. Then we can view x as a distribution over $\mathcal{X}$. Then, conditioning on the event i over the solution x corresponds to conditioning on i over the distribution x.

4 Minimizing Makespan Under Precedence Constraints For General Job Lengths

In this section we consider the problem of minimizing the makespan when jobs have arbitrary processing lengths, and prove Theorem 1.1. The input to the problem consists of a set of jobs J, where each job $j \in J$ has a processing length p_j, and the precedence constraints $\prec$ between jobs. We imagine j as being made up of p_j tasks (or atoms) $a_{1,j}, a_{2,j}, \ldots, a_{p_j,j}$. The set of tasks of a job j is denoted by $A(j)$. Similarly, we define $A(J')$ as the set of tasks of jobs in J' for every $J' \subseteq J$. Formally, $A(J') := \bigcup_{j \in J'} A(j)$. We take the transitive closure of the precedence relations among jobs; that is, if $j_1 \prec j_2$ and $j_2 \prec j_3$, then we have $j_1 \prec j_3$.

We use the notation $a \sim a'$ to mean that the tasks a and a' belong to same job. Formally, $a \sim a'$ if there is $j \in J$ such that $a, a' \in A(j)$. The precedence constraints between jobs are extended to the tasks in the following natural way. For each job j, first we assume that $a_{1,j} \prec a_{2,j} \prec \ldots \prec a_{p_j,j}$. Consider any two jobs j and j' with precedence constraint $j \prec j'$. Then, for tasks $a \in A(j)$ and $a' \in A(j')$, we introduce a precedence constraint $a \prec a'$. Sometimes we also overload the precedence relation and write $a \prec j'$ to mean that every task of j' needs to be scheduled after the task a; that is, there is $j \in J$ such that $a \in A(j)$ and $j \prec j'$.

During the description of our algorithm we often go back and forth between two views. In the *task view* of the problem, we imagine our input as consisting of a set of tasks $A(J)$, each task with unit size, and precedence constraints as defined above. In the *job view*, we treat each job as a separate entity.

Our goal is to assign each job to a single machine and specify a schedule of tasks such that the precedence constraints among jobs are satisfied. Our objective is to minimize the makespan of the schedule, which is defined as the completion time of the last task. Formally, we define a valid schedule as follows.

A schedule $\mathcal{S}$ for a subset $A' \subseteq A(J)$ of tasks on an interval $I \subseteq [T]$ with integer length is a function $\mathcal{S} : A' \to [m] \times I$ that indicates the (machine, time slot) pair that each task is assigned to. For every $a \in A'$, we then use $\mathcal{S}_{\mathrm{mac}}(a)$ and $\mathcal{S}_{\mathrm{time}}(a)$ to denote the first and second component of $\mathcal{S}(a)$ respectively.

Definition 4.1. *A schedule $\mathcal{S}$ for $A' \subseteq A(J)$ is* valid *if it satisfies the following constraints.*

- *Capacity Constraints: for every two tasks $a \neq a' \in A'$, we have $\mathcal{S}(a) \neq \mathcal{S}(a')$.*
- *No-migration Constraints: For every pair of tasks $a \sim a' \in A'$, we have $\mathcal{S}_{\mathrm{mac}}(a) = \mathcal{S}_{\mathrm{mac}}(a')$.*
- *Precedence Constraints: For every pair of tasks $a \prec a' \in A'$, we have $\mathcal{S}_{\mathrm{time}}(a) < \mathcal{S}_{\mathrm{time}}(a')$.*

So, a valid schedule for A' guarantees that for a job j all the tasks of j in A' are assigned to a single machine. We define the completion time of job j as the time at which the last task of j is scheduled and denote it by C_j. If $j \prec j'$, then $C_j < C_{j'}$. Throughout the section, we use $N = \sum_j p_j$ to denote the total number of tasks. We can assume by appropriate scaling of the input that $N \leq n^2/\epsilon$; see Appendix B for details.

4.1 LP Relaxation

Our algorithm that proves Theorem 1.1 is based on rounding the Sherali-Adams lift of a natural LP for the problem. The variables of LP are $x_{(a,i,t)}$, which are intended to be 1 if the task $a \in A(J)$ is assigned to machine $i \in [m]$ at time $t \in [T]$. A natural LP formulation to decide if there is a valid schedule with makespan at most T is as follows:

$$\sum_{i,t} x_{(a,i,t)} = 1 \quad \forall a \quad (2) \qquad \sum_{i,t' \leq t+1} x_{(a',i,t')} \leq \sum_{i,t' \leq t} x_{(a,i,t')} \quad \forall a \prec a', t \in [T-1] \quad (4)$$

$$\sum_{a} x_{(a,i,t)} \leq 1 \quad \forall i,t \quad (3) \qquad \sum_{t} x_{(a',i,t)} = \sum_{t} x_{(a,i,t)} \quad \forall a \sim a', i \quad (5)$$

$$x_{(a',i,t)} \in [0,1] \quad \forall a \in A(J), i \in [m], t \in [T] \quad (6)$$

The constraints (2) guarantee that every task is feasibly scheduled, while (3) are the capacity constraints. The constraints (4) impose the precedence order among tasks. Here we appeal to the task view of the problem. Finally, (5) are intended to enforce the no-migration constraints: in a non-migratory schedule all the tasks $A(j)$ of a job j are scheduled on a single machine. Hence a valid schedule satisfies those constraints. Therefore, if there is an optimal integral solution with makespan at most T, then there is a feasible solution to the LP. We use $\mathcal{P}(T)$, and simply $\mathcal{P}$ when T is clear from the context, to denote the polytope defined by the above LP relaxation.

Towards proving the main result (Theorem 1.1), we first design a LP rounding algorithm that only schedules a subset $A(J) \setminus A_{\text{discarded}}$ of tasks. Our main goal in this section is to prove the following lemma.

Lemma 4.2. *Let T be the smallest value for which the Sherali-Adams lift of LP (2-6) to $r = (\log n)^{O((m^2/\epsilon^2)\cdot \log\log n)}$ rounds has a feasible solution x. In time $n^{O(r)}$ we can find a valid schedule $\mathcal{S} : A(J) \setminus A_{\text{discarded}} \to [m] \times [T]$ with $|A_{\text{discarded}}| \leq \epsilon T$.*

We give a brief sketch of the proof that a partial schedule $\mathcal{S}$ satisfying the guarantees of the above lemma can be easily extended to a valid schedule $\mathcal{S}^*$ for $A(J)$ with makespan $[T + |A_{\text{discarded}}|]$ as follows. Consider a discarded task a. Let t be the earliest time slot such that scheduling a at t satisfies all the precedence constraints. We create a separate time slot for a at time t, and schedule it there. We do not schedule any other task at t, and shift the entire schedule of tasks following t by one time step. Since this time slot can be created on any machine, the non-migratory constraints can be satisfied. We repeat this procedure for every discarded task. Hence scheduling the tasks in $A_{\text{discarded}}$ increases the makespan by an additive factor of $|A_{\text{discarded}}|$. As $|A_{\text{discarded}}| \leq \epsilon T$, this implies that makespan of our final schedule $\mathcal{S}^*$ is at most $(1+\epsilon)T$. As the Sherali-Adams lift of the linear program is a valid relaxation of the optimal solution, the optimal makespan has to be at least T. Moreover, one can solve the lifted linear program in time $n^{O(r)}$. Putting together all these facts, we conclude that Lemma 4.2 implies Theorem 1.1.

4.2 Rounding Algorithm

Our algorithm to prove Lemma 4.2 is a generalization of the algorithm in [25]. Hence, for easier reading we try to keep the notation and structure of our paper similar to [25] as much as possible.

We begin by partitioning the interval $[T]$ into a balanced binary family $\mathcal{I}$ of intervals of length $T, T/2, T/4, \ldots, 2, 1$. W.l.o.g., we can assume that T is a power of two using the padding trick in [25]. One way to visualize $\mathcal{I}$ is to think of a balanced binary tree, where the root node corresponds to the interval $[T]$, and nodes at level ℓ correspond to the 2^ℓ intervals obtained by partitioning $[T]$ into sub-intervals of length $T/2^\ell$.

We define some notation that will be used throughout the paper. For an interval $I \in \mathcal{I}$ and a subset of jobs $J' \subseteq J$ we define

$$J'(I,x) = \left\{ j \in J' : \forall a \in A(j), \sum_i \sum_{t \in I} x_{(a,i,t)} = 1 \right\}$$

as the subset of jobs with support completely in the interval I in the LP-hierarchy solution x. Similarly, we define

$$A(I,J',x) = \left\{ a \in A(J') : \sum_i \sum_{t \in I} x_{(a,i,t)} = 1 \right\}$$

as the set of tasks belonging to jobs in J' that have their entire support in the interval I. We emphasize that I may contain partial support of some tasks of $A(J')$, but they are not included in the set $A(I,J',x)$. We use $A(I,j,x) \subseteq A(j)$ as a shorthand for $A(I,\{j\},x)$, i.e, as the subset of tasks of job j that have entire support in I.

Finally, we define four global constants k, δ, K and K'. We set $k = \frac{O(1)m}{\epsilon} \cdot \log\log T$ to be large enough and $\delta = \frac{\epsilon}{8k^2 m 2^{2k^2} \log T}$. Let $K = \frac{1}{\delta} \cdot mk^2 \cdot 2^{k^2}$ and $K' = K \cdot 2^{k^2} = \frac{1}{\delta} \cdot mk^2 \cdot 2^{2k^2}$. The parameter δ is used to define what constitutes a long chain. The roles of constants k, K and K' will become clear when we give the description of our algorithm. For a set $J' \subseteq J$ of jobs, we use $\Delta(J')$ to denote the maximum possible *total size* of jobs in a precedence-chain in J'.

The recursive algorithm PARTIAL-SCHEDULE used to prove Lemma 4.2 is given as Algorithm 1. The main claim of [25] is that if we fix any interval $I \in \mathcal{I}$ and consider the set of tasks that are entirely scheduled in the interval I by the LP solution x, then one can find a partial schedule of the tasks in the interval I that discards few tasks. The main contribution of this paper is that a similar statement can be shown even when jobs have arbitrary lengths and we enforce no-migration constraints. In order to give the main lemma, we shall first define a *partial-scheduling* instance.

Definition 4.3. *In a partial-scheduling instance, we are given an interval $I^* \in \mathcal{I}$, an LP-hierarchy solution $x \in \mathrm{SA}(\mathcal{P}, r)$ for $r = \log|I^*| \log T \cdot K'$, a set $J^* \subseteq J(I^*, x)$ of jobs that have complete support in I^* according to x, and a special set $J^*_{\text{special}} \subseteq J$ of jobs disjoint from J^*, with $|J^*_{\text{special}}| \leq \log \frac{T}{|I^*|} \cdot K$. Further, we are given a function $\sigma : J^*_{\text{special}} \to [m]$ such that for every $j \in J^*_{\text{special}}$, we have*

1. *j is only scheduled on $\sigma(j)$ in x, i.e, $x_{(a,i,t)} = 0$ if $a \in A(j)$ and $i \neq \sigma(j)$,*
2. *every task a of j is completely scheduled in I^* w.r.t. x, or not scheduled in I^* at all. That is, $\sum_{t \in I^*} x_{(a,\sigma(j),t)}$ is either 0 or 1.*

The goal of the scheduling problem is to schedule (a subset of) $A(J^*) \cup A(I^*, J^*_{\text{special}}, x)$ in I^*. We denote the partial-scheduling instance by $(I^*, J^*, J^*_{\text{special}}, \sigma, x)$.

Observe in the above definition that for each special job $j \in J^*_{\text{special}}$ in a partial-scheduling instance, we are given a machine $\sigma(j)$ on which j must be scheduled. Moreover, we know exactly the set of special-job tasks that must be scheduled: these are the tasks that are completely scheduled in I^* on $\sigma(j)$ in the LP solution x, and the other tasks are not scheduled at all. Notice that the tasks to be scheduled are consecutive in the task chain for job j due to the precedence constraints.

The following main lemma bounds the number of tasks discarded by the recursive algorithm PARTIAL-SCHEDULE that solves the partial scheduling problem.

Lemma 4.4 (Main Lemma). *Let $(I^*, J^*, J^*_{\text{special}}, \sigma, x)$ be a partial-scheduling instance, and let $A^* := A(J^*) \cup A(I^*, J^*_{\text{special}}, x)$ be the tasks we need to schedule. Then,* PARTIAL-SCHEDULE *returns a valid schedule $\mathcal{S}$ for $A^* \setminus A_{\text{discarded}}$ of makespan I^* for a set $A_{\text{discarded}}$ of discarded tasks of size at most*

$$|A_{\text{discarded}}| \leq \frac{\epsilon}{2} \cdot \frac{\log |I^*|}{\log T} \cdot |I^*| + \frac{\epsilon}{2m} \cdot |A^*|.$$

Moreover, for every job $j \in J^*_{\text{special}}$, the set $A(I^*, j, x)$ of tasks is scheduled on the machine $\sigma(j)$.

Observe that Lemma 4.2 follows immediately by instantiating the above lemma for the entire interval $[T]$ and the set of jobs $J^* = J$ and $J^*_{\text{special}} = \emptyset$. Then the total number of discarded tasks will be

$$\frac{\epsilon}{2} \cdot \frac{\log T}{\log T} \cdot T + \frac{\epsilon}{2m} \cdot |A(J)| \leq \epsilon \cdot T,$$

where we used the fact that $|A(J)| \leq mT$. Definition 4.3 also requires an LP-hierarchy solution x of level r at least $(\log T)^2 \cdot K'$.

We have $T \leq N \leq \text{poly}(n)$. As we set $k = \frac{O(1)m}{\epsilon} \cdot \log\log T$ and $\delta = \frac{\epsilon}{8k^2 m 2^{2k^2} \log T}$ and $K' = \frac{1}{\delta} \cdot mk^2 \cdot 2^{2k^2}$, we have

$$r = (\log T)^2 \cdot K' = (\log n)^{O((\frac{m}{\epsilon})^2 \log\log n)}.$$

The term that determines the asymptotic running time of our algorithm in the above equation is 2^{2k^2}, which is at most $(\log n)^{O((\frac{m}{\epsilon})^2 \log\log n)}$.

From now on we focus solely on proving Lemma 4.4, and we assume we are given an instance $(I^*, J^*, J^*_{\text{special}}, \sigma, x)$. During our algorithm, I^* does not change, but we shall move jobs from J^* to J^*_{special} and extend σ accordingly. The LP-hierarchy solution x will also be updated using the conditioning operation. Let T^* always denote $|I^*|$; note that T^* is some power of two. Let $\mathcal{I}^*$ denote the set of intervals in $\mathcal{I}$ that are sub-intervals of I^*. For an integer $\ell \in [0, \log T^*]$, we use $\mathcal{I}^*_\ell$ to denote the intervals in $\mathcal{I}^*$ with length $T^*/2^\ell$; thus, $\mathcal{I}^* := \mathcal{I}^*_0 \cup \mathcal{I}^*_1 \cup \ldots \cup \mathcal{I}^*_{\log T^*}$, and each $\mathcal{I}^*_\ell$ contains 2^ℓ intervals of length $\frac{T^*}{2^\ell}$ each.

For a job $j \in J^*$, let $I' \in \mathcal{I}^*$ be the interval of smallest length such that it contains the entire support of j. We say that I' *owns* the job j. If $I' \in \mathcal{I}^*_\ell$, then we also say that the level ℓ *owns* the job j. We use the notation $J^*_{\ell'}(x)$ to denote the subset of jobs in J^* that are owned by a level $\ell' \in [\log T^*]$. That is,

$$J^*_{\ell'}(x) := \left\{ j \in J^* : \text{ level } \ell' \text{ owns } j \right\}.$$

Contrast this with notation $J^*(I, x)$, which indicates the set of jobs that have full support in the interval I. Also notice that only jobs in J^* are owned by intervals or levels. We say that "the algorithm conditions on job j" or "condition on an event (a, i, t)" to mean that our

Algorithm 1 PARTIAL-SCHEDULE$\left(I^*, J^*, J^*_{\text{special}}, \sigma, x\right)$

Input: a partial-scheduling instance $(I^*, J^*, J^*_{\text{special}}, \sigma, x)$ satisfying Definition 4.3
Output: a schedule $\mathcal{S}^* : A(J^*) \cup A(I^*, J^*_{\text{special}}, x) \setminus A_{\text{discarded}} \to [m] \times I^*$ for some $A_{\text{discarded}}$

```
1: if $|I^*| < 2^{k^2}$ then schedule tasks by conditioning and return
2: for every job $j \in J^*_{\text{special}}$ do: $x \leftarrow \mathsf{SPLIT}(x, j)$
3: while there exists $I \in \mathcal{I}^*_0 \cup \ldots \cup \mathcal{I}^*_{k^2-1}$ and a chain $\mathcal{C}$ of jobs in $J^*$ owned by $I$ with total
   size at least $\delta|I|$ do
4:     $j \leftarrow$ the first job in $\mathcal{C}$, $a \leftarrow$ last task of $j$
5:     take $(i, t)$ such that $x_{(a,i,t)} > 0$ with the largest $t$
6:     $x \leftarrow x$ conditioned on the event $(a, i, t)$
7:     $J^* \leftarrow J^* \setminus \{j\}$, $J^*_{\text{special}} \leftarrow J^*_{\text{special}} \cup \{j\}$, $\sigma(j) \leftarrow i$
8:     $x \leftarrow \mathsf{SPLIT}(x, j)$
9: Partition the jobs in the set $J^*$ as follows:
       $J^*_{\text{top}} = \bigcup_{\ell=0}^{\ell^*-k-1} J^*_\ell(x)$; $J^*_{\text{mid}} = \bigcup_{\ell=\ell^*-k}^{\ell^*-1} J^*_\ell(x)$; $J^*_{\text{bot}} = \bigcup_{\ell=\ell^*}^{\log T^*} J^*_\ell(x)$,
10: where $\ell^* \in \{k, \ldots, k^2\}$ is chosen satisfying the condition below:
       $$|A(J^*_{\text{mid}})| \le \frac{\epsilon}{4} \cdot \frac{T^*}{\log T} + \frac{\epsilon}{2m} \cdot \left(|A(J^*_{\text{mid}})| + |A(J^*_{\text{top}})|\right)$$
11: for every interval $I \in \mathcal{I}^*_{\ell^*}$ do
12:     PARTIAL-SCHEDULE$\big(I, J^*_{\text{bot}}(I, x), J^*_{\text{special}}, \sigma, x\big)$
13: Insert $J^*_{\text{top}}$ into $I^*$ using Lemma 4.6.
```

Algorithm 2 SPLIT(x, j), where $j \in J^*_{\text{special}}$

```
1: for every $I \in \mathcal{I}^*_{k^2}$ from left to right do
2:     $t^* := \max\left\{t \in I : (\exists a \in A(I^*, j, x))\ x_{(a,\sigma(j),t)} > 0\right\}$
3:     if $t^*$ is defined then
4:         let $a^* \in A(j)$ be any task with $x_{(a^*,\sigma(j),t^*)} > 0$
5:         $x \leftarrow x$ conditioned on $(a^*, \sigma(j), t^*)$
6: return $x$
```

algorithm conditions on the event $x_{(a,i,t)} = 1$ for a task $a \in A(j)$. Similarly, we use the phrase "the algorithm conditions on task a". Note that as the solution x changes due to conditioning, the sets of jobs owned by certain intervals and levels can change as well. Recall that we call a subset of jobs $J' \subseteq J$ a *chain* if the precedence relation $\prec$ gives a *total ordering on* J'. For any subset J' of jobs, $\Delta(J')$ is defined as the maximum of $\sum_{j\in J''} p_j$ over all chains $J'' \subseteq J'$.

4.3 Main Steps of PARTIAL-SCHEDULE

Now we give details about our algorithm, which consists of five main steps. The pseudo-code for the algorithm, which we call PARTIAL-SCHEDULE, is given in Algorithm 1.

