Scheduling Precedence-Constrained Jobs on Related Machines with Communication Delay

Biswaroop Maiti* Rajmohan Rajaraman† David Stalfa‡ Zoya Svitkina§
Aravindan Vijayaraghavan¶

Abstract

We consider the problem of scheduling n precedence-constrained jobs on m uniformly-related machines in the presence of an arbitrary, fixed communication delay ρ. Communication delay is the amount of time that must pass between the completion of a job on one machine and the start of any successor of that job on a different machine. We consider a model that allows job duplication, i.e. processing of the same job on multiple machines, which, as we show, can reduce the length of a schedule (i.e., its makespan) by a logarithmic factor. Our main result is an $O(\log m \log \rho / \log\log \rho)$-approximation algorithm for minimizing makespan, assuming the minimum makespan is at least ρ. Our algorithm is based on rounding a linear programming relaxation for the problem, which includes carefully designed constraints capturing the interaction among communication delay, precedence requirements, varying speeds, and job duplication. To derive a schedule from a solution to the linear program, we balance the benefits of duplication in satisfying precedence constraints early against its drawbacks in increasing overall system load. Our result builds on two previous lines of work, one with communication delay but identical machines [20], and the other with uniformly-related machines but no communication delay [10, 21].

We next show that the integrality gap of our mathematical program is $\Omega(\sqrt{\log \rho})$. Our gap construction employs expander graphs and exploits a property of robust expansion and its generalization to paths of longer length, which may be of independent interest. Finally, we quantify the advantage of duplication in scheduling with communication delay. We show that the best schedule without duplication can have makespan $\Omega(\rho/\log \rho)$ or $\Omega(\log m/\log\log m)$ or $\Omega(\log n/\log\log n)$ times that of an optimal schedule allowing duplication. Nevertheless, we present a polynomial time algorithm to transform any schedule to a schedule without duplication at the cost of a $O(\log^2 n \log m)$ factor increase in makespan. Together with our makespan approximation algorithm for schedules allowing duplication, this also yields a polylogarithmic-approximation algorithm for the setting where duplication is not allowed.

*Northeastern University, Boston, MA, USA. Email: m.biswaroop@gmail.com
†Northeastern University, Boston, MA, USA. Email: rraj@ccs.neu.edu
‡Northeastern University, Boston, MA, USA. Email: stalfa@ccis.neu.edu
§Google Research, Mountain View, CA, USA. Email: zoya@google.com
¶Northwestern University, Evanston, IL, USA. Email: aravindv@northwestern.edu

1 Introduction

As computational workloads get larger and more complex, it becomes necessary to distribute tasks across multiple heterogenous processors. For example, the process of training and evaluating neural network models is often distributed over diverse devices such as CPUs, GPUs, or other specialized hardware; this process, commonly referred to as *device placement* has gained significant interest [24, 12, 23]. This gives rise to a multiprocessor scheduling problem of optimizing both the assignment of tasks to processors and the order of their execution. We address this problem, taking into account several complications that such a distributed setting presents, including job dependencies, heterogeneous machine speeds, and a communication delay between them.

The jobs comprising a workload can have data dependencies between them, where the output of one job serves as the input to another. As is common in scheduling literature, we model these dependencies using a directed acyclic graph (DAG), where a directed edge uv represents that job u must be scheduled before v. However, if these two jobs are executed on different machines, additional time is needed to transfer the data from one machine to the other. We model this time as a communication delay: this delay is zero if the two jobs run on the same machine, and is equal to some value ρ if they run on different machines. Considering that the communication delay can be substantial, another aspect of the problem comes into play. Instead of a machine waiting for the result of some computation to be communicated from another machine, it may be advantageous for it to perform this computation itself, thus *duplicating* work in order to obtain the result sooner (as highlighted in early work [29]). Indeed, the technique of duplication to hide latency has been incorporated in schedulers proposed for grid computing and cloud environments [7, 8, 16, 35]. In addition, jobs may have different processing sizes and the devices may run at different speeds, representing either different types (e.g. CPU, GPU, or TPU), or differences in machine model.

Optimization problems associated with scheduling under communication delays have been studied over the last three decades, but provably good approximation bounds are few and several challenging open problems remain [1, 3, 11, 15, 20, 26, 25, 28, 29, 31, 33]. It is known that scheduling a DAG of uniform size jobs on identical machines with a communication delay is NP-hard, even when the number of machines is infinite [33, 31]. Several inapproximability results have also been derived [3, 15]. However, these results are very limited and the approximability status of scheduling under communication delay is listed as one of the top ten open problems in scheduling surveys [4, 34]. For the special case of uniform speeds and unit jobs, a logarithmic-approximation algorithm is presented in [20]. In recent work [18], a quasi-polynomial time approximation scheme is developed for the problem when the number of machines is $O(1)$, communication delays are $O(1)$, and the machines are identical. Our focus in this paper is on deriving approximation algorithms for scheduling a DAG with *non-uniform size* jobs on an *arbitrary* number of *related machines* (arbitrary speeds) and an *arbitrary communication delay*.

1.1 Our results and techniques

We study the problem of scheduling a DAG with n jobs of arbitrary sizes on m *related machines*, connected by a network with a fixed communication delay. In the related machines model, machine i has a speed s_i, and the time taken to complete a job v of size p_v on i is given by p_v/s_i. We represent the network communication delay as ρ times the processing time of the smallest job on the fastest machine.

Approximation algorithm for makespan. We focus on the *makespan* objective, which is defined as the time taken by a given schedule to complete the given DAG on the machines. We consider scheduling policies that allow duplication of jobs, which, as we discuss below, can reduce makespan when compared to schedules that do not allow duplication.

Theorem 1 (Makespan approximation)**.** There is a polynomial time algorithm that, given an instance of DAG scheduling with fixed communication delay, computes a schedule whose makespan is $O(\log m \log \rho / \log\log \rho)(OPT + \rho)$, where OPT is the optimal makespan for the given instance.

We thus obtain an $O(\log m \log \rho / \log\log \rho)$-approximation algorithm as long as $OPT \geq \rho$, which is a natural requirement since it takes ρ time to distribute the jobs to the machines at the start of the schedule as well as to synchronize termination at the end of the schedule. We note that the $\log m$ factor in our approximation corresponds to an upper bound on the number of geometrically separated speed groups. This entails that, for the special case of uniform speeds, our algorithm constructs a schedule with makespan upper bounded by $O(\log \rho / \log\log \rho)(OPT + \rho)$, thus extending the result of [20] to non-uniform job sizes.

A central component of our algorithm is a linear programming relaxation. A significant challenge in this regard is to capture the precedence requirement in the presence of communication delays: we would like to determine where and when to schedule individual jobs while, at the same time, adjusting the start time of each job to account for communication delays in relation to *all* its predecessors. We consider several related LPs and their natural extensions and show that these approaches are inadequate for our algorithm (see Appendix C). To overcome these challenges, we introduce a set of variables that indicate whether a job and its predecessor are scheduled within ρ time of each other, and incorporate these variables into two new sets of constraints. The first enforces the delay requirement on jobs that do not start within ρ time of each other, and the second upper bounds the total size of all predecessors that can be executed within ρ time of their successor. The addition of these constraints exponentially reduces the integrality gap of our program.

Our rounding algorithm has two components. First, we process a fractional solution to our linear relaxation to determine a tentative assignment of jobs to groups of machines, along the lines of [10]. Next, we convert the group assignment to an actual schedule. Unlike in the case of related machines with no communication delay, we cannot invoke a list scheduling type of policy. Furthermore, our algorithm needs to duplicate jobs judiciously so as to hide the communication latency and achieve the desired approximation ratio. The main challenge in this regard is that, in order to make sufficient progress on the LP solution, we must duplicate some jobs on machines much slower than their assigned machine. We overcome this obstacle by upper bounding the total size of any duplicated jobs and structuring the machines such that those with slower speed have, as a whole, higher capacity.

Integrality gap. We next study the integrality gap of the linear program underlying our approximation algorithm, and its dependence on the communication delay ρ. Previous work of [10] on scheduling on related machines implies an integrality gap of $\Omega(\log m / \log\log m)$ for non-uniform speeds and non-uniform job sizes, but it does not consider communication delays and hence does not yield any gap in terms of ρ.

Theorem 2 (Integrality gap)**.** There is a family of instances with uniform speeds and uniform job sizes such that for any ρ that is at least some sufficiently large constant, our linear programming relaxation has a gap of at least $\Omega(\sqrt{\log \rho})$.

This integrality gap gives the first evidence that constant factor approximations may not be tractable or may be out of reach of existing techniques when the communication delay ρ is super-constant, even with uniform job sizes and identical machines. The integrality gap also extends to variants of time-indexed linear programs and, we suspect, to a wider class of mathematical programming relaxations. Given that without communication delay, the unit speed and unit job size case has an integrality gap of at most 2 by Graham's list scheduling [13], our result suggests a separation in the approximability between the variants of precedence-constrained scheduling with and without communication delays.

Our gap construction consists of a layered DAG with $L = \omega(1)$ layers, where the dependency graph between successive layers corresponds to a random graph. The main technical challenge is to argue that $\Omega(L)$ phases (a phase here corresponds to roughly ρ time units) are needed in order to schedule all the jobs for the optimal integral solution. The expansion of the random graph implies that at most $o(1)$ fraction of the jobs can be scheduled in the first phase. However, in the next phase, the jobs that were completed previously are now available on all the machines; moreover, the remaining graph (on the unscheduled jobs) in subsequent phases is *not random* any longer! To overcome this technical hurdle, we identify and exploit a property of "robust expansion" and its generalization to paths of longer length, which may be of independent interest. Section 3.2 provides an overview of our integrality gap result, and Section 5 contains the full proof.

Bounding the duplication advantage. Given the potential of duplication to effectively hide communication latency, a natural question arises: how much smaller can the makespan of a schedule with duplications be, when compared to a *no-duplication* schedule, i.e., a schedule in which each job is processed exactly once? Our final set of results formally quantifies the *duplication advantage*.

Theorem 3 (Bounding the duplication advantage)**. Upper bound:** Given any instance with n jobs, m machines, communication delay ρ, and a schedule with makespan $C^* \geq \rho$, there exists a polynomial-time computable no-duplication schedule with makespan $O(C^* \cdot \log^2 n \log m)$. **Lower bound:** There exists an instance with $n/2 = m = 2^\rho$ for which any no-duplication schedule has makespan at least $\rho/\log \rho$ times the optimal makespan.

Together with our makespan algorithm for general schedules, the algorithm of the above theorem yields a *polylogarithmic approximation makespan algorithm for no-duplication schedules*. Note that the preceding approximation ratio holds even when the makespan of an optimal no-duplication schedule is less than ρ since it is straightforward to determine whether there is a no-duplication schedule that completes all jobs in less than ρ time without any communication. Section 3.3 gives an overview of our algorithm that transforms a general schedule to a no-duplication schedule, and Section 6 contains the full proofs for bounding the duplication advantage.

1.2 Related work

Scheduling theory has a rich history and there is extensive work on scheduling jobs with precedence constraints dating back to over three decades. In the following, we review scheduling work most closely related to this paper: scheduling DAGs on related machines, and scheduling DAGs under communication delays.

Scheduling DAGs on related machines. The problem of scheduling DAGs on related machines (with no communication delays) to minimize weighted completion time was first studied by Jaffe, who gave an $O(\sqrt{m})$ approximation algorithm [17]. This was significantly improved by Chudak and Shmoys who first derived an $O(\log m)$ asymptotic approximation ratio for minimizing makespan [10] and then invoked a general framework due to Hall et al [14] and Queyranne and Sviridenko [32] to convert an approximation algorithm for makespan to an approximation algorithm for weighted completion time. The Chudak-Shmoys algorithm for makespan minimization first solves an LP relaxation for the problem, and then assigns each job to a group of machines whose speeds are within a factor of two of one another. Using Graham's list scheduling [13], they then schedule the jobs within each group of machines. The $O(\log m)$ factor arises due to the number of machine groups. In subsequent work, Chekuri and Bender derived the same $O(\log m)$ approximation via a combinatorial algorithm [9]. In recent work, Shi Li improved the approximation ratio

to $O(\log m/\log\log m)$ by a more careful tradeoff between the factor lost for organizing the machines into groups and the factor lost while assigning jobs to machine groups [21].

With regard to hardness, it is known that the problem is hard to approximate to within a constant factor even for the special case of identical machines, where the particular constant depends on underlying complexity theory assumptions [19, 5, 36]. Recent work has also shown that the problem is hard to approximate to within any constant assuming the hardness of a particular optimization problem on k-partite graphs [6].

Scheduling under communication delays. As discussed above, optimization problems associated with scheduling under communication delays have been studied for three decades since the early work of [33, 29, 37], but provably good approximation bounds are few. All previous work assumes uniform machines and either uniform job sizes or special cases such as $O(1)$ machines and $O(1)$ communication delay. For instance, in the special case of unit-size jobs, identical machines, and unit communication delay, a 7/3-approximation is presented in [25], while [15] show that it is NP-hard to approximate better than a factor of 5/4. Hardness results are also shown in [3, 31, 33]. To the best of our knowledge, our work is the first to develop algorithms for scheduling non-uniform jobs with precedence constraints on related machines connected by an arbitrary communication network with fixed delay.

The natural idea of duplication to hide communication latency was first studied by Papadimitriou and Yannakakis, who proposed a 2-approximation algorithm for scheduling DAGs on an unbounded number of identical machines with a fixed communication delay [29]. Improved bounds for infinite machines have been given in [1, 11, 27, 28]. For the case of a bounded number of machines, [26, 25] give approximation algorithms under some special cases of either very small or very large communication delay or with the DAG restricted to be a tree-precedence graph. The only provable guarantee for a bounded number of machines with an arbitrary communication delay parameter is the work of Lepere and Rapine, who present an approximation algorithm for scheduling a DAG of unit-size jobs on identical machines with communication delay of ρ units, which achieves a makespan $O((OPT+\rho)\log\rho/\log\log\rho)$ [20]. The recent work of [18] presents a novel quasi-polynomial time approximation scheme, based on the Sherali-Adams hierarchy framework, for the problem with $O(1)$ identical machines, non-uniform job sizes, and $O(1)$ communication delays.

2 Problem formulation and notation

An instance of precedence constrained scheduling with fixed communication delay is a triple (G, M, ρ) where G is a directed acyclic graph, M is a set of machines, and ρ is the communication delay. In the graph $G = (V, E)$, the n nodes of V represent jobs and the edges of E represent precedence constraints. Each job v has a *size* $p_v > 0$ and for any subset $U \subseteq V$, we define $p(U) = \sum_{u\in U} p_u$. In the set of machines $M = \{1, \ldots, m\}$, each machine $i \in M$ has a speed s_i. We order the machines such that $s_1 \leq s_2 \leq \ldots \leq s_m$. Processing a job v on a machine i takes p_v/s_i units of time. We normalize these values so that the shortest job has size 1 and the fastest machine has speed 1, in which case one time unit is defined as the time needed to process the shortest job on the fastest machine. Each job may be *duplicated*, i.e. copies of it processed on different machines. Preemption is not allowed, and at most one job can run on a machine at any given time.

We say that u is a *predecessor* of v, denoted $u \prec v$, if there is some (non-zero length) directed path from u to v in G. We denote the set of all predecessors of v by A_v (note that $v \notin A_v$). The parameter ρ specifies the time needed to communicate the result of a job computed on one machine to a different machine. So if $u \prec v$

m	number of machines	n	number of jobs
i, j	machines	v, u	jobs
s_i	speed of machine i	p_v	size of job v
ρ	communication delay	A_v	predecessors of v

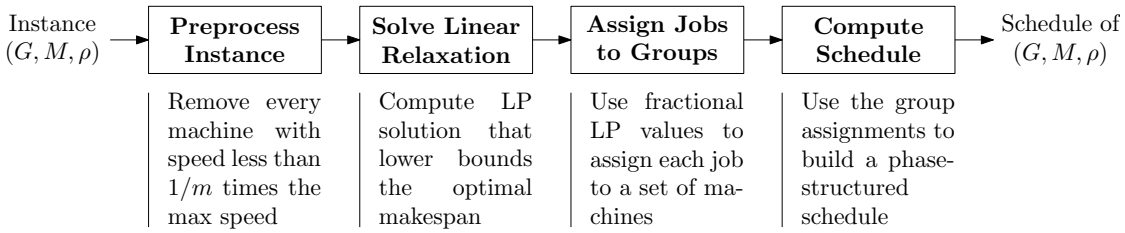
and v starts on machine i at time t, then there must be a copy of u that completes either on machine i by time t or on a different machine by time $t - \rho$.

We represent a schedule as a function $\sigma : V \times M \to \mathbb{R} \cup \{\infty\}$ mapping pair (v, i) to the start time of v on i, or to ∞ if v is not scheduled on i. We say that σ is a schedule of (G, M, ρ) if all jobs in G have a finite start time on some machine in M subject to the constraints listed above. The objective is to find a σ with minimum makespan, which is the maximum (finite) completion time in σ of any copy of any job. Since this objective is trivial if there is only one job or one machine, we assume $n, m \geq 2$. In the three field notation, this problem is denoted $Q|\text{duplication, prec}, c|C_{\max}$ where c indicates uniform communication delay.

3 Overview of the results and techniques

3.1 Approximation algorithm for makespan

At a high level, our algorithm finds a fractional solution to the scheduling problem and then, through a series of refinements, constructs a final schedule for the given instance. The various components of the algorithm are highlighted in the figure below. The first step is a standard preprocessing of the instance, in which we eliminate machines that are slower than the fastest machine by a factor of m or more, while incurring at most a constant factor increase in makespan. We refer the reader to Section 4 for details.



Approaches based on previous related work. We briefly review natural approaches to the problem of scheduling on related machines with communication delay, based on previous related work and indicate the ways in which these approaches are inadequate for our setting.

One approach is that taken in [20], which uses a combinatorial algorithm for the case with unit-speed machines, unit-size jobs, communication delay ρ, and duplication allowed. The crux of the algorithm is to repeatedly find jobs that can be completed in ρ steps and schedule them (duplicating their uncompleted predecessors, if necessary) until all such jobs have at least half their remaining predecessors already scheduled. At this point the algorithm introduces a delay on all machines, removes all previously scheduled jobs, and repeats. While this approach may work for arbitrary job sizes, accounting for variable speeds is more difficult. In Appendix C.1 we show that a natural extension of this combinatorial algorithm fails.

A more effective approach is to develop a suitable LP relaxation. We consider natural variants of two relaxations developed in related work. The first captures precedence constraints and communication delays by relating job-machine assignment variables to job start and completion time variables. In this way, precedence constraints can be addressed effectively, as has been shown by [10, 21], but communication delays are much more challenging to capture. One natural approach is to add same-machine indicator variables $\delta_{u,v,i}$. Intuitively $\delta_{u,v,i} = 1$ if u and v are both scheduled on machine i, otherwise $\delta_{u,v,i} = 0$. We could then add the following constraint, where S_v and C_v represent the start and completion times of job v.

$$S_v \geq C_u + \rho\Big(1 - \sum_i \delta_{u,v,i}\Big) \qquad \forall u, v, i : u \prec v$$

We can think of the constraint as stating that any job v must begin at least ρ steps after any of its predecessors u, if v and u are not executed on the same machine. Unfortunately, a simple instance with a fractional solution that spreads each job among all the machines and sets the δ values to $1/m$ leads to an integrality gap as large as a polynomial in ρ, m, and n. See Appendix C.2 for details.

A different strategy for constructing a linear relaxation is to use time-indexed job-machine assignment variables $x_{v,i,t}$ to indicate the completion time t of job v on machine i. Indeed, such a program capturing both precedence constraints and communication delays is used in [18] to obtain a quasi-polynomial time approximation scheme when the number of machines m is $O(1)$, the communication delays are $O(1)$, and all machines are identical. Unlike [18], however, we are working with an arbitrary number of machines of arbitrary speeds, and arbitrarily large communication delay. In this case, the time-indexed relaxation has an integrality gap as large as a polynomial in ρ, m, and n. See Appendix C.3.

Developing our relaxation. To overcome the challenges mentioned above, we introduce two new sets of constraints - *delay constraints* and *phase constraints* - in addition to the usual related machines scheduling constraints of [10, 21], where a *phase* is any interval of ρ time in a schedule. To build intuition, we introduce these constraints in the setting with unit speeds and unit job sizes. We then provide a natural (but weak) generalization of these constraints to the setting with arbitrary speeds and job sizes which, unfortunately, has a large integrality gap. Finally, we refine the constraints yielding our linear relaxation.