Step 1: Reducing Chain Length Among Top Jobs. Let $\widehat{J} := \bigcup_{\ell=0}^{k^2-1} J^*_\ell(x)$ be the set of jobs that are owned by the first $k^2 - 1$ levels of $\mathcal{I}^*$ in the solution x. By appropriately choosing certain jobs in $\widehat{J}$ and conditioning on events of the form $x_{a,i,t} = 1$, our algorithm maintains the invariant that there are no long chains in $\widehat{J}$. In particular, the maximum chain length $\Delta(\widehat{J}) \le k^2\delta T^*$.

The LP-hierarchy solution x changes in the following way. For every job j on which our algorithm does the conditioning, the entire support of job j gets concentrated on a single machine. The reason is that when we condition on an event $x_{a,i,t} = 1$, the non-migratory constraints in our LP (Eq. 5) force all the other tasks a' of j to also have their entire support on machine i. Since conditioning can only shrink the support of jobs, this property remains true regardless of future conditioning operations. For every job j on which our algorithm does conditioning, we define $\sigma(j)$ as the machine on which the entire support of j resides in x. Further, we insert the job into the set J^*_{special} of special jobs and delete it from J^*. Hence, by induction, we are guaranteed that in any recursive call to PARTIAL-SCHEDULE with input parameters J' and J'_{special}, the invariant $J' \cap J'_{\text{special}} = \emptyset$ is satisfied.

Another consequence of conditioning is that the support of some jobs in the set $\widehat{J}$ shrinks, and they move down the levels in $\mathcal{I}^*$. Note however that a job $j \in \widehat{J}$ can move down only k^2 levels. We use this observation along with a simple counting argument to show that the total number of conditioning operations required to reduce the chain length among jobs in the set $\widehat{J}$ is at most K', which is independent of the length of the interval I^*. In particular, we prove the following lemma in Section 4.5.

Lemma 4.5. *Let $\widehat{J} := \bigcup_{\ell=0}^{k^2-1} J^*_\ell(x)$ denote the set of jobs owned by the intervals in $\mathcal{I}^*_0, \mathcal{I}^*_1, \ldots, \mathcal{I}^*_{k^2-1}$ according to the LP hierarchy solution x. Then, after line 8 in* PARTIAL-SCHEDULE, *we have that $\Delta(\widehat{J}) \leq k^2\delta T^*$. Moreover, the number of iterations we run the loop on line 3 is at most K.*

Step 2: Splitting Special Jobs. For every job in the set J^*_{special}, we perform the operation of *splitting*, which is given by the procedure SPLIT defined in Algorithm 2. Recall that for every $j \in J^*_{\text{special}}$, $\sigma(j)$ is defined. Our algorithm guarantees that all the tasks of j are scheduled on $\sigma(j)$. The idea behind splitting is to ensure that every task of a job $j \in J^*_{\text{special}}$ is pushed to the lowest level of the recursion, where it will eventually get scheduled by conditioning. This guarantees that even if different tasks of a special job get assigned to different intervals, and hence may be part of different recursive calls, all of them will be scheduled on the machine $\sigma(j)$. In order to do so, we need the sub-instances to satisfy Property 2 in Definition 4.3. That is, each task $a \in A(j)$ should either be completely scheduled in the sub-interval I in the solution x or not scheduled at all. This is exactly what the procedure SPLIT does.

Here is how the splitting procedure for a job $j \in J^*_{\text{special}}$ works. Let $A(I^*, j, x) := \{1, 2, \ldots, g\}$. Starting from the left, let us denote the intervals in $\mathcal{I}^*_{k^2}$ by $I_1, I_2, \ldots, I_{2^{k^2}}$. Now, consider an interval I_u for $u \in [2^{k^2}]$. Define $t^* := \max\{t \in I_u : (\exists a \in A(I^*, j, x))\ x_{(a,\sigma(j),t)} > 0\}$ as the rightmost time slot t in the interval I_u for which some $a \in \{1, 2, \ldots, g\}$ has positive support on machine $\sigma(j)$. Let a^* be such a task. Now we condition on the event $x_{a^*,\sigma(j),t^*} = 1$. After conditioning, the fractional solution has the following property: every task $a' > a^*$ is scheduled completely in $I_{u+1} \cup I_{u+2} \cup \cdots \cup I_{2^{k^2}}$ and every task $a' \leq a^*$ is scheduled completely in $I_1 \cup I_2 \cup \cdots \cup I_u$. This follows from the precedence constraints in our LP and the choice of t^*. So, by repeating the operation for u from 1 to 2^{k^2}, we partition the tasks in $A(I^*, J^*_{\text{special}}, x)$ into 2^{k^2} "chunks", each of which contains a subset of consecutive tasks from $A(I^*, j, x)$. We apply the procedure first for all the jobs in the original set J^*_{special}. When a new job j is added to J^*_{special} because of Step 1 of our algorithm, we also apply the procedure to j. Notice that splitting may shrink the support of other jobs $j' \in J^* \cup J^*_{\text{special}}$ because of the conditioning operations; the running of step 2 is actually interleaved with the running of step 1.

For each special job we perform at most 2^{k^2} conditioning operations during splitting, which we show is acceptable for our targeted running time. A crucial observation is that the total number of special jobs at any level of recursion is small, hence the number of extra conditioning operations performed by our algorithm is small.[4] This guarantees that the number of levels left in the LP hierarchy solution x satisfies the requirements of Lemma 4.4.

Step 3: Partitioning Jobs into Top, Middle and Bottom Jobs. Let x be the LP-hierarchy solution after the first two steps, and J^* be the current set of jobs which have the entire support in I^*. Note that the set J^* may have reduced in size after the first two steps, as we move some jobs from J^* to J^*_{special} in Step 1. Next, we select an index $\ell^* \in \{k, \dots, k^2\}$ and partition the jobs in J^* into three sets.

1. A set of *top jobs* denoted by J^*_{top}. These are the jobs owned by the levels $0, \dots, \ell^* - k - 1$; formally, $J^*_{\text{top}} = \bigcup_{\ell=0}^{\ell^*-k-1} J^*_\ell(x)$.
2. A set of *middle jobs* denoted by J^*_{mid}. These are the jobs owned by the levels $\ell^* - k, \dots, \ell^* - 1$; formally, $J^*_{\text{mid}} = \bigcup_{\ell=\ell^*-k}^{\ell^*-1} J^*_\ell(x)$.
3. A set of *bottom jobs* denoted by J^*_{bot}. These are the jobs owned by the levels $\ell^*, \dots, \log T^*$; formally, $J^*_{\text{bot}} = \bigcup_{\ell=\ell^*}^{\log T^*} J^*_\ell(x)$.

Our algorithm completely discards the middle jobs. [25] showed using a counting argument that there exists an index $\ell^* \in \{k, \dots, k^2\}$ satisfying the following relation:

$$|A(J^*_{\text{mid}})| \leq \frac{\epsilon}{4} \cdot \frac{T^*}{\log T} + \frac{\epsilon}{2m} \cdot \left(|A(J^*_{\text{mid}})| + |A(J^*_{\text{top}})|\right) \tag{7}$$

In this paper we assume that such an index exists and we refer the reader to [25] for more details.

Step 4: Recursing on Bottom Jobs. In this step we find a partial schedule for the bottom jobs. Notice that J^*_{bot} is the union of the 2^{ℓ^*} disjoint sets $\{J^*(I,x)\}_{I \in \mathcal{I}^*_{\ell^*}}$. For each interval $I \in \mathcal{I}^*_{\ell^*}$, the algorithm finds a partial schedule of jobs in $J^*(I,x) \subseteq J^*_{\text{bot}}$ in the interval I by recursively invoking the procedure with input parameters $(I, J^*(I,x), J^*_{\text{special}}, \sigma, x)$. It is crucial to note that every invocation of the recursive algorithm receives an *independent* copy of the LP-hierarchy solution x; a programmer might say that x is passed by value. In other words, the conditioning done in recursive call $\mathsf{PARTIAL\text{-}SCHEDULE}(I, J^*(I,x), J^*_{\text{special}}, x)$ has no effect on the recursive call $\mathsf{PARTIAL\text{-}SCHEDULE}(I', J^*(I',x), J^*_{\text{special}}, x)$ if $I, I' \in \mathcal{I}^*_{\ell^*}$ are different intervals. In fact, it can be imagined as being done in parallel.

A recursive application of Lemma 4.4 returns a feasible schedule of some tasks in $A(J^*(I,x)) \cup A(I, J^*_{\text{special}}, x))$ satisfying the conditions stated in the lemma. Let $A_{I,\text{discarded}}$ be the set of tasks discarded by our algorithm in the interval $I \in \mathcal{I}^*_{\ell^*}$. Let

$$\mathcal{S}_I : A(J^*(I,x)) \cup A(I, J^*_{\text{special}}, x) \setminus A_{I,\text{discarded}} \rightarrow [m] \times I$$

denote the schedule returned by our recursive calls for each $I \in \mathcal{I}^*_{\ell^*}$. Let $A_{\text{bottom-discarded}} = \bigcup_{I \in \mathcal{I}^*_{\ell^*}} A_{I,\text{discarded}}$. Then a schedule $\mathcal{S}$ of non-discarded tasks in bottom jobs and the tasks of special jobs which have complete support in I can be obtained by combining the schedules $\mathcal{S}_I$. Let

$$\mathcal{S} : A(J^*_{\text{bot}}) \cup A(I^*, J_{\text{special}}, x) \setminus A_{\text{bottom-discarded}} \rightarrow [m] \times I^*$$

denote this combined schedule. Formally, for a task a that belongs to an interval I, $\mathcal{S}(a) := \mathcal{S}_I(a)$. From our construction, $\mathcal{S}$ is a valid partial schedule.

Step 5: Scheduling Top Jobs. At this stage, it remains to assign tasks in the set $A(J^*_{\text{top}})$ in a non-migratory fashion. Recall that after the first four steps of our algorithm, we have a

[4]Note that our algorithm never conditions on negative events.

partial schedule of tasks belonging to the bottom jobs and the special jobs. For convenience, define $\widehat{A} := A(J^*_{\text{bot}}) \cup A(I^*, J_{\text{special}}, x) \setminus A_{\text{bottom-discarded}}$. We want to extend the schedule $\mathcal{S}$ to include $A(J^*_{\text{top}})$. We achieve this in two stages.

- In the first stage, we build a *tentative assignment* of tasks in $A(J^*_{\text{top}})$ in the slots left by $\mathcal{S}$. During this step, we pretend that each task in $A(J^*_{\text{top}})$ is an independent entity with no precedence constraints to any other task in the set $A(J^*_{\text{top}})$. Furthermore, we *do not* enforce the non-migratory constraints as well. However, this step guarantees that the capacity constraints and the precedence constraints between tasks in $A(J^*_{\text{top}})$ and $\widehat{A}$ are satisfied. The precedence constraints are satisfied using the following strategy. For every top job j we define an interval $[r_j, d_j]$, such that if j is scheduled within this interval, then it satisfies all the precedence constraints to jobs in the bottom intervals. The interval $[r_j, d_j]$ is defined assuming the worst possible schedule of the bottom jobs. Hence, it is possible that $[r_j, d_j]$ may be shorter than the interval defined by the support of LP solution for j. This implies that that some of the slots used to schedule top jobs in the LP solution may not be available for our algorithm. To argue that this does not create serious concerns, we make use of the fact that our algorithm completely discarded the jobs in J^*_{mid} (middle jobs). Because of this, the length of a top interval is signficantly longer than a bottom interval. This helps us to argue that for most jobs the interval $[r_j, d_j]$ in which j needs to be scheduled is not much shorter than the interval in which LP schedules job j. In fact, an easy argument from [25] shows that most of the jobs in J^*_{top} can still be scheduled in the intervals $[r_j, d_j]$, and the number of discarded tasks is small.
- In the second stage, we convert the tentative schedule into a feasible schedule. Here we run into a bigger technical hurdle: How do we schedule the top jobs such that a) Every job is scheduled in the interval $[r_j, d_j]$ on a single machine; b) The precedence constraints among the top jobs are satisfied. To accomplish this, we design a new algorithm, which considers the top jobs in Earliest Deadline First (EDF) order of their d_j values assigned in the first stage. Then, the job is assigned to the machine on which it will have the Earliest Completion Time (ECT). During this process we give our algorithm the flexibility of scheduling a job partially and discard the tasks which it cannot schedule feasibly. We give a careful argument to show that EDF and ECT policies together will guarantee that only a small number of tasks are discarded during this process. Moreover, our partial schedule also satisfies the constraints (a) and (b).

Our algorithms for the above two steps guarantee that the final partial schedule obtained is indeed a valid partial schedule respecting all the precedence constraints. Let $A_{\text{top-discarded}}$ denote the total number of tasks discarded from the set $A(J^*_{\text{top}})$ in the above stages. We show the following lemma.

Lemma 4.6. *The valid schedule $\mathcal{S} : \widehat{A} \to I^*$ can be extended to a valid schedule*

$$\mathcal{S}^* : \left(\widehat{A} \cup A(J^*_{\text{top}})\right) \setminus A_{\text{top-discarded}} \to [m] \times I^*$$

such that $|A_{\text{top-discarded}}| \leq \frac{\epsilon}{4} \cdot \frac{T^*}{\log T}$.

4.4 Proof of the Main Lemma (Lemma 4.4)

With all the components, we can finish the proof of the main lemma.

Proof. We first check that the instance given to a recursive call of PARTIAL-SCHEDULE satisfies the properties of Definition 4.3. By Lemma 4.5, the total number of jobs we add to J^*_{special} in the loop on line 3 is at most K, since we add one job to J^*_{special} in each iteration of the loop. Initially, we have $|J^*_{\text{special}}| \leq \log \frac{T}{|I^*|} \cdot K$. Thus, after line 8 in PARTIAL-SCHEDULE, we have that

$$|J^*_{\text{special}}| \leq \log \frac{T}{|I^*|} \cdot K + K \leq \log \frac{T}{|I^*|/2^{\ell^*}} \cdot K.$$

Hence, the number of special jobs given as input is small. Now we consider the number r' of levels the LP-hierarchy solution x has after line 8. Initially, the number of levels x had was $r = \log |I^*|(\log T)K'$. The total number of conditioning operations performed on line 2 and in the loop on line 3 is at most

$$\log \frac{T}{|I^*|} \cdot K \cdot 2^{k^2} + K \cdot 2^{k^2} = \left(\log \frac{T}{|I^*|} + 1\right) K' \leq (\log T)K'.$$

Thus, the number r' of levels x has after line 8 is at least $\log |I^*|(\log T)K' - (\log T)K' \geq \log \frac{|I^*|}{2^{\ell^*}} \cdot (\log T)K'$.

To sum up, since the interval I in each sub-instance has length $\frac{|I^*|}{2^{\ell^*}}$, we have that $|J^*_{\text{special}}|$ is small. Further, the number of levels x has is sufficiently large for each sub-instance. All the other properties in Definition 4.3 follow directly from the description of PARTIAL-SCHEDULE.

Now we prove that the algorithm returns a schedule satisfying the properties stated in the lemma. The schedule $\mathcal{S}^*$ guarantees that every job $j \in J^*$ is assigned to a single machine. On the other hand, our splitting operation guarantees that for every job $j \in J^*_{\text{special}}$, the set $A(I, j, x)$ of tasks that have complete support in I is scheduled on machine $\sigma(j)$. This follows from the fact that all the tasks of special jobs are scheduled by conditioning at the lowest level of recursion.

Thus to prove the lemma, it remains to bound the number of tasks discarded by our algorithm. This calculation is same as in [25] and we do it for the sake of completeness. In the following, the values of J^* and J^*_{special} are considered at the time after line 8.

Our recursive algorithm discards tasks in the following three steps.

- The entire set of middle jobs is discarded. From the discussion in Step 3 (Equation 7) of our algorithm we know that

$$|A(J^*_{\text{mid}})| \leq \frac{\epsilon}{4} \cdot \frac{T^*}{\log T} + \frac{\epsilon}{2m} \cdot \left(|A(J^*_{\text{mid}})| + |A(J^*_{\text{top}})|\right). \tag{8}$$

- For each interval $I \in \mathcal{I}^*_{\ell^*}$, our algorithm recursively schedules tasks in $A(J^*(I,x)) \cup A(I, J^*_{\text{special}}, x)$ in the interval I. By recursive application of Lemma 4.4 we conclude that

$$A_{I,\text{discarded}} \leq \frac{\epsilon}{2} \cdot \frac{\log(\frac{T^*}{2^{\ell^*}})}{\log T} \cdot \frac{T^*}{2^{\ell^*}} + \frac{\epsilon}{2m} \cdot |A(J^*(I,x)) \cup A(I, J^*_{\text{special}}, x))|,$$

where we used the fact that $|I| = \frac{T^*}{2^{\ell^*}}$. Therefore,

$$\begin{aligned} |A_{\text{bottom-discarded}}| &= \sum_{I \in \mathcal{I}^*_{\ell^*}} |A_{I,\text{discarded}}| \\ &\leq \sum_{I \in \mathcal{I}^*_{\ell^*}} \left(\frac{\epsilon}{2} \cdot \frac{\log(\frac{T^*}{2^{\ell^*}})}{\log T} \cdot \frac{T^*}{2^{\ell^*}} + \frac{\epsilon}{2m} \cdot |A(J^*(I,x)) \cup A(I, J^*_{\text{special}}, x))|\right) \\ &= \frac{\epsilon}{2} \cdot \frac{\log T^* - \ell^*}{\log T} \cdot T^* + \frac{\epsilon}{2m} \cdot \left(|A(J^*_{\text{bot}})| + |A(I^*, J^*_{\text{special}}, x)|\right). \end{aligned} \tag{9}$$

We used the fact that for every $a \in A(I^*, J^*_{\text{special}}, x)$, a is completely scheduled inside some $I \in \mathcal{I}^*_{\ell^*}$.

- We discard some more tasks from the set $A(J^*_{\text{top}})$ while scheduling top jobs. By Lemma 4.6,

$$|A_{\text{top-discarded}}| \leq \frac{\epsilon}{4} \cdot \frac{T^*}{\log T}. \tag{10}$$

By combining (8), (9) and (10), we get

$$\begin{aligned} |A_{\text{discarded}}| &= |A(J^*_{\text{mid}})| + |A_{\text{bottom-discarded}}| + |A_{\text{top-discarded}}| \\ &\leq \frac{\epsilon}{4} \cdot \frac{T^*}{\log T} + \frac{\epsilon}{2} \cdot \frac{\log T^* - \ell^*}{\log T} \cdot T^* + \frac{\epsilon}{4} \cdot \frac{T^*}{\log T} \\ &+ \frac{\epsilon}{2m} \left(|A(J^*_{\text{mid}})| + |A(J^*_{\text{top}})| + |A(J^*_{\text{bot}})| + |A(I^*, J^*_{\text{special}}, x)| \right) \\ &\leq \frac{\epsilon}{2} \cdot \frac{\log T^*}{\log T} \cdot T^* + \frac{\epsilon}{2m} \cdot |A(J^*) \cup A(I^*, J^*_{\text{special}}, x)|, \end{aligned}$$

where we used $\ell^* \geq 1$ and the fact that the sets $J^*_{\text{top}}, J^*_{\text{mid}}, J^*_{\text{bot}}$ define a partition of J^* and $J^* \cap J^*_{\text{special}} = \emptyset$. We want to bound the number of discarded jobs using the original J^* and J^*_{special}, i.e, the sets specified in the input. However, it is fairly straightforward to see that the set $A(J^*) \cup A(I^*, J^*_{\text{special}}, x)$ does not change when moving jobs from J^* to J^*_{special}. The lemma follows by the definition of T^* and I^*. □

Organization of the rest of Section 4: The rest of this section is devoted to proving the lemmas mentioned in Steps 1 and 5. In Subsection 4.5 we prove Lemma 4.5. In Subsection 4.6 we prove Lemma 4.6.

4.5 Reducing the Chain Length and Bounding Number of Conditionings

In this section, we give more details about the first two steps of our algorithm. We first show that the total number of conditionings required to reduce chain length among top jobs is small, which in turn implies small number of conditionings during splitting operations. We begin by proving Lemma 4.5 from Step 1, which we restate below for convenience.