For unit speeds and unit jobs size, the phase constraints require that if a job v is scheduled to start at time t on machine i, then the total number of v's predecessors that are scheduled to start in the interval $[t-\rho, t)$ is at most ρ because they must all be scheduled on the same machine. To capture this property, we introduce *same-phase* variables $y_{u,v}$ for each pair of jobs u, v such that $u \prec v$. We can view $y_{u,v}$ as indicating whether u is scheduled within ρ steps of the start of v. We can then give the following constraints.

$$S_v \geq S_u + \rho(1 - y_{u,v}) \quad \forall u, v : u \prec v \qquad \text{and} \qquad \rho \geq \sum_{u \prec v} y_{u,v} \quad \forall v$$

The first is the delay constraint and states that the difference in start times for v and u is at least ρ if u is not scheduled within ρ of the start time of v. The second is the phase constraint and states that the total number of v's predecessors that are scheduled to start within ρ time of v is at most ρ. While this relaxation has a small integrality gap in the unit case, adapting it to the non-unit case is not straightforward.

In the case with arbitrary speeds and job sizes, we would like to capture the property analogous to the one used in the unit case: if a job v is scheduled to start at time t on machine i then the set of all v's predecessors that are scheduled to start in the interval $[t-\rho, t)$ should have total size at most ρs_i. The following relaxation, which retains the same-phase variables of the unit relaxation as well as the unit delay constraint, shows a natural way to extend the phase constraint to capture this property.

$$\rho \sum_i s_i x_{v,i} \geq \sum_{u \prec v} p_u y_{u,v} \qquad \forall v$$

However, these constraint have a flaw. If, say a small fraction of v is placed on the fastest machine and the rest on the slowest, then the left-hand term will allow too many predecessors to be scheduled in the same phase. As shown in Appendix C.4, this leads to an integrality gap as large as ρ or polynomial in m and n.

A key idea in our linear relaxation is the introduction of *machine-dependent same-phase* variables, which tie the notion of a phase to the speed of a particular machine. Using these variables, we introduce new phase and delay constraints which rely crucially on our ordering of machines by increasing speed. Our linear relaxation LP minimizes C subject to the following constraints.

$$C \geq S_v + p_v \sum_i x_{v,i}/s_i \qquad \forall v \quad (1)$$

$$S_v \geq S_u + p_u \sum_i x_{u,i}/s_i \qquad \forall u, v : u \prec v \quad (2)$$

$$S_v \geq S_u + \rho\Big(\sum_{j \leq i} x_{v,j} - z_{u,v,i}\Big) \qquad \forall u, v, i : u \prec v \quad (3)$$

$$\sum_{j \leq i} x_{v,j} \geq \sum_{u \prec v} p_u z_{u,v,i}/\rho s_i \qquad \forall v, i \quad (4)$$

$$Cs_i \geq \sum_v p_v x_{v,i} \qquad \forall i \quad (5)$$

$$\sum_i x_{v,i} = 1 \qquad \forall v \quad (6)$$

$$S_v \geq 0 \qquad \forall v \quad (7)$$

$$x_{v,i} \in (0,1) \qquad \forall v, i \quad (8)$$

$$z_{u,v,i} \in (0,1) \qquad \forall u, v, i : u \prec v \quad (9)$$

We provide some intuition behind the variables and constraints. We interpret the variables $x_{v,i}$ as giving the "primary" placement of v and S_v as the corresponding start time of v. Then, for any jobs u and v such that $u \prec v$ and for any machine i, we can understand the variable $z_{u,v,i}$ as indicating, first, whether v is executed on a machine indexed i or lower, and second, whether the start time of u is within ρ of the start time of v. The significance of this indication is that, if these conditions are met, then some *copy* of u must execute on the same machine as v within ρ time of v and, therefore, only predecessors of total size at most ρs_i can meet these conditions. The remaining variable C captures the makespan of the resulting schedule.

The delay constraint (3) states that if v is scheduled on a machine slower than i, then v should start at least ρ time after any predecessor u unless u is scheduled in the same phase as v. The phase constraint (4) states that if v is scheduled on a machine slower than i, then the total size of v's predecessors scheduled in the same phase is at most ρs_i. The remaining constraints ensure that no job completion time exceeds the makespan (1), that jobs are executed completely and in order (2, 6), and that the total load on any machine does not exceed the makespan (5).

Group assignment. The fractional solution we obtain for the relaxation of LP gives us a fractional assignment of jobs to machines, as well as lower bounds on start times of jobs. The objective function is the maximum over all job completion times as well as over all machine loads, and so lower bounds the optimal makespan. The next step is to convert this solution into an assignment κ of each job to some set of machines. This assignment will guide our final construction of the schedule. We partition the set of machines into $K \leq \log m$ groups $\Gamma_1, \ldots, \Gamma_K$ of increasing speed and define a job's "median" machine group as the lowest (slowest) one such that the job's total fractional assignment to this and slower groups is at least $1/2$. Our group assignment follows an approach similar to [10, 21]: we assign each job to the highest capacity group that is at least as fast as its median group. Note that, if there are jobs assigned to groups Γ_k and $\Gamma_{k'}$, with $k < k'$, then the minimum speed in group Γ_k is less than that in group $\Gamma_{k'}$, but the capacity of Γ_k is at least that of $\Gamma_{k'}$, since the jobs assigned to Γ_k could have been assigned to group $\Gamma_{k'}$ but were not.

Computing the schedule. Our scheduling algorithm (Algorithm 1) takes the group assignment κ and produces a schedule, with possible duplications, for all jobs. The main challenge in constructing the schedule is balancing two conflicting incentives. On the one hand, the more we allow a set of jobs to be duplicated, the faster we can finish any jobs preceded by jobs in the set. On the other hand, if we duplicate too often, then we risk overloading machines with too many jobs to execute. Specifically, we want to avoid scheduling too much load assigned to higher capacity groups on lower capacity (faster speed) groups, even when doing so would allow us to complete some jobs earlier. We strike this balance by allowing a job to be duplicated only in groups with capacity higher than its assigned group. Furthermore, similar to [20], when the scheduler places a set of jobs on a machine, we require that at least a $1/\eta$ fraction of the total size of that set be from jobs that have not yet been placed on any machine, where η will be set later.

The scheduling algorithm proceeds in a series of rounds. In each round, the algorithm iterates through

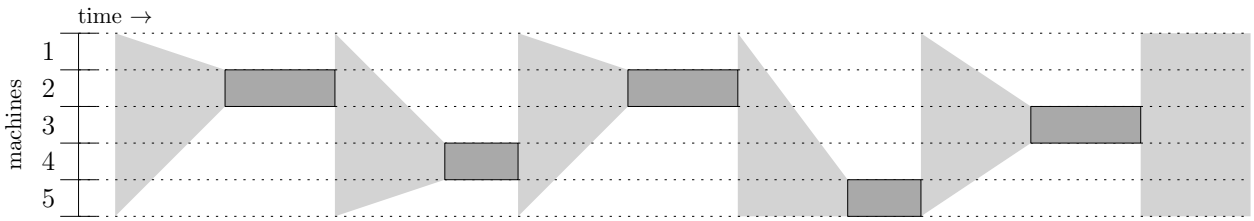


Figure 1: Machines are shown vertically on the left and time increases from left to right. The chain is shown as dark gray boxes. Each Light gray, borderless area represents the set of jobs that precede the chain job to its right (if it exists) and complete after the chain job to its left (if it exists).

each machine group Γ_k and considers each job v with $\kappa(v) = k$ that has not yet been scheduled. On a machine $i \in \Gamma_k$ the algorithm schedules v and its predecessors that have not been completed in earlier phases if the following three conditions are satisfied: **(a)** v's incomplete predecessors can be completed on i in time $O(\rho)$; **(b)** the total size of v and its predecessors not already scheduled (on any machine) is at least a $1/\eta$ fraction of the total size of its uncompleted predecessors; and **(c)** all of v's remaining predecessors have been assigned to higher indexed groups. Condition **(c)** ensures that we duplicate jobs only from lower capacity groups to higher capacity groups. Condition **(a)** ensures that any jobs we duplicate from lower capacity, higher speed groups won't take too long on the lower speed group. Condition **(b)** ensures two things: first, it guarantees that the total increase in load from duplication is no more than η and, second, it guarantees any large gaps in the schedule result from the fact that all those jobs with a small number of predecessors have the total size of their remaining predecessor reduced by a factor of η. The usefulness of these conditions is made more explicit in the analysis section.

Overview of the analysis. For the purposes of analysis, we divide our schedule into phases of length ρ and partition these phases into three types. We then bound the makespan of our schedule by bounding the total number of phases of each type. Our analysis combines elements of the analysis in [10] and [20].

The three types of phases are *chain* phases, *load* phases, and *height* phases. We define a chain $\mathscr{C}$ such that each element in $\mathscr{C}$ precedes the next, and each element has an instance which takes a sufficiently long time in the schedule. Chain phases are those phases in which some machine spends most of its time working on some chain element. All non-chain phases are divided into load and height phases. Load phases are those non-chain phases in which every machine of some group is working on jobs for most of the phase. The remaining phases are height phases. We can think of the three categories more intuitively as follows. Chain phases primarily reduce the remaining execution time of the chain. Load phases primarily reduce the remaining execution time of the set of all jobs. Height phases primarily reduce the amount of time before the next chain phase (or the end of the schedule if the chain has been completed). Figure 1 depicts the relationship between the chain and the sets of jobs on which height phases make progress.

We now briefly overview how we bound the number of phases of each type. We first discuss chain phases. Since chain jobs take a long time in the schedule, condition **(a)** ensures that every chain job is scheduled only on machines in its assigned group. Since we derived the group assignments from LP, the time spent executing jobs in the chain is at most $O(OPT)$, so the total number of chain phases is at most $O(OPT/\rho)$. We next consider load phases. Condition **(c)** guarantees that the set of jobs scheduled on groups $\Gamma_k, \ldots, \Gamma_K$ is a subset of the jobs assigned to these groups by κ. So, by condition **(b)**, we have that for any k, the total load on groups $\Gamma_k, \ldots, \Gamma_K$ is at most an η factor above the total load assigned to those groups by κ. Using a lemma from [10], this entails that the total number of phases is no more than $O(OPT \cdot K\eta/\rho)$.

Bounding the number of height phases is more involved as it requires a closer analysis of the linear program as well as a more detailed understanding of the step-by-step operation of the scheduling algorithm. We first partition the jobs in *bands* $B_1, B_2, \ldots$ according to their start times as given by LP. We show that, for each job v in a band, the total size of v's predecessors in the same band is small enough to be completed in $O(\rho)$ time on v's assigned group. Then, for each height phase τ, we consider the lowest band B_r with some job scheduled after phase τ and the slowest group Γ_k with a job in that band. Let v be some unscheduled job in B_r assigned to group Γ_k. We consider a series of height phases separated by at most $O(1)$ height phases. We show, for each height phase in this series, that there is some iteration of our scheduling algorithm in which the algorithm *considers* placing v with its remaining predecessors on some machine in Γ_k and in which all of v's predecessors that started in the previous height phase in the series have completed with enough time to communicate the results to all machines. Due to our choice of v, we can then infer that, if the algorithm does not place v in this iteration, it is because v's uncompleted predecessor set violates condition **(b)**. This entails that by the next height phase in the series, the size of v's remaining predecessor set is reduced by a factor of η. Since v's predecessors within the band can be completed in $O(\rho)$ time on any machine in group Γ_k, we have that after $O(\log_\eta \rho)$ height phases v's predecessor set is empty. This entails that v is scheduled before (or during) the next height phase in the series. Letting r^* be the number of bands, this argument upper bounds the number of height phases by $O(Kr^* \log_\eta \rho)$. We then show that the number of bands r^* is $O((OPT + \rho)/\rho)$, which gives the desired bound on the number of height phases.

Finally, we set η to $\log \rho / \log\log \rho$. Summing over the number of phases of each type, we have that the length of our schedule is upper-bounded by $O(K \cdot \log \rho / \log\log \rho)(OPT + \rho)$.

3.2 Integrality gap

We construct a new integrality gap instance that achieves a $\omega(1)$ integrality gap in the presence of communication delays. The gap construction consists of a layered DAG with $L = \omega(1)$ layers and n vertices in each layer, where each job in layer ℓ has dependencies on d randomly chosen jobs in $V_{\ell+1}$ as shown in Figure 2. In particular, $\rho = d^L = n^c$ for a small constant $c > 0$. The parameters of the construction are set up in such a way that fractionally all the jobs can be assigned in one phase (hence the LP solution value is at most ρ).

The main technical challenge is to argue that $\Omega(L)$ phases are needed to schedule all the jobs in order to get a lower bound of $\Omega(L\rho)$ for the integer solution value. This gives a gap of $\Omega(L) = \Omega(\sqrt{\log \rho})$. From the expansion of the random graph in each layer, it is easy to argue that at most a $o(1)$ fraction of the jobs in layers $\{1, \ldots, L-2\}$ can be scheduled in the first phase (since at most $\rho \ll n$ of the jobs can be on one machine). However, in the next phase the results of all jobs that were scheduled previously are now available to all the machines; moreover the choice of these jobs could depend on the randomness in the DAG. Hence the remaining graph in each layer (after removing vertices that have already been scheduled) in the subsequent phases is *not random* any longer!

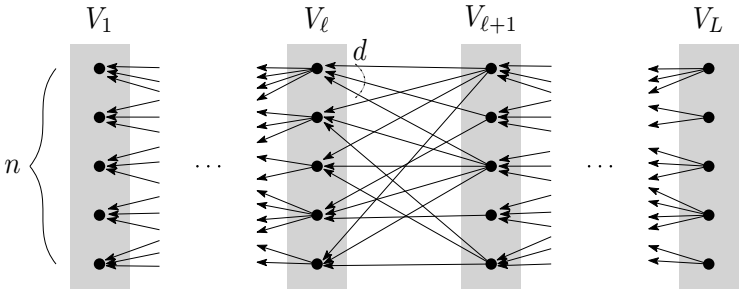


Figure 2: The figure shows the DAG with L layers $V_1, \ldots, V_L$ representing the nL jobs. Each of the n jobs in V_ℓ has dependencies on d randomly chosen jobs in $V_{\ell+1}$. We set $\rho = d^L$, $m = \rho$, and the parameters $L = c_1\sqrt{\log n}, d = 2^{c_2\sqrt{\log n}}$ for some appropriate constants $c_1, c_2 > 0$.

To overcome this technical hurdle, we identify and exploit a property of *robust expansion*, which may

be of independent interest. The standard vertex expansion property of a random graph says that w.h.p. any subset $S \subset V_\ell$ of size $|S| \leq n/d$ has a neighborhood of size $|\Gamma(S)| = \Omega(d|S|)$. However, random graphs have the stronger property that no subset T of size $o(d|S|)$ can have $\Omega(d|S|)$ of the edges from S incident on it. For our analysis, we need to prove a generalization for paths of length $\ell < L$ (Lemma 5.3): *w.h.p. for any $S \subset V_i$ (of sufficiently small size), there is no subset of size $o(d^\ell|S|)$ that can have $\Omega(d^\ell|S|)$ of the length-ℓ paths going into S.*

Each job u in layer i (i.e. a vertex in V_i) has $d^{\ell-i}$ incoming paths from layer V_ℓ, and all of the vertices in these paths need to be scheduled before scheduling u – either in a previous phase, or on the same machine in the current phase. The above robust expansion property is used to upper bound the number of jobs completed in each phase in two different ways: 1) to upper bound the number of jobs in V_i whose dependencies in V_ℓ "mostly" consists of jobs scheduled in previous phases, and 2) to upper bound the number of jobs in V_i such that most of their dependencies in V_ℓ need to be resolved in the current phase. This allows us to prove that we need at least $L/2$ phases before most of the jobs in V_1 can be scheduled.

We believe that our integrality gap argument applies to a wider class of relaxations for the problem. Any program that captures communication delay and precedence requirements through individual constraints for each job and has independent load constraints for each machine is likely to incur a similar gap.

3.3 Bounding the duplication advantage

The final contribution of this paper is to quantitatively characterize the duplication advantage. While it is easy to construct instances where the makespan of a schedule allowing duplication (which we refer to as a general schedule) is better than that of a no-duplication schedule (one in which all jobs are processed exactly once), our goal is to place upper and lower bounds on the duplication advantage.

Lower bound. We first present a simple family of instances with m identical machines, $n = 2m$ unit jobs, and $\rho = \log m$, for which any no-duplication schedule has makespan $\Omega(\rho^2/\log\rho)$, while the optimal makespan is at most ρ. The DAG for such an instance consists of a rooted binary tree with m leaves and edges directed away from the root, such that an optimal schedule executes each root-leaf path on a separate machine (with necessary duplication), while any no-duplication schedule is essentially forced to decompose the tree into $\rho/(\log\rho)$ phases, interspersed with communication delays. Note that we thus have $\Omega(\log m/\log\log m)$ and $\Omega(\log n/\log\log n)$ bounds on the duplication advantage.

Upper bound. Our main result in this section is that the duplication advantage is, in fact, also upper-bounded by a polylogarithmic factor $O(\log^2 n \log m)$. Our proof is through a polynomial-time algorithm that transforms any schedule to a no-duplication schedule with the polylogarithmic factor loss in makespan. The algorithm processes a given (general) schedule in "phases" of length ρ. The core of the algorithm is to transform each phase into a no-duplication schedule of length $O(\rho \log^2 n \log m)$. There are technical complications since (i) processing of jobs may span multiple phases of the schedule, and (ii) the number of phases may be super-polynomial in the size of the instance. Both these can be handled relatively easily by considering machines that are processing "long" jobs separately, and ignoring phases where no jobs are started or completed. These are detailed in Section 6.

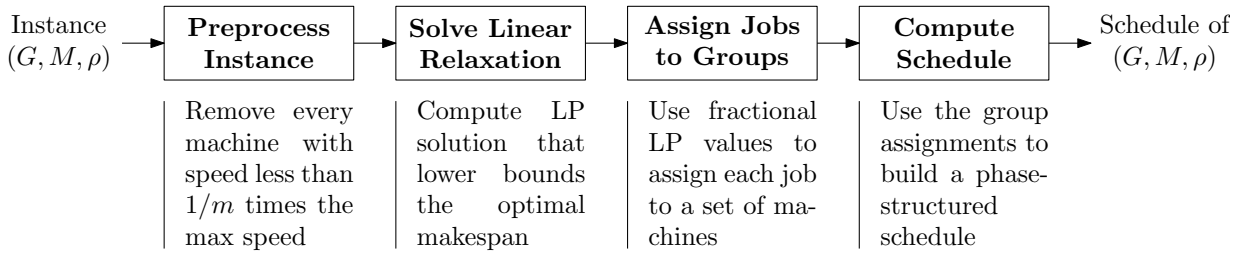
We now give an overview of the algorithm's core. Consider the sub-DAG D of the original DAG formed by the jobs that are processed within a particular phase of the general schedule. We face several technical challenges while designing a no-duplication schedule for this sub-DAG. First, we need to determine the relative order between the jobs. For example, if a node serves as an predecessor of many other jobs, it could be given higher priority, but those successors may have been processed in the general schedule at many distinct machines, along with a copy of the predecessor, something we cannot do in the no-duplication

schedule. Second, if we choose to process two jobs on two different machines in a phase, we have to ensure that they do not share a common predecessor.

To address these challenges, we organize and process the jobs of D as follows. First, we divide them into $O(\log n \log m)$ groups based on their level of duplication in the general schedule; each group consists of jobs whose duplication level is within a factor of $(1 + 1/(2 \log n))$ of one another. We then process the groups from the highest level of duplication down to the lowest, since the duplication level of a job in D is at least that of any of its successors. Within a given group, we focus on the sink jobs (which have no predecessors in the group) and construct an undirected graph H over them in which an edge exists between two sinks if they share a common predecessor. Our key insight about H is that any subset of jobs in H that is composed of regions of diameter $O(\log n)$ that do not share any common neighbors among them can be processed in a single phase in a no-duplication schedule. We show that using a classic low-diameter decomposition technique from approximation algorithms and distributed computing (e.g., see [2, 22, 30]), we can find a subset of $\Omega(H)$ jobs that has the desired structure in H. A recursive use of this subroutine, together with the other techniques indicated above, yields the desired no-duplication schedule.