Lemma 4.5. *Let $\widehat{J} := \bigcup_{\ell=0}^{k^2-1} J^*_\ell(x)$ denote the set of jobs owned by the intervals in $\mathcal{I}^*_0, \mathcal{I}^*_1, \ldots, \mathcal{I}^*_{k^2-1}$ according to the LP hierarchy solution x. Then, after line 8 in* PARTIAL-SCHEDULE*, we have that $\Delta(\widehat{J}) \leq k^2 \delta T^*$. Moreover, the number of iterations we run the loop on line 3 is at most K.*

Proof. After line 8 in PARTIAL-SCHEDULE, we have that for every $\ell \in \{0, 1, \ldots, k^2 - 1\}$ and every interval $I \in \mathcal{I}^*_\ell$, the maximum chain length of jobs owned by I is at most $\delta|I| = \delta \frac{T^*}{2^\ell}$. Thus, the maximum chain length of jobs owned by level ℓ is at most δT^*, since there are exactly 2^ℓ intervals. There are at most k^2 levels, which implies that the maximum chain length of $\widehat{J}$, i.e, jobs owned by levels $0, 1, \ldots, k^2 - 1$, is at most $k^2 \delta T^*$.

Thus it remains to bound the number of iterations of the loop on line 3. We focus on some interval $I \in \mathcal{I}^*_\ell$ for some $\ell \in \{0, 1, \ldots, k^2 - 1\}$. Consider the chain $\mathcal{C}$ of jobs owned by the interval I of length (total size) at least $\delta|I|$ and the job $j \in \mathcal{C}$ that we condition on in an iteration of the loop. Let I_1 and I_2 be the left and right children of I in the laminar tree; so $I_1, I_2 \in \mathcal{I}^*_{\ell+1}$. Notice that j will be removed from J^* and thus $\widehat{J}$ during the iteration. On the other hand, all the other jobs in $\mathcal{C}$ will be owned by some sub-interval of I_2 after the conditioning. This is true for the following reason. Since j was owned by the interval I, it means that for the last task

a of j and some time $t \in I_2$ we have $x_{(a,i,t)} > 0$. Furthermore, j was the first job in the chain $\mathcal{C}$. Therefore the entire support of the jobs in $\mathcal{C} \setminus \{j\}$ will lie inside I_2 after the conditioning on (a, i, t). Thus, every $j' \in \mathcal{C} \setminus \{j\}$ is now owned by some level $\ell' \geq \ell + 1$.

Appealing to the fact that support can only shrink upon conditioning, a single job (thus a task) can move down at most k^2 times before it is removed from $\widehat{J}$. Due to the capacity constraints, there are at most mT^* tasks owned by all intervals $I \subseteq I^*$ in total, so there will be at most mT^*k^2 events that a task moves down. On the other hand, in one iteration of the loop on line 3, each task in $\mathcal{C}$, of which there are at least $\delta|I|$ many, is either removed from the set $A(\widehat{J})$ or moved down by at least one level. Note that $\delta|I| \geq \frac{\delta T^*}{2^{k^2}}$. Together we get that the number of iterations of the loop is at most

$$\frac{mT^*k^2}{\delta \frac{T^*}{2^{k^2}}} = \frac{mk^2 \cdot 2^{k^2}}{\delta} = K. \qquad \square$$

4.6 Scheduling the Top Jobs

In this section we give an algorithm to schedule the top jobs J^*_{top}. Recall that after the first four steps of our algorithm, we have a partial schedule of tasks belonging to the bottom jobs and the special jobs. Formally,

$$\mathcal{S} : (A(J^*_{\text{bot}}) \cup A(I^*, J_{\text{special}}, x)) \setminus A_{\text{bottom-discarded}} \to [m] \times I^* .$$

Recall that for convenience we have defined $\widehat{A} := (A(J^*_{\text{bot}}) \cup A(I^*, J_{\text{special}}, x)) \setminus A_{\text{bottom-discarded}}$. We want to extend $\mathcal{S}$ to a schedule $\mathcal{S}^*$ that includes most tasks of the set $A(J^*_{\text{top}})$.

As described in Step 5 of the algorithm, we accomplish this in two stages. In the first stage, we build a *tentative assignment* of tasks in $A(J^*_{\text{top}})$ in the slots left by $\mathcal{S}$. At this stage, we *do not* satisfy the non-migratory constraints but we guarantee that the capacity constraints and the precedence constraints between tasks in $A(J^*_{\text{top}})$ and $\widehat{A}$ are satisfied. During this step we discard some tasks from the set $A(J^*_{\text{top}})$, which we denote by $A^1_{\text{top-discarded}}$. In the second stage, we convert the tentative schedule into a valid partial schedule respecting all the precedence constraints. This step also involves discarding some tasks from the set $A(J^*_{\text{top}})$, which we denote by $A^2_{\text{top-discarded}}$.

Define $A_{\text{top-discarded}} := A^1_{\text{top-discarded}} + A^2_{\text{top-discarded}}$. The goal of this section is to prove Lemma 4.6, which we re-state for convenience.

Lemma 4.6. *The valid schedule $\mathcal{S} : \widehat{A} \to I^*$ can be extended to a valid schedule*

$$\mathcal{S}^* : \left(\widehat{A} \cup A(J^*_{\text{top}})\right) \setminus A_{\text{top-discarded}} \to [m] \times I^*$$

such that $|A_{\text{top-discarded}}| \leq \frac{\epsilon}{4} \cdot \frac{T^*}{\log T}$.

4.6.1 Tentative Schedule

As we allow migration of jobs in this step, it gives us the flexibility to treat each task in $A(J^*_{\text{top}})$ almost independently. This lets us use the algorithm in [25] as a black box to build the tentative schedule. We briefly sketch proofs of the claims made in this subsection, and refer the reader to Section 5 in [25] for full details.

Recall that T^* denotes the length of interval I^*. For convenience we re-index the time slots in I^* so that $I^* := \{1, 2, \ldots, T^*\}$. Let $\mathcal{I}^*$ denote the binary laminar family of intervals, and let $I_1, I_2, \ldots, I_p$ be the set of bottom intervals in $\mathcal{I}^*_{\ell^*}$. From our construction $p = 2^{\ell^*}$. For each interval $I_v, v \in \{1, 2, \ldots, p\}$, let $\text{begin}(I_v)$ and $\text{end}(I_v)$ denote the first and the last time slots that belong to I_v.

Our recursive algorithm schedules most of the tasks in $A(J^*_{\text{bot}}) \cup A(I, J^*_{\text{special}}, x)$ in the intervals $I_1, I_2, \ldots, I_p$. Recall that our algorithm recurses for each interval $I \in \mathcal{I}^*_{\ell^*}$ on the subset of jobs $J^*(I, x)$ and J^*_{special}. By definition, every job $j \in J^*(I, x)$ has its entire support in the interval I. Furthermore, the tasks of jobs in the set J^*_{special} that are scheduled in I are precisely those tasks which have entire support in I. For each $I \in \mathcal{I}^*_{\ell^*}$, our algorithm schedules a subset of tasks in $A(J^*(I, x)) \cup A(I, J^*_{\text{special}}, x)$. This implies that $|A(J^*_{\text{top}})| \leq m|I^*| - |\widehat{A}|$. In other words, there are enough slots to schedule tasks in $A(J^*_{\text{top}})$ in the slots left open by $\mathcal{S}$.

In fact, we can say something stronger. Let $[r_j, d_j]$ denote the *smallest* interval in which the entire support of any job $j \in J^*_{\text{top}}$ lies in the solution x. That is, $\sum_i \sum_{t \in [r_j, d_j]} x_{(i,a,t)} = 1$ for every task $a \in A(j)$. Let u_j be the largest index such that $\text{begin}(I_{u_j}) \leq r_j$. Similarly let v_j be the smallest index such that $\text{end}(I_{v_j}) \geq d_j$. In other words, the interval $[\text{begin}(I_{u_j}), \text{end}(I_{v_j})]$ is the smallest interval that completely contains $[r_j, d_j]$ and whose beginning and ending coincide with a beginning and ending of intervals in $I_1, I_2, \ldots, I_p$. Let $[r'_j, d'_j] := [\text{begin}(I_{u_j}), \text{end}(I_{v_j})]$. The current LP solution x also guarantees that there is enough space to *fractionally* schedule tasks in $A(J^*_{\text{top}})$ so that for every job $j \in J^*_{\text{top}}$, all the tasks in $A(j)$ are scheduled in the interval $[r'_j, d'_j]$. This can be proved by setting up a fractional bipartite matching between the jobs and the empty slots in the intervals $I_1, I_2, \ldots, I_p$. The existence of a fractional matching in a bipartite graph implies that there exists an integral matching. Hence, there is an assignment of tasks of $A(J^*_{\text{top}})$ such that every job j is scheduled in the interval $[r'_j, d'_j]$.

Unfortunately, the schedule obtained using the above argument can violate the precedence constraints between tasks in $A(J^*_{\text{top}})$ and $\widehat{A}$. The main reason is that the schedule of tasks in $\widehat{A}$ had been constructed without taking $A(J^*_{\text{top}})$ into account. We will define a *release time* r^*_j and a *deadline* d^*_j for all jobs $j \in J^*_{\text{top}}$ in such a way that if all the tasks of j are scheduled in the interval $[r^*_j, d^*_j]$, then the precedence constraints between $A(J^*_{\text{top}})$ and $\widehat{A}$ will be satisfied. A natural way to define the interval $[r^*_j, d^*_j]$ is as follows:

$$r^*_j := \text{begin}(I_{u_j+1}) \quad \text{and} \quad d^*_j := \text{end}(I_{v_j-1}). \tag{11}$$

We claim that if our algorithm schedules all tasks of the job j in the interval $[r^*_j, d^*_j]$, then the precedence constraints between tasks in $A(J^*_{\text{top}})$ and $\widehat{A}$ will be satisfied. For this, note that every task in $\widehat{A}$ is supported on a single interval in $\mathcal{I}^*_{\ell^*}$ in x; for tasks of bottom jobs this follows from the definition, and for tasks of special jobs it is a consequence of our splitting procedure (indeed every task of a special job is supported on a single interval in $\mathcal{I}^*_{k^2}$, and $\ell^* \leq k^2$). Consider a task $a \in \widehat{A}$ and a job $j \in J^*_{\text{top}}$ with $a \prec j$. Then a must be supported on an interval I_u with $u \leq u_j$. (Suppose otherwise that $u > u_j$, and let a' be the first task of j; then a' is scheduled with positive x-fraction on the interval I_{u_j} by the definition of r_j, yet the entire support of a in x lies to the right of I_{u_j}, contrary to $a \prec a'$ and the constraints (4) of the LP.) Thus, as long as j is scheduled after $r^*_j = \text{begin}(I_{u_j+1})$, the constraint $a \prec j$ will be satisfied. The argument for precedence constraints of the form $j \prec a$ is symmetric.

Now observe that $[r^*_j, d^*_j] \subsetneq [r'_j, d'_j]$: the interval in which we are allowed to schedule the job j is shorter than the interval in which the LP solution schedules the tasks. Therefore, it is no longer clear that we will have enough slots to schedule all tasks in $A(J^*_{\text{top}})$ and our algorithm may have to discard some tasks. An argument in [25] shows that the number of tasks discarded is not large.

Lemma 4.7 ([25])**.** *A feasible partial schedule $\mathcal{S} : \widehat{A} \to [m] \times I^*$ can be extended to a new schedule $\mathcal{S}' : \left(\widehat{A} \cup A(J^*_{\text{top}})\right) \setminus A^1_{\text{top-discarded}} \to [m] \times I^*$ satisfying the following properties:*

1. *Consider $j \in J^*_{\text{top}}$ and let $a \in A(j)$. Then in the schedule $\mathcal{S}'$, either a is assigned in the interval $[r^*_j, d^*_j]$ or $a \in A^1_{\text{top-discarded}}$.*
2. *The total number of discarded tasks $|A^1_{\text{top-discarded}}| \leq 4m2^{-k}T^*$.*
3. *The precedence constraints between tasks in $A(J^*_{\text{top}}) \setminus A^1_{\text{top-discarded}}$ and $\widehat{A}$ are respected.*
4. *The capacity constraints are satisfied.*

The proof of this lemma in [25] crucially relies on the following observation. Notice that although the interval in which we can schedule the job j shrinks, the loss is small. More precisely, the LP solution x would have scheduled some portion of the job in the intervals I_{u_j} and I_{v_j}, and now we will not be able to use them. However, the intervals in $\mathcal{I}^*_{\ell^*}$ are much shorter than the intervals that own the top jobs. This is because we discarded a set of k consecutive middle intervals. Hence, each interval $I \in \mathcal{I}^*_{\ell^*}$ is at least 2^{-k} times shorter than an interval $I' \in \mathcal{I}_0 \cup \cdots \cup \mathcal{I}_{\ell^*-k-1}$. This fact can be used to show that we only need to throw away few tasks to account for the loss of flexibility in scheduling top jobs.

4.6.2 From Tentative Schedule to Final Schedule

In this final step of our algorithm, we convert the schedule $\mathcal{S}' : \left(\widehat{A} \cup A(J^*_{\text{top}})\right) \setminus A^1_{\text{top-discarded}} \to [m] \times I^*$ into a non-migratory schedule that satisfies all the precedence constraints among top jobs. The main goal of this subsection is to prove Lemma 4.6. Our final schedule $\mathcal{S}^*$ satisfying the requirements of the lemma guarantees the following two invariants:

- For every job $j \in J^*_{\text{top}}$, all the non-discarded tasks are scheduled within $[r^*_j, d^*_j]$.
- The assignment of tasks in $\widehat{A}$ remains the same as that in the schedule $\mathcal{S}'$.

The above two invariants together will imply that all the precedence constraints are satisfied among non-discarded tasks in $\mathcal{S}^*$. We build schedule $\mathcal{S}^*$ by first designing an algorithm for a new stand-alone scheduling problem, then using it as a black box for scheduling top jobs. We first define the new scheduling problem.

A Deadline Scheduling Problem with Precedence Constraints: Consider a set of jobs J with processing lengths p_j, release times r_j, deadlines d_j and precedence constraints. As before, we assume that each job $j \in J$ is made up of p_j unit-length tasks. The precedence constraints among jobs satisfy the property that if $j \prec j'$, then $r_j \leq r_{j'}$ and $d_j \leq d_{j'}$. Recall that we use $\Delta(J)$ to denote the maximum chain length in J. The time horizon $[T]$ is partitioned into p equal sized intervals $I_1, I_2, \ldots, I_p$. The release times and deadlines of jobs fall at beginnings and ends of the intervals.

For each machine i, we are given a capacity function $\mathsf{cap}_i : [T] \to \{0,1\}$. If $\mathsf{cap}_i(t) = 1$, then the time slot t on machine i is available to schedule a task in $A(J)$. For an interval $I = \{t', \ldots, t''\}$, we overload the notation to refer $\mathsf{cap}_i(I)$ as the number of available slots in the interval I; that is, $\mathsf{cap}_i(I) = \sum_{t=t'}^{t''} \mathsf{cap}_i(t)$. Similarly, we use $\mathsf{cap} : [T] \to [m]$ to denote the number of available slots at time t across all the m machines; $\mathsf{cap}(t) = \sum_{i=1}^{m} \mathsf{cap}_i(t)$.

Suppose there is a schedule of tasks $\mathcal{S}' : A(J) \to [m] \times [T]$ that assigns each task in $A(J)$ to a machine-timeslot pair such that no two tasks are assigned to the same machine and the same time slot; that is, the capacity constraints on machines are satisfied. Moreover, $\mathcal{S}'$ ensures that for each job $j \in J$, all the tasks in $A(j)$ are scheduled within $[r_j, d_j]$. We remark that $\mathcal{S}'$ may not respect the precedence constraints in J and the no-migration constraints. Our goal is to schedule each $j \in J$ on a single i so that precedence constraints among the jobs are satisfied and enforce the no-migration constraints.

We prove the following theorem in this section.

Algorithm 3 EDF+ECT

Input: A set of jobs J with release times, deadlines and precedence constraints; capacity function $\mathsf{cap}_i : [T] \to \{0,1\}$ for each machine i.

Output: Schedule $\mathcal{S} : A(J) \to [m] \times [T]$ such that for $a \sim a'$, either a or a' belongs to $A_{\text{discarded}}$ or $\mathcal{S}_{\text{mac}}(a) = \mathcal{S}_{\text{mac}}(a')$.

```
1: Sort the jobs in $J$ in the increasing order of their deadlines. Reindex the jobs so that
   $J := \{1, 2, \dots, n\}$ and $d_1 \le d_2 \le \dots \le d_n$.
2: Initialize $A_{\text{discarded}} = \emptyset$.
3: for $j = 1$ to $n$ do
4:     Find the earliest time slot $t \in [T]$ such that following conditions hold: i) $\mathsf{cap}_i(t) = 1$ for
       some machine $i \in [m]$ and $r_j \le t \le d_j$; ii) $C_{j'} < t$ for all $j' \prec j$.
5:     If no such $t$ exists, then set $B_j :=$ DISCARDED and add all the tasks $A(j)$ to the set
       $A_{\text{discarded}}$.
6:     If there is a $t$ satisfying the conditions above, set $B_j = t$ and do the following.
7:     Find the set of machines $M' = \{i \in [m] : \mathsf{cap}_i(B_j, d_j) \ge p_j\}$.
8:     if $|M'| = 0$ then
9:         $i^* = \text{argmax}_i\{\mathsf{cap}_i(B_j, d_j)\}$.
10:        Schedule $\mathsf{cap}_{i^*}(B_j, d_j)$ tasks of the job $j$ in the interval $[B_j, d_j]$ on the machine $i^*$.
11:        Set $C_j = d_j$. Add the remaining $p_j - \mathsf{cap}_{i^*}(B_j, d_j)$ tasks to the set $A_{\text{discarded}}$.
12:        Update the capacity function $\mathsf{cap}_{i^*}$ for the machine $i^*$ and the schedule $\mathcal{S}$.
13:    if $|M'| \ge 1$ then
14:        Find the earliest time slot $t^*$ such that there exists a machine $i^* \in M'$ and
           $\mathsf{cap}_i(B_j, t^*) = p_j$.
15:        Set $C_j = t^*$. Schedule all the tasks $A(j)$ of $j$ in the interval $[B_j, C_j]$ on the machine
           $i^*$.
16:        Update the capacity function $\mathsf{cap}_{i^*}$ for the machine $i^*$ and the schedule $\mathcal{S}$.
17: return $\mathcal{S}$.
```

Theorem 4.8. *There exists an algorithm that in polynomial time converts the schedule $\mathcal{S}'$ into a valid schedule that satisfies the following properties:*

1. *It partially schedules every job on exactly one machine (no-migration). For a job that is partially scheduled, we discard the remaining tasks; for the sake of the precedence constraints, we assume that every partially scheduled job or a fully discarded job is completely processed.*
2. *The precedence constraints among the jobs are satisfied.*
3. *The total number of discarded tasks is at most $2p^2 m\Delta(J)$.*

We prove the above theorem using the following scheduling algorithm, which considers jobs in the Earliest Deadline First (EDF) order and assigns jobs to machines on which they will have *the earliest completion time* (ECT). Our algorithm is given in Algorithm 3, which we call EDF+ECT. See Figure 1 for an illustration.

For a job j, we call the interval $[B_j, C_j]$ the *active* interval. B_j denotes the first time instant when the job is *ready* to be scheduled *and there is an available slot* on some machine i. Note that it can be the case that no task of job j is scheduled at the time slot B_j, because we choose the machine on which the job will have the earliest completion time. If for a job j, $B_j =$ DISCARDED then we say that the job is *fully discarded*; otherwise, if not all of its tasks are scheduled, we say that it is *partially discarded.* For a fully discarded job, all its tasks belong

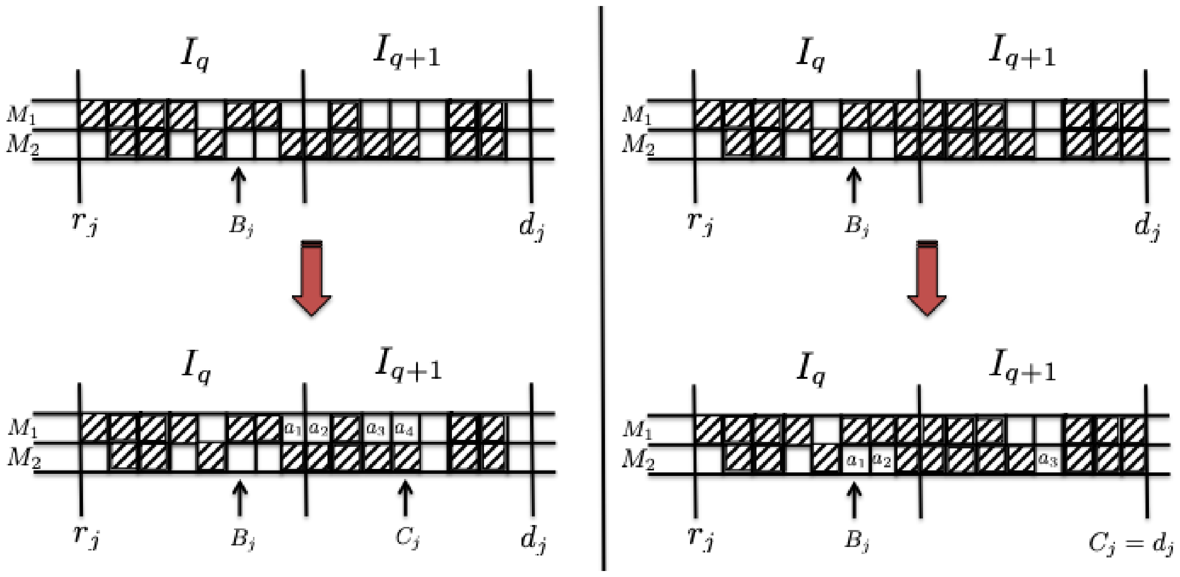

Figure 1: Figure illustrates scheduling a job j with $p_j = 4$ in the non-migratory fashion when $m = 2$. On the left, j is fully scheduled in the interval $[B_j, C_j]$ on M_1. On the right, the first three tasks are scheduled on machine M_2, and the last task a_4 is discarded.

to the set $A_{\text{discarded}}$. We say that a job j is either *partially or fully discarded in the interval* $I \in \{I_1, I_2, \dots, I_p\}$ if $d_j \in I$ and j is either partially or fully discarded.

We call a time slot t on machine i *idle* if $\mathsf{cap}_i(t) = 1$ and our schedule $\mathcal{S}$ does not assign any task at time t on machine i. Roughly speaking, the idle time slots correspond to the number of tasks we discard, and our goal going forward is to show that there are not too many idle time slots in $\mathcal{S}$. We make the following observations that will help in proving Theorem 4.8.

Observation 4.9. *Fix an interval I_q for some $q \in [p]$. Suppose the following two conditions hold:*

- *There is a time slot $t^* \in I_q$ that is idle on machine i in $\mathcal{S}$.*
- *There exists a job j^* with $t^* \in [r_{j^*}, d_{j^*}]$ and either $B_{j^*} > t^*$ or $B_{j^*} =$ DISCARDED.*

Then there exists a job j such that $t^* \in [B_j, C_j]$ and $j \prec j^*$. Moreover, j is not scheduled on machine i.

Proof. If no such job j exists, then, when our algorithm considers the job j^* for scheduling, it would set $B_{j^*} = t^*$. This follows from line 4 of our algorithm. Moreover, if j were scheduled on i, then it would have been scheduled on every time slot between B_j and C_j that is idle on i, which contradicts the facts that $t^* \in [B_j, C_j]$ and that t^* is idle on i. □

Observation 4.10. *Consider a job $j \in J$ with $B_j \neq$ DISCARDED that is active in the interval $[B_j, C_j]$ on machine i. Let p'_j be the total number of tasks of j scheduled in $[B_j, C_j]$. Then for any other machine $i' \neq i$, the number of idle time slots in the interval $[B_j, C_j]$ is at most p'_j.*

Proof. Consider the case when $p'_j = p_j$. In this case, the lemma follows since our algorithm assigns jobs to machines on which they will have the earliest completion time. Now consider the case when $p'_j \neq p_j$. As job j was scheduled on machine i, from line 9, it follows that i had the maximum number of empty slots in the interval $[B_j, d_j]$, which is equal to p'_j. Since in this case we set $C_j = d_j$, no machine $i' \neq i$ had more than p'_j empty slots. □

The above two observations will help us argue that the number of idle time slots on any machine is small.

Lemma 4.11. *Consider any arbitrary time interval $I := \{t', \dots, t''\} \subseteq I_q$ for some $q \in [p]$. Suppose there is at least one job j^* with $I \subseteq [r_{j^*}, d_{j^*}]$ and either $B_{j^*} > t''$ or $B_j = \text{DISCARDED}$. Then, for any machine $i \in [m]$ the number of idle time slots in I is at most $\Delta(J)$.*

Proof. Consider a machine i; we will show that it has at most $\Delta(J)$ idle time slots in the interval I. Let $t^* \in I$ be the last such time slot. Since j^* is available at t^*, by Observation 4.9 there must exist a job $j_1 \prec j^*$ that is active at time t^*, i.e. $t^* \in [B_{j_1}, C_{j_1}]$. Now consider the last time slot $t_1 < B_{j_1}$ that is empty on machine i. We claim that j_1 is available for processing at time t_1. This follows from our assumptions that if $j_1 \prec j^*$, then $r_{j_1} \le r_{j^*}$ and the release times and the deadlines of J align with the beginnings and the endings of the intervals. Therefore, there must be a job $j_2 \prec j_1$ such that $t_1 \in [B_{j_2}, C_{j_2}]$. Moreover, $C_{j_2} < B_{j_1}$ as $j_2 \prec j_1$. We continue by induction to construct a chain of jobs $j_y \prec j_{y-1} \prec \dots \prec j_1$ such that $[B_{j_y}, C_{j_y}] \cup [B_{j_{y-1}}, C_{j_{y-1}}] \cup \dots \cup [B_{j_1}, C_{j_1}]$ covers all the empty slots on machine i in the interval I. These intervals are pairwise disjoint. The total size of the jobs in the chain $j_y \prec j_{y-1} \prec \dots \prec j_1$, $\sum_{v=1}^{y} p_{j_v}$, is at most $\Delta(J)$, since the maximum chain length among J is at most $\Delta(J)$. By Observation 4.9, jobs $j_1, \dots, j_y$ are not scheduled on i. Thus, by Observation 4.10, for any job j_v belonging to the chain, there can be at most p_{j_v} empty slots on the machine i in the interval $[B_{j_v}, C_{j_v}]$. Therefore, the number of all time slots idle on i in the union of these intervals is at most $\sum_{v=1}^{y} p_{j_v} \le \Delta(J)$. We conclude by recalling that $[B_{j_y}, C_{j_y}] \cup [B_{j_{y-1}}, C_{j_{y-1}}] \cup \dots \cup [B_{j_1}, C_{j_1}]$ covers all the idle slots on machine i. □

The above lemma implies the following useful corollary.

Corollary 4.12. *Suppose there is an interval I_q and a machine i with more than $\Delta(J)$ idle time slots. If there is a job j^* such that $B_{j^*} \in I_{q+1} \cup I_{q+2} \cup \dots \cup I_p$, then its release time $r_{j^*} \in I_{q+1} \cup I_{q+2} \cup \dots \cup I_p$.*

Proof. For contradiction, let us assume that the release time of j^* belongs to $I_{q'}$, where $q' < q+1$. Recall that all jobs are released at the beginning of the intervals. Now, we invoke the previous lemma on the interval I_q and the job j^*, which gives us the contradiction. □

The next lemma shows that even if a job is partially discarded, the number of idle time slots on a machine cannot be too large. This lemma accounts for the number of idle slots due to the no-migration constraints, that is, due to the fragmentation of jobs.

Lemma 4.13. *Consider any arbitrary time interval $I := \{t', \dots, t''\} \subseteq I_q$ for some $q \in [p]$. Suppose there is at least one job j^* with $I \subseteq [r_{j^*}, d_{j^*}]$ and $A(j^*) \cap A_{\text{discarded}} \ne \emptyset$. Then, for any machine i the number of idle time slots in I is at most $2\Delta(J)$.*

Proof. If $B_{j^*} > t''$, then the statement follows from Lemma 4.11. Therefore, $B_{j^*} \le t''$ and some tasks of j^* got discarded. By Observation 4.10, in the interval $[B_{j^*}, C_{j^*}]$ there cannot be more than $p_{j^*} \le \Delta(J)$ idle slots on any machine i where j^* was not scheduled (and there cannot be any such slots on the machine where j^* was scheduled since it was partially discarded). *A fortiori*, there are at most $\Delta(J)$ such slots in the interval $[B_{j^*}, t''] \subseteq [B_{j^*}, d_{j^*}] = [B_{j^*}, C_{j^*}]$ (we have $d_{j^*} = C_{j^*}$ since j^* was partially discarded). If $B_{j^*} \le t'$, then we are done since $I \subseteq [B_{j^*}, t'']$. Otherwise, i.e. if $B_{j^*} > t'$, we apply Lemma 4.11 to the interval $[t', B_{j^*} - 1]$ and the job j^*, obtaining that in the interval $[t', B_{j^*} - 1]$ there can be at most $\Delta(J)$ idle time slots. We conclude by combining the two intervals $[t', B_{j^*} - 1]$ and $[B_{j^*}, t'']$. □

With all the above lemmas, it is easy to prove Theorem 4.8. For brevity, we use $\mathcal{S}^{-1}(I)$ to denote the set of tasks scheduled in the interval I in $\mathcal{S}$.

Proof of Theorem 4.8. Our algorithm guarantees that in the schedule $\mathcal{S}$ all the tasks $A(j)$ of a job $j \in J$ are assigned to a single machine and the precedence constraints among the jobs are satisfied. Therefore, it only remains to show that the number of discarded tasks $|A_{\text{discarded}}| \leq 2p^2 m\Delta(J)$. The proof is based a double-counting argument, which was also used in [25]. Let I_h, $h \in [p]$ be any interval, and let α be the total number of tasks discarded in I_h. We argue that $\alpha \leq 2pm\Delta(J)$. As there are p intervals, this will complete the proof.

Let j_s be the lowest-priority (i.e. the highest-index) job that got discarded either completely or partially in the interval I_h (if no such job exists, then we are done). Consider the set of jobs $J' := \{j_1, j_2, \ldots, j_s\}$ and the intervals $I_1, I_2, \ldots, I_h$. Imagine the following thought experiment: Let us reschedule the jobs in the set J' using our algorithm in the intervals $I_1, I_2, \ldots, I_h$. Let $\mathcal{S}^*$ be this new schedule. As our algorithm considers the jobs in EDF order, the schedule $\mathcal{S}^*$ will be identical to $\mathcal{S}$ for the set J'. In particular, this means that the interval I_h will have α discarded tasks in $\mathcal{S}^*$. This is true as we never discard a job until we hit its deadline, and by our choice j_s is the highest-indexed job that got discarded. So all the tasks discarded in the interval I_h belong to the jobs in J' in both schedules $\mathcal{S}$ and $\mathcal{S}^*$.

From now on we will focus on the schedule $\mathcal{S}^*$ and the set of jobs J'. By Lemma 4.13, the interval I_h cannot have more than $2\Delta(J)$ idle time slots on any machine i as there is a job j_s that got discarded in I_h. Let $g \in \{1, 2, \ldots, h\}$ be the minimal index such that no machine i has more than $2\Delta(J)$ empty slots in any of the intervals $I_g, \ldots, I_h$. Let $I' := I_g \cup \ldots \cup I_h$. Consider the following set of jobs:

$$J'' := \left\{ j \in \{j_1, j_2, \ldots, j_s\} : [B_j, C_j] \subseteq I' \text{ or } B_j = \text{DISCARDED} \right\}.$$

For every job $j \in J''$ we claim that $[r_j, d_j] \subseteq I'$. To get this, it is enough to show that $r_j \geq \text{begin}(I_g)$. Then we have $\text{begin}(I_g) \leq r_j \leq d_j \leq d_{j_s} = \text{end}(I_h)$. Our claim is clearly true if $g = 1$. Otherwise we argue as follows. From our choice of the index g, I_{g-1} has more than $\Delta(J)$ (in fact, $2\Delta(J)$) many idle time slots on some machine i. Now consider a job $j \in J''$. The first case is that $B_j \in I'$. Then from Corollary 4.12 (applied to I_{g-1}) we have $r_j \in I'$. The second case is that $B_j = \text{DISCARDED}$. Towards a contradiction suppose that $r_j \leq \text{begin}(I_{g-1})$. By applying Lemma 4.11 to I_{g-1}, we get that I_{g-1} has at most $\Delta(J)$ idle slots on any machine, which contradicts our choice of the index g. Therefore again $r_j \in I'$.

Conversely, any job $j \in J'$ that has a task scheduled in I' must be in J''. Suppose otherwise; then we would have $g > 1$, $B_j < \text{begin}(I_g) \leq C_j$, and $r_j \leq \text{begin}(I_{g-1})$. This, together with the fact that I_{g-1} has more than $2\Delta(J)$ idle slots on some machine, implies that j should have been scheduled fully in $I_1 \cup \cdots \cup I_{g-1}$ (via an argument similar to the proof of Lemma 4.11), which is a contradiction.

Now we can use a simple double-counting trick to bound α. Let $A_{\text{discarded}}(J'') \subseteq A(J'')$ denote the subset of tasks discarded by our schedule $\mathcal{S}^*$. Then,

$$|A(J'')| \geq |(\mathcal{S}^*)^{-1}(I')| + |A_{\text{discarded}}(J'')| \geq \mathsf{cap}(I') - 2pm\Delta(J) + \alpha. \tag{12}$$

The last inequality above follows from the fact that there are at most p intervals in $\{I_g, \ldots, I_h\}$ and, by our definition of g, any interval $I_g, \ldots, I_h$ has at most $2\Delta(J)$ empty slots on any machine. But note that every task in $A(J'')$ needs to be scheduled in I', because for every job $j \in J''$, $[r_j, d_j] \subseteq I'$. By our assumption, there is a (possibly invalid) schedule $\mathcal{S}'$ that assigns every task in $A(J'')$ in the interval I'. Therefore, $A(J'') \leq \mathsf{cap}(I')$. Combining this with (12) we have $\alpha \leq 2pm\Delta(J)$. This completes the proof. $\square$

Now we are ready to prove Lemma 4.6, which bounds the total number of tasks discarded in the process of scheduling top jobs.

Proof of Lemma 4.6. We obtain the schedule $\mathcal{S}^*$ by applying Theorem 4.8 on the jobs in the set J^*_{top}. For every job $j \in J^*_{\text{top}}$, we define the truncated job length p'_j by taking into account the discarded tasks in $A^1_{\text{top-discarded}}$. We define the release time of j as r^*_j and the deadline as d^*_j, where r^*_j, d^*_j are defined in Equation (11). For each machine i, $\mathsf{cap}_i(t) = 1$ if time t on machine i is not assigned any task in the schedule $\mathcal{S}$. Recall that $\mathcal{S}$ gives an assignment of a subset of tasks of the bottom jobs and the special jobs $(\widehat{A})$ in the interval I^*.

By Lemma 4.5, the maximum chain length of jobs in J^*_{top} is at most $k^2\delta T^*$, where T^* is the length of interval I^*. Therefore, by Theorem 4.8, the total number of tasks discarded in converting the tentative schedule into an actual schedule is given by

$$|A^2_{\text{top-discarded}}| \leq 2p^2 m \cdot k^2 \delta T^*.$$

By Lemma 4.7, we have

$$|A^1_{\text{top-discarded}}| \leq 4m2^{-k}T^*.$$

Therefore,

$$\begin{aligned}|A_{\text{top-discarded}}| &= |A^1_{\text{top-discarded}}| + |A^2_{\text{top-discarded}}| \\ &\leq 2p^2 m \cdot k^2\delta T^* + 4m2^{-k}T^* \\ &\leq \frac{\epsilon}{4} \cdot \frac{T^*}{\log T}\end{aligned}$$

The last inequality follows from substituting $p = 2^{\ell^*} \leq 2^{k^2}$, $k = \frac{O(1)m}{\epsilon}\log\log T$, and $\delta = \frac{\epsilon}{8k^2 m 2^{2k^2}\log T}$. $\square$

5 Minimizing Makespan Under Precedence Constraints with Communication Delays

Now we consider the problem of minimizing the makespan when jobs have precedence constraints and communication delay constraints. For the rest of this section, we sometimes refer to the problem we studied in the first part of the paper , $Pm|\text{prec}, \text{pmtn}|C_{\max}$, as "the no-delay problem" or "the first problem" and we refer the problem, $Pm|\text{prec}, \text{pmtn}, c_{j,j'}|C_{\max}$, as "the delay problem". We follow the notation developed for the no-delay problem, and whenever necessary remind the readers their meaning. If we use a notation without definition in this section, it implies that its meaning is same as in the no-delay problem.

Similar to the no-delay problem, the input to the problem consists of a set of jobs J, where each job $j \in J$ has a processing length p_j, the jobs have precedence constraints, and we are given a set of m machines. If there is a precedence constraint $j \prec j'$, then we require that job j' can only start after job j is completed. Furthermore, if j and j' are processed *on two different machines*, then the processing of job j' cannot start earlier than $c_{j,j'} > 0$ time units after the completion time of j. We assume without loss of generality that $c_{j,j'}$ are natural numbers. If, however, j and j' are processed on the same machine, then j' can start right after the completion of job j. We study the case when $\max_{j\prec j'}\{c_{j,j'}\} = O(1)$ case, which is a generalization of the well studied case of when communication delays are all equal to 1. However, to keep the notation simple, we first present our proof assuming $c = 1$. At the end, when it will be clear that our framework is general enough to handle $\max_{j\prec j'}\{c_{j,j'}\} = O(1)$, and we give a sketch of our proof for $\max_{j\prec j'}\{c_{j,j'}\} = O(1)$. The goal is to schedule jobs satisfying the precedence and communication delay constraints so as to minimize makepsan. In the three field notation, the problem is denoted by $Pm|\text{prec}, \text{pmtn}, c = 1|C_{\max}$.

Our goal is to assign each job to a single machine and specify the schedule of tasks such that the precedence constraints and communication delay constraints among jobs are satisfied. Formally, we define a valid schedule as follows.

A schedule $\mathcal{S}$ for a subset $A' \subseteq A(J)$ of tasks to an interval $I \subseteq [T]$ with integer length is a function $\mathcal{S} : A' \to [m] \times I$ that indicates the (machine, slot) pair that each task is assigned to. For every $a \in A'$, we then use $\mathcal{S}_{\text{mac}}(a)$ and $\mathcal{S}_{\text{time}}(a)$ to denote the first and second component of $\mathcal{S}(a)$ respectively.

Definition 5.1. *A schedule $\mathcal{S}$ for $A' \subseteq A(J)$ is valid if it satisfies the following constraints.*

- *Capacity Constraints: for every two tasks $a \neq a' \in A'$, we have $\mathcal{S}(a) \neq \mathcal{S}(a')$.*
- *No-migration Constraints: For every pair of tasks $a \sim a' \in A'$, we have $\mathcal{S}_{\text{mac}}(a) = \mathcal{S}_{\text{mac}}(a')$.*
- *Precedence Constraints: For every pair of tasks $a \prec a' \in A'$, we have $\mathcal{S}_{\text{time}}(a) < \mathcal{S}_{\text{time}}(a')$.*
- *Communication Delay Constraints: Consider a pair of jobs $j \prec j'$ with precedence constraints. Suppose $a_{1,j'} \in A'$ and $a_{p_j,j} \in A'$. If $\mathcal{S}_{\text{mac}}(a_{1,j'}) \neq \mathcal{S}_{\text{mac}}(a_{p_j,j})$, then*

$$\mathcal{S}_{\text{time}}(a_{1,j'}) > \mathcal{S}_{\text{time}}(a_{p_j,j}) + 1$$

The first three constraints of the above definition is same as Definition 4.1 for the no-delay problem. So, let us focus on the communication delay constraints. For a pair of jobs $j \prec j'$ with precedence constraints, we enforce the communication delay constraints only if the first task of j' (that is $a_{1,j'}$) and the last task of j (that is $a_{p_j,j}$) are in the set A'. The reason is that if either one of them is not in the set A', it means that our algorithm has *discarded* that task. For such jobs, we enforce the communication delay constraints when inserting the discarded tasks back.