4 Approximation Algorithm for Makespan Minimization

Our algorithm for minimizing makespan is based on a linear programming relaxation of DAG Scheduling with Communication Delay. This linear program is rounded to get an assignment of jobs to groups of machines, and the assignment is then used to schedule each job. In Section 4.1 we present the first subroutine of our algorithm, which removes all machines that are too slow to be useful. Section 4.2 contains the second subroutine, which consists of solving the linear programming relaxation. In Section 4.3 we present the third subroutine, which consists of finding an assignment of each job to a group of machines, given a fractional assignment of jobs to machines given by the LP solution. In Section 4.4 we present our scheduling subroutine which takes an assignment of jobs to groups of machines and computes a schedule. Finally, in Section 4.5 we show that the schedule produced by our algorithm has makespan within a polylogarithmic factor of the optimal makespan.



4.1 Preprocessing the Instance

In this section, we present the first subroutine of our algorithm. Given an instance (G, M, ρ), this subroutine outputs a new instance (G, M', ρ) such that $M' \subseteq M$ where the speed of each machine in M' is within a factor m of the speed of the fastest machine in M. More formally,

$$M' = \{i \in M : s_i \geq s_m/m\}$$

where machine m is the fastest machine in M. The following lemma is proved in Appendix B.

Lemma 4.1. Let C^* be the optimal makespan of any schedule of (G, M, ρ) and let C' be the optimal makespan of any schedule of (G, M', ρ). Then $C' = O(C^*)$.

The proof in Appendix B shows that any schedule σ_1 on (G, M, ρ) with makespan C_1 can be converted into a schedule σ_2 of (G, M', ρ) with makespan $C_2 \leq 6C_1$. We note that an alternative strategy for eliminating these machines is given in [10]. There, the authors argue that any solution to their linear program defined over (G, M) can be converted into a solution to their linear program on (G, M'). A corresponding result can be proved for our linear program. We choose to use the method given in Appendix B because the bound proved there holds for the optimal *makespan*, independent of a particular relaxation or algorithm used to solve the problem, and is therefore more general.

4.2 Linear Programming Relaxation

In this subroutine, we formulate and solve a linear programming relaxation for the given instance of DAG Scheduling with Communication Delay. The following linear program, called LP, minimizes C subject to the following constraints. Indices u and v refer to jobs and i and j refer to machines.

$$C \geq S_v + p_v \sum_i x_{v,i}/s_i \qquad \forall v \quad (1)$$

$$S_v \geq S_u + p_u \sum_i x_{u,i}/s_i \qquad \forall u, v : u \prec v \quad (2)$$

$$S_v \geq S_u + \rho\Big(\sum_{j \leq i} x_{v,j} - z_{u,v,i}\Big) \qquad \forall u, v, i : u \prec v \quad (3)$$

$$\sum_{j \leq i} x_{v,j} \geq \sum_{u \prec v} p_u z_{u,v,i}/\rho s_i \qquad \forall v, i \quad (4)$$

$$Cs_i \geq \sum_v p_v x_{v,i} \qquad \forall i \quad (5)$$

$$\sum_i x_{v,i} = 1 \qquad \forall v \quad (6)$$

$$S_v \geq 0 \qquad \forall v \quad (7)$$

$$x_{v,i} \in (0, 1) \qquad \forall v, i \quad (8)$$

$$z_{u,v,i} \in (0, 1) \qquad \forall u, v, i : u \prec v \quad (9)$$

We give an intuitive interpretation of the variables and constraints. We interpret the variables $x_{v,i}$ as giving the "primary" placement of v and S_v as the corresponding start time of v. Then, for any jobs u and v such that $u \prec v$ and for any machine i, we can understand the variable $z_{u,v,i}$ as indicating, first, whether v is executed on a machine indexed i or lower, and second, whether the start time of u is within ρ of the start time of v. The significance of this indication is that, if these conditions are met, then some *copy* of u must execute on the same machine as v within ρ time of v and, therefore, only predecessors of total size at most ρs_i can meet these conditions. The remaining variable C captures the makespan of the resulting schedule.

Constraint (1) states that the makespan should be at least as the amount of time to execute any job after its start time. Constraint (2) states that a job should start after the completion of its predecessor. The delay constraint (3) states that if v is scheduled on a machine slower than i, then v should start at least ρ time after any predecessor u unless u is scheduled in the same phase as v. The phase constraint (4) states that if v is scheduled on a machine slower than i, then the total size of v's predecessors scheduled in the same phase is at most ρs_i. Constraint (5) states that the makespan should be at least as large as the total load on any machine. Constraint (6) states that each job should be completely scheduled.

Lemma 4.2 (LP is a valid relaxation). For any instance (G, M, ρ) for which the optimal makespan is C^*, the value of the optimal solution to LP is at most $2C^*$.

Proof. Let σ be a schedule of (G, M, ρ) with makespan C'. We construct a solution to LP with objective value $2C'$. We first construct the schedule σ' as follows. We define *phase* τ to be the interval of time $[\tau\rho, (\tau+1)\rho)$. For any job-machine pair (v, i) such that v is scheduled on i and $\sigma(v, i) \in [\tau\rho, (\tau+1)\rho)$, we

set $\sigma'(v,i) = \sigma(v,i) + \tau\rho$. Note that the makespan of σ' is at most $2C'$. We show that σ' is a valid schedule of the instance (G, M, ρ). To show that the precedence and communication requirements are satisfied, suppose job v is scheduled on machine i and job u is scheduled on machine j. By definition of the phases, we have $\sigma'(v,i) - \sigma'(u,j) \geq (\sigma(v,i) - \sigma(u,j) + \rho \cdot \lfloor \sigma(v,i) - \sigma(u,j)/\rho \rfloor \geq \sigma(v,i) - \sigma(u,j)$. So, the time between two executions in σ' is at least the between the same executions in σ. This entails that both precedence and communication requirements are satisfied in σ'. The other requirements are easy to check. Therefore, σ' is a valid schedule.

Given σ', we set the variables of LP as follows. We assume, without loss of generality, that each job is executed at most once on each machine. For each v, let i^* be some machine on which σ' first completes v, choosing arbitrarily if there is more than one. Set $x_{v,i^*} = 1$ and $x_{v,i} = 0$ for $i \neq i^*$. Let t^* be the start time of v on i^* in σ' and set $S_v = t^*$. For all $u \prec v$ and $i \geq i^*$, set $z_{u,v,i} = 1$ if $S_v - S_u \leq \rho$, and $z_{u,v,i} = 0$ otherwise. Set $C = 2C'$.

We now show that our assignment of values to each variable satisfies all constraints of the linear program. It is easy to verify that constraints (1), (2), (5) - (9) are satisfied. For Constraint (3), let us fix u, v and i, and let $\sum_{j \leq i} x_{v,j} = \alpha$ and $z_{u,v,i} = \beta$. Then there are four cases to verify: (a) $\alpha = \beta = 1$, (b) $\alpha = 1$ and $\beta = 0$, (c) $\alpha = 0$ and $\beta = 1$, or (d) $\alpha = \beta = 0$. In cases (a), (c), and (d) it is easy to see that the constraint is satisfied by the fact that $S_v \geq S_u$. In case (b), we see that $z_{u,v,i}$ is set to 0 only if $S_v - S_u > \rho$, which entails that constraint is satisfied, or if $i < i^*$, which entails $\sum_{j \leq i} x_{v,j} = 0$.

To show that Constraint (4) is satisfied, fix v and i and let A be the set of jobs $u \in A_v$ for which $z_{u,v,i} = 1$. By our setting of $z_{u,v,i}$, if $\sum_{j \leq i} x_{v,j} = 0$, then $A = \varnothing$ and the constraint is trivially satisfied. So assume that $\sum_{j \leq i} x_{v,j} = 1$. This implies that v is first completed on a machine j no faster than s_i. Since all jobs in A start their first completed execution less than ρ time before the start of v, they all must have some copy executed on j that also completes less than ρ time before the start of v. In σ, let u be some job that completes on j less than ρ time before $\sigma(v,j)$ and starts on j more than ρ time before $\sigma(v,j)$. In this case, u starts in a lower phase than v. Therefore, by our construction of σ', there is a gap of at least ρ in σ' between the completion of u and the start of v during which no jobs are executed. Therefore, in σ', all jobs in the set A both start and complete on j less than ρ time before $\sigma'(v,j)$. Therefore, $p(A) \leq \rho s_i$, from which (4) follows. □

4.3 Assigning Jobs to Groups

In this section, we present the third subroutine of our algorithm, which takes as input a given instance of DAG Scheduling with Communication Delay, as well as a fractional assignment $\{x_{v,i}\}_{v,i}$ of jobs to machines, and returns an integral assignment of jobs to groups of machines. Per Lemma 4.1, suppose that $s_i \geq s_m/m$ for each $i \in M$.

Recall that the machines are ordered such that $s_i \leq s_j$ if $i < j$. We first partition the set of machines M into groups $\Gamma_1, \Gamma_2, \ldots, \Gamma_K$. We define the groups iteratively, with group $\Gamma_1 = \{i : s_i \in [s_1, 2s_1)\}$ and group $\Gamma_{k+1} = \{i : s_i \in [s_j, 2s_j)$ where $j = \arg\min_{j'}\{j' \notin \bigcup_{k'=1}^{k} \Gamma_{k'}\}\}$. Since the speeds of all machines are within a factor of m, we have that the number of groups K is at most $\log m$.

Definition 4.3. For each job v, we define the group $\kappa(v)$ of v as follows. Let $\mathsf{median}(v) = \min\{k : \sum_{k' \leq k} \sum_{i \in \Gamma_k} x_{v,i} \geq 1/2\}$. Then, for any v,

$$\kappa(v) = \arg\max_k \{m_k \overline{s}_k : k \geq \mathsf{median}(v)\}.$$

We also define $\kappa^{-1}(k) = \{v : \kappa(v) = k\}$ and $\kappa_{\geq}^{-1}(k) = \{v : \kappa(v) \geq k\}$.

Informally, $\kappa(v)$ is the highest capacity group such that at least half of job v is assigned to groups no faster than the slowest machine in Γ_k. In the remainder of the section, we prove several properties of this assignment. Toward this end, we consider a set of start times $\{S_v\}_v$ and makespan $\{C\}$ such that $\{S_v\}_v \cup \{C\} \cup \{x_{v,i}\}_{v,i}$ is a feasible solution to LP for the given problem instance. We partition V into sets called *bands* where the elements v of each band are determined by the value S_v.

Definition 4.4. Band $B_r = \{v : \rho(r-1)/4 \leq S_v < \rho r/4\}$.

In the following lemma, we show that, for any job v, the total size of v's predecessors that occupy the same band is no more than the amount that can be completed in a constant number of communication phases on some machine in v's assigned group.

Lemma 4.5 (Upper bound on total size of predecessors of a job in a band). For any job $v \in B_r$, we have $p(B_r \cap A_v) \leq 8\rho\overline{s}_{\kappa(v)}$.

Proof. We fix r and choose any $v \in B_r$. Let $A = B_r \cap A_v$ and let $i^* = \arg\max_{i \in \mathsf{median}(v)}\{s_i\}$. By definition of the groups, if $i^* \in \Gamma_k$ then $k \leq \kappa(v)$. In this case, $s_{i^*} \leq 2\overline{s}_{\kappa(v)}$ and it is sufficient to show that $p(A) \leq 4\rho s_{i^*}$. For any $u \in A$,

$$\begin{aligned} S_v &\geq S_u + \rho\Big(\sum_{j \leq i^*} x_{v,j} - z_{u,v,i^*}\Big) && \text{by (3)} \\ &\geq S_u + \rho(1/2 - z_{u,v,i^*}) && \text{by definition of } i^*. \end{aligned}$$

By definition of B_r we have that $S_v \leq S_u + \rho/4$, so $z_{u,v,i^*} \geq 1/4$ for all $u \in A$. Suppose for the sake of contradiction that $p(A) > 4\rho s_{i^*}$. Then

$$\begin{aligned} \sum_{u \in A_v} p_u z_{u,v,i^*} \geq \sum_{u \in A} p_u z_{u,v,i^*} &\geq p(A)/4 > \rho s_{i^*} && \text{by assumption} \\ = \rho s_{i^*} \sum_j x_{v,j} &\geq \rho s_{i^*} \sum_{j \leq i^*} x_{v,j} && \text{by (6).} \end{aligned}$$

Therefore, Constraint (4) is violated. □

Although we can upper-bound the size of a job's predecessor set within a band, there is no limit on the total amount of work that can be put on a single band. For example, if all jobs are independent, then the solution to LP might place all jobs on the first band. The following lemma, due to [10], addresses this issue by providing a lower bound on the optimal linear programming solution in terms of the total load across all groups. We repeat the lemma and provide a proof for our linear relaxation.

Lemma 4.6 ([10] Lower bound on C^* in terms of load). $\displaystyle\sum_{k=1}^{K} \frac{p(\kappa^{-1}(k))}{m_k \overline{s}_k} = O(KC^*)$.

Proof. Let $y_{v,k} = 1$ if $\kappa(v) = k$ and 0 otherwise, for all $v \in V$ and $k = 1, \ldots, K$. Then, by definition of

the group assignment, the set of all variables $y_{v,k}$ gives an optimal solution to the following linear program.

$$\min \sum_{k=1}^{K} \sum_{v \in V} \frac{p_v y_{v,k}}{m_k \overline{s}_k}$$

$$\text{subject to} \quad \sum_{k=1}^{K} y_{v,k} = 1 \qquad \text{for all } v \in V, k = 1, \ldots, K \tag{10}$$

$$y_{v,k} = 0 \qquad \text{for all } v \in V, k = 1, \ldots, \mathsf{median}(v) - 1 \tag{11}$$

$$y_{v,k} \geq 0 \qquad \text{for all } v \in V, k = 1, \ldots, K \tag{12}$$

We give a feasible solution to this linear program whose objective values is at most $4KC$, thereby proving the lemma.

Let $x_{v,i}, S_v, C_v$, and C, for all $v \in V$ and $i \in M$, represent the variables in the the feasible solution to LP. Then, for all jobs v and groups Γ_k, we set $y_{v,k}$ as follows.

$$y_{v,k} \leftarrow \begin{cases} \frac{\sum_{i \in \Gamma_k} x_{v,i}}{\sum_{\ell=\mathsf{median}(v)}^{K} \sum_{i \in \Gamma_\ell} x_{v,i}} & \text{if } k \geq \mathsf{median}(v) \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that this assignment satisfies all constraints (10) – (12). We show that the objective is no more than $4KC$.

$$\begin{aligned}
\sum_{k=1}^{K} \sum_{v} \frac{p_v y_{v,k}}{m_k \overline{s}_k} &= \sum_{k=1}^{K} \frac{1}{m_k \overline{s}_k} \sum_{v:\mathsf{median}(v) \leq k} \frac{p_v \sum_{i \in \Gamma_k} x_{v,i}}{\sum_{\ell=\mathsf{median}(v)}^{K} \sum_{i \in \Gamma_\ell} x_{v,i}} && \text{by assignment of } y_{v,k} \\
&\leq \sum_{k=1}^{K} \frac{2}{\sum_{i \in \Gamma_k} s_i} \sum_{v:\mathsf{median}(v) \leq k} \frac{p_v \sum_{i \in \Gamma_k} x_{v,i}}{\sum_{\ell=\mathsf{median}(v)}^{K} \sum_{i \in \Gamma_\ell} x_{v,i}} && \text{by definition of } m_k \overline{s}_k \\
&\leq \sum_{k=1}^{K} \frac{2C}{\sum_{i \in \Gamma_k} \sum_{v} p_v x_{v,i}} \sum_{v:\mathsf{median}(v) \leq k} \frac{p_v \sum_{i \in \Gamma_k} x_{v,i}}{\sum_{\ell=\mathsf{median}(v)}^{K} \sum_{i \in \Gamma_\ell} x_{v,i}} && \text{by Constraint (5)} \\
&\leq \sum_{k=1}^{K} \frac{2C}{\sum_{i \in \Gamma_k} \sum_{v} p_v x_{v,i}} \sum_{v:\mathsf{median}(v) \leq k} 2p_v \sum_{i \in \Gamma_k} x_{v,i} && \text{by } \mathsf{median}(v) \\
&\leq \sum_{k=1}^{K} \frac{2C}{\sum_{i \in \Gamma_k} \sum_{v} p_v x_{v,i}} \cdot \sum_{v} 2p_v \sum_{i \in \Gamma_k} x_{v,i} = \sum_{k=1}^{K} 4C = 4KC.
\end{aligned}$$

The lemma follows. □

We note that an alternative group construction is given in [21] where the author reduces the approximation ratio of the algorithm in [10] from $O(\log m)$ to $O(\log m / \log \log m)$. However, given our analysis, we have been unable to apply the method of [21] to similarly reduce our approximation ratio. While the speeds of all machines within a group in our algorithm are within a factor of 2 of each other, the construction of [21] allows for these speeds to be off by up to a factor of $\log m / \log \log m$. Use of this alternative construction incurs an additional factor of $O(\log m / \log \log m)$ for the upper bound proved in Lemma 4.5 on the processing time of a job's predecessors in a band, resulting in a worse approximation ratio than our current bound.

4.4 Computing the Schedule

In this section, we describe the fourth subroutine of our algorithm that outputs a schedule σ for a given instance (G, M, ρ) and a given assignment of each job v to a group $\kappa(v)$ of machines in M. Our algorithm builds on ideas used in the scheduling algorithms given in [20] and [10]. We apply a similar approach as that used in [20] to decide when to place a job (and all its remaining predecessors). However, unlike [20], our algorithm accounts for arbitrarily slow machines and arbitrarily large jobs, both of which may cause a job's execution to last well beyond after the time of its placement. As in [10], we use the group assignment to guide where jobs are scheduled. One fundamental difference between our algorithm and that in [10] is that, given certain conditions, we allow a job to be scheduled on machines in groups to which the job is not assigned. The subroutine in given in Algorithm 1.

Algorithm 1: Group-Based Scheduling with Duplication and Communication Delay

Data: instance (G, M, ρ), assignment κ of jobs to groups, and overlap parameter $\eta \geq 1$
Result: a schedule σ of G on M

1 **Initialize:** $T \leftarrow 0$; $\mathsf{Placed} \leftarrow \varnothing$; $\forall j : T_j \leftarrow 0$; $\forall v, i : \sigma(v, i) = \infty$
2 **while** $\mathsf{Placed} \neq V$ **do**
3 **forall** *machine groups* $k = 1, \ldots, K$ **do**
4 **forall** *jobs* $v \in \kappa^{-1}(k)$ **do**
5 $i \leftarrow \arg\min_{j \in \Gamma_k} \{T_j\}$
6 $A \leftarrow (A_v \cup \{v\}) \setminus \{u : \sigma(u, i) + p_u/s_i \leq T_i \text{ or } \exists\ j,\ \sigma(u, j) + p_u/s_j \leq T_i - \rho\}$
7 **if (a)** $p(A \setminus \{v\}) \leq 8\rho\bar{s}_k$ *and* **(b)** $p(A \setminus \mathsf{Placed}) \geq p(A)/\eta$ *and* **(c)** $A \subseteq \kappa_{\geq}^{-1}(k)$ **then**
8 **forall** $u \in A$ *in topological order* **do**
9 $\sigma(u, i) \leftarrow T_i$
10 $T_i \leftarrow T_i + p_u/s_i$
11 $\mathsf{Placed} \leftarrow \mathsf{Placed} \cup A$
12 $T \leftarrow \min\{t : t > T \text{ and either } t = \sigma(v, i) + p_v/s_i \text{ or } t = \rho + \sigma(v, i) + p_v/s_i \text{ for some } v, i\}$
13 $\forall j :\ T_j \leftarrow \max\{T, T_j\}$

We introduce notation to keep track of how the algorithm variables change during the execution. We refer to each execution of a line of Algorithm 1 as a *step* of the algorithm. For a given step ς of Algorithm 1, we use $\langle \mathsf{term} \rangle^{(\varsigma)}$ to denote the value of $\langle \mathsf{term} \rangle$ immediately before executing step ς, where $\langle \mathsf{term} \rangle$ may be $v, i, A, k, T, \mathsf{Placed}$, or T_i for a given i. We also use σ to denote the schedule output by Algorithm 1. Define

$$\mathcal{T} = \{0\} \cup \{t : \exists\ (v, i), \sigma(v, i) \neq \infty \text{ and } t = \sigma(v, i) + p_v/s_i \text{ or } t = \rho + \sigma(v, i) + p_v/s_i\}$$

and $V(j, \varsigma)$ to be the set of jobs that have been placed on machine j prior to executing step ς. The following two lemmas characterize how T_i and T vary, respectively, during the execution of the algorithm.

Lemma 4.7 (T_i tracks maximum completion time on i). For any machine j and any step ς executing line 10 or 13, we have $T_j^{(\varsigma+1)} = \max\{T^{(\varsigma+1)}, \max_{v \in V(j,\varsigma)}\{\sigma(v, j) + p_v/s_j\}\}$.

Proof. Let $\varsigma_0, \varsigma_1, \varsigma_2, \ldots$ be the sequence of steps defined as follows: ς_0 is first step of the algorithm (executing line1) and ς_{d+1} is the d^{th} step, in order, in which either line 10 or 13 is executed. We show that, for any j and d, $T_i^{(\varsigma_d+1)} = \max\{T^{(\varsigma_d+1)}, \max_{v \in V(j,\varsigma_d)}\{\sigma(v, j) + p_v/s_j\}\}$. The proof is by induction on

d. The equality holds easily for $d = 0$ by the initialization step. So we assume the induction hypothesis equality holds up to $d \geq 0$. We consider two cases. First, suppose that ς_{d+1} executes line 10. Then the job $u = u^{(\varsigma_{d+1})}$ was just placed on machine $i^{(\varsigma_{d+1})}$ in line 9. For $j \neq i^{(\varsigma_{d+1})}$, neither T_j nor T has changed, and $V(j, \varsigma_{d+1})$ is the same as $V(j, \varsigma_d)$, so the claim follows from the induction hypothesis. For $j = i^{(\varsigma_{d+1})}$, by the execution of line 9 on step $\varsigma_{d+1} - 1$ and line 10 on step ς_{d+1}, we have $T_j^{(\varsigma_{d+1}+1)} = \sigma(u, j) + p_u/s_j$. We thus derive

$$\begin{aligned}
T_j^{(\varsigma_{d+1}+1)} &= \sigma(u,j) + p_u/s_j && \text{lines 9 and 10} \\
&\leq \max_{v \in V(\varsigma_{d+1}, j)} \{\sigma(v,j) + p_v/s_j\} && u \in V(\varsigma_{d+1}+1, j) \\
&= \max\{\sigma(u,j) + p_u/s_j, \max_{v \in V(\varsigma_d, j)} \{\sigma(v,j) + p_v/s_j\}\} && V(i, \varsigma_{d+1}) = V(i, \varsigma_d) \cup \{u\} \\
&\leq \max\{\sigma(u,j) + p_u/s_j, T_j^{(\varsigma_d+1)}\} && \text{induction hypothesis} \\
&= \max\{\sigma(u,j) + p_u/s_j, T_j^{(\varsigma_{d+1})}\} && T_j^{(\varsigma_{d+1})} = T_j^{(\varsigma_d+1)} \\
&= \max\{\sigma(u,j) + p_u/s_j, \sigma(u,j)\} && \text{line 9} \\
&= \sigma(u,j) + p_u/s_j = T_j^{(\varsigma_{d+1}+1)}.
\end{aligned}$$

It thus follows that every inequality in the above sequence is, in fact, an equality yielding the equation

$$T_j^{(\varsigma_{d+1}+1)} = \max_{v \in V(\varsigma_{d+1}, j)} \{\sigma(v,j) + p_v/s_j\}.$$

We also have

$$T_j^{(\varsigma_{d+1}+1)} \geq T_j^{(\varsigma_{d+1})} = T_j^{(\varsigma_d+1)} \geq T^{(\varsigma_d+1)} = T^{(\varsigma_{d+1}+1)},$$

yielding the desired claim for induction step:

$$T_j^{(\varsigma_{d+1}+1)} = \max\{T^{(\varsigma_{d+1}+1)}, \max_{v \in V(\varsigma_{d+1}, j)} \{\sigma(v,j) + p_v/s_j\}\}.$$

We next consider the second case of the lemma, in which ς_{d+1} executes line 13. Fix any machine j. We derive

$$\begin{aligned}
T_j^{(\varsigma_{d+1}+1)} &= \max\{T^{(\varsigma_{d+1})}, T_j^{(\varsigma_{d+1})}\} && \text{line 13} \\
&= \max\{T^{(\varsigma_{d+1}+1)}, T_j^{(\varsigma_{d+1})}\} && T^{(\varsigma_{d+1}+1)} = T^{(\varsigma_{d+1})} \\
&= \max\{T^{(\varsigma_{d+1}+1)}, T_j^{(\varsigma_d+1)}\} && T^{(\varsigma_{d+1})} = T_j^{(\varsigma_d+1)} \\
&= \max\{T^{(\varsigma_{d+1}+1)}, \max_{v \in V(j, \varsigma_d)} \{\sigma(v,j) + p_v/s_j\}\} && \text{induction hypothesis} \\
&= \max\{T^{(\varsigma_{d+1}+1)}, \max_{v \in V(j, \varsigma_{d+1})} \{\sigma(v,j) + p_v/s_j\}\},
\end{aligned}$$

thus completion the induction step for the second case of the lemma. □

Lemma 4.8 (T iterates through $\mathcal{T}$). For any integer $1 \leq \ell \leq |\mathcal{T}|$, if step ς is the ℓth execution of line 12, then $T^{(\varsigma)}$ is the ℓth minimum element of $\mathcal{T}$.

Proof. Our proof is by induction on ℓ. Let ς_ℓ denote the ℓth execution of line 12. For the base case $\ell = 1$, we note that $T^{(\varsigma_1)} = 0$, which is the smallest element of $\mathcal{T}$. For the induction hypothesis, suppose the claim holds up to $\ell \geq 1$. Consider the claim for $(\ell+1)$th execution of line 12. Since T is only updated in line 12, $T^{(\varsigma_{\ell+1})} = T^{(\varsigma_\ell+1)}$.

By Lemma 4.7 and the setting of σ in line 9, for every job v placed after the $(\ell-1)$th execution of line 12 and before the ℓth execution of line 12, there exists an i such that $\sigma(v,i) + p_v/s_i$ is finite and exceeds $T^{(\varsigma_\ell)}$, which by induction is the ℓth minimum element of $\mathcal{T}$. So, if there is any job v such that $\sigma(v,i) + p_v/s_i + \rho$ is finite and exceeds $T^{(\varsigma_\ell)}$, then $T^{(\varsigma_\ell+1)}$ equals the $(\ell+1)$th minimum element of $\mathcal{T}$ since every subsequent job placed has its start time at least $T^{(\varsigma_\ell+1)}$.

It remains to show that there is at least one such job. For the sake of contradiction, suppose not. Then, it must be the case that for every v already placed, $\sigma(v,i) + p_v/s_i + \rho$ is at most $T^{(\varsigma_\ell)}$, and no job was placed after the $(\ell-1)$th execution of line 12 and before the ℓth execution of line 12. If Placed equals V before the ℓth execution of line 2, then there is nothing to prove since then there is no ℓth execution of line 12. Otherwise, there is a job v not placed, all of whose ancestors already have been placed. Then, in an execution of line 6 between the $(\ell-1)$th and ℓth executions of line 12, A is set to $\{v\}$, ensuring that all conditions of line 7 are satisfied and v is placed, yielding a contradiction. □

Lemma 4.9 (Correctness and Runtime). Algorithm 1 outputs a valid schedule in $\mathsf{poly}(n,m)$ time.

Proof. We show that precedence and communication delay requirements are satisfied. It is straightforward to establish that remaining requirements (given in Section 2) are met. We show first that precedence requirements are met. This follows from the fact that when any job is placed on a machine, all of its uncompleted predecessors are placed in topological order on the same machine. Therefore, for any job v with predecessor u, the start time of u is less than the start time of v. We now show that communication delay requirements are met. This follows from the fact that when Algorithm 1 places a set of jobs on a machine i starting at time t, it duplicates any jobs that have not completed previously on i or have not completed by time $t-\rho$ on any other machine.

We now show that Algorithm 1 runs in time polynomial in n and m. We first argue that there are at most $2m(n-1)+2$ executions of line 2. For the sake of contradiction, suppose there is an $(2m(n-1)+3)$th execution of line 2. By Lemma 4.8, if ς represents the $(2m(n-1)+1)$th execution of line 12, then $T^{(\varsigma)}$ equals the $(2m(n-1)+1)$ smallest element of $\mathcal{T}$. Since every job v has at most $2m$ finite values of the form $\sigma(v,i) + p_v/s_i$ or $\sigma(v.i) + p_v/s_i + \rho$ in $\mathcal{T}$, it follows from the pigeon hole principle that every job v has at least one finite value of the form $\sigma(v,i) + p_v/s_i$ that is at most the $(2m(n-1)+1)$ smallest element of $\mathcal{T}$, indicating that every job has been placed before the $(2m(n-1)+1)$th execution of line 12. This ensures that Placed equals V no later than the $(2m(n-1)+2)$th execution of line 2, after which the algorithm terminates, leading to a contradiction. We thus obtain that line 2 iterates at most $O(nm)$ times. All other loops iterate at most n or m times, and all other instructions take at most time nm. □

The following lemma establishes that, for $u \prec v$ and a gap that occurs before v and sufficiently long after u, there are steps in the algorithm that consider placing v and its remaining predecessors A to start inside the gap such that $u \notin A$. This lemma is used in the proof of Lemma 4.13.

Lemma 4.10 (Predecessor removal in gaps). Let $u \prec v$, let c be the earliest completion time of u in σ, let b be the earliest start time of v in σ, and suppose that $b > c + \rho$. If there is some gap (g_1, g_2) on any machine in $\Gamma_{\kappa(v)}$ such that $g_1 < b$ and $g_2 \geq c + \rho$, then there is step ς checking the conditions of line 7 such that $v^{(\varsigma)} = v$, $i^{(\varsigma)} \in \Gamma_{\kappa(v)}$, $T^{(\varsigma)}_{i^{(\varsigma)}} = \max\{g_1, c+\rho\}$, and $u \notin A^{(\varsigma)}$.

Figure 3 depicts an instance for which the conditions of Lemma 4.10 are met.

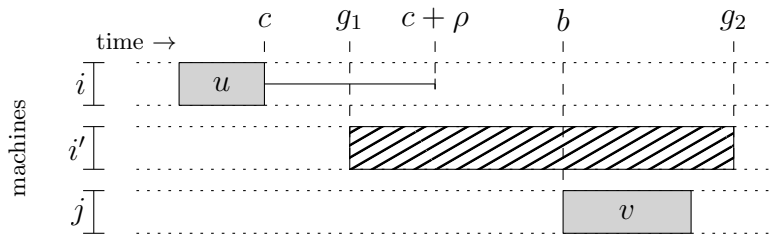


Figure 3: A depiction of jobs u and v for which the conditions of Lemma 4.10 are met, given that $i' \in \Gamma_{\kappa(v)}$. Machines are listed vertically on the left and time proceeds from left to right. The jobs u and v are shown as gray boxes, and the gap (g_1, g_2) is shown as a diagonal striped box.

Proof. Suppose there are jobs u and v and a gap (g_1, g_2) on machine j satisfying the conditions of the lemma. Let $c_{\max} = \max\{c + \rho, g_1\}$ and let $b_{\min} = \min\{g_2, b\}$. By definition of a gap we have that $(c_{\max}, b_{\min})$ is a gap. Also, since c and g_1 is both completion times of some jobs, $c_{\max}$ is in $\mathcal{T}$. Therefore, by Lemma 4.8, there is a step ς executing line 12 , immediately before which T equals $c_{\max}$. Since T does not change between executions of line 12, and line 4 iterates through all jobs between two consecutive executions of line 12, there exists a step $\varsigma^* < \varsigma$ executing line 7 for which $T^{(\varsigma^*)} = c_{\max}$ and $v^{(\varsigma^*)} = v$. By the assignment of i at line 5, we have that $i^{(\varsigma^*)} \in \Gamma_{\kappa(v)}$. The fact that $(c_{\max}, b_{\min})$ is a gap entails that the max completion time of any job placed on j prior to step ς^* is at most $c_{\max}$. Therefore, $T_j^{(\varsigma^*)} = T^{(\varsigma^*)}$ by Lemma 4.7, so $T_{i^{(\varsigma^*)}}^{(\varsigma^*)} = T^{(\varsigma^*)}$ by the assignment at line 5. Finally, $u \notin A^{(\varsigma^*)}$ follows from the assignment of A at line 6 and the fact that $c + \rho \leq c_{\max} = T_{i^{(\varsigma^*)}}^{(\varsigma^*)}$. □

Lemma 4.11 (Total load within η factor of total job size). Let $V(i)$ be the set of all jobs with some copy scheduled on machine i. Then, for any k,

$$\sum_{k' \geq k} \sum_{i \in \Gamma_{k'}} p(V(i)) \leq \eta \cdot p(\kappa_{\geq}^{-1}(k)).$$

Proof. Let $\varsigma_1, \varsigma_2, \ldots$ be the series of steps such that ς_d is the first execution of line 9 when placing the set $A^{(\varsigma_d)}$ on any machine in $\bigcup_{k' \geq k} \Gamma_k$. Let $V(i, \varsigma)$ be the set of jobs placed on machine i prior to executing step ς. We show by induction on d^* that

$$\lambda(d^*) \equiv \sum_{d=1}^{d^*} \sum_{k' \geq k} \sum_{i \in \Gamma_{k'}} p(V(i, \varsigma_d)) \leq \eta \cdot p\Bigg(\bigcup_{d=1}^{d^*} \bigcup_{k' \geq k} \bigcup_{i \in \Gamma_{k'}} V(i, \varsigma_d) \Bigg). \tag{13}$$

Since $\bigcup_{k' \geq k} \bigcup_{i \in \Gamma_{k'}} V(i) \subseteq \kappa_{\geq}^{-1}(k)$ by condition **(c)** in line 7, inequality (13) is sufficient to prove the lemma. Inequality (13) holds trivially for $d^* = 1$, so we suppose for induction that it holds up to step

$d^* \geq 1$. Then

$$\begin{aligned}
\lambda(d^*+1) &= \lambda(d^*) + p(A^{(\varsigma_{d^*+1})}) && \text{by definition of } \lambda \\
&\leq \lambda(d^*) + \eta \cdot p(A^{(\varsigma_{d^*+1})} \setminus \mathsf{Placed}^{(\varsigma_{d^*+1})}) && \text{by condition } \textbf{(b)} \\
&\leq \eta \cdot p\Bigg(\bigcup_{d=1}^{d^*} \bigcup_{k' \geq k} \bigcup_{i \in \Gamma_{k'}} V^{(\varsigma_d)}(i)\Bigg) + \eta \cdot p(A^{(\varsigma_{d^*+1})} \setminus \mathsf{Placed}^{(\varsigma_{d^*+1})}) && \text{by induction} \\
&\leq \eta \cdot p\Bigg(\bigcup_{d=1}^{d^*+1} \bigcup_{k' \geq k} \bigcup_{i \in \Gamma_{k'}} V^{(\varsigma_d)}(i)\Bigg)
\end{aligned}$$

where the last inequality follows from the fact that for all jobs $v \in \mathsf{Placed}^{(\varsigma_{d^*+1})}$, v is either an element of $\bigcup_{d=1}^{d^*} \bigcup_{k' \geq k} \bigcup_{i \in \Gamma_{k'}} V^{(\varsigma_d)}(i)$ or placed on machines in groups lower indexed than k. □

4.5 Analysis

In this section, we prove that our algorithm yields an approximation ratio with respect to the optimal makespan C^*. We assume that we have a preprocessed instance in which all machines of speed less than $1/m$ have been removed, that each job v has been assigned to a group $\kappa(v)$ via the rounding of Section 4.3 based on an optimal solution to the linear program LP of Section 4.2 with makespan $C \leq C^*$, and that these assignments have been given as input to Algorithm 1.

Our analysis combines elements of the analysis given in [10] and [20]. As in [10], we define a chain of jobs whose execution time serves as a lower bound on the optimal length of any schedule. The chain also has the following property: for any non-chain job v that precedes a job in the chain and is not executed in parallel with the chain, if v is scheduled on any machine i then v's execution time on i is no longer than 8ρ. This property allows us to leverage the analysis from [20] to show that the time spent in σ not executing the chain can be bounded by the highest band number and the load on all groups, both of which give lower bounds on the optimal makespan.

We define the *chain* $\mathscr{C} = \{c_1, c_2, \ldots, c_{|\mathscr{C}|}\}$ as follows. Let D be the set of pairs (v, i) such that σ schedules some copy of v on i. Let L be the set of pairs $(v, i) \in D$ such that $p_v/s_i > 8\rho$. Then

$$\begin{aligned}
c_1 &= \arg\max_{(v,i) \in L} \{\sigma(v, i) + p_v/s_i\} \\
c_{q+1} &= \arg\max_{(v,i) \in L} \{\sigma(v, i) + p_v/s_i : c_q = (u, j) \text{ and } v \in A_u\}.
\end{aligned}$$

We also define sets of jobs that precede jobs in the chain and execute between jobs in the chain. Let V_0 be the set of all jobs that complete after c_1, i.e. for $c_1 = (i, v)$, V_0 is the set of $u \in V$ such that, for some machine j we have $(u, j) \in D$ and $\sigma(u, j) + p_u/s_j \geq \sigma(v, i) + p_v/s_i$. Then, for $0 < q < |\mathscr{C}|$, let V_q be the set of predecessors of c_q that complete after c_{q+1}, i.e. for $c_q = (v, i)$, V_q is the set of all jobs u such that $u = v$ or $u \prec v$ and for some machine j, $(u, j) \in D$ and $\sigma(u, j) + p_u/s_j \leq \sigma(v, i)$ and $\sigma(u, j) + p_u/s_j \geq \sigma(v', i') + p_{v'}/s_{i'}$ where $c_{q+1} = (v', i')$. Finally, let $V_{|\mathscr{C}|}$ be the set of jobs that precede $c_{|\mathscr{C}|}$. Figure 4 depicts the construction of $\mathscr{C}$ and each V_q.

We divide the schedule into phases of length ρ where phase $\tau = [\rho(\tau - 1), \rho\tau)$. We say that a machine i is *busy* in phase τ if the total time spent executing jobs on i in phase τ is at least $\rho/2$. Otherwise, we say that i is idle in τ. We classify the phases into three different types. Phase τ is a *chain phase* if at least $\rho/2$ time is spent working on any job in the chain on any machine. A c_q-phase is a chain phase in which at least

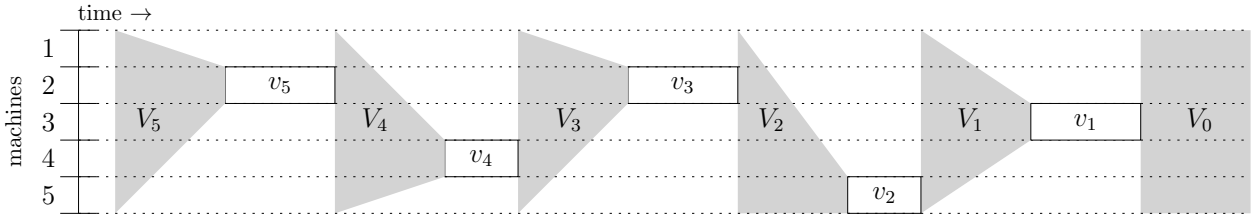


Figure 4: Machines are shown vertically on the left and time increases from left to right. The dotted lines track the jobs being executed on each machine. Jobs v_1 through v_5 are the only jobs with copies that take longer than ρ time. Other jobs are scheduled but not shown. In this case, $\mathscr{C} = \{c_1 : (v_1, 3), c_2 : (v_2, 5), c_3 : (v_3, 2), c_4 : (v_4, 4), c_5 : (v_5, 2)\}$. The sets V_q are shown as shaded regions. All jobs $u \in A_{v_q}$ such that some copy of v is scheduled on i and some completion time of v on i falls inside the shaded trapezoidal region to the left of v_q are elements of V_q.