5.1 LP Relaxation

Similar to our first result, the algorithm to prove Theorem 1.2 is also based on rounding Sherali-Adams lift of a LP for the problem. We now give a new LP relaxation for minimizing makespan with precedence and communication delay constraints, which extends the LP for the no-delay problem. Similar to the LP for the no-delay problem, we use the variables $x_{(a,i,t)}$, which are intended to be 1 if the task $a \in A(J)$ is assigned to machine $i \in [m]$ at time $t \in [T]$. Let $[m]_{-i}$ denote the set $\{[m] \setminus \{i\}\}$. In our new LP, we have all the constraints in the LP for the precedence constraints problem (i.e, Constraints (2) to (6)). The only new set of constraints we introduce is the following one:

$$x_{(a_{1,j'},i,t+1)} + \sum_{i' \in [m]_{-i}} x_{(a_{p_j,j},i',t)} \leq 1 \qquad \forall j \prec j', i \in [m], t \in [T-1] \tag{13}$$

Consider a pair of jobs $j \prec j'$. We want to guarantee that if j and j' are scheduled on different machines, then j' starts at least 1 time step after the completion of job j. This is equivalent to satisfying the communication delay constraint on the last task of job j and the first task of job j'. This further implies that if $x_{(a_{p_j,j},i,t)} = 1$ for some machine i and time slot t, then $x_{(a_{1,j'},i',t)} = 0$ for all machines i'. The constraints (13) precisely guarantee this. Therefore, if there is an optimal integral solution with makespan at most T, then there is a feasible solution to the LP. We use $\mathcal{P}(T)$ (or $\mathcal{P}$) to denote the polytope define by LP (2-6, 13). Our main goal in this section is to prove the following theorem.

Theorem 5.2. *Let T be the smallest value for which the Sherali-Adams lift of LP (2-6) to $r = (\log n)^{((m^2/\epsilon^2) \cdot \log\log n)}$ rounds has a feasible solution x. Then any feasible solution x can be rounded to produce valid a schedule $\mathcal{S} : A(J) \to [m] \times [(1+\epsilon)T]$ satisfying the Definition 5.1. Therefore, the makespan of $\mathcal{S}$ is at most $(1+\epsilon)T$.*

Note that above theorem immediately implies Theorem 1.2.

Remark: We note that constraints (13) are not the strongest inequalities we can write for the communication delay constraints. We can indeed describe exactly the convex hull of all integral schedules restricted to the 2 jobs $j \prec j'$. However, our algorithm uses the constraints only at the lowest level of recursion where it schedules jobs by conditioning. So any constraint that gives the correct set of integral solutions will be sufficient for our goal.

5.2 Rounding Algorithm

Towards proving Theorem 5.2, we first design an algorithm that only schedules a subset $A(J) \setminus A_{\text{discarded}}$ of tasks.

Lemma 5.3. *Let T be the smallest value for which the Sherali-Adams lift of LP (2-6, 13) to $r = (\log n)^{((m^2/\epsilon^2) \cdot \log\log n)}$ rounds has a feasible solution x. Then there is a valid schedule $\mathcal{S} : A(J) \setminus A_{\text{discarded}} \to [m] \times [T]$ with $|A_{\text{discarded}}| \leq \epsilon T$.*

Our algorithm to prove Lemma 5.3 uses PARTIAL-SCHEDULE from the first part, but modifies it to satisfy the communication delay constraints. Before we proceed with description of how we accomplish that, let us summarize the three places in which the algorithm for the no-delay problem actually makes the assignment of tasks to time slots.

1. Scheduling of tasks by conditioning: Consider the first line of PARTIAL-SCHEDULE procedure. Here, if the length of the interval $|I^*| < 2^{k^2}$, then the algorithm schedules tasks by conditioning on the LP solution.
2. Scheduling of top jobs: The second place where our algorithm assigns tasks to time slots is when inserting the top jobs in the last line of the procedure PARTIAL-SCHEDULE.
3. Scheduling of discarded tasks: Lastly, our algorithm assigns tasks to time slots when scheduling the discarded tasks.

The high level idea of how we deal with the communication delay constraints is the following: The constraints (13) of the LP guarantee that if tasks are scheduled by conditioning, then the communication delay constraints are satisfied. For every discarded task, we simply create an empty time slot on either side of the time slot on which it is scheduled. This guarantees that no matter how other jobs that have precedence relationship with it are scheduled, the communication delay constraints are satisfied. Finally, when scheduling top jobs we will make use of the fact that the chain lengths among top jobs are small, hence the communication delay constraints do not pose a big problem. We give a formal proof of validity of our schedule later in proof of Theorem 5.2.

We give the modified PARTIAL-SCHEDULE algorithm for the delay problem is given in Algorithm 4, which we call PARTIAL-SCHEDULE-COMM. First we highlight the places where the two algorithms differ.

1. Consider the first line of PARTIAL-SCHEDULE-COMM. Here, if the length of the interval is sufficiently small, then our algorithm schedules the remaining tasks by conditioning the LP solution x. Next, it also discards completely the set of the tasks scheduled in the last time slot of the interval I^*; there can be m such tasks. We will make use of this fact in the proof of Theorem 5.3 to show that the communication constraints are satisfied across all jobs. Note that the length of the interval at the last level of recursion is at least $\Omega(\log n)$, hence, the number of such discarded tasks across all the intervals is very small.
2. The main difference between PARTIAL-SCHEDULE-COMM and PARTIAL-SCHEDULE is in scheduling the top jobs. For this step we give a new algorithm to insert top jobs such that communication delay constraints taken into account.

Algorithm 4 PARTIAL-SCHEDULE-COMM$\left(I^*, J^*, J^*_{\text{special}}, \sigma, x\right)$

Input: a partial-scheduling instance $(I^*, J^*, J^*_{\text{special}}, \sigma, x)$ satisfying Definition 4.3
Output: a schedule of $\mathcal{S}^* : A(J^*) \cup A(I^*, J^*_{\text{special}}, x) \setminus A_{\text{discarded}} \to [m] \times I^*$ for some $A_{\text{discarded}}$

```
1: if $(|I^*| < 2^{k^2})$ then
2:     schedule the tasks by conditioning on the LP solution $x$.
3:     discard all the tasks scheduled at the last time slot $|I^*|$ of the interval $I^*$.
4:     return.
5: for every job $j \in J^*_{\text{special}}$ do: $x \leftarrow$ SPLIT$(x, j)$
6: while there exists $I \in \mathcal{I}^*_0 \cup \ldots \cup \mathcal{I}^*_{k^2-1}$ and a chain $\mathcal{C}$ of jobs owned by $I$ with total size at least $\delta|I|$ do
7:     $j \leftarrow$ the first job in $\mathcal{C}$, $a \leftarrow$ last task of $j$,
8:     take $(i, t)$ such that $x_{(a,i,t)} > 0$ with the largest $t$
9:     $x \leftarrow x$ conditioned on the event $(a, i, t)$
10:    $J^* \leftarrow J^* \setminus \{j\}, J^*_{\text{special}} \leftarrow J^*_{\text{special}} \cup \{j\}, \sigma(j) \leftarrow i$
11:    $x \leftarrow$ SPLIT$(x, j)$
12: Partition the jobs in the set $J^*$ as follows:
        $J^*_{\text{top}} = \bigcup_{\ell=0}^{\ell^*-k-1} J^*_\ell(x)$; $J^*_{\text{mid}} = \bigcup_{\ell=\ell^*-k}^{\ell^*-1} J^*_\ell(x)$; $J^*_{\text{bot}} = \bigcup_{\ell=\ell^*}^{\log T^*} J^*_\ell(x)$,
13: where $\ell^* \in \{k, \ldots, k^2\}$ is chosen satisfying the condition below:
        $$|A(J^*_{\text{mid}})| \le \frac{\epsilon}{4} \cdot \frac{T^*}{\log T} + \frac{\epsilon}{2m} \cdot \left(|A(J^*_{\text{mid}})| + |A(J^*_{\text{top}})|\right)$$
14: for every interval $I \in \mathcal{I}^*_{\ell^*}$ do
15:     PARTIAL-SCHEDULE$\left(I, J^*_{\text{bot}}(I, x), J^*_{\text{special}}, \sigma, x\right)$
16: Insert $J^*_{\text{top}}$ to $I^*$ using Lemma 5.4.
```

From the pseudo-code of PARTIAL-SCHEDULE-COMM, it is clear that most of the lemmas proved for the precedence constrained scheduling directly extends to the communication delay. Thus it only remains to argue that one can schedule the top jobs without discarding too many tasks even when there are communication delay constraints.

5.3 Scheduling the Top Jobs With Communication Delay Constraints

Now we give more details about PARTIAL-SCHEDULE-COMM to schedule top jobs respecting the communication delay constraints. Fix a recursive invocation of the procedure with the input instance instance $(I^*, J^*, J^*_{\text{special}}, \sigma, x)$. We assume that the input satisfies Definition 4.3 given in the first half of the paper. As, the first 4 steps of PARTIAL-SCHEDULE-COMM procedure remains exactly the same as PARTIAL-SCHEDULE, we assume that we have a partial schedule of tasks belonging to the bottom jobs and the special jobs. Formally,

$$\mathcal{S} : (A(J^*_{\text{bot}}) \cup A(I^*, J_{\text{special}}, x)) \setminus A_{\text{bottom-discarded}} \to [m] \times I^*$$

Let $\widehat{A} := (A(J^*_{\text{bot}}) \cup A(I^*, J_{\text{special}}, x)) \setminus A_{\text{bottom-discarded}}$ for rest of this subsection. We want to extend the schedule $\mathcal{S}$ to $\mathcal{S}^*$ which includes most of the tasks of the set $A(J^*_{\text{top}})$. In particular, we want to prove the following lemma which is a counterpart to Lemma 4.6, but with communication delay constraints.

Lemma 5.4. *The valid schedule $\mathcal{S} : \widehat{A} \to I^*$ can be extended to a valid schedule*

$$\mathcal{S}^* : \left(\widehat{A} \cup A(J^*_{\text{top}})\right) \setminus A_{\text{top-discarded}} \to [m] \times I^*$$

such that $|A_{\text{top-discarded}}| \leq \frac{\epsilon}{4} \cdot \frac{T^*}{\log T}$.

Our strategy to prove the above lemma follows the same framework as in the no-delay problem. In the first stage, we build a *tentative assignment* of tasks in $A(J^*_{\text{top}})$ in the slots left open by $\mathcal{S}$. At this stage, we only make sure that capacity constraints (that is, only 1 task is scheduled in 1 time slot on a machine) and the precedence constraints between tasks in $A(J^*_{\text{top}})$ and $\widehat{A}$ are satisfied. Both the communication delay constraints and the non-migratory constraints may be violated, and will be fixed in the second stage. During this step we discard some tasks from the set $A(J^*_{\text{top}})$, which we denote by $A^1_{\text{top-discarded}}$. We obtain this schedule by applying the tentative schedule algorithm from Section 4.6.1 for the no-delay problem. Then we get the following lemma.

Lemma 5.5 ([25])**.** *A feasible partial schedule $\mathcal{S} : \widehat{A} \to I^*$ of tasks in bottom jobs can be extended to a new schedule $\mathcal{S}' : \left(\widehat{A} \cup A(J^*_{\text{top}})\right) \setminus A^1_{\text{top-discarded}} \to [m] \times I^*$ satisfying following properties:*

1. *Consider $j \in J^*_{\text{top}}$ and let $a \in A(j)$. Then in the schedule $\mathcal{S}'$, either a is assigned in the interval $[r^*_j, d^*_j]$ or $a \in A^1_{\text{top-discarded}}$.*
2. *The total number of discarded tasks $|A^1_{\text{top-discarded}}| \leq 4m2^{-k}T^*$.*
3. *The precedence constraints between tasks in $\left(A(J^*_{\text{top}}) \setminus A^1_{\text{top-discarded}}\right)$ and $\widehat{A}$ are respected.*
4. *The capacity constraints are satisfied.*

Note that above conditions do not guarantee that the communication delay constraints hold between jobs in $\left(A(J^*_{\text{top}}) \setminus A^1_{\text{top-discarded}}\right)$ and $\widehat{A}$. Next, we convert the tentative schedule into a valid partial schedule respecting all the precedence constraints and communication delay constraints. During this step our algorithm discards some more tasks from the set $A(J^*_{\text{top}})$, which we denote by $A^2_{\text{top-discarded}}$. Define $A^1_{\text{top-discarded}} = A^1_{\text{top-discarded}} + A^2_{\text{top-discarded}}$. Our final schedule

$$\mathcal{S}^* : \left(\widehat{A} \cup A(J^*_{\text{top}})\right) \setminus A_{\text{top-discarded}} \to [m] \times I^*$$

satisfies the following guarantees:

- For every job $j \in J^*_{\text{top}}$, all the non-discarded tasks are scheduled within $[r^*_j, d^*_j]$.
- The assignment of tasks in $\widehat{A}$ remains same as that in the schedule $\mathcal{S}'$.

- The communication delay constraints are satisfied among all non-discarded job.

The first two invariants guarantee that the precedence constraints are satisfied among all non-discarded tasks. We will argue $\mathcal{S}^*$ also satisfies the communication delay constraints, and hence is a valid partial schedule of non-discarded tasks satisfying Definition 5.1. Similar to what we did in the first half of the paper, we build schedule $\mathcal{S}^*$ by first designing an algorithm for a new stand-alone scheduling problem, then using it as a black-box for scheduling top jobs.

5.4 A Deadline Scheduling Problem with Precedence and Communication Delay Constraints

We are given a set of jobs J with processing lengths p_j, release times r_j, deadlines d_j. The jobs have precedence constraints and communication delay constraints. The precedence constraints among jobs satisfy the property that if $j \prec j'$, then $r_j \leq r_{j'}$ and $d_j \leq d_{j'}$. Recall that we use $\Delta(J)$ to denote the maximum chain length in J. The time horizon $\{1, 2, \ldots, T\}$ is partitioned into p equal sized intervals $I_1, I_2, \ldots, I_p$. The release times and deadlines of jobs correspond to the beginning and the end of the intervals. For each machine i, we are given a capacity function $\mathsf{cap}_i : [T] \to \{0, 1\}$. If $\mathsf{cap}_i(t) = 1$, then the time slot t on machine i is available to schedule a task in $A(J)$.

Suppose there is a schedule of tasks $\mathcal{S}' : A(J) \to [m] \times [T]$ that assigns each task in $A(J)$ to a machine, time slot pair such that no two tasks are assigned to the same machine and the same time slot; that is, capacity constraints on machines are satisfied. Moreover, $\mathcal{S}'$ ensures that for each job $j \in J$, all its tasks $A(j)$ are scheduled within $[r_j, d_j]$. However, $\mathcal{S}'$ may not respect the precedence and communication delay constraints in J, the schedule may not be non-migratory. Our goal is to schedule each $j \in J$ on a single i such that the precedence and communication delay constraints among the jobs is satisfied, and the schedule is non-migratory. We prove the following theorem in this subsection.

Theorem 5.6. *There exists an algorithm that in polynomial time converts the schedule $\mathcal{S}'$ into a valid schedule σ that satisfies the following properties:*

1. *It partially schedules every job on exactly one machine (no-migration). For a job that is partially scheduled, we discard the remaining tasks; For the sake of the precedence constraints, we assume that every partially scheduled job or a fully discarded job is completely processed.*
2. *The precedence constraints among the jobs is satisfied.*
3. *The communication delay constraints among the jobs is satisfied, as given in Definition 5.1.*
4. *The total number of discarded tasks is at most $6p^2 m \Delta(J)$.*

We prove the above theorem by extending the procedure EDF+ECT described in Algorithm 3 to satisfy the communication delay constraints. We give the pseudocode in Algorithm 5, and refer to the procedure as EDF+ECT+COMM.

To prove how our algorithm satisfies the communication delay constraints, we need the following simple observation.

Observation 5.7. *Fix a job $j \in J$ for which $B_j \neq$ DISCARDED. Consider the active interval $[B_j, C_j]$ as set by the procedure EDF+ECT+COMM. Then, no task of job j is scheduled at the time slots B_j and C_j.*

Proof. See Figure 2 for a proof by pictures. Consider the definition of B_j in line 6 of the algorithm. This is the first time slot at which job j can be scheduled while respecting the

Algorithm 5 EDF+ECT+COMM

Input: A set of jobs J with release times, deadlines and precedence constraints; capacity function $\mathsf{cap}_i : [T] \to \{0,1\}$ for each machine i.

Output: Schedule $\mathcal{S} : A(J) \to [m] \times ([T])$ such that for $a \sim a'$, either a or a' belongs to $A_{\text{discarded}}$ or $\mathcal{S}_{\text{mac}}(a) = \mathcal{S}_{\text{mac}}(a')$.

```
1: Sort the jobs in $J$ in the increasing order of their deadlines. Reindex the jobs so that
   $J := \{1, 2 \dots, n\}$ and $d_1 \le d_2 \le \dots \le d_n$.
2: Initialize $A_{\text{discarded}} = \emptyset$.
3: for $j = 1$ to $n$ do
4:     Find the earliest time slot $t \in [T]$ such that following conditions hold: i) $\mathsf{cap}_i(t) = 1$ for
       some machine $i \in [m]$ and $r_j \le t \le d_j$; ii) $C_{j'} < t$ for all $j' \prec j$.
5:     If no such $t$ exists, then set $B_j$ := DISCARDED and add all tasks $A(j)$ to the set
       $A_{\text{discarded}}$.
6:     If there is a $t$ satisfying the conditions above, set $B_j = t$ and do the following.
7:     Find the set of machines $M' \subset [m]$ such that $\forall i \in M'$, $\mathsf{cap}_i(B_j + 1, d_j - 1) \ge p_j$.
8:     if $|M'| = 0$ then
9:         $i^* = \max_i \{\mathsf{cap}_i(B_j + 1, d_j - 1)\}$.
10:        Schedule $\mathsf{cap}_i(B_j + 1, d_j - 1)$ tasks of the job $j$ in the interval $[B_j + 1, d_j - 1]$ on the
           machine $i^*$.
11:        Set $C_j = d_j$. Add the remaining $p_j - \mathsf{cap}_i(B_j + 1, d_j - 1)$ tasks to the set $A_{\text{discarded}}$.
12:        Update the capacity function $\mathsf{cap}_{i^*}$ for the machine $i^*$.
13:    if $|M'| \ge 1$ then
14:        Find the earliest time slot $t^* \le d_j - 1$ such that there exists a machine $i^* \in M'$ and
           $\mathsf{cap}_i(B_j + 1, t^*) = p_j$.
15:        Set $C_j = t^* + 1$. Schedule the tasks $A(j)$ in the interval $[B_j + 1, C_j - 1]$ on the
           machine $i^*$ and update $\mathcal{S}$.
16:        Update the capacity function $\mathsf{cap}_{i^*}$ for the machine $i^*$.
17: return $\mathcal{S}$.
```

precedence constraints; that is, for every $j' \prec j$, $C_{j'} < B_j$. Similarly, we define C_j as either d_j or the earliest time slot $t^* + 1$ such that there are p_j empty time slots on some machine i in the interval $[B_j, t^*]$; See the lines 15 and 11. From lines 10 and 15, it is clear that our algorithm does not schedule any task of job j at the time steps B_j and C_j. □

We will argue in the proof of Theorem 5.2 that above observation immediately implies that the communication delay constraints are satisfied for a pair of jobs j and j' if either one of them happen to be a top job.