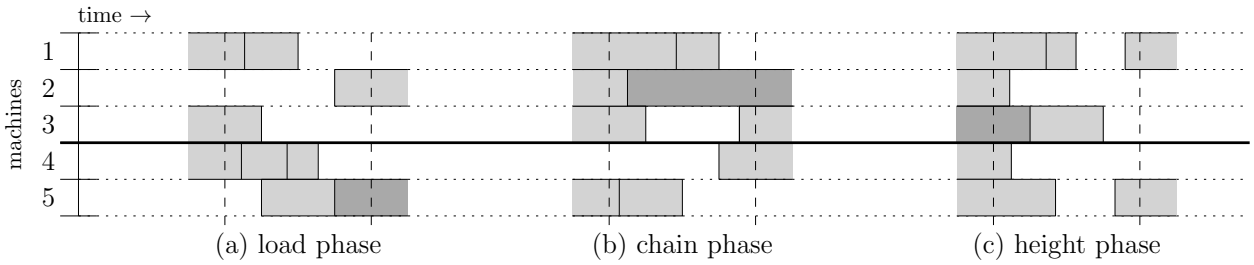


Figure 5: Phase types. Machines are shown vertically on the left and time proceeds from left to right. There are two groups of machines $\Gamma_1 = \{1, 2, 3\}$ and $\Gamma_2 = \{4, 5\}$, separated by a dark line. Chain jobs are shown in dark gray and all other jobs are shown in light gray. For each phase, the time between two dashed lines is ρ. (a) A load phase: all machines in group $\{4, 5\}$ are busy for at least $\rho/2$ time in the phase. (b) A chain phase: some element in the chain is executing for at least $\rho/2$ time in the phase. (c) A height phase: all groups have some machine that is busy for less than $\rho/2$ time in the phase: machine 2 in Γ_1 and machine 4 in Γ_2.

$\rho/2$ time is spent working on c_q. Phase τ is a *load phase* if τ is not a chain phase and every machine in some group is busy in τ. The remaining phases are height phases. The different phase types are illustrated in Figure 5.

Lemma 4.12 (Chain Phase Bound). The number of chain phases is $O(C^*/\rho)$.

Proof. First we observe that for any job v,

$$\sum_i \frac{x_{v,i}}{s_i} \;\geq\; \sum_{i \leq \kappa(u)} \frac{x_{v,i}}{s_i} \;\geq\; \sum_{i \leq \kappa(u)} \frac{x_{v,i}}{2\overline{s}_{\kappa(v)}} \;=\; \frac{1}{2\overline{s}_{\kappa(v)}} \sum_{i \leq \kappa(u)} x_{v,i} \;\geq\; \frac{1}{4\overline{s}_{\kappa(v)}}.$$

Recall that $\mathscr{C} = \{c_1, \ldots, c_{|\mathscr{C}|}\}$. Let $c_q = (v_q, i_q)$. Let us sum up constraint (2) for $v = v_1, ..., v_{|\mathscr{C}|-1}$, and, respectively, $u = v_2, ..., v_{|\mathscr{C}|}$, as well as constraint (1) for v_1. We get

$$C^* + \sum_{q=1}^{|\mathscr{C}|-1} S_{v_q} \;\geq\; \sum_{q=1}^{|\mathscr{C}|} S_{v_q} + \sum_{q=1}^{|\mathscr{C}|} p_{v_q} \sum_i \frac{x_{v_q,i}}{s_i} \;\geq\; \sum_{q=1}^{|\mathscr{C}|} S_{v_q} + \sum_{q=1}^{|\mathscr{C}|} p_{v_q} \frac{1}{4\overline{s}_{\kappa(v)}},$$

where the second inequality uses the bound above. This implies that $C^* \geq S_{v_{|\mathscr{C}|}} + \sum_{v \in \mathscr{C}} \frac{p_v}{4\overline{s}_{\kappa(v)}}$, and thus $\sum_{v \in \mathscr{C}} \frac{p_v}{\overline{s}_{\kappa(v)}} \leq 4C^*$.

Therefore, in order to prove the lemma, all that remains is to show that all chain jobs are scheduled on the groups to which they are assigned. Specifically, we show that if σ schedules job v on machine i and $p_v > 8\rho s_i$, then $i \in \Gamma_{\kappa(v)}$. Suppose Algorithm 1 places job v on machine $i \in \Gamma_k$ and $p_v/s_i > 8\rho$. We fix the step ς to the last execution of line 7 prior to placing v on i and show that $v^{(\varsigma)} = v$. Suppose otherwise. Then $v \in A^{(\varsigma)} \setminus \{v\}$. By condition **(a)** we have that $8\rho\overline{s}_k \geq A^{(\varsigma)} \setminus \{v\} \geq p_v$. This, however, contradicts our supposition that $(v, i) \in \mathscr{C}$. Therefore, $v = v^{(\varsigma)}$ which entails, by selection of v at line 4, that $i \in \Gamma_{\kappa(v)}$. □

We define *long* jobs be those jobs v such that σ schedules some copy of v on any machine i and $p_v/s_i > 8\rho$. All jobs that are not long are *short*.

Lemma 4.13 (Height Phase Bound). The number of height phases is $O(K(C^* + \rho) \cdot \log_\eta(\rho)/\rho)$.

Proof Idea. We provide a high level proof idea. Consider any two consecutive chain jobs, such as v_4 and v_3 in Figure 4. The goal is to upper bound the number of height phases which occur between these chain jobs, in this case after the completion of v_4 on machine 4 and before the start of v_3 on machine 2. So we consider such a height phase τ. Because of the way the algorithm introduces gaps in the schedule, all jobs placed while $T < t$, for some time t, will start their execution before the next height phase after time t. So we focus on the steps of the algorithm in which $T \leq \tau\rho$. By the fact that all jobs (with the exception of v_3) in V_3 are short, if all jobs in V_3 are placed on any group during these steps, then v_3 starts on machine 2 within 8 height phases phases after τ. So we can assume that some job in V_q has not been placed on any group during these steps.

Consider all such jobs that are in the lowest band according to the LP values (see Definition 4.4) and, among those, pick a job v which is assigned to the slowest machine group Γ_k. We know that group Γ_k has a machine with idle time in phase τ, so the only reason v is not placed on some machine in Γ_k during these steps is that it fails to meet one of the three conditions given in line 7 of Algorithm 1. We can infer that v meets condition **(a)** by virtue of being in the lowest band and by Lemma 4.5. We can also infer that v meets condition **(c)** by virtue of being assigned to the slowest group. Thus, it must fail condition **(b)**. This means that a $(1-1/\eta)$ fraction of its incomplete predecessors has already been placed. By definition of V_q, all these predecessors must be short. Since these predecessors have all been placed on machines prior to checking v, these predecesors will begin before next height phase and have time to complete and communicate $O(1)$ phases after that. By Lemma 4.5 and scaling of speeds and job sizes, v has at most 8ρ predecessors in V_q. So, in $O(\log_\eta \rho)$ phases, they all complete and v can be scheduled. Repeating this argument for all K groups and all $O((C^* + \rho)/\rho)$ bands gives the bound in the lemma. We now present the complete proof.

Proof. We specify a sequence of phases in σ that will structure our argument. We let τ_0 denote the first phase of the schedule. For $1 \leq q \leq |\mathscr{C}|$, we let τ_q denote the first phase in which v is scheduled on i, where chain element $c_q = (v, i)$. We let $\tau_{|\mathscr{C}|+1}$ denote the last phase in which some job is started, and $\tau_{|\mathscr{C}|+2}$ to be the last phase in σ. We also define the function ϕ as follows, where $r_{\max} = \max\{r : B_r \neq \varnothing\}$.

$$\phi(q) = \begin{cases} 0 & \text{if } q = 0 \\ r & \text{if } 1 \leq q \leq |\mathscr{C}| \text{ and } c_{|\mathscr{C}|-(q-1)} = (v, i) \text{ and } v \in B_r \\ r_{\max} + d & \text{if } q = |\mathscr{C}| + 1 + d \text{ with } d \in \{0, 1\}. \end{cases}$$

Intuitively, ϕ provides us a way to to reason over any pre-chain bands ($\phi(0)$ to $\phi(1)$), those bands of jobs between chain elements ($\phi(q)$ to $\phi(q+1)$), any post-chain bands ($\phi(|\mathscr{C}|)$ to $\phi(|\mathscr{C}|+1)$), and the final

runtime of any jobs once all jobs have been started ($\phi(|\mathscr{C}|+1)$ to $\phi(|\mathscr{C}|+2)$). Finally, we let H_q be the number of height phases up to and including τ_q. We argue that, for any $0 \leq q \leq |\mathscr{C}|+2$,

$$H_q \leq \phi(q) \cdot K \log_\eta \rho. \tag{14}$$

For any job $v \in B_{r_{\max}}$, we have that $S_v \geq \rho(r_{\max}-1)/4$ by definition of $B_{r_{\max}}$. So $\rho r_{\max} = 4\frac{\rho(r_{\max}-1)}{4} + \rho \leq 4S_v + \rho \leq 4(C^* + \rho)$ by Constraint (1). So proving (14) is sufficient to prove the lemma.

We prove (14) by induction on q. The inequality holds trivially for $q = 0$, so we suppose it holds for some $0 \leq q \leq |\mathscr{C}|+1$. Since $H_{q+1} = H_q + (H_{q+1} - H_q)$, by induction it suffices to show that $H_{q+1} - H_q \leq (\phi(q+1) - \phi(q)) \cdot K \log_\eta \rho$.

We first consider the case where $0 \leq q \leq |\mathscr{C}|$. Recall that τ is a height phase if, for every k, there is some $i \in \Gamma_k$ that is idle in τ. We consider an arbitrary band B_{r^*} such that $\phi(q-1) < r^* \leq \phi(q)$ and an arbitrary group Γ_{k^*}. Let τ^* be the last phase during which any job in $B_{r^*-1} \cap \kappa_{\geq}^{-1}(k^*) \cap V_q$ is started. (Recall the sets V_q from Figure 4.) Note that, if $q > 0$ then τ^* is after the last chain phase for c_q. For a given phase τ, we define $\varsigma(\tau)$ to be the set of steps ς of Algorithm 1 for which $T^{(\varsigma)} \leq \tau\rho$. Let v be some job in $B_{r^*} \cap \kappa_{\geq}^{-1}(k^*) \cap V_q$ such that no copy of v is placed on any machine in any step $\varsigma \in \varsigma(\tau^*)$. Let τ_v be the phase in which the first copy of v is started. We define the sequence $\tau_1^*, \tau_2^*, \ldots$ where $\tau_0^* = \tau^*$ and

$$\tau_{d+1}^* = \min_{\tau}\{\tau < \tau_v : \exists \tau' > \tau_d^* \text{ such that } \tau \text{ and } \tau' \text{ are height phases and } \tau \geq \tau' + 10\}.$$

We prove that the length of this sequence is $O(\log_\eta \rho)$ by establishing the following claim: for all $d \geq 1$ there is some step ς of line 7 such that $v^{(\varsigma)} = v$, $i^{(\varsigma)} \in \Gamma_{k^*}$, $T_{i^{(\varsigma)}}^{(\varsigma)} \leq \rho\tau_d^*$ and $p(A^{(\varsigma)}) \leq 8\rho\overline{s}_{k^*}/\eta^d$, i.e. while scheduling each phase in the sequence, there is a check to schedule v on some machine in $\Gamma_{\kappa(v)}$ during which the total size of v's uncompleted predecessor set has been reduced by a factor of η. The claim entails that for every step ς executing line 7 in which $v^{(\varsigma)} = v$, $i^{(\varsigma)} \in \Gamma_{k^*}$, and $T^{(\varsigma)} \geq \rho\tau_{1+\log_\eta \rho}^*$, we have that $p(A^{(\varsigma)}) = 0$. Therefore, by Lemma 4.10, v is scheduled before (or during) the height phase immediately after $\tau_{1+\log_\eta \rho}^*$. Because we have chosen an arbitrary group, this entails that, if $c_q = (v^*, i)$, then all predecessors of v^* in B_{r^*} are started within $10K \log_\eta \rho$ height phases after τ^*. Also, because we have chosen an arbitrary band, all bands r such that $\phi(q) \leq r \leq \phi(q+1)$, are started in $(\phi(q+1) - \phi(q)) \cdot K \log_\eta \rho$ height phases after the first chain phase of c_{q+1}. Therefore, $H_{q+1} - H_q \leq 10 \cdot (\phi(q+1) - \phi(q)) \cdot K \log_\eta \rho$.

We prove the claim by induction on d. By definition of r^*, all jobs in band $r^* - 1$ are placed during steps $\varsigma(\tau^*)$. Note that, since $c_{q+1} = (u, j)$ was the last long predecessor of v to complete, none of v's predecessors placed after c_q are long. Since these predecessors have all been placed while $T \leq \tau^*\rho$ and since gaps are introduced only when T exceeds some T_i at line 13, all these predecessors have started before (or during) the next height phase. This entails that all are completed at least ρ time before the start of phase τ_1^*. Since there is some gap on some machine in phase τ_1^*, by Lemmas 4.5 and 4.10, we have that there is some step ς executing line 7 in which $v^{(\varsigma)} = v$, $i^{(\varsigma)} \in \Gamma_k$, $T_{i^{(\varsigma)}}^{(\varsigma)} \leq \rho\tau^*$, and $p(A^{(\varsigma)}) \leq 8\rho\overline{s}_{\kappa(v)}$. This proves the base case.

We now prove the claim for $d+1$ assuming it holds for d. Consider phase τ_d^* and let ς be any step executing line 7 in which $v^{(\varsigma)} = v$, $i^{(\varsigma)} \in \Gamma_{k^*}$, $T_{i^{(\varsigma)}}^{(\varsigma)} \leq \rho\tau_d^*$ and $p(A^{(\varsigma)}) \leq 8\rho\overline{s}_{k^*}/\eta^d$, which exists by induction. By definition of the sequence and our selection of v, we know that conditions **(a)** and **(c)** are met in step ς. Since v was not placed in step ς it must be that condition **(b)** was not met. This entails that $p(A^{(\varsigma)} \setminus \mathsf{Placed}^{(\varsigma)}) < p(A^{(\varsigma)})/\eta$. Since all jobs in $A^{(\varsigma)} \cap \mathsf{Placed}^{(\varsigma)}$ are short and all are placed while $T \leq \rho\tau_d^*$, we have that they all start before the next height phase and complete at least ρ time before the start of phase τ_{d+1}^*. Therefore, by Lemma 4.10, we have that there exists a step ς' in which $v^{(\varsigma')} = v$,

$A^{(\varsigma')} \cap A^{(d)} \cap \mathsf{Placed}^{(d)} = \varnothing$, $i^{(\varsigma')} \in \Gamma_{k^*}$, and $T^{(\varsigma')}_{i^{(\varsigma')}} \leq \rho\tau^*_{d+1}$. This entails that $A^{(\varsigma')} \leq p(A^{(\varsigma)})/\eta$, which proves the inductive step.

Finally, we consider the case where $q = |\mathscr{C}| + 1$. We show that $H_{q+1} - H_q = O(1)$. By definition, all jobs have been started by the end of phase τ_q. Note that if any of these jobs are chain jobs then the remainder of the phases, with the possible exception of the last, are chain phases, so $H_{q+1} - H_q \leq 2$. On the other hand, if no remaining jobs are chain jobs, then all remaining jobs take less than 8ρ time on the machines where they've been scheduled, so $H_{q+1} - H_q \leq 8$. This completes the proof of inequality (14). □

Lemma 4.14 (Load Phase Bound). The number of load phases is $O(KC^*\eta/\rho)$.

Proof. For this proof, we assume that $\Gamma_1, \ldots, \Gamma_K$ is the set of groups Γ_k for which $\kappa^{-1}(k) \neq \varnothing$. By definition of κ, this entails that $m_k\overline{s}_k \geq m_{k+1}\overline{s}_{k+1}$. Note that the algorithm does not schedule any jobs on the other groups.

We begin with a useful technical claim. Suppose that we wanted to find values $x_1, ..., x_K \geq 0$ that optimize the following linear program:

$$\text{maximize} \sum_k \frac{x_k}{m_k\overline{s}_k} \quad \text{subject to} \quad \forall k' : \sum_{k \geq k'} x_k \leq \eta \cdot \sum_{k \geq k'} p(\kappa^{-1}(k)). \tag{15}$$

Note that all $p(\kappa^{-1}(k)) > 0$. We claim that setting the variables in such a way that all these constraints are tight, namely $x^*_k = \eta \cdot p(\kappa^{-1}(k))$, is the optimal solution. For contradiction, assume that there is some other solution that gives a higher objective, and let $x'_1, ..., x'_K$ be the lexicographically smallest optimal solution. Note that if $x'_k = 0$ for some k, then the constraint for $k' = k$ is not tight, since the constraint bounds strictly increase as k' decreases. If $x'_1 = 0$, then we can increase x'_1 and improve the objective. Otherwise, let k^* be such that the constraint for k^* is not tight and $x'_{k^*-1} > 0$. Then we can increase x'_{k^*} by a small value ε and decrease x'_{k^*-1} by the same amount. This is neutral for all constraints except k^*, and the objective increases by $\varepsilon/m_{k^*}\overline{s}_{k^*} - \varepsilon/m_{k^*-1}\overline{s}_{k^*-1} \geq 0$, since $m_{k^*-1}\overline{s}_{k^*-1} \geq m_{k^*}\overline{s}_{k^*}$. Thus, the solution x' is either not optimal or not lexicographically smallest, which proves the claim.

Let $V(i)$ be the set of jobs scheduled on machine i. Now, the values $x_k \equiv \sum_{i \in \Gamma_k} p(V(i))$ constitute a feasible solution to (15), with Lemma 4.11 showing that they satisfy the constraints. Thus, their objective value is at most that of x^*:

$$\sum_k \sum_{i \in \Gamma_k} \frac{p(V(i))}{m_k\overline{s}_k} \leq \sum_k \frac{\eta \cdot p(\kappa^{-1}(k))}{m_k\overline{s}_k}. \tag{16}$$

Let L_k be the number of load phases in which group k is the slowest group with all machines busy for at least $\rho/2$ time. The amount of time that the machines in Γ_k are busy during such phases is at least $\frac{\rho}{2} \cdot m_k \cdot L_k$. The total amount of time that the machines in Γ_k are busy is $\sum_{i \in \Gamma_k} p(V(i))/s_i$, so

$$\frac{\rho}{2} m_k L_k \;\leq\; \sum_{i \in \Gamma_k} \frac{p(V(i))}{s_i} \;\leq\; \sum_{i \in \Gamma_k} \frac{p(V(i))}{\overline{s}_k}.$$

Thus, for all k,

$$L_k \;\leq\; \frac{2}{\rho} \sum_{i \in \Gamma_k} \frac{p(V(i))}{m_k\overline{s}_k}.$$

Summing over k, using (16), and applying Lemma 4.6, the total number of load phases is

$$\sum_k L_k \;\leq\; \frac{2}{\rho} \sum_k \sum_{i \in \Gamma_k} \frac{p(V(i))}{m_k\overline{s}_k} \;\leq\; \frac{2}{\rho} \sum_k \frac{\eta \cdot p(\kappa^{-1}(k))}{m_k\overline{s}_k} \;=\; O(KC^*\eta/\rho),$$

which proves the lemma. □

Theorem 1 (**Makespan approximation).** There is a polynomial time algorithm that, given an instance of the DAG scheduling with fixed communication delay problem, computes a schedule whose makespan is $O(\log m \log \rho / \log\log \rho)(C^* + \rho)$, where C^* is the optimal makespan for the given instance.

Proof. We set $\eta = \log \rho / \log\log \rho$. By Lemma 4.13, we have the number of height phases is $O(K(C^* + \rho) \log \rho / \log\log \rho)$. By Lemma 4.14 we have that the number of load phases is $O(C^* K \log \rho / \log\log \rho)$. By Lemma 4.12 we have that the number of chain phases is $O(C^*)$. Since there are no other phases, if C is the makespan of σ we have

$$C = O(K(C^*+\rho)\log\rho/\log\log\rho) + O(C^*K\log\rho/\log\log\rho) + O(C^*) = O(K\log\rho/\log\log\rho)(C^*+\rho).$$

Since the number of groups $K \leq \log m$, this yields a bound of $O(\log m \log \rho / \log\log \rho)(C^* + \rho)$. □

5 Integrality gap of $\Omega(\sqrt{\log \rho})$

The main result of this section is the development of an instance of the problem for which the integrality gap of LP is $\Omega(\sqrt{\log \rho})$.

Theorem 2 (**Integrality gap).** There is a family of instances such that for any ρ that is at least some sufficiently large constant, the linear programming relaxation LP has a gap of at least $\Omega(\sqrt{\log \rho})$.

Input Instance The input to the scheduling problem is a layered directed acyclic graph G (see Figure 6). Each vertex in G represents a unit size job. All machines are of unit speed so that every job can be processed at any machine in unit time. The instance is parametrized by $d, L \in \mathbb{N}$, where L is the number of levels, and d is the number of immediate predecessors for any job at any of the levels 1 through $L-1$. We set the communication delay $\rho = d^L$. The set V_i of vertices at level i has size $n = m\rho/L$, where m is the number of machines. Finally, we specify the precedence constraints. Between any two levels V_i, V_{i+1} we have a random graph G_i that is bipartite and d left-regular (for each vertex in V_i, d random vertices in V_{i+1} are chosen as its neighbors). For our purposes, it will suffice to pick $L = \varepsilon_1\sqrt{\log n}$, $d = 2^{\varepsilon_2\sqrt{\log n}}$ and $m = \rho$, for some appropriate absolute constants $\varepsilon_1, \varepsilon_2 > 0$. In particular, for our setting we have $\log \rho = \Theta(\log n)$; so our gap will be $\Omega(\sqrt{\log n})$.

Overview of Challenges and Proof Outline We construct a new integrality gap instance that achieves a $\omega(1)$ integrality gap in the presence of delays. The gap construction consists of a layered DAG (with $L = \Theta(\sqrt{\log n})$ layers), where each layer corresponds to a random graph with (left)-degree $d = 2^{\Theta(\sqrt{\log n})}$. The parameters of the construction are set up in such a way that fractionally, all the jobs can be assigned in one phase (hence the LP solution value is 1).

The main technical challenge is to argue that $\Omega(L)$ phases are needed in order to schedule all the jobs. From the expansion of the random graph in each layer, it is easy to argue that at most $o(1)$ fraction of the jobs (in layers $\{1, \ldots, L-2\}$) can be scheduled in the first phase (since at most $\rho \ll n$ of the jobs can be on one machine). However, in the next phase the jobs that were scheduled previously are now available in all the machines; moreover the choice of these jobs could depend on the randomness in the DAG. Hence the remaining graph in each layer (after removing vertices that have already been scheduled) in the subsequent phases is *not random* any longer!

To overcome this technical hurdle, we identify and exploit the property of *robust expansion*, which may be of independent interest. The vertex expansion property of a random graph says that w.h.p. any subset

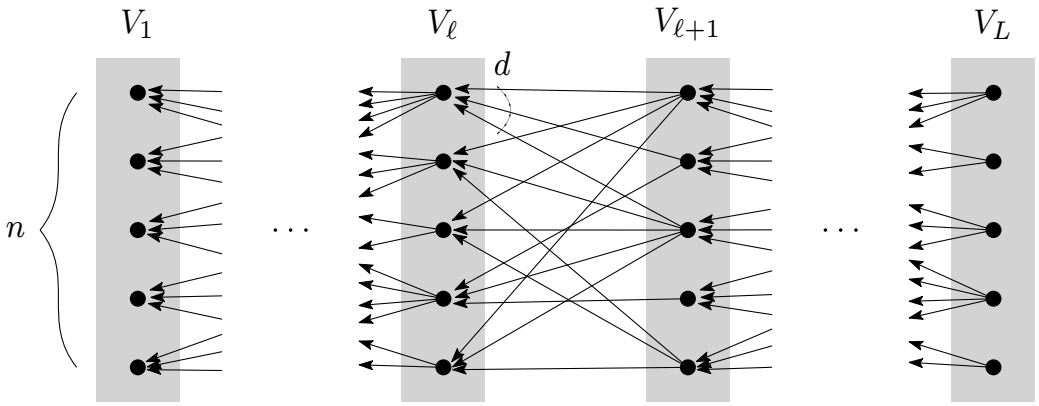


Figure 6: The figure shows the DAG with L layers $V_1, \ldots, V_L$ each with n vertices, representing the nL jobs. Each of the n jobs in V_ℓ has dependencies on d randomly chosen jobs in $V_{\ell+1}$.

$S \subset V_\ell$ of size $|S| \leq n/d$ has a neighborhood of size $|\Gamma(S)| = \Omega(d|S|)$. However, random graphs have the stronger property that no subset T of size $o(d|S|)$ can have $\Omega(d|S|)$ of the edges from S incident on it. This property of robust expansion along with its generalization to paths of length $\ell < L$ (in Lemma 5.3) is crucial in the analysis. This property will allow us to prove that we need at least $L/2$ phases before most of the jobs in V_1 can be scheduled.

Recall that the relaxation in Section 4 tries to minimize the number of phases, where each phases corresponds to roughly ρ time units (see Lemma 4.2). In the rest of the section, we will measure the length of the schedule in terms of the number of phases. We first show that the LP has value at most 1 in Lemma 5.1. Then we prove the necessary robust expansion properties of the instance in Section 5.1, and then use this property to lower bound the number of phases in the optimal schedule by $\Omega(L)$ in Proposition 5.4. Theorem 2 follows directly by just combining Lemma 5.1 and Proposition 5.4.

Upper bound on the LP solution value.

We begin by proving that the LP has a fractional solution of value at most ρ.

Lemma 5.1. The linear program LP has a value of at most ρ.

Proof. We distribute each job uniformly across all machines, so $x_{v,i} = 1/m$ for every v and every i. Set $z_{u,v,i} = i/m$ for all u, v, i. Finally the variables C_v, S_v respect the layered structure of the instance; we set $C_v = (L - \ell + 1)\rho/L, S_v = (L - \ell)\rho/L$ for all $v \in V_\ell$, and set $C = \rho$. Constraints (1) and (2) are immediate, since our assignments of C_v, S_v satisfy the layer structure of the DAG (note all the $p_u, s_i = 1$). Constraint (5) is satisfied with equality since $m = nL/\rho$. Constraint (3) is satisfied since $\sum_{j \leq i} x_{v,j} = z_{u,v,i}$ for all u, v, i; this combined with $|A_v| \leq \rho$ also establishes (4). The other constraints are easily seen to hold, completing the proof of the desired claim about LP. □

The remainder of this section establishes that there is a setting of the parameters for which an optimal integral schedule has makespan $\Omega(\sqrt{\log \rho})$. For our purposes, $L = \varepsilon_1 \sqrt{\log n}$, $d = 2^{\varepsilon_2 \sqrt{\log n}}$ and $m = \rho$, for some appropr absolute constants $\varepsilon_1, \varepsilon_2 > 0$. We begin by establishing certain robust expansion properties of G in Section 5.1, and then prove in Section 5.2 the lower bound on the makespan of any integer solution.

5.1 Robust expansion properties

In the rest of this section, we will ignore the directionality of the edges, and treat the graphs $\{G_i\}$ between layers as undirected graphs for convenience. The following lemma shows that each of the layers G_i has a certain *robust expansion* property, that states that for any $S \subset V_i$ and any $T \subseteq \Gamma(S)$ that has a constant fraction of the edges from S incident on it should be of size $\Omega(d|S|)$.

Lemma 5.2. For every $i \in [L-1]$, there is an absolute constant $c_1 \leq 8$ such that the following holds with probability $1 - o(1)$: for every set $S \subset V_i, T \subset V_{i+1}$ with $|S| \leq n/d^2$ and $|E(S,T)| \leq c_1|S| + c_1|T|$.

Proof. Let $\mathcal{E}$ represent the event that there exists S, T satisfying $|E(S,T)| \geq c_1|S|$ and $|E(S,T)| \geq c_1|T|$. Fix such a S, T. Let $m := |E(S,T)| \geq c_1|S|$. We can also assume without loss of generality that $|T| := t = m/c_1$; note that $t \leq ds \leq n/d$. Also fix an index set $J \subseteq \{1, \ldots, d|S|\}$ with $|J| = m$, to represent a subset of edges incident on S, and let E_J represent the corresponding edges. The probability that the edges corresponding to E_J are all incident on T is at most $(|T|/n)^m$. By a union bound over the choice of the index set J, we have

$$\Pr[|E(S,T)| \geq m] = \Pr\left[\exists J \subset \{1, \ldots, d|S|\} \text{ s.t. } E_J \subset S \times T, |J| = m\right] \leq \binom{d|S|}{m}\left(\frac{|T|}{n}\right)^m.$$

Now performing a union bound over all S, T,

$$\begin{aligned}
\Pr[\mathcal{E}] &\leq \sum_{s \leq \frac{n}{d^2}} \sum_{S:|S|=s} \binom{n}{s}\binom{n}{|T|}\binom{d|S|}{m}\left(\frac{|T|}{n}\right)^m \\
&\leq \sum_s \exp\Big(s\log(en/s) + t\log(en/t) + m\log(eds/m) - m\log(n/t)\Big) \\
&\leq \sum_s \exp\Big(s\log(en/s) + t\log\Big(en/t\Big) - m\log\Big(\frac{nm}{etds}\Big)\Big). \\
&\leq \sum_s \exp\Big(-s\Big((\tfrac{m}{2s}-1)\log(\tfrac{en}{s}) - \tfrac{m}{2s}\log(\tfrac{e^2td}{m})\Big) - t\Big((\tfrac{m}{2t}-1)\log(\tfrac{en}{t}) - \tfrac{m}{2t}\log(\tfrac{e^2ds}{m})\Big)\Big). \quad (17)
\end{aligned}$$

For the first expression in (17), recall that $\frac{m}{2s} > \frac{c_1}{2} \geq 4$, and $n/s \geq d^2$. Similarly $\frac{m}{2t} \geq \frac{c_1}{2} \geq 4$. Hence,

$$\begin{aligned}
&(\tfrac{m}{2s}-1)\log(en/s) \geq (\tfrac{m}{2s}-1)\log(ed^2) \geq \tfrac{3}{2}\cdot\tfrac{m}{2s}\log(d) \geq \tfrac{3}{2}\cdot\tfrac{m}{2s}\log(e^2td/m), \text{ and} \\
&(\tfrac{m}{2t}-1)\log(en/t) \geq (\tfrac{m}{2t}-1)\log(\tfrac{en}{\beta m}) \geq (\tfrac{m}{2t}-1)\log(\tfrac{e^2sd^2}{m}) \geq \tfrac{3m}{8t}\log(d\cdot\tfrac{e^2sd}{m}) \geq \tfrac{3}{2}\cdot\tfrac{m}{2t}\log(\tfrac{e^2sd}{m}),
\end{aligned}$$

since $e^2ds/m \leq d$ for our choice of $m \geq c_1 s \geq e^2 s$. Hence, substituting in (17) we have

$$\Pr[\mathcal{E}] \leq \sum_s \exp\left(-\tfrac{s}{3}(\tfrac{m}{2s}-1)\log(\tfrac{en}{s})\right) \leq \sum_s \exp\left(-s\log(\tfrac{en}{s})\right) = O(\tfrac{1}{n}).$$

This concludes the proof. □

The following lemma generalizes the above lemma to paths of longer length $\ell \geq 1$ as well, and will be crucially used in the analysis.

Lemma 5.3 (Robust Expansion for Paths)**.** In the above construction G, there exists universal constants $c > 1$ such that the following holds for any $\ell \leq L$ and $i \in \{1, 2, \ldots, L-\ell\}$ with probability $1 - o(1)$. For any $S \subseteq V_i$ s.t. $|S| \leq n/(4d^{\ell+1})$ and for any $T \subseteq V_{i+\ell}$, the number of length-ℓ paths between S and T is at most $c^\ell|T| + (2cd)^{\ell-1}|S|$.

Proof. We will prove this by induction. The base case when $\ell = 1$ follows immediately from Lemma 5.2.

Set $c := 4c_1$. Let us assume that the statement of the lemma holds of all $\ell' < \ell$. We will prove the statement for ℓ. Let $\mathcal{P}_\ell$ be the set of length-ℓ paths between $S \subseteq V_i$ and $T \subseteq V_{i+\ell}$. Let $T_{\ell-1} \subseteq V_{i+\ell-1}$ be the subset of vertices on which $\mathcal{P}_\ell$ are incident.

For $u \in V_{\ell-1}$, let $d_{\ell-1}(u)$ denote the number of length $(\ell-1)$ paths between u and S. Let $T_{bad} = \{u \in V_{\ell-1} : d_{\ell-1}(u) > 2c^{\ell-1}\}$, and let $T_{good} = T_{\ell-1} \setminus T_{bad}$. We will count the number of length $(\ell-1)$ paths through T_{bad} and T_{good} separately.

First for T_{bad}, we get an upper bound on $|T_{bad}|$ as follows. From the inductive hypothesis, and since the number of length $(\ell-1)$ paths incident on each vertex in T_{bad} is at least $2c^{\ell-1}$, we have

$$\begin{aligned} 2c^{\ell-1}|T_{bad}| &\le c^{\ell-1}|T_{bad}| + (2cd)^{\ell-2}|S| \\ |T_{bad}| &\le \frac{(2c)^{\ell-1}}{c^{\ell-1}}|S|d^{\ell-2} \le 2^{\ell-2}d^{\ell-2}|S|. \end{aligned}$$

But the total number of length-ℓ paths through T_{bad} is at most d times the number of length $(\ell-1)$ paths incident on T_{bad}. Hence, the number of length ℓ paths through T_{bad} is at most

$$d \times \Big(c^{\ell-1}|T_{bad}| + (2c)^{\ell-2}|S|d^{\ell-2}\Big) \le \Big(\tfrac{1}{2}(2c)^{\ell-1} + (2c)^{\ell-2}\Big)|S|d^{\ell-1} \le (2c)^{\ell-1}|S|d^{\ell-1}.$$

On the other hand for T_{good}, we first observe that $|T_{good}| \le |S|d^{\ell-1} \le n/(4d^2)$. Hence from Lemma 5.2, w.h.p. the number of edges $|E(T_{good}, T)| \le c_1(|T_{good}| + |T|) \le \frac{c}{4}|S|d^{\ell-1} + \frac{c}{4}|T|$. The total number of length-ℓ paths through T_{good} is at most $\frac{1}{2}c^{\ell-1} \times (|S|d^{\ell-1} + |T|) \le c^\ell|T| + c^\ell|S|d^{\ell-1}$. Hence, in total the number of ℓ paths between S and T is at most $c^\ell|T| + (2c)^\ell|S|d^{\ell-1}$. □

5.2 Bounding the Integer Solution Value

For any $i \in [L]$ and $t \le L$, let $n_i(t)$ is the number of jobs in layer i that can be scheduled in the first t phases. The following proposition upper bounds the number of phases required to schedule all the jobs.

Proposition 5.4. In the above notation and construction, there is an absolute constant $\beta > 0$ such that for any layer $i \in [L]$, and any phase $t \le [L/2]$, we have with probability $1 - o(1)$,

$$n_i(t+1) \le n \times \min\Big\{\frac{8L}{(\beta d)^{L-2t-i}}, 1\Big\}. \tag{18}$$

In particular, with probability $1 - o(1)$, we have $n_1(\lfloor \frac{L}{2} \rfloor) \le n/2$; hence the number of phases needed in the optimal solution is at least $\lfloor L/2 \rfloor$ with probability $1 - o(1)$.

The following lemma is key to upper bound the number of jobs scheduled in the first t phases.

Lemma 5.5. For any $i \in [L]$ and $t \le L$, we have for some absolute constant $c > 0$, and any $\ell \in [L]$ with $\ell \ge i$ such that $n_\ell(t) \le n/(16c^\ell d)$ that

$$n_i(t+1) \le \frac{4c^{\ell-i}}{d^{\ell-i}}\Big(nL + n_\ell(t)\Big). \tag{19}$$

We will apply the lemma with $\ell = L - 2t$ to obtain our desired bounds.

Proof. The proof of the above lemma will use the robust expansion property for path (Lemma 5.3) in two different ways. Let $T'_\ell \subset V_\ell$ be the subset of jobs in their respective layers that have been completed in the previous phases (hence $|T'_\ell| = n_\ell(t)$).

For a job to get scheduled in phase $(t+1)$ on a machine $r \in [m]$, all of its ancestor jobs should have either been completed already, or should be scheduled in the same machine r in phase $t+1$. Let $T^{(r)}$ be the jobs in layer V_ℓ that will be scheduled on machine r in the current phase $t+1$. Let $S^{(r)} \subset V_i$ represent the subset of jobs in V_i that can be completed in phase $t+1$ on machine $r \in [m]$ in the $(t+1)$th phase. Hence all of the $|S^{(r)}|d^\ell$ paths of length ℓ to $S^{(r)}$ have to be incident on the vertices in $T^{(r)} \cup T'_\ell$. We divide such vertices into two cases depending on whether most of its length-ℓ paths go from T'_ℓ or $T^{(r)}$. Let $S' \subset V_i$ be the subset of jobs such that at least $\frac{1}{2}d^\ell$ paths to S' that are incident on T'_ℓ. Note that S' also includes the jobs from V_i that have been completed in the previous phases. We will first show that $|S'| \le 4c^\ell n_\ell(t)/d^\ell$. Suppose for contradiction that $S'' \subset S'$ with $|S''| = \lfloor \frac{4c^\ell n_\ell(t)}{d^\ell} \rfloor + 1$. Recall that $|T'_\ell| = n_\ell(t)$, and from assumption $|S''| \le n/(4d^{\ell+1-i})$. Hence we can apply Lemma 5.3 with $S = S'' \subset V_i$ and $T = T'_\ell \subset V_\ell$ (along with the fact that $(2c)^{\ell-1} < d/4$) to conclude that

$$\frac{1}{2}|S''|d^\ell \le c^\ell |T'_\ell| + (2cd)^{\ell-1}|S''| \le c^\ell |T'_\ell| + \tfrac{1}{4}|S''|d^\ell \quad \implies \quad |S''| \le \frac{4c^\ell |T'_\ell|}{d^\ell}$$

$$\text{Hence, } |S'| \le \frac{4c^\ell |T'_\ell|}{d^\ell} \text{ by contradiction.} \tag{20}$$

On the other hand, for any job in $S^{(r)} \setminus S'$, at least $\frac{1}{2}d^\ell$ of the length-ℓ paths to it have to be incident on $T^{(r)}$, which has size at most ρ. Assume for contradiction that $|S^{(r)}| > \frac{4c^\ell \rho}{d^\ell}$, and let $S \subset S^{(r)}$ of size $|S| = \lfloor \frac{4c^\ell \rho}{d^\ell} \rfloor + 1$. Again, from our choice of parameters $|S| \le n/(4d^\ell)$. Hence, from Lemma 5.3 applied to set S and $T^{(r)}$

$$\frac{1}{2}|S|d^\ell \le c^\ell |T^{(r)}| + (2cd)^{\ell-1}|S| \le c^\ell \rho + \tfrac{1}{4}|S|d^\ell \quad \implies \quad |S| \le \frac{4c^\ell \rho}{d^\ell}$$

$$\text{Hence by contradiction, } |S^{(r)}| \le \frac{4c^\ell \rho}{d^\ell} \ \forall r \in [m]. \tag{21}$$

Combining (20) and (21) and using $|T'_\ell| = n_\ell(t)$, we get

$$n_i(t+1) \le |S'| + \sum_{r \in [m]} |S^{(r)}| \le \frac{4c^\ell}{d^\ell}(m\rho + n_{i+\ell}(t)) = \frac{4c^\ell}{d^\ell}(nL + n_{i+\ell}(t)),$$

where the last equality follows from our setting of parameters. □

We now proceed to the proof of Proposition 5.4. Note that Proposition 5.4 implies that we need at least $L/2$ phases to schedule all the jobs. Hence, the optimum is $\Omega(L)$.

Proof of Proposition 5.4. We will prove this by induction using the relation in Lemma 5.5. Set $\beta := 1/c$.

Firstly, $n_\ell(0) = 0$ for all $\ell \in [L]$. For the base case when $t = 0$, we have from Lemma 5.5 applied with $\ell = L$, that for every $i \in [L]$, $n_i(1) \le nL/(d/c)^{L-i}$ as required.