For now, we focus on arguing that EDF+ECT+COMM did not discard too many tasks. Interestingly, we show that the total number of tasks discarded by our algorithm to enforce communication delay constraints is only factor 5 more than EDF+ECT. The intuition is that every time that was left empty to satisfy a communication delay constraint should also account for the decrease in chain length. As the chain length among top jobs is small, we argue that wasted time slots is also small. We now give more details about EDF+ECT+COMM.

We call a time slot t on machine i as *idle* if $\mathsf{cap}_i(t) = 1$ and our schedule $\mathcal{S}$ does not assign any task at time t on machine i. The idle time slots correspond to the number of tasks we discard, and our goal going forward is to show that there are not too many idle time slots in $\mathcal{S}$. The following observations are needed for proving Theorem 4.8.

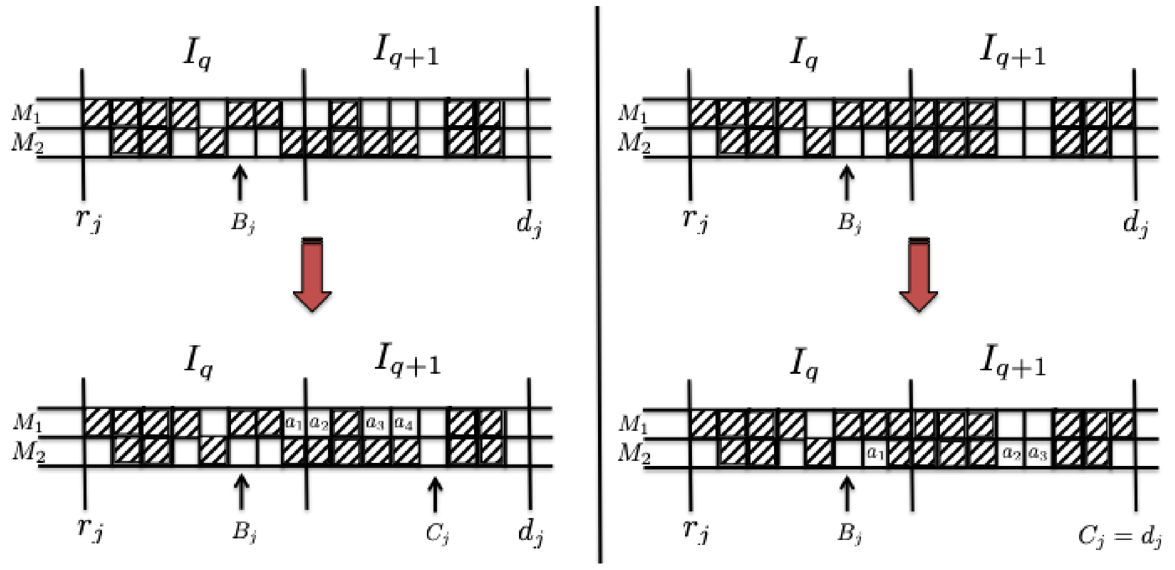

Figure 2: The figure illustrates inserting a job j with $p_j = 4$ on two machines. On the left, the job is fully scheduled in the interval $[B_j, C_j]$ on machine M_1. On the right, the task a_4 is discarded but the remaining tasks are scheduled on M_2. In the both cases, notice that no task of j is scheduled either at B_j or at C_j.

Observation 5.8. *Fix an interval I_q for some $q \in [p]$. Suppose the following two conditions hold:*

- *There is a time slot $t^* \in I_q$ that is idle on machine i in $\mathcal{S}$.*
- *There exists a job j^* with $t^* \in [r_{j^*}, d_{j^*}]$ and $B_{j^*} > t^*$ or all the tasks $A(j^*) \in A_{\text{discarded}}$.*

Then there exists a job j such that $t^* \in [B_j, C_j]$ and $j \prec j^*$.

Proof. Proof of the above observation follows by the fact that if no such job j exists, then when our algorithm considers the job j^* then it would set $B_j = t^*$; refer to line 6 in EDF+ECT+COMM. □

Observation 5.9. *Consider a job $j \in J$ with $B_j \neq$ DISCARDED that is active in the interval $[B_j, C_j]$. Let p'_j be the total number of tasks scheduled in $[B_j, C_j]$ on machine i. Then for any other machine $i' \neq i$, the number of idle time slots in the interval $[B_j, C_j]$ is at most $p'_j + 2$.*

Proof. See Figure 2 for a proof by pictures. Consider the case when $p'_j = p_j$. In this case, the lemma follows from the observation that our algorithm assigns jobs to machines on which they will have earliest completion time; See line 14 of the algorithm. Now consider the case when $p'_j \neq p_j$. As job j was scheduled on machine i, from line , it follows that i had the maximum number of empty slots in the interval $[B_j + 1, d_j - 1]$, which is at most p'_j. Therefore, the maximum number of empty slots on any machine in the interval $[B_j, d_j]$ is at most $p'_j + 2$. Since in this case we set $C_j = d_j$, we complete the lemma. □

We will use the above facts to argue that the number of idle time slots on any machine is small.

Lemma 5.10. *Consider any arbitrary time interval $I := \{t', \ldots, t''\} \subseteq I_q$ for some $q \in [p]$. Suppose there is at least one job j^* with $I \subseteq [r_{j^*}, d_{j^*}]$ and $B_{j^*} > t''$ or all the tasks $A(j^*) \in A_{\text{discarded}}$ ($B_j =$ DISCARDED). Then, for any machine $i \in [m]$ the number of idle time slots in I is at most $3\Delta(J)$.*

Before we proceed with the proof, contrast the above lemma with Lemma 4.11; the number of idle slots increases in communication delay case increases by a factor of 3.

Proof. We prove this by showing a contradiction that if there are more than $3\Delta(J)$ idle time slots on a machine, then the maximum chain length among jobs in J is more than $\Delta(J)$. Consider a machine i with $3\Delta(J)+1$ idle time slots in the interval I. Let $t^* \in I$ be the latest time slot on machine i that is idle. Since j^* is available at t^*, by Observation 5.8, it must be the case that there exists a job $j_1 \prec j^*$ that is active at time t^*, which implies $t^* \in [B_{j_1}, C_{j_1}]$. Now consider the latest time slot $t' < B_{j_1}$ that is empty on machine i. We claim that j_1 is available for processing at time t'. This follows from our assumptions that if $j_1 \prec j^*$, then $r_{j_1} \leq r_{j^*}$ and the release times and the deadlines of J align with the beginnings and the endings of the intervals. Therefore, there must be a job $j_2 \prec j_1$ such that $t' \in [B_{j_2}, C_{j_2}]$. Moreover, $C_{j_2} < B_{j_1}$ as $j_2 \prec j_1$. We continue by induction to construct a chain of jobs $j_y \prec j_{y-1} \prec \ldots \prec j_1 \prec j^*$ such that $[B_y, C_y] \cup [B_{y-1}, C_{y-1}] \cup \ldots \cup [B_{j_1}, C_{j_1}]$ covers all the empty slots on machine i in the interval I. Furthermore, for any two intervals $I', I'' \in \{[B_y, C_y], [B_{y-1}, C_{y-1}], \ldots, [B_{j_1}, C_{j_1}]\}$, $I' \cap I'' = \emptyset$. The total processing lengths of the jobs in the chain $j_y \prec j_{y-1} \prec \ldots \prec j_1 \prec j^*$, $\sum_{v=1}^{y} p_v$, is at most $\Delta(J)$, since the maximum chain length among J is at most $\Delta(J)$. By Observation 5.9, for any job j_v belonging to the chain, there can be at most $p_{j_v}+2$ empty slots on the machine i in the interval $[B_{j_v}, C_{j_v}]$. Therefore, $\sum_{v=1}^{y} \text{Idle}_i(B_{j_v}, C_{j_v}) \leq \sum_{v=1}^{y}(p_v+2) \leq 3\Delta(J)$. However, $[B_y, C_y] \cup [B_{y-1}, C_{y-1}] \cup \ldots \cup [B_{j_1}, C_{j_1}]$ covers all the empty slots on machine i, which is a contradiction. □

The above lemma implies the following useful corollary.

Corollary 5.11. *Suppose there is an interval I_q and a machine i with more than $3\Delta(J)$ idle time slots. If there is a job j^* such that $B_{j^*} \in I_{q+1} \cup I_{q+2} \cup \ldots \cup I_p$, then the release time of job $r_{j^*} \in I_{q+1} \cup I_{q+2} \cup \ldots \cup I_p$.*

Proof. For contradiction, let us assume that the release time of j^* belongs to $I_{q'}$, where $q' < q+1$. Recall that all jobs are released at the beginning of the intervals. Now, we invoke the previous lemma on the interval I_q and the job j^*, which gives us a necessary contradiction. □

The next lemma shows that even if a job is partially discarded then the number of idle time slots on a machine cannot be too much.

Lemma 5.12. *Consider any arbitrary time interval $I := \{t', \ldots, t''\} \subseteq I_q$ for some $q \in [p]$. Suppose there is at least one job j^* with $I \subseteq [r_{j^*}, d_{j^*}]$ and $B_{j^*} > t'$ or a subset of the tasks $A(j^*) \in A_{\text{discarded}}$. Then, for any machine i the number of idle time slots in I is at most $6\Delta(J)$.*

Proof. If $B_{j^*} \geq t''$, then the proof follows from Lemma 5.10. Therefore, $B_{j^*} \leq t''$ and some tasks of j^* got discarded. By Observation 5.9, in the interval $[B_j, t'']$ there cannot be more than $p_{j^*}+2 \leq \Delta(J)+2$ idle slots on any machine i. By applying Lemma 5.10 to the interval $[t', B_{j^*}-1]$ and job j^*, we conclude that in the interval $[t', B_{j^*}-1]$ there can be at most $3\Delta(J)$ idle time slots. Therefore, there can be at most $3\Delta(J)+\Delta(J)+2 \leq 6\Delta(J)$ idle slots on any machine i. □

With above two observations, it is easy to prove Theorem 5.6. For brevity, we use $\mathcal{S}^{-1}(I)$ to denote the set of tasks scheduled in the interval I in $\mathcal{S}$.

Proof of Theorem 5.6. Our algorithm guarantees that in the schedule $\mathcal{S}$ all the tasks $A(j)$ of a job $j \in J$ are assigned to a single machine and the precedence constraints among the jobs is satisfied. Moreover, communication delay constraints are satisfied as our algorithm while scheduling a top job j always leaves the first slot in the active interval $[B_j, C_j]$ empty. Therefore, it only remains to show that the number of discarded tasks $|A_{\text{discarded}}| \leq 6p^2 m\Delta(J)$, which now readily follows from the proof of Theorem 4.8 from the first part. $\square$

Now we are ready to prove Lemma 5.4, which bounds the total number tasks discarded in inserting top jobs.

Proof. We obtain the schedule $\mathcal{S}^*$ by applying Theorem 5.6 on the jobs in the set J^*_{top}. For every job $j \in J^*_{\text{top}}$, we define the truncated job length p'_j by taking into account the discarded tasks in $A^1_{\text{top-discarded}}$. We define the release time of j as r^*_j and the deadline as d^*_j, where r^*_j, d^*_j are defined in Lemma 5.5 For each machine i, $\mathsf{cap}_i(t) = 1$ if time t on machine i is not assigned any task in the schedule $\mathcal{S}$. Recall that $\mathcal{S}$ gives an assignment of subset of tasks in the bottom jobs and the special jobs $(\widehat{A})$ in the interval I^*.

The maximum chain length of jobs in J^*_{top} is at most $k^2\delta T^*$, where T^* is the length of interval I^*. This follows by applying Lemma 4.5 for the no-delay problem to our setting. Therefore, by Theorem 5.6, the total number of tasks discarded in converting the tentative schedule into an actual schedule is given by

$$|A^2_{\text{top-discarded}}| \leq 5p^2 m \cdot k^2 \delta T^*.$$

By Lemma 5.5,

$$|A^1_{\text{top-discarded}}| \leq 4m2^{-k}T^*.$$

Therefore,

$$\begin{aligned} |A_{\text{top-discarded}}| &= |A^1_{\text{top-discarded}}| + |A^2_{\text{top-discarded}}| \\ &\leq 5p^2 m \cdot k^2\delta T^* + 4m2^{-k}T^* \\ &\leq \frac{\epsilon}{4} \cdot \frac{T^*}{\log T} \end{aligned}$$

The last inequality follows from substituting $p = 2^{\ell^*} \leq 2^{k^2}$, $k = \frac{O(1)m}{\epsilon} \log\log T$, and we set $\delta = \frac{\epsilon}{16k^2 m 2^{2k^2} \log T}$.

$\square$

5.5 Proof of Theorem 5.2

First we note that guarantee of Lemma 5.4 is exactly same as the guarantee of Lemma 4.6 for the no-delay problem. As rest of the steps of PARTIAL-SCHEDULE-COMM remains exactly same as PARTIAL-SCHEDULE, all the lemmas we proved for PARTIAL-SCHEDULE procedure also directly extend to the PARTIAL-SCHEDULE-COMM. Hence it is not hard to see that a proof of Lemma 5.3 follows by repeating the arguments in the proof of Lemma 4.2 for the no-delay problem. We omit the proof as the details are fairly straightforward.

Now we have all the ingredients to prove our main result for the delay problem.

Proof of Theorem 5.2. Set $\epsilon' = \epsilon/3$. By Lemma 5.3, there is a partial schedule for $\mathcal{S}$ for $A(J) \setminus A_{\text{discarded}}$ of makespan $[T]$ with $|A_{\text{discarded}}| \leq \epsilon' T$. We extend $\mathcal{S}$ to a valid schedule $\mathcal{S}^*$ for $A(J)$ with makespan $[T + 3|A_{\text{discarded}}|]$ as follows. We give a procedure to insert one discarded task such that all the precedence and communication constraints are satisfied. The final schedule is constructed by repeating this procedure for each discarded task. Consider a discarded task

a. Let t be the earliest time step in $\mathcal{S}$ where a can be scheduled respecting the precedence constraints. Now, create three new private slots at $t, t+1$ and $t+2$ for the task a. Schedule the task a at the time step $t+1$ on the machine i such that non-migratory constraints are satisfied. The makespan increases by an additive factor of 3 for every discarded task, and hence the total increase in makespan is $3|A_{\text{discarded}}|$. As $|A_{\text{discarded}}| \leq \epsilon T/3$, we conclude that makespan of the our schedule is at most $(1+\epsilon)T$.

It is easy to see that the precedence constraints and non-migratory constraints are satisfied by our schedule. It remains to argue about the communication delay constraints.

Fix any two jobs j and j' such that $j \prec j'$. Recall that $a_{p_j,j}$ denotes the last task of job j and $a_{1,j'}$ denotes the first task of job j'. Let t, t' be the time slots at which the tasks $a_{p_j,j}$ and $a_{1,j'}$ are scheduled by our algorithm. We will argue that if $a_{p_j,j}$ and $a_{1,j'}$ are scheduled on two different machines then, $t' > t+1$. We consider the following cases.

- Our algorithm discarded either $a_{p_j,j}$ or $a_{1,j'}$. Let us, without loss of generality, assume that $a_{1,j'}$ was discarded. From our description of the algorithm to schedule discarded jobs, it follows that the time slot $t'-1$ was empty. Therefore, $t' > t+1$.

- Our algorithm scheduled either $a_{p_j,j}$ or $a_{1,j'}$ using the procedure EDF+ECT+COMM. Consider the case when $a_{1,j'}$ was in the set of top jobs. From Observation 5.7, it follows that $t' \geq B_{j'}+1$, where $B_{j'}$ denotes the earliest time step when the job j' can be scheduled respecting the precedence constraints. Note that $B_{j'} > t$. Hence, $t' \geq B_{j'}+1 > t+1$. On the other hand, consider the case when $a_{p_j,j}$ was scheduled using EDF+ECT+COMM. Again from Observation 5.7, it follows $t < C_j \leq d_j$, as no task of job j is scheduled at the time step d_j. Further, due to the precedence constraints $t' > d_j$. Thus, $t' > t+1$.

  Observe that in these cases we simply assumed the worst scenario that the jobs are scheduled on different machines by our algorithm.

- Both $a_{p_j,j}$ and $a_{1,j'}$ was scheduled by our algorithm by conditioning on the LP solution in the first step of PARTIAL-SCHEDULE-COMM. We consider two cases. Suppose both $a_{p_j,j}$ and $a_{1,j'}$ belonged to the same interval I^*, and were scheduled on two different machines. In this case, $t' > t+1$ follows by the LP constraints (13). Consider the second case where $a_{p_j,j}$ is scheduled by conditioning in the interval I^* and $a_{1,j}$ is scheduled by conditioning in a different interval I^{**}. Clearly, I^* has to be to the left of I^{**}. Now consider the first line of procedure PARTIAL-SCHEDULE-COMM. Here, we completely discard all the tasks scheduled in the last time slot in the interval I^*. If there was at least one such discarded task, it is clear that $t' > t+1$ as every discarded task creates one empty time slot to the right of it. If no tasks were discarded, then it implies that in last time slot of the interval I^* was completely empty. In this case also, $t' > t+1$.

  Therefore, we conclude that our algorithm satisfies communication delay constraints.

□

Thus to prove our second main Theorem, we only need to argue that proof of Theorem 5.2 also extends when.

Proof of Theorem 1.2. Define $\beta = \max_{j \prec j'}\{c_{j,j'}\}$. We need the following changes to complete our proof.

- The communication constraints in our LP become:

$$x_{(a_{1,j'},i,t+1)} + \sum_{i' \in [m]_{-i}} \sum_{t'=t-\beta}^{t} x_{(a_{p_j,j},i',t')} \leq 1 \quad \forall j \prec j', i \in [m], t \in [T-1]$$

- In the first line of procedure PARTIAL-SCHEDULE-COMM, we completely discard all the tasks scheduled in the last β time slots in the interval I^*.
- We modify the schedule obtained by running EDF+ECT+COMM on the set of top jobs as follows: For every top job j, we discard first $\beta - 1$ and the last $\beta - 1$ tasks scheduled in the interval $[B_j, C_j]$.
- While inserting a discarded task a, we create $2\beta + 1$ time slots, and insert the task in the middle.

By repeating the proofs for the case of $\beta = 1$, it is easy to see that the total number of discarded tasks increases by a factor of $O(\beta)$. So, by appropriately choosing $\epsilon' = \epsilon/O(\beta)$, and running our entire algorithm by fixing ϵ' we can show that the makespan of the schedule is at most $(1+\epsilon)$T. It is also easy to see that all the communication delay constraints and the precedence constraints are also satisfied by this strategy. This completes the proof. $\square$

6 Integrality Gap Instance For Sherali-Adams Hierarchy for $P2|\text{prec}|C_{\max}$?

In this section, we give some evidence that an $o(\log n)$-level Sherali-Adams lift of the basic LP for $P2|\text{prec}|C_{\max}$ may not lead to a $(1+\epsilon)$-approximation for non-preemptive precedence constraints problems with arbitrary job lengths. Due to a certain technical difficulty, which will become clear later, we do not quite prove this exact statement; instead, we introduce a new scheduling problem that is equivalent to a special case of $P2|\text{prec}|C_{\max}$, and show an integrality gap result for the new problem. However, we believe the instance we construct is the right one for proving an integrality gap for $P2|\text{prec}|C_{\max}$.

6.1 A Scheduling Problem on Single Machine

In our problem, we have a single machine and n jobs J, each $j \in J$ having a size $p_j \in \mathbb{Z}_{>0}$, a release time $r_j \in \mathbb{Z}_{\geq 0}$ and a deadline $d_j \geq r_j + p_j$. There are no precedence constraints among jobs, but they have to be processed during their respective $(r_j, d_j]$ windows.