Assume the inductive hypothesis is true for the first t phases. Consider $\ell := L - 2t$. From the inductive hypothesis,

$$n_\ell(t) \le n \cdot \frac{8L}{(\beta d)^{L-2(t-1)-(L-2t)}} \le n \cdot \frac{8L}{(\beta d)^2} \le \frac{n}{16c^\ell d},$$

since for our choice of parameters $c^L < 128\beta^2 d$. Hence, applying Lemma 5.5 with $\ell = L - 2t$, we get for every $i \leq \ell$

$$n_i(t+1) \leq \Big(\frac{4nL}{(\beta d)^{L-2t-i}} + \frac{n_\ell(t)}{(\beta d)^{L-2t-i}} \Big) \leq n \cdot \frac{8L}{(\beta d)^{L-2t-i}},$$

as required. Hence by induction the proposition follows. □

6 Bounding the duplication advantage

In this section, we quantify the advantage that job duplication may offer. We say that a schedule for a given instance is a *no-duplication* schedule if every job in the instance is processed exactly once in the schedule.

Theorem 3 (Bounding the duplication advantage). There exists an instance with $n/2 = m = 2^\rho$ for which any no-duplication schedule has makespan at least $\rho/\log\rho$ times the optimal makespan. Given any instance I with n jobs, m machines, communication delay ρ, and a schedule σ with makespan $M^* \geq \rho$, there exists a polynomial-time computable no-duplication schedule for I with makespan $O(M^* \cdot \log^2 n \log m)$.

We first present the proof for the case where no job takes more than ρ steps in σ. We then show how to extend the theorem to the general case.

We divide time into phases of length ρ (the communication delay). Consider the ith phase $[i\rho, (i+1)\rho)$ of σ for integer $i \geq 0$. Let σ_i denote the schedule σ restricted to job executions that begin in the time interval $[i\rho, (i+1)\rho)$. Let G_0 denote the subgraph of G induced by the jobs processed in σ during phase 0. For $i > 0$, let G_i denote the subgraph of G induced by jobs not in $\cup_{\ell<i} G_\ell$ whose first execution in σ begins in $[i\rho, (i+1)\rho)$. For convenience, we use $G_{<i}$ to denote $\cup_{\ell<i} G_\ell$.

Lemma 6.1. For any job u processed on machine j in σ_i, every predecessor of u is either in $G_{<i}$ or processed on j in σ_i.

Proof. Consider a job u processed on machine j in σ_i; that is $\sigma[u, j] \in [i\rho, (i+1)\rho)$. Let v be any predecessor of u. If v is not in $G_{<i}$, then v is first processed at time at least $i\rho$. Since u is processed at time less than ρ after $i\rho$ and has v as its predecessor, v must be processed in phase i on every machine where u is processed in phase i. □

Our algorithm for transforming an arbitrary schedule σ to a no-duplication schedule $\widehat{\sigma}$ consists of transforming each σ_i, $i \geq 0$, to a no-duplication schedule $\widehat{\sigma}_i$ that completes G_i in $O(\rho \log^2 n \log m)$ time. Since each σ_i is of length ρ, it follows that $\widehat{\sigma}$ is of length $O(M^* \cdot \log^2 n \log m)$.

Fix i, and consider schedule σ_i. The no-duplication schedule $\widehat{\sigma}_i$ begins with ρ steps allocated for communication delay so that the results of the execution of all jobs in $G_{<i}$ are available to every machine. So the remainder of the schedule focuses on completing G_i.

Lemma 6.2. Suppose jobs u and v in G_i share a common predecessor p in G_i, and let m_u, m_v, and m_p denote the number of machines that process u, v, and p, respectively, in σ_i. Then, there exist at least $m_u + m_v - m_p$ machines that process both u and v in σ_i.

Proof. Let M_p (resp., M_u and M_v) denote the set of m_p (resp., m_u and m_v) machines processing p (resp., u and v) in σ_i. Since $M_p \supseteq M_u, M_v$, it follows that at most $m_p - m_u$ (resp., $m_p - m_v$) of the machines in M_p do not process u (resp., v). Thus, at least $m_p - (m_p - m_u) - (m_p - m_v) = m_u + m_v - m_p$ machines in M_p process both u and v, yielding the desired claim. □

Lemma 6.3. There exists a polynomial-time computable no-duplication schedule that can complete G_i in $O(\log^2 n \log m)\rho$ steps.

Proof. Recall that σ_i is a one-phase schedule for completing G_i (with possible duplication of jobs). We divide the jobs of G_i into groups based on the number of machines they are replicated on in σ_i. For any integer $r \geq 0$, let G_{ir} denote the subset of jobs in G_i with the amount of duplication in $[(1+\mu)^r, (1+\mu)^{r+1})$, where $\mu = 1/(2\log n)$. Since the maximum amount of duplication is m, we obtain that the number of subgroups r^* is at most $\log_{1+\mu} m = O(\log m \log n)$.

We first argue that for any r, any job in G_{ir} has no predecessor in $G_{ir'}$ for $r' < r$. To see this, note that any job u in G_{ir} is replicated on at least $(1+\mu)^r$ machines. Since there is no communication in σ_i, it follows that any predecessor of u needs to be executed on every machine where u is executed, which implies that that any predecessor of u is in $G_{ir'}$ for some $r' \geq r$.

Our algorithm consists of computing a no-duplication schedule for G_{ir}, in order from $r = r^*$ to $r = 0$. In the remainder, we show that any G_{ir} can be completed by a no-duplication schedule in $O(\log n)$ phases. Together with the bound on the number of subgroups, this yields the desired bound.

Fix integer $r \in [0, r^*]$. We show how to construct a no-duplication schedule that completes at least $1/4$ of the sinks (jobs with no successors) in G_{ir} in one phase. Repeating this at most $2\log n$ times completes the scheduling of all the sinks of G_{ir}, and hence also all of G_{ir} in $2\log n$ phases (non-sink jobs are scheduled with sink jobs for which they are required).

We construct an auxiliary undirected graph H over the sinks in G_{ir} as follows: there is an edge between sink job u and sink job v if and only if u and v share a common predecessor in G_{ir}. Using a standard ball-growing technique (or the notion of sparse partitions), we determine a collection $\{S_\ell\}$ of disjoint sets of sinks in H such that (a) in every set S_ℓ, there exists a sink s_ℓ that is within $\log n$ hops of every sink in S_ℓ, (b) for any distinct ℓ, ℓ' and any two sinks $s \in S_\ell$ and $s' \in S_{\ell'}$, s is not adjacent to s'; and (c) the total number of sinks in the collection is at least $|H|/2$. Our algorithm for obtaining a collection $\{S_\ell\}$ is as follows. For any undirected graph K, vertex $v \in K$, and integer $x \geq 0$, let $B_x(K, v)$ denote the ball of radius x around v in K.

1. Set H' to H and ℓ to 0.
2. Repeat until H' is empty:
   (a) Let s_ℓ be an arbitrary node in H'.
   (b) Determine the smallest x such that $|B_{x+1}(H', s_\ell)| \leq 2|B_x(H', s_\ell)|$.
   (c) Set S_ℓ to $B_x(H', s_\ell)$ and H' to $H' \setminus B_{x+1}(H', s_\ell)$.

We now argue the three properties we desire. For (a), we note that in step 2a, $x \leq \log n$ since otherwise $|B_{y+1}(H', s_\ell)| > 2|B_y(H', s_\ell)|$ for $0 \leq y < \log n$, implying that $|B_{\log n}(H', s_\ell)|$ exceeds $n \geq |H'|$, a contradiction. For (b), we note that once we include a set S_ℓ, we remove all sinks in $H' \setminus S_\ell$ that are adjacent to a sink in S_ℓ, which ensures that any sink in S_ℓ is not adjacent to any sink in $S_{\ell'}$ for $\ell' > \ell$, thus establishing (b). Finally, for (c), we observe that when S_ℓ is included in the collection, we remove a set of size at most $2|S_\ell|$ from H', implying that the total number of sinks in the collection $\{S_\ell\}$ is at least $|H'|/2$, as desired.

Consider any edge (u, v) in H. By Lemma 6.2, since u and v share a predecessor in G_{ir}, it follows that there exist at least $(1+\mu)^r(1-\mu)$ machines that process both u and v. Let u be any job in S_ℓ. By a repeated application of the lemma along the shortest path from s_ℓ to u, we obtain that s_ℓ and u are processed on at least $(1+\mu)^r(1-\mu)^{\log n} \geq (1+\mu)^r/2$ machines. By a standard averaging argument, it follows that there is a machine j_ℓ that processes a subset S'_ℓ of at least $|S_\ell|/2$ of the jobs in S_ℓ in σ_i.

The desired no-duplication schedule, which we denote by $\widehat{\sigma}_H$ then consists of processing S'_ℓ and all of its predecessors in G_{ir} on machine j_ℓ, for every ℓ. Since no two jobs in S_ℓ and $S_{\ell'}$ share any predecessors, it follows that no job is executed on more than one machine, hence ensuring that $\widehat{\sigma}_H$ is indeed a no-duplication schedule. Furthermore, since the jobs scheduled by $\widehat{\sigma}_H$ on a given machine j is a subset of the jobs scheduled by σ_i on j, $\widehat{\sigma}_H$ completes in a phase. Finally, since $|S'_\ell| \geq |S_\ell|/2$ and $|\cup_\ell S_\ell| \geq |H|/2$, it follows that at least $|H|/4$ of the sinks are completed in $\widehat{\sigma}_H$. We thus have obtained a no-duplication schedule that completes at least $1/4$ of the sinks in G_{ir} in one phase, thus completing the proof of the lemma. □

For the special case where every job completes in ρ steps in σ, the theorem follows immediately from Lemma 6.3. The final no-duplication schedule consists of appending the no-duplication schedules for G_i, $i \geq 0$.

For the general case, we extend the above proof by first marking the jobs in G_i that begin in a phase but end at a different phase; there is at most one job on each machine in a given phase. We then apply the above proof to all the jobs in G_i. In our schedule, any marked job, if executed, would be the last job scheduled on the respective machine. We add an additional delay of ρ so that any marked jobs that complete in the following phase in σ are completed in the no-duplication schedule. If a machine works on its marked job for the next $t > 1$ phases in σ, then we remove the machine from consideration for the next t iterations since it is not executing any jobs in G_{i+1} through G_{i+t}.

7 Open Problems

We have presented the first approximation algorithms for scheduling precedence-constrained jobs of non-uniform sizes on related machines with a fixed communication delay, with the objective of minimizing makespan. Using standard arguments, we can extend our results to the objective of weighted completion times. Our work leaves several open problems and directions for future research. Can we improve on the approximation factor achieved for general schedules? Is there a $\omega(1)$ hardness of approximation for the problem? We conjecture that the integrality gap of the relaxations is $\Omega(\log \rho / \log \log \rho)$. Improving the current bound and broadening the class of programs is of interest. Also, there is a gap between the lower and upper bounds for the duplication advantage. Narrowing this gap, and finding better approximation algorithms for no-duplication schedules would be useful for scenarios where job duplication is not a viable option.

We believe the most significant direction for future research is to study the scheduling problem under more general communication delay environments. From a practical standpoint, developing algorithms that account for delays in a hierarchical network, which may be modeled for instance by a hierarchically well-separated metric, would be valuable for many datacenter scheduling problems.

A Scheduling Without Duplication

Lemma A.1 (Lower bound on duplication advantage). There is an instance for which the makespan of an optimal no-duplication schedule is at least $\Omega(\frac{\rho}{\log \rho})$ times that of an optimal schedule.

Proof. Our lower bound instance is a directed complete binary out-tree $G = (V, E)$ whose edges define the precedence order from the root v_0 to the leaves. In order to simplify the arguments below, we also include an additional node v' that precedes v_0. So $u \prec v$ if and only if u lies on the path from v' to v. G has $2^{\rho-1}$ leaves and ρ levels (not including v'). All jobs have unit processing time, i.e. for all $v \in V, p_v = 1$. We set $m = 2^{\rho-1}$, and assume all machines have unit speed, i.e. for all $i \in M, s_i = 1$.

If duplication is allowed, then G can be scheduled on M in $\rho + 1$ time by executing each path from v' to leaf in topological order on a separate machine. We prove that the makespan of any no-duplication schedule of G on M is $\Omega(\rho^2/\log\rho)$.

Let σ be any no-duplication schedule of G on M. We define the tth ρ-phase to be the time interval $[(t-1)\rho, t\rho)$. In the following, we will show that the maximum number of jobs that can be completed in tth phase ρ-phase of σ is at most $2\rho^t$. Since the union of all levels $(t-1)\log(\rho)+1$ to $t\log(\rho)+1$ contains at least ρ^t jobs, this implies that the number of ρ-phases in σ is at least $\rho/(2\log\rho)$. So we have that $C^* \geq \rho^2/(2\log\rho)$, from which the lemma follows.

In the remainder of this proof, we show by induction on t that the maximum number of jobs that can be completed in tth phase ρ-phase of σ is at most $2\rho^t$. For the base case, consider $t = 1$. By the structure of the graph, v' must be executed before any other job, and this job can be scheduled on only one machine. Therefore, only one machine can run in the first ρ-phase, and it can execute at most ρ jobs.

Suppose the claim holds up to some $t \geq 1$. Then the total number of jobs executed in the first t ρ-phases is at most $2\rho^t$, by the induction hypothesis. Since there is no duplication, the only jobs in the remaining graph that can be scheduled on different machines are those that share no predacessors. Let V_t be the set of jobs completed in the first t ρ-phases. Then the number of machines on which we can schedule jobs in ρ-phase $t+1$ is equal to the number of independent trees in the subgraph G_t of G induced on $V \setminus V_t$. We show that this number is no more than $2\rho^t$.

We show that there are at most $2\rho^t$ independent trees in G_t using a simple exchange argument. Note that, for any set of jobs V_t completed in the first t ρ-phases of σ, for every $v \in V_t$, $\{u : v_0 \in A_u \text{ and } u \in A_v\} \subseteq V_t$. Let $V[\ell]$ be the union of all sets of jobs in levels $0, \ldots, \ell$ of G. Define $V_t = V[t\log\rho]$. Then the number of independent subtrees in G_t is at most $2\rho^t$.

We can reach any other configuration by swapping one job at a time: if $v \in V_t$ and v has no descendants in V_t and $u \notin V_t$ and $A_u \subseteq V_t$, then we replace v with u. This swap maintains the property that all jobs in V_t have all their predacessors also in V_t. Note, however, that executing a swap can only decrease the number of resulting independent subtrees–if we swap internal node for internal node, the number of subtrees remains the same, but if we swap internal node for leaf, the number of resulting subtrees decreases. Therefore, the maximum number of independent subtrees in G_t is $2\rho^t$.

This entails that the total number of machines that can execute jobs in ρ-phase $t+1$ is at most $2\rho^t$. Since each machine can execute at most ρ jobs in a single ρ-phase, the total number jobs completed after ρ-phase $t+1$ is at most $2\rho^{t+1}$. □

We note that the above lower bound is tight for this instance as we can construct an $O(\rho^2/\log\rho)$ length schedule of G on M as follows. In the first two ρ-phases, run one machine for ρ steps to execute all jobs in the first $\log(\rho)$ levels. In the next two ρ-phases, we can complete the next $\log(\rho)$ levels by repeating the same process for all ρ subtrees freed up by the previous phases. In general, for each ρ-phases t and $t+1$, we can run ρ^{t-1} machines to complete the first $\log(\rho)$ levels of all independent subtrees of G_t. This yields a schedule with $O(\rho/\log\rho)$ ρ-phases with makespan $O(\rho^2/\log\rho)$.

B Proof of Lemma 4.1

Let σ_0 be some schedule of (G, M, ρ). We define phase τ of σ_0 to be the period of time $[\rho(\tau-1), \rho\tau)$ in σ_0. We also define $M_0 = \{i \in M : s_i < s_m/m\}$. We first construct a schedule σ_1 on $(G, (M \setminus M_0) \cup \{m'\}, \rho)$ where $s_{m'} = s_m$ (see Figure 7). The construction maps all the job start times in a given phase of σ_0 to a start time in a *step* of the new schedule σ_1. We define a step of σ_1 as follows. For any phase τ of σ_0, let U_τ

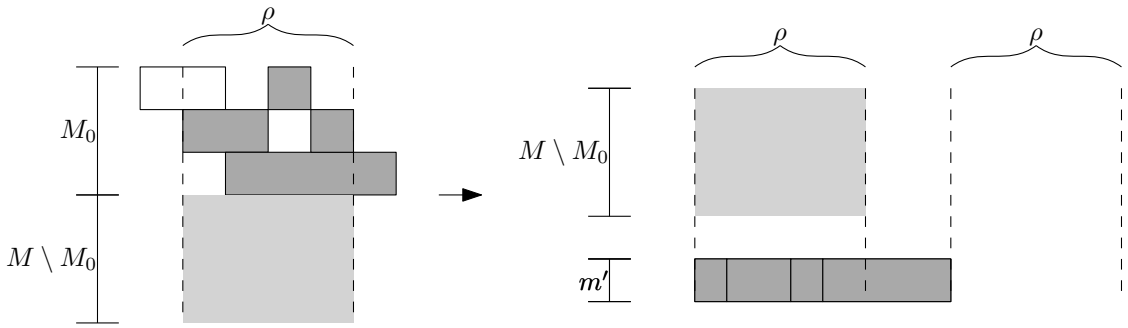


Figure 7: Constructing a step of σ_1 (right) from a phase of σ_0 (left). In each, machines are shown vertically on the left and time increases from left to right. Bordered boxes represent jobs. The light gray borderless box represents the jobs being executed on $M \setminus M_0$. All jobs starting in this phase are placed in topological order on the new machine m' and a ρ delay inserted after their completion. The white job is started in a previous phase and so is ignored.

be the set of all jobs started on some machine in M_0 in phase τ. Then step τ begins at time

$$b(\tau) = \sum_{\tau' < \tau} \Big(\max\{\rho, p(U_{\tau'})/s_m\} + \rho \Big)$$

and ends at time $b(\tau + 1) = e(\tau)$. We construct an arbitrary step τ of σ_1. For each job $v \in U_\tau$, schedule v on machine m' at the earliest possible time after $b(\tau)$, maintaining topological order. For each job v that starts on some machine in $M \setminus M_0$ in phase τ, we set

$$\sigma_1[v, i] = b(\tau) + \sigma_0[v, i] - (\tau - 1)\rho.$$

We now construct a schedule σ_2 of $(G, M \setminus M_0, \rho)$ (see Figure 8). We order all jobs that are executed on machine m' by their start times on m'. This gives the ordering $v_1, \ldots, v_L$ where $\sigma_1[v_\ell, m'] < \sigma_1[v_{\ell+1}, m']$. We also define the set W_ℓ of pairs (v, i) such that $i \neq m'$ and $\sigma_1[v_\ell, m'] \leq \sigma_1[v, i]$ and $\sigma_1[v, i] < \sigma_1[v_{\ell+1}, m']$ if $v_{\ell+1}$ exists (and $\sigma_1[v, i] \neq \perp$). Finally, we define $\mathsf{fin}(v, i, x) = \sigma_x[v, i] + p_v/s_i$ and $u_\ell = \arg\max_{v:(v,m)\in W_\ell}\{\sigma_1[v, m]\}$. We then construct σ_2 in L stages, where L is the number of jobs on m'. $\sigma_{(2,0)} = \sigma_1$ and

$$\sigma_{(2,\ell+1)}[v, i] = \begin{cases} \sigma_{(2,\ell)}[v, i] & \text{if } (v, i) \in \bigcup_{\ell'=1}^{\ell} W_{\ell'} \\ \mathsf{fin}(u_\ell, m, (2, \ell)) & \text{if } v = v_{\ell+1} \\ \mathsf{fin}(u, m, (2, \ell)) + (\sigma_1[v, m'] - \mathsf{fin}(u_\ell, m, 1)) + & \text{if } (v, i) \in \bigcup_{\ell'=\ell+1}^{L} W_{\ell'}. \\ \quad \min\{0, \mathsf{fin}(v_{\ell+1}, m', 1) - \sigma_1[v, m']\} & \end{cases}$$

$\sigma_2 = \sigma_{2,L}$.

Claim B.1. If σ_0 is a valid schedule of (G, M, ρ), then σ_2 is a valid schedule of $(G, M \setminus M_0, \rho)$.

Proof. Suppose σ_0 is a valid schedule. We first show that σ_1 is valid. For any job u, let $t_0^c(u)$ and $t_1^c(u)$ be the completion times of u in σ_0 and σ_1, respectively, and let $t_0^s(u)$ and $t_1^s(u)$ be the starting times. We show that the communication delay restriction is satisfied with the following claim: if u and v are two jobs such that $u \prec v$, then either u and v are on the same machine in σ, or $t_1^c(u) - t_1^s(u) \geq \rho$.