In the problem, any job $j \in J$ can be "partially processed". More specifically, we can choose a length $p'_j \in [0, p_j]$ for any $j \in J$ and process j non-preemptively only for p'_j units of time in $(r_j, d_j]$. The objective we consider is then to minimize $\sum_{j \in J}(p_j - p'_j)$, the total job units that are not processed (or "discarded"). We shall use $\sum_j p_j U'_j$ to denote this objective of minimizing $\sum_{j\in J}(p_j - p'_j)$, where $U'_j = \frac{p_j - p'_j}{p_j}$ is the fraction of job j that is unfinished.[5] We denote the problem by $1|r_j, d_j|\sum_j p_j U'_j$, and use the tuple (J, p, r, d) to denote an instance of this problem.

Suppose we are given an instance $\mathcal{I} = (J, p, r, d)$ of $1|r_j, d_j|\sum_j p_j U'_j$. Let $T = \sum_{j \in J} p_j$ and we assume $T \geq \max_{j \in J} d_j$. We construct an equivalent instance $\mathcal{I}'$ of $P2|\text{prec}|C_{\max}$ as follows. The jobs in $\mathcal{I}'$ will be $J \cup J^{\text{chain}}$, where $J^{\text{chain}} = \{j_1^{\text{chain}}, j_2^{\text{chain}}, \cdots, j_T^{\text{chain}}\}$ is a set of T unit-length jobs. For simplicity we define the lengths, release times and deadlines of J^{chain} as follows: for every $t \in [T]$, we have $p_{j_t^{\text{chain}}} = 1$ and $r_{j_t^{\text{chain}}} = t - 1$ and $d_{r_t^{\text{chain}}} = t$. Then the precedence constraints are defined as follows: for every $j, j' \in J \cup J^{\text{chain}}$, $j \prec j'$ if and only if $d_j \leq r_{j'}$. In particular, this implies $j_1^{\text{chain}} \prec j_2^{\text{chain}} \prec \cdots \prec j_T^{\text{chain}}$.

Our goal for $\mathcal{I}'$ is to schedule the precedence-constrained jobs $J \cup J^{\text{chain}}$ non-preemptively on 2 machines so as to minimize the makespan. We use $\mathcal{I}' = (J \cup J^{\text{chain}}, p, \prec)$ to denote this instance of $P2|\text{prec}|C_{\max}$. Ideally, we would like to use one machine to process J, and the other

[5]In the literature, U_j indicates if j is not scheduled; so we use $U'_j \in [0, 1]$ to indicating the fraction of j that is not processed.

one to process J^{chain}. Then the time window constraints in $\mathcal{I}$ will correspond to the precedence constraints in $\mathcal{I}'$. By discarding and inserting job units, a good schedule for $\mathcal{I}$ can be converted to a good schedule for $\mathcal{I}'$ and vice versa. This technique was crucially used the in [25] for the makespan minimization problem on unit-length jobs.

Formally, the following lemma establishes the equivalence between the two instances $\mathcal{I}$ and $\mathcal{I}'$:

Lemma 6.1. *Let $\mathcal{I} = (J, p, r, d)$ be an instance of $1|r_j, d_j|\sum_j p_j U'_j$, $T := \sum_{j \in J} p_j$ and assume $T \geq \max_{j \in J} d_j$. Let $\mathcal{I}' = (J \cup J^{\text{chain}}, p, \prec)$ be the instance of $P2|\text{prec}|C_{\max}$ constructed using the above procedure. Let $\delta \geq 0$ be any constant. Then,*

- *Any solution to $\mathcal{I}$ with cost δT can be efficiently converted to a solution to $\mathcal{I}'$ with makespan at most $(1+\delta)T$.*
- *Any solution to $\mathcal{I}'$ with makespan $(1+\delta)T$ can be efficiently converted to a solution to $\mathcal{I}$ with cost at most $2\delta T$.*

Proof. Given a schedule for $\mathcal{I}$ on one machine with at most δT job units discarded, we construct a schedule for $\mathcal{I}'$ as follows. We first add a second machine and schedule all the jobs J^{chain} naturally on the machine: we schedule j_t^{chain} at slot $(t-1, t]$. The precedence constraints are satisfied since they defined according to r_j and d_j values of jobs, and the schedule respects the time window constraints. Then we can insert the discarded job units back, increasing the makespan by at most δT.

Now suppose we are given a solution to $\mathcal{I}'$ with makespan $(1+\delta)T$. We focus on the time slots where no jobs in J^{chain} are scheduled. We remove from the schedule these time slots, as well as the job units processed in these slots. We removed at most $2\delta T$ job units. The resulting schedule has makespan exactly T and each job j_t^{chain} is scheduled in $(t-1, t]$. We can assume jobs in J^{chain} are processed on the same machine. Since the precedence constraints are satisfied, we have that all jobs $j \in J$ are scheduled within the window $(r_j, d_j]$. Removing the machine for J^{chain} gives us a schedule for $\mathcal{I}$ with at most $2\delta T$ job units discarded. □

The factor of 2 in the lemma does not create an issue for our reduction: we are interested in deciding whether the instance $\mathcal{I}'$ has makespan at most $(1+\epsilon)T$ or at least cT for some absolute constant $c > 1$. This is equivalent to deciding whether the instance $\mathcal{I}$ has cost at most ϵT or at least $c'T$ for some absolute constant $c' > 0$. Our main theorem is that an $o(\log n)$-level Sherali-Adams lifting of some natural LP relaxation for $1|r_j, d_j|\sum_j p_j U'_j$ can not distinguish between the two cases. We define the natural LP relaxation first and then give our theorem.

6.2 Integrality Gap Result for $1|r_j, d_j|\sum_j p_j U'_j$

As is typical in the LP/SDP lifting framework, we specify an upper bound B on the solution cost and impose a constraint for the objective function. In the LP, $x_{j,t,p'}$ indicates weather j is processed during the interval $(t, t+p'] \subseteq (r_j, d_j]$. We require $p' \in [p_j]$ and $t \in [r_j, d_j - p']$; notice that we do not have variables for the cases in which a job j is not processed at all. For simplicity, we assume all the variables $x_{j,t,p'}$ with (j, t, p') not satisfying the property are identically 0. The LP relaxation is as follows:

$$\sum_j \left(p_j - \sum_{t,p'} x_{j,t,p'} p' \right) \leq B \qquad (14) \qquad\qquad \sum_{j,t,p' : t' \in (t, t+p']} x_{j,t,p'} \leq 1 \qquad \forall t' \in [T] \qquad (16)$$

$$\sum_{t,p'} x_{j,t,p'} \leq 1 \qquad \forall j \in J \qquad (15) \qquad\qquad x_{j,t,p'} \geq 0 \qquad \forall j, p', t \qquad (17)$$

(14) says that we can discard at most B job units; (15) says each job j can be scheduled at most once. (16) requires that at any time t', at most 1 job can be processed, and (17) requires all variables to be non-negative. For a fixed instance $\mathcal{I} = (J, p, r, d)$ and a bound B, we use $\mathcal{P}_{\mathcal{I}}(B)$ to denote the above polytope.

Our main theorem of the section is as follows:

Theorem 6.2. *There exists some constant $c > 0$ such that the following holds for every small enough $\epsilon > 0$ and infinitely many integers $n > 0$. There is an instance $\mathcal{I} = (J, p, r, d)$ of $1|r_j, d_j|\sum_j p_j U'_j$ with $n = |J|$ jobs and $T := \sum_{j\in J} p_j = \max_{j\in J} d_j$ such that the following holds.*

- *The optimum solution to $\mathcal{I}$ has cost at least $0.2T$.*
- $\mathrm{SA}(\mathcal{P}_{\mathcal{I}}(\epsilon T), \lfloor c\epsilon \log n\rfloor) \neq \emptyset$.

The remaining part of this section is to prove the theorem. Throughout, let $\epsilon > 0$ be a small enough constant as in the theorem statement. Let $L > 0$ be a large enough integer; we assume that $L + 1$ is an integer power of 2. We then define the instance $\mathcal{I} = (J, p, r, d)$ of $1|r_j, d_j|\sum_j p_j U'_j$ with $n := |J| = 2^{L+1} - 1$ and $T := \sum_{j\in J} p_j = (L + 1)2^L$.

The set of jobs J in $\mathcal{I}$ form a full binary tree of $L + 1$ levels, where each node corresponds to a job. The root of the tree is at level 0 and the leaves are at level L. So there are 2^ℓ jobs at level ℓ; all these jobs j have size $p_j = 2^{L-\ell}$; for every $\ell \in [0, L]$ and $k \in [2^\ell]$, the k-th job from the left-side in the ℓ-th level of the tree has release time $r_j = (L + 1)(k - 1)2^{L-\ell}$ and deadline $d_j = (L+1)k2^{L-\ell}$. So, the window size $d_j - r_j$ for every job j is exactly $(L+1)p_j$; the windows of all the jobs form a laminar family represented by the tree structure. It is easy to verify that the number n of jobs is $2^{L+1} - 1$ and the total size of the jobs is $T = (L + 1)2^L$, which is equal to $\max_{j\in J} d_j$. See Figure 3 for the illustration of the instance.

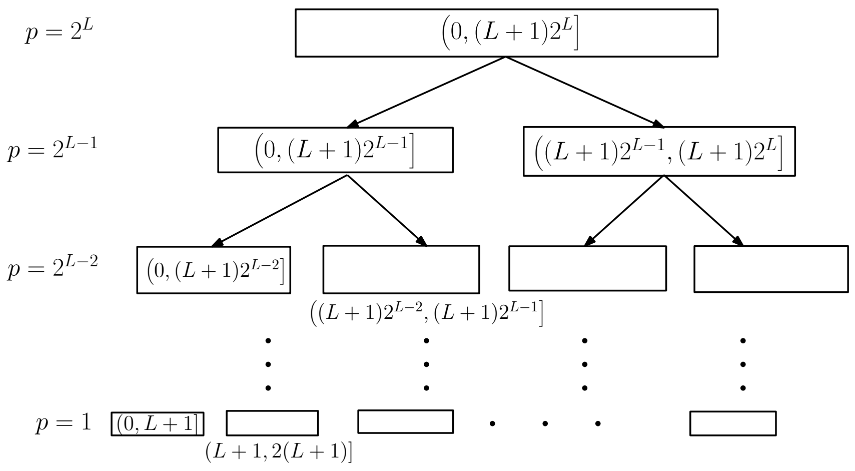
$p = 2^L$
$\big(0, (L+1)2^L\big]$
$p = 2^{L-1}$
$\big(0, (L+1)2^{L-1}\big]$
$\big((L+1)2^{L-1}, (L+1)2^L\big]$
$p = 2^{L-2}$
$\big(0, (L+1)2^{L-2}\big]$
$\big((L+1)2^{L-2}, (L+1)2^{L-1}\big]$
$p = 1$
$(0, L+1]$
$(L+1, 2(L+1)]$

Figure 3: The gap instance for $1|r_j, d_j|\sum_j p_j U'_j$. The jobs form a tree; the sizes of jobs are $2^L, 2^{L-1}, 2^{L-2}, \cdots, 1$ from top to bottom of the tree. The windows of jobs form a laminar family.

First we show that the optimum solution to the $1|r_j, d_j|\sum_j p_j U'_j$ instance $\mathcal{I} = (J, p, r, d)$ is large.

Lemma 6.3. *Any valid solution to the instance $\mathcal{I}$ of $1|r_j, d_j|\sum_j p_j U'_j$ has cost at least $0.2T$.*

Proof. We choose some integer A much smaller than L, whose exact value will be decided later. We break the $L+1$ levels of the job tree into 3 classes: the top $(L+1)/2 - A$ levels, the middle

A levels and the bottom $(L+1)/2$ levels. Roughly speaking, we shall assume that the jobs in the middle levels are processed for free and analyze the conflicts between top and bottom levels.

Focus on a bottom-level $\ell \in [(L+1)/2, L]$, and a top-level job j and assume it is scheduled in $(a_j, b_j]$ in the optimum solution. Let $a'_j \geq a_j$ be the smallest integer that is a multiple of $(L+1)2^{L-\ell}$ and $b'_j \leq b_j$ be the smallest integer that is a multiple of $(L+1)2^{L-\ell}$. Then we have that $(a'_j, b'_j]$ is the disjoint union of windows of some level-ℓ jobs. So, these jobs at level ℓ can not be processed at all in the optimum solution; we call these jobs forbidden jobs. Taking all top-level jobs j into consideration (notice that they have disjoint scheduling intervals in the optimum solution), the total number of level-ℓ forbidden jobs is at least:

$$\sum_{j \text{ top job}} \frac{(b_j - a_j - 2\cdot(L+1)2^{L-\ell})}{(L+1)2^{L-\ell}} = \frac{1}{(L+1)2^{L-\ell}} \sum_{j \text{ top job}} (b_j - a_j) - 2(\#\text{top jobs}).$$

Let $P_{\text{top}} = \sum_{j \text{ top job}}(b_j - a_j)$; notice that this is the total job units processed for top jobs. The total number of top jobs is $2^{(L+1)/2-A} - 1 < 2^{(L+1)/2-A}$. Since each level-$\ell$ job has size $2^{L-\ell}$, the total size of level-ℓ forbidden jobs is at least

$$\frac{P_{\text{top}}}{L+1} - 2^{(L+1)/2-A}\cdot 2^{L-\ell} \geq \frac{P_{\text{top}}}{L+1} - 2^{L-A} = \frac{P_{\text{top}}}{L+1} - \frac{T}{(L+1)2^A}.$$

The total size of forbidden jobs at all bottom levels is at least $(L+1)/2$ times the above quantity, which is $\frac{P_{\text{top}}}{2} - \frac{T}{2^{A+1}}$.

Let P_{bot} be the total job units processed for bottom jobs in the optimal solution. Then, we have

$$P_{\text{bot}} \leq \frac{T}{2} - \frac{P_{\text{top}}}{2} + \frac{T}{2^{A+1}}.$$

So, we have

$$P_{\text{top}} + P_{\text{bot}} = P_{\text{bot}} + \frac{P_{\text{top}}}{2} + \frac{P_{\text{top}}}{2} \leq \frac{T}{2} + \frac{T}{2^{A+1}} + \frac{1}{2}\cdot\frac{((L+1)/2 - A)T}{L+1} = \frac{3T}{4} + \frac{T}{2^{A+1}} - \frac{AT}{2(L+1)}.$$

Considering the middle level jobs, the total scheduled job units in the optimum solution is at most $\frac{3T}{4} + \frac{T}{2^{A+1}} + \frac{AT}{2(L+1)}$. Let $A = \log(L+1)$, then the scheduled jobs is at most $\frac{3T}{4} + O(\log L/L)T = \left(\frac{3}{4} + o(1)\right)T$. So, if L is large enough, the total scheduled job units is at most $0.8T$, finishing the proof of the lemma. □

Then we shall give an LP hierarchy solution to $\mathcal{I}$ with small cost. Recall that we are allowed to partially process a job; however our fractional solution does not need to take the advantage: for each job j, it either processes it completely, or does not process it at all. Moreover, the fractional solution only starts a job j at a time that is a multiple of p_j. Since $d_j - r_j = (L+1)p_j$, there are exactly $L+1$ possible starting times for a fixed job j. For simplicity, we use $x_{j,t}$ to indicate the event that j is processed in $(t, t+p_j]$. We let $\mathcal{D}$ denote the set of (j,t) pairs for which the variables can take positive value; that is $\mathcal{D} := \{(j,t) : j \in J, t \in [r_j, d_j) \text{ is a multiple of } p_j\}$. For convenience, we also treat each (j,t) pair as the interval $(t, t+p_j]$. Then, removing the variables that are identically 0, and replace B with ϵT, the LP (14)-(17) becomes

$$\sum_{(j,t)\in\mathcal{D}} x_{j,t}p_j \geq (1-\epsilon)T \qquad (18)$$

$$\sum_{t:(j,t)\in\mathcal{D}} x_{j,t} \leq 1 \qquad \forall j \in J \qquad (19)$$

$$\sum_{(j,t)\in\mathcal{D}: t' \in (t, t+p_j]} x_{j,t} \leq 1 \qquad \forall t' \in [T] \qquad (20)$$

$$x_{j,t} \geq 0 \qquad \forall (j,t) \in \mathcal{D} \qquad (21)$$

Let $\epsilon' = \epsilon/4$ let $q = \lfloor \epsilon'(L+1) \rfloor$ be the number of rounds we shall allow. Since $L = \Theta(\log n)$, we have $q \geq \lfloor c\epsilon \log n \rfloor$ if c is small enough. As $\mathcal{I}$ and $B = \epsilon T$ are fixed, we can use $\mathcal{P}'$ to denote the above polytope. Our goal is to prove that $\mathrm{SA}(\mathcal{P}_{\mathcal{I}}(\epsilon T), q) \neq \emptyset$ is not empty. Since from $\mathcal{P}'$ is obtained from $\mathcal{P}_{\mathcal{I}}(\epsilon T)$ by setting some variables to be 0, it suffices to prove $\mathrm{SA}(\mathcal{P}', q) \neq \emptyset$; this is our goal for remaining part of the section.

For a 1-round solution, we can simply set $x_{j,t} = 1/(L+1)$ for every $(j,t) \in \mathcal{D}$. Notice that there are $L+1$ levels of jobs and each job has $L+1$ possible intervals, the LP solution is valid and has cost 0. We show that with a small loss, we can raise the solution by q levels using the Sherali-Adams hierarchy.

Now, we define the solution in $\mathrm{SA}(\mathcal{P}', q)$. For a set $S \subseteq \mathcal{D}$, $|S| \leq r$ of variables, we set $x_S = \left(\frac{1-\epsilon'}{L+1}\right)^{|S|}$ if S does not lead to a contradiction and 0 otherwise. Here, S leads to a contradiction iff either some $j \in J$ appears in more than one pair in S, or for two distinct pairs $(j,t), (j',t') \in S$, $(t, t+p_j]$ and $(t', t'+p_{j'}]$ overlap.

We consider the constraints (18)-(20) one by one, and prove that their respective induced constraints in the LP hierarchy are satisfied. First consider (19); we need to prove

$$\sum_{R' \subseteq R, t:(j,t) \in \mathcal{D}} (-1)^{|R'|} x_{S \cup R' \cup \{(j,t)\}} \leq \sum_{R' \subseteq R} (-1)^{|R'|} x_{S \cup R'}, \qquad \forall S, R \subseteq \mathcal{D} \text{ with } |S| + |R| \leq r, j \in J, \tag{22}$$

We can assume S does not lead to a contradiction. If j appears in S, then t has to be the value satisfying $(j,t) \in S$ to make sure $x_{S \cup R' \cup \{(j,t)\}} \neq 0$. Then (22) holds with equality. So, we assume j does not appear in S. We can also assume that R and S are disjoint; otherwise, both sides of (22) are 0. For the fixed S, R and j, (22) is equivalent to

$$\sum_{R' \subseteq R} (-1)^{|R'|} \left(x_{S \cup R'} - \sum_{t:(j,t) \in D} x_{S \cup R' \cup \{(j,t)\}} \right) \geq 0.$$

Consider any $R' \subseteq R$. If $S \cup R'$ leads to a contradiction, or j appears in R', then it is easy to see that $x_{S \cup R'} - \sum_t x_{S \cup R' \cup \{(j,t)\}} = 0$. Otherwise, we have $x_{S \cup R'} - \sum_t x_{S \cup R' \cup \{(j,t)\}} \geq \epsilon' x_{S \cup R'}$. This holds since $x_{S \cup R'} = \left(\frac{1-\epsilon'}{L+1}\right)^{|S \cup R'|}$ and for each relevant t, we have $x_{S \cup R' \cup \{(j,t)\}}$ is either 0 or $\left(\frac{1-\epsilon'}{L+1}\right)^{|S \cup R'|+1}$.

Let $*$ be the family of subsets R' of R such that $S \cup R'$ does not lead to a contradiction and j does not appear in R'. Then, in order to prove (22), it suffices to show the following:

$$\sum_{R' \in *, |R'| \text{ even}} \epsilon' x_{S \cup R'} - \sum_{R'' \in *, |R''| \text{ odd}} x_{S \cup R''} \geq 0. \tag{23}$$

Notice that we assumed that S and R are disjoint. For every $R' \in *$ of even size, the $\epsilon' x_{S \cup R'}$ budget can be used to cover the negative sum $\sum_{R'' \in *: R' \subseteq R'', |R''| = |R'|+1} x_{S \cup R''}$. This holds since for every R'' in the summation, we have $x_{S \cup R''} = \frac{(1-\epsilon') x_{S \cup R'}}{L+1}$. Since there are at most $|R| \leq r \leq \epsilon'(L+1)$ terms in the summation,the budget $x_{S \cup R'}$ is at least the sum. Also, notice that each odd set $R'' \in *$ is covered at least once. So, we have proved (23), which implies (22).

Now we turn to (20). We need to show

$$\sum_{(j,t) \in \mathcal{D}: t' \in (t, t+p_j]} \sum_{R' \subseteq R} (-1)^{|R'|} x_{S \cup R' \cup \{(j,t)\}} \leq \sum_{R' \subseteq R} (-1)^{|R'|} x_{S \cup R'},$$
$$\forall S, R \subseteq \mathcal{D} \text{ with } |S| + |R| \leq r, t' \in [T]. \tag{24}$$

This can be proved similarly as (22). First, we can assume that S does not lead to a contradiction and the intervals in S do not cover t'. Then, we can prove that $x_{S\cup R'} - \sum_{(j,t) \text{ covers } t'} x_{S\cup R'\cup\{(j,t)\}}$ is 0 if $S \cup R'$ leads to a contradiction or intervals in $S\cup R'$ cover t'. Otherwise, the quantity is at least $\epsilon' x_{S\cup R'}$. We can similarly define $*$ to be the family of subsets R' of R for which we have the latter case. Then using the same way we can prove (23), which implies (24).

Finally we consider (18), the constraint for the objective value. By reorganizing the terms, we need to prove that for every $S, R \subseteq \mathcal{D}$ with $|S| + |R| \le r$,

$$\sum_{R'\subseteq R} (-1)^{|R'|} \left(\sum_{(j,t)\in\mathcal{D}} x_{S\cup R'\cup\{(j,t)\}} \cdot p_j - (1-\epsilon)T \cdot x_{S\cup R'} \right) \ge 0 \tag{25}$$

Let us focus on some $R' \subseteq R$ such that $S \cup R'$ does not lead to a contradiction. Define $\mathcal{D}'$ be the set of (j,t) pairs in $\mathcal{D} \setminus (S\cup R')$ such that $S \cup R' \cup (j,t)$ does not lead to a contradiction. Then

$$Q := \sum_{(j,t)\in\mathcal{D}} x_{S\cup R'\cup\{(j,t)\}} \cdot p_j = x_{S\cup R'} \left(\sum_{(j,t)\in S\cup R'} p_j + \frac{1-\epsilon'}{L+1} \sum_{(j,t)\in\mathcal{D}'} p_j \right).$$

We are interested in upper and lower bounds of Q. For the upper bound , we have

$$Q \le x_{S\cup R'} \left(q \cdot \frac{T}{L+1} + \frac{1-\epsilon'}{L+1} \cdot T(L+1) \right) \le x_{S\cup R'} \big(\epsilon' T + (1-\epsilon')T\big) = x_{S\cup R'} T.$$

Above, we used that $|S\cup R'| \le q \le \epsilon'(L+1)$, every $j \in J$ has $p_j \le \frac{T}{L+1}$, $\sum_{j\in J} p_j = T$ and every j appears in $\mathcal{D}$ exactly $L+1$ times.

For the lower bound of Q, we lower bound the term $\sum_{(j,t)\in\mathcal{D}'} p_j$. This is at least

$$(L+1)T - \sum_{(j,t)\in\mathcal{D}: j \text{ is in some pair in } S\cup R'} p_j - \sum_{(j,t)\in\mathcal{D} \text{ intersects some interval in } S\cup R'} p_j.$$

We upper bound the subtrahends one by one. The first subtrahend is at most $q(L+1)\cdot\frac{T}{L+1} \le \epsilon'(L+1)T$. The second term is maximized when $S\cup R'$ contains q disjoint intervals of length $T/(L+1)$, with boundaries being multiply of $T/(L+1)$. (Recall that we assumed $L+1$ is a power of 2; this does not correspond to an actual $S \cup R'$ since we only have 1 job of length $T/(L+1)$; but it will give an upper bound.) In this case, for any job length 2^ℓ, we have $\Pr_{(j,t)\sim\mathcal{D}}\left[(j,t) \text{ intersects a interval in } S\cup R' | p_j = 2^\ell\right] = q/(L+1) \le \epsilon'$. So, the second subtrahend is at most $\epsilon'(L+1)T$. Overall, we have $\sum_{(j,t)\in\mathcal{D}'} p_j \ge (L+1)T - 2\epsilon'(L+1)T = (1-2\epsilon')(L+1)T$. This implies $Q \ge x_{S\cup R'} \cdot \frac{1-\epsilon'}{L+1}(1-2\epsilon')(L+1)T \ge (1-3\epsilon')x_{S\cup R'}T$.

With the upper and lower bounds, we can prove (25). Let $*$ be the family of subsets R' of R such that $S\cup R'$ does not lead to a contradiction. Then, the left side of (25) is at least

$$\sum_{R'\in *:|R'| \text{ even}} \big((1-3\epsilon')x_{S\cup R'}T - (1-\epsilon)x_{S\cup R'}T\big) - \sum_{R'\in *:|R'| \text{ odd}} \big(x_{S\cup R'}T - (1-\epsilon)x_{S\cup R'}T\big)$$
$$= \epsilon' T \sum_{R\in *:|R'|\text{even}} x_{S\cup R'} - \epsilon T \sum_{R\in *:|R'|\text{odd}} x_{S\cup R'},$$

where we used that $\epsilon' = \epsilon/4$. Using the same covering idea as before and that $q \le (L+1)/4$, we can prove the quantity is at least 0. This implies that (25) holds, which finishes the proof of Theorem 6.2.

6.3 Discussion

We showed that an $o(\log n)$-level Sherali-Adams lift of a natural LP relaxation for $1|r_j, d_j|\sum_j p_j U'_j$ can not distinguish between whether an instance $\mathcal{I}$ has cost at most ϵT or at least $0.2T$. The instance $\mathcal{I}'$ of $P2|\text{prec}|C_{\max}$ constructed from $\mathcal{I}$ will have makespan at most $(1+\epsilon)T$ and at least $1.1T$ for the two cases.

One could ask if the natural LP relaxation for $P2|\text{prec}|C_{\max}$ on the instance $\mathcal{I}'$ has large intergrality gap when raised to $o(\log n)$ levels. Unfortunately we could not prove such a result. It is known that some small modifications with no effect on the basic LP can change the feasibles solutions in the hierarchy. For example, we can introduce new variables $x_{j,\text{bot}}$ in LP (14-17) to indicate whether j is not scheduled, and require $x_{j,\text{bot}} = 1 - \sum_{t,p'} x_{j,t,p'}$. This does not change the LP, but we need to consider the new variables when deriving the constraints in the lifted LP. We do not know how to give a fractional solution for this new lifted LP. This seems to be a barrier to extend the negative result to $P2|\text{prec}|C_{\max}$, as in the problem, we do require every job to be processed to an extent of 1. Nevertheless, we believe the instance we constructed is the right one for proving an integrality gap result for $P2|\text{prec}|C_{\max}$.

We also remark that in the gap instance for $1|r_j, d_j|\sum_j p_j U'_j$, the windows of all jobs form a laminar tree of depth $O(\log n)$. Such an instance can be solved efficiently and exactly using dynamic programming. Thus for this problem, the Sherali-Adams hierarchy does not capture the dynamic programming idea using a small number of rounds. Moreover, using the dynamic programming technique in [19], one may obtain a QPTAS for $1|r_j, d_j|\sum_j p_j U'_j$. This suggests that our reduction from $1|r_j, d_j|\sum_j p_j U'_j$ to $P2|\text{prec}|C_{\max}$ is unlikely to give APX-hardness for the latter problem, if such a result indeed holds.

Acknowledgments

We thank Shashwat Garg for several helpful discussions on the topic.

References

[1] Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallel DAG jobs online to minimize average flow time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 176–189, 2016.

[2] Nikhil Bansal. Scheduling open problems: Old and new. MAPSP 2017. http://www.mapsp2017.ma.tum.de/MAPSP2017-Bansal.pdf, 2017.

[3] Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 453–462. IEEE Computer Society, 2009.

[4] Soumen Chakrabarti, Cynthia A. Phillips, Andreas S. Schulz, David B. Shmoys, Cliff Stein, and Joel Wein. *Improved scheduling algorithms for minsum criteria*, pages 646–657. Springer Berlin Heidelberg, 1996.

[5] C. Chekuri and S. Khanna. Approximation algorithms for minimizing average weighted completion time. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Inc., Boca Raton, FL, USA*, 2004.

[6] Fabián A. Chudak and David B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, pages 581–590. Society for Industrial and Applied Mathematics, 1997.

[7] Dryad. https://www.microsoft.com/en-us/research/project/dryad/.

[8] Devdatta Gangal and Abhiram Ranade. Precedence constrained scheduling in $2 - 7/3p + 1$ optimal. *Journal of Computer and System Sciences*, 74(7):1139 – 1146, 2008.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[10] Shashwat Garg. Quasi-ptas for scheduling with precedences using LP hierarchies. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 59:1–59:13, 2018.

[11] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM JOURNAL ON APPLIED MATHEMATICS*, 17(2):416–429, 1969.

[12] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, 4:287–326, 1979.

[13] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 81–97, 2016.

[14] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22(3):513–544, August 1997.

[15] Claire Hanen and Alix Munier. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. *Discrete Applied Mathematics*, 108:239–257, 2001.

[16] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, and Phillip B. Gibbons. Pipedream: Fast and efficient pipeline parallel DNN training. *CoRR*, abs/1806.03377, 2018.

[17] Han Hoogeveen, Petra Schuurman, and Gerhard J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. In *Proceedings of the 6th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 353–366. Springer-Verlag, 1998.

[18] J.A. Hoogeveen, J.K. Lenstra, and B. Veltman. Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, 16(3):129 – 137, 1994.

[19] Sungjin Im, Shi Li, Benjamin Moseley, and Eric Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines: Extended abstract. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1070–1086, Philadelphia, PA, USA, 2015. Society for Industrial and Applied Mathematics.

[20] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. *CoRR*, abs/1901.05758, 2019.

[21] Janardhan Kulkarni. Scheduling jobs with dependencies: New applications, classic problems. In *https://www.cs.umd.edu/%7Esamir/DCscheduling18/ slides/Janardhan%20Kulkarni.pdf.*

[22] Jean B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[23] M. Laurent. A comparison of the sherali-adams, lovsz-schrijver and lasserre relaxations for 0-1 programming. *Mathematics of Operations Research*, 28:470–496, 2001.

[24] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26(1):22–35, February 1978.

[25] Elaine Levey and Thomas Rothvoss. A (1+epsilon)-approximation for makespan scheduling with precedence constraints using LP hierarchies. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 168–177. ACM, 2016.

[26] L. Lovasz and A. Schrijver. Cones of matrices and set-functions and 01 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.

[27] Alix Munier and Jean-Claude Knig. A heuristic for a scheduling problem with communication delays. *Operations Research*, 45(1):145–147, 1997.

[28] Alix Munier, Maurice Queyranne, and Andreas S. Schulz. *Approximation Bounds for a General Class of Precedence Constrained Parallel Machine Scheduling Problems*, pages 367–382. Springer Berlin Heidelberg, 1998.

[29] Christos Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 510–513, New York, NY, USA, 1988. ACM.

[30] Maurice Queyranne and Maxim Sviridenko. Approximation algorithms for shop scheduling problems with minsum objective. *Journal of Scheduling*, 5(4):287–305, 2002.

[31] Thomas Rothvoss. The lasserre hierarchy in approximation algorithms, 2013.

[32] Thomas Rothvoss. The lasserre hierarchy in approximation algorithms lecture notes for the mapsp 2013 tutorial preliminary version. 2013.

[33] Petra Schuurman and Gerhard J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems, 1999.

[34] Hanif Sherali and Warren P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. 3:411–430, 05 1990.

[35] Martin Skutella. A 2.542-approximation for precedence constrained single machine scheduling with release dates and total weighted completion time objective. *Operations Research Letters*, 44(5):676 – 679, 2016.

[36] SparkSQL. https://spark.apache.org/sql/.

[37] Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 745–754. ACM, 2010.

[38] Data Center Scheduling: From Theory to Practice. https://www.cs.umd.edu/%7Esamir/DCscheduling18/.

[39] J. D. Ullman. Np-complete scheduling problems. *J. Comput. Syst. Sci.*, 10(3):384–393, June 1975.

[40] Bart Veltman, BJ Lageweg, and Jan Karel Lenstra. Multiprocessor scheduling with communication delays. *Parallel computing*, 16(2-3):173–182, 1990.

[41] Zhicheng Yin, Jin Sun, Ming Li, Jaliya Ekanayake, Haibo Lin, Marc Friedman, José A. Blakeley, Clemens A. Szyperski, and Nikhil R. Devanur. Bubble execution: Resource-aware reliable analytics at cloud scale. *PVLDB*, 11(7):746–758, 2018.

[42] Hongyu Zhu, Mohamed Akrout, Bojian Zheng, Andrew Pelegris, Anand Jayarajan, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko. Benchmarking and analyzing deep neural network training. In *2018 IEEE International Symposium on Workload Characterization, IISWC 2018, Raleigh, NC, USA, September 30 - October 2, 2018*, pages 88–100, 2018.

A Comparison of Three Optimal Schedules

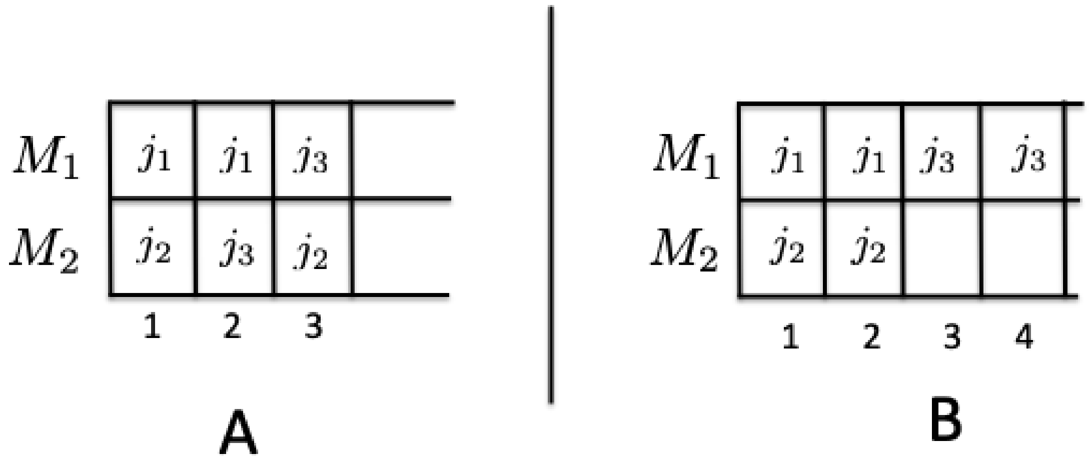

Figure 4: The figure illustrates a gap of at least 4/3 between the optimal schedules for A and B.

Recall that when jobs have arbitrary processing lengths, the optimal schedule for minimizing makespan with precedence constraints can be of three types:

A) Fully preemptive ($Pm|\text{prec, pmtn, migration}|C_{\max}$);
B) Preemptive but non-migratory ($Pm|\text{prec, pmtn}|C_{\max}$);
C) Non-preemptive ($Pm|\text{prec}|C_{\max}$).

Clearly, the value of optimal makespan for A is at most B which is at most C. Here we give instances to prove that these three schedules can be constant factor away from each other, and hence ruling out a black-box approach to the design of $(1+\epsilon)$-approximation algorithms for these problems.

A.1 Gap Between A and B

Refer to Figure 4. In the instance we have 3 jobs each of length 2, and no precedence constraints. The optimal schedules for A and B are shown in the figure. There is a gap of 4/3 between A and B. We can generalize the instance to have m machines and $m+1$ jobs of length m. Then model A can achieve makespan $m+1$, whereas model B needs to have makespan $2m$. Thus the gap can be made arbitrarily close to 2. This also gives a gap close to 2 between A and C, since C is more restricted than B.

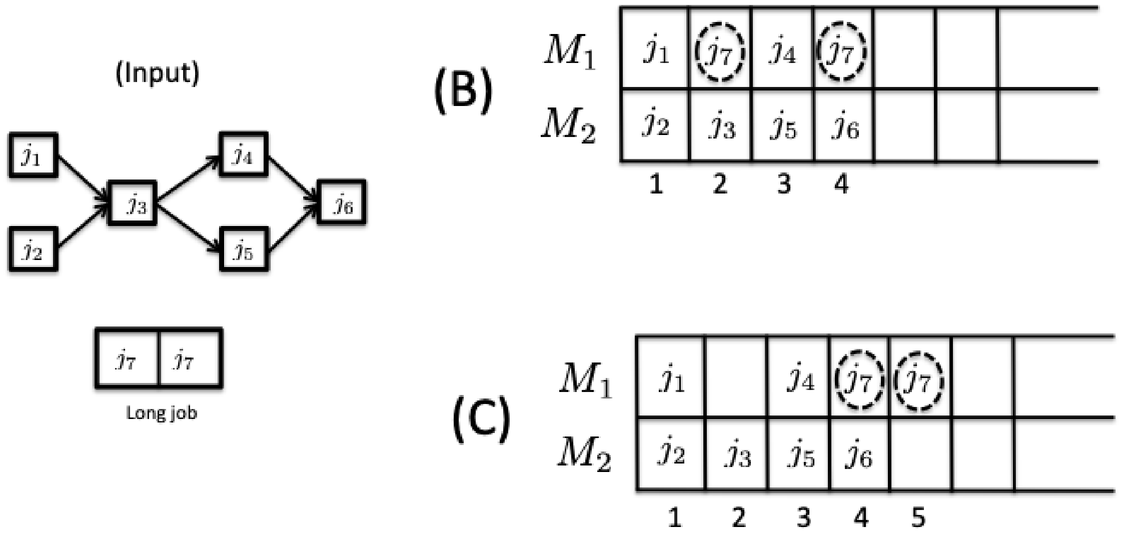
(Input)
j_1
j_2
j_3
j_4
j_5
j_6
j_7 j_7
Long job
(B)
M_1 j_1 j_7 j_4 j_7
M_2 j_2 j_3 j_5 j_6
1 2 3 4
(C)
M_1 j_1 j_4 j_7 j_7
M_2 j_2 j_3 j_5 j_6
1 2 3 4 5

Figure 5: The figure illustrates a gap of at least 5/4 between the optimal schedules B and C.

A.2 Gap Between B and C

Refer to Figure 5. In the instance we have 7 jobs. The first six jobs have unit processing lengths, and the precedence relationship is as shown in the figure. The job 7 is long and has processing length of 2. The optimal schedules for B and C are shown in the figure. By making the DAG contain $3n$ unit length jobs, and the processing length of long job as n, we can make the gap approach 1.5.

B Removing the Polynomial Size Assumption on Job Lengths

We give a brief sketch of how to handle the case when p_j's are not polynomially bounded. Let $p_{\max} = \max_j p_j$. We can round each job size down to the nearest multiple of $\epsilon p_{\max}/n$; in particular, if there is a job j such that $p_j < \epsilon p_{max}/n$, we discard it. Thus, the total size of jobs we discarded is at most ϵp_{max}. It is easily seen that the optimum value must be at least p_{max} and hence the total size we discarded is at most ϵ times the optimum makespan. Scaling down all job sizes by a factor of $\epsilon p_{\max}/n$, we obtain an instance where all job sizes are integers between 1 and n/ϵ. Hence, we can assume that $\sum_j p_j$ is polynomial in n.