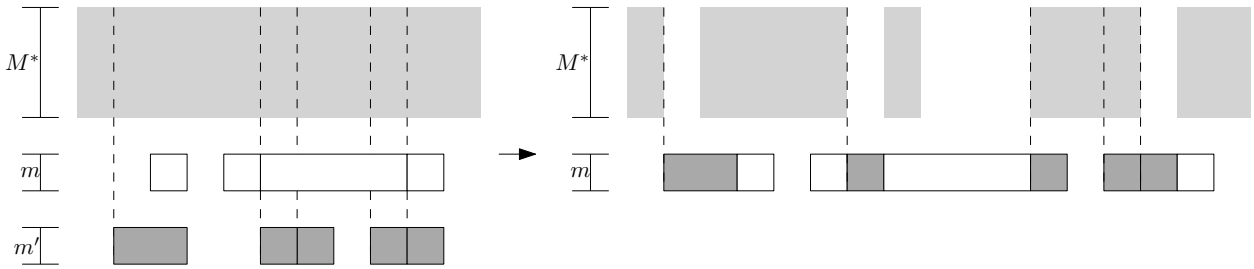


Figure 8: Constructing σ_2 (right) from σ_1 (left). In each, machines are shown vertically on the left and time increases from left to right. $M^* = (M \setminus M_0) \setminus \{m\}$. Dashed lines indicate starting times of jobs on m'. Bordered boxes represent jobs. Unbordered regions represent starting times of jobs. Note that jobs on M^* may be executing in the white regions of σ_2. All jobs on m' in σ_1 are placed immediately prior to later jobs on m in σ_2, and all starting times are correspondingly delayed.

Suppose u is completed in phase τ_u and v is started in phase τ_v. If $\tau_u = \tau_v$ then, since there is no communication within a phase, u and v are executed in order on the same machine in both σ_0 and σ_1, by construction. So we assume that $\tau_u < \tau_v$. If both u and v execute on some machine in M_0 in σ_0, then they are on the same machine and in order in σ_1 by construction. Also, if u is executed on m' in σ_1, then the result follows immediately since it finishes at least ρ time before the end of its step. Also, if u and v both execute on machines in $M \setminus M_0$ then, by construction $t_1^s(v) - t_1^c(u) \geq t_0^s(v) - t_0^c(u)$ so the result follows by the validity of σ_0. So suppose that u executes on some machine in $M \setminus M_0$ and v on machine m'. Note that any job's starting phase in σ_0 is the same as its starting step in σ_1, and the length of each step is at least twice the length of each phase. So, the ending step of u in σ_1 is at least as large as its ending phase in σ_0 and, in its ending step, u completes at least ρ time before the end of the step, by construction. This shows that σ_1 is valid.

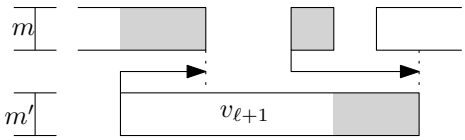
We now show that σ_2 is valid if σ_1 is valid. Consider any two jobs u and v in G. By construction of σ_2, if v starts time t after the completion of u in σ_1, then v starts at least time t after u in σ_2. The claim follows by the validity of σ_1. □

Claim B.2. If σ_0 has makespan C_0 and σ_2 has makespan C_2 the $C_2 \leq 6C_0$.

Proof. Let C_0, C_1, and C_2 be the makespans of σ_0, σ_1, and σ_2 respectively. We first show that $C_1 \leq 3C_0$. Let $W = \{v : \sigma_0[v, i] \neq \perp \text{ for some } i \in M_0\}$. The extra communication phase at the end of each step adds C_0 to the length of the whole schedule σ_1. So we have that

$$C_1 \leq 2C_0 + p(W)/s_m \leq 2C_0 + \sum_{i \in M_0} C_0 s_i / s_m \leq 2C_0 + C_0 \sum_{i \in M} 1/m \leq 3C_0.$$

We now show that $C_2 \leq 2C_1$. Consider difference $\delta_{\ell+1}$ in makespan between $\sigma_{(2,\ell)}$ and $\sigma_{(2,\ell+1)}$. This difference is partially made up of the extra time needed to place $v_{\ell+1}$ on m after the last job on executed on m from a prior $W_{\ell'}$ with $\ell' \leq \ell$. The difference is also partially made up of the amount of time needed to delay all start times in $W_{\ell'}$ with $\ell' \geq \ell$. Notice that these two amounts can be charged to the processing time of $v_{\ell+1}$ and to the processing times of those jobs executed in parallel to $v_{\ell+1}$ on machine m, and note that all charged regions are disjoint.



So, $\sum_{\ell=1}^{L} \delta_\ell \leq C_1$. Therefore, by construction of σ_2, we have that $C_2 = C_1 + \sum_{\ell=1}^{L} \delta_\ell \leq 2C_1$. □

C Additional Motivation for our Approach

C.1 A Combinatorial Approach

To motivate our linear program formulation, we show that a natural extension of the combinatorial algorithm in [20] performs poorly in the worst case. We summarize the original algorithm here, making some simplifications.

The algorithm given in [20] assumes all jobs have unit size and all machines have unit speed. The algorithm constructs the schedule in a series of rounds. In a given round, the algorithm picks the machine i with the least load and finds those jobs that have fewer than ρ predecessors. Let v be one such job. If fewer than $1/2$ of v's uncompleted predecessors have already been scheduled on other machines, v and *all* its uncompleted predecessors are placed on i. The algorithm then finds the machine with the least load after adding v and its predecessors and continues scheduling the phase. Once no jobs satisfy the above conditions, the algorithm inserts a communication delay and starts the next round.

We extend this algorithm to the setting with unit job sizes and arbitrary speeds. In this setting, we can load balance using the machine with minimum load as above while looking at the sets of jobs that can be completed in ρ steps for each machine. The extended algorithm is given as Algorithm 2.

Algorithm 2: Extended Combinatorial Algorithm of [20]

$T \leftarrow 0$; $\forall i, T_i \leftarrow 0$
while *there is some unscheduled job* **do**
  $W \leftarrow \{v : \text{some copy of } v \text{ is scheduled after time } T\}$
  $i \leftarrow \arg\min_j \{T_j\}$
  $V_i \leftarrow \{v : v \text{ has fewer than } \rho s_i \text{ remaining predecessors}\}$
  $\forall v \in V_i, U_v \leftarrow (A_v \cup \{v\}) \setminus \{u : \text{some copy of } u \text{ finishes by time } T\}$
  if *there is some* $v \in V_i$ *such that* $|U_v \setminus W| \geq |U_v|/2$ **then**
    place U_v on i in topological order starting at time T_i
    $T_i \leftarrow T_i + |U_v|$
  else
    $T \leftarrow \rho + \max\{\text{completion time of any currently scheduled job}\}$
    $\forall i, T_i \leftarrow \max\{T_i, T\}$

Lemma C.1. For a given instance of precedence constrained scheduling, let C^* be the optimal makespan for that instance. Then there is some instance such that the schedule output by Algorithm 2 has makespan upper bound by $O(C^*/\rho) = O(C^*/\sqrt{m}) = O(C^*/n)$.

Proof. Consider the DAG shown in Figure 9(b) and let v_ℓ be the job in the chain at level ℓ (starting at level 1) and let $u_{\ell,d}$ be the d^{th} non-chain successsor of v_ℓ, in any order. Suppose that $\rho = m^2$ and that we are

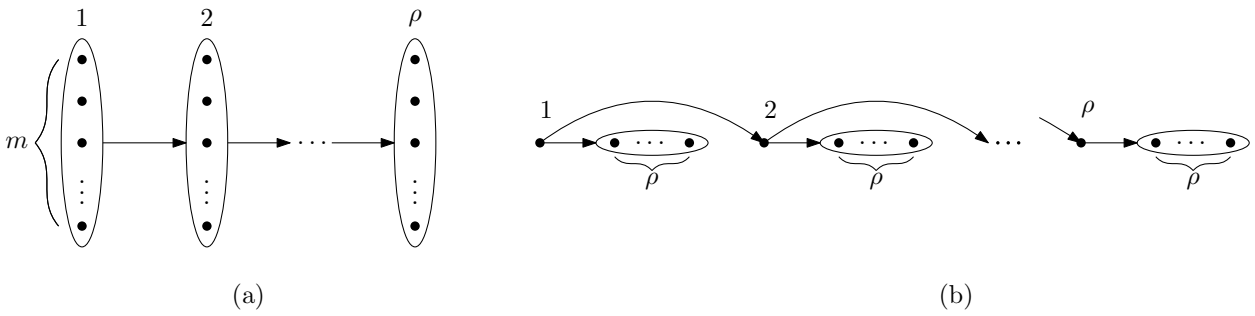


Figure 9: Integrality gap constructions for alternate LP's and the combinatorial heuristic. (a) has ρ layers, each containing m independent jobs. All jobs in layer ℓ are predecessors of every job in layer $\ell + 1$. (b) shows a length ρ chain, where each job in the chain precedes a group of ρ independent jobs.

given $m^2 - 1$ machines of speed 2 and one machine of speed ρ. In this case, Algorithm 2 may repeatedly place the pair $(v_\ell, u_{\ell,1}$ on the speed ρ machine and, for $d = 2$, place the pairs $(v_\ell, u_{\ell,d}$ on the $d - 1^{\text{th}}$ speed 1 machine. This would result in a makespan of $\rho^2 = m = \Omega(n)$. However, the optimal makespan is at most $\rho + 2$ by placing the entire chain on the speed ρ machine in the first step, then introducing a communication delay, then placing one remaining job on each machine. □

We note that the given proof applies to other natural extensions of the combinatorial algorithm, including prioritizing higher speed machines and prioritizing higher capacity groups of machines.

C.2 Same-Machine Variable Relaxation

We refer to the following linear program as LP (22 - 31), which minimizes C subject to the given constraints.

$$C \geq S_v + 1 \qquad \forall v \quad (22)$$

$$Cs_i \geq \sum_v x_{v,i} \qquad \forall i \quad (23)$$

$$S_v \geq S_u + \rho \left(1 - \sum_{i \in M} \delta_{i,u,v}\right) \qquad \forall u \prec v \quad (24)$$

$$S_v \geq S_u + 1 \qquad \forall u \prec v \quad (25)$$

$$S_v \geq 0 \qquad \forall v \quad (26)$$

$$\sum_i x_{v,i} = 1 \qquad \forall v \quad (27)$$

$$x_{v,i} \in [0,1] \qquad \forall v, i \quad (28)$$

$$\delta_{i,u,v} \leq x_{v,i} \qquad \forall v \quad (29)$$

$$\delta_{i,u,v} \leq x_{u,i} \qquad \forall u \quad (30)$$

$$\delta_{i,u,v} \in [0,1] \qquad \forall u, v, i \quad (31)$$

Consider the instance given in Figure 9(a). The instance is divided into ρ *levels*, each of which contains m independent jobs. All jobs at level ℓ precede those jobs at level $\ell + 1$. There are $n = m^2$ jobs, m machines, and communication delay of $\rho = m$. All jobs have unit sizes and all machines have unit speed. We show that the optimal duplication schedule has makespan $\Omega(\rho^2)$. Note that it is trivial to construct a $\rho^2 + \rho$ schedule by scheduling all jobs in level ℓ to start at time $\rho\ell + (\ell - 1)$, and so complete by time $\rho\ell + \ell$. We prove the lower bound by induction on ℓ for the following claim, which straightforwardly entails the lower bound.

Lemma C.2. In any schedule of the instance given in Figure 9(a) with m unit speed machines and communication delay ρ, no job in level ℓ can be scheduled before time $\rho\ell$.

Proof. For $\ell = 0$ the claim is trivially true, so we suppose the claim is true for all levels up to $\ell \geq 0$. In this case, all jobs in level ℓ are scheduled after time $\rho\ell$. Consider any job v in level $\ell + 1$. Since $m > \rho$, we can schedule v earliest by scheduling all jobs in level ℓ on different machines at time $\rho\ell$ and scheduling v on some machine at time $\rho\ell + 1 + \rho$. Thus, the claim is proved. □

Lemma C.3. For a given instance of precedence constrained scheduling with fixed communication delay, let C^* be the optimal makespan for that instance. Then there is some instance for which the optmal value of LP (22 - 31) is upper bounded by $O(C^*/\rho) = O(C^*/m) = O(C^*/\sqrt{n})$.

Proof. For each v, i, we set $x_{v,i}$ to $1/m$. For each v in level ℓ, we set set $S_v = \ell$. For all u, v, i we set $\delta_{u,v,i} = 1/m$. Finally, we set $C = \rho$. It is easy to see that constraints (22), (23), (25) - (31) are satisfied, so we focus on (24). Consider any two jobs u, v such that $u \prec v$. In this case, we have $\sum_i \delta_{i,u,v} = 1$, so (24) is satisfied. □

C.3 A Time-Indexed Relaxation

We refer to the following time-indexed program as LP (32-36). Here, we use notation $[a]$ to refer to the set of all positive integers less than or equal to a.

$$\sum_{i,t} x_{v,i,t} = 1 \qquad \forall v \quad (32)$$

$$x_{v,i,t+1} + \sum_{i' \in [m] \setminus \{i\}} \sum_{t'=t-\rho}^{t} x_{u,i',t} \leq 1 \qquad \forall u \prec v, i, t \quad (33)$$

$$\sum_{i,t' \leq t+1} x_{v,i,t'} \leq \sum_{i,t' \leq t} x_{u,i,t'} \qquad \forall u \prec v, t \quad (34)$$

$$\sum_{v} x_{v,i,t} \leq 1 \qquad \forall i, t \quad (35)$$

$$x_{v,i,t} \in [0,1] \qquad \forall v, i, t \quad (36)$$

Lemma C.4. For a given instance of precedence constrained scheduling with fixed communciation delay, let C^* be the optimal makespan for that instance. Then there is an instance for which the optimal value of LP (32-36) is upper bound by $O(C^*/\rho) = O(C^*/m) = O(C^*/\sqrt{n})$.

Proof. We consider the same instance as Lemma C.3 for which the optimal schedule has makespan $\Omega(\rho^2)$ by Lemma C.2. We assign the variables of LP (32-36) as follows. For each ℓ and $v \in \ell$ and i, we assign $x_{v,i,\ell} = 1/m$ and $x_{v,i,t} = 0$ for $t \neq \ell$. Note that, since every level is completed in a single step, the total number of steps with any nonzero variable is ρ.

We show that this assignment satisfies all constraints of LP (32-36). Constraint 36 is trivially satisfied. Since every job has a $1/m$ fraction assigned to m machines, constraint 32 is satisfied. Since there are m jobs with a $1/m$ fraction assigned to each machine at each step t, constraint 35 is satisfied. There is no constraint 34 for those jobs in the first level, so we show that 34 is satisfied for those jobs in levels $\ell \geq 1$. Let v be a job in level $\ell \geq 1$ and let u be any predecessor of v. By construction, u is in some level $\ell' < \ell$. Let v be fractionally scheduled to start in step t. By construction, all predecessors are completed by time t. So we have that 34 is satisfied for all $t' < t$. Consider $t' \geq t$. In this case, both the right and left sums equal 1, so the constraint is satisfied.

All that remains is to show that constraint 33 is satisfied. Again, we consider a job v in level $\ell \geq 1$ and some predecessor u of v in level $\ell' < \ell$. We fix the machine i and suppose that v is scheduled in step t^*. By construction, we have that $\sum_{i' \in [m] \setminus \{i\}} \sum_{t'=t-\rho}^{t} x_{u,i',t}$ equals 0 or $(1 - 1/m)$, depending on t. Similarly, we have that $x_{v,i,t}$ equals 0 or $1/m$ depending on t. For a given time $t < t^* - 1$, the term $x_{v,i,t+1} = 0$, so

the constraint is trivially satisfied. So we assume $t \geq t^* - 1$. In this case, $x_{v,i,t+1} = 1/m$. Let $\hat{t}$ be the step when u is scheduled. By construction, $\hat{t} \geq t^* - \rho$. So, $\sum_{i' \in [m] \setminus \{i\}} \sum_{t'=t-\rho}^{t} x_{u,i',t} = 1 - 1/m$, so the constraint is satisfied (and tight).

Thus, the makespan of the fractional schedule resulting from the variable assignments is $O(\rho) = O(m)$ by definition of m, and the best duplication schedule has makespan $\Omega(\rho^2) = \Omega(m^2)$. The lemma follows. □

Note that, since duplicated schedules include non-duplicated schedules, the above lemma applies to non-duplicated schedules as well. Finally, note that, since our algorithm finds an approximation within $\log \rho$ of the optimal solution to our LP, this implies that there is a gap of $\Omega(\rho / \log \rho)$ between LP (32-36) and our LP.

C.4 A Same-Phase Variable Relaxation

The linear program LP (37 - 45) minimizes C subject to the following constraints.

$$C \geq S_v \qquad \forall v \quad (37)$$

$$Cs_i \geq \sum_v p_v x_{v,i} \qquad \forall i \quad (38)$$

$$S_v \geq S_u + \rho(1 - y_{u,v}) \qquad \forall u \prec v \quad (39)$$

$$\rho \sum_i s_i x_{v,i} \geq \sum_{u \prec v} p_u y_{u,v} \qquad \forall v \quad (40)$$

$$S_v \geq S_u \qquad \forall u \prec v \quad (41)$$

$$S_v \geq 0 \qquad \forall v \quad (42)$$

$$\sum_i x_{v,i} = 1 \qquad \forall v \quad (43)$$

$$x_{v,i} \in [0,1] \qquad \forall v, i \quad (44)$$

$$y_{u,v} \in [0,1] \qquad \forall u, v \quad (45)$$

Lemma C.5. For a given instance of precedence constrained scheduling with fixed communication delay, let C^* be the optimal makespan for that instance. Then there is an instance for which the optimal solution to LP (37 - 45) is upper bound by $O(C^*/m^{1/12}) = O(C^*/\rho^{1/6}) = O(C^*/n^{1/18})$.

Proof. We consider graph with $\sqrt{m}$ copies of a graph similar to the construction in Figure 9(a), except these instances we have $\sqrt{m}$ levels with $\sqrt{m}$ jobs per level. For machines, we have m machines of speed 1 and one machine of speed $m^{1/3}$. The communication delay $\rho = \sqrt{m}$ and the number of jobs $n = m^{3/2}$.

A similar argument to the one used in the proof of Lemma C.2 entails that any schedule of this graph has makespan $\Omega(\rho\sqrt{m})$. We now prove an upper bound on the optimal value of LP (37 - 45). We assign values to each variable as follows. Let i^* be the speed $m^{1/3}$ machine. For each v, i, if $i = i^*$ then we set $x_{v,i} = 1/m^{1/4}$ and we distribute the remaining $x_{v,i} = 1 - 1/m^{1/4}$ evenly over all remaining machines. We partition the levels into $\sqrt{m}/m^{1/12}$ classes where class $c_r = \{v \in \text{level } \ell \text{ for } (r-1)m^{1/12} \leq \ell < rm^{1/12}\}$. For each $v \in c_r$, we set $S_v = \rho r$ and $C_v = S_v + 1$. We set $y_{u,v} = 1$ if, for some r, $u \in c_r$ and $v \in c_r$ and set $y_{u,v} = 0$ otherwise. Finally, we set $C = \rho\sqrt{m}/m^{1/12}$.

We show that all constraints of LP (37 - 45) are satisfied. It is straightforward to check that constraints (41) - (45) are satisfied, so we focus on constraints (37), - (40). We note that $\max_v\{S_v\} = \rho(\sqrt{m}/m^{1/12})$, so (37) is satisfied. For i^*, we have that $Cs_{i^*} = m^{17/12} > m^{15/12} = \sum_v p_v x_{v,i^*}$. For $i \neq i^*$, (38) follows easily, so the constraint is satisfied. (39) follows from the fact that, if $y_{u,v} = 1$ then $S_v = S_u$ and, if $y_{u,v} = 0$

and $u \prec v$ then u is in a lower class than v so $S_v \geq S_u + \rho$. We now show that (40) is satisfied. For any v,

$$\begin{aligned}
\rho \sum_i s_i y_{v,i} &= \rho \sum_{i \neq i^*} y_{v,i} + \rho y_{v,i^*} m^{1/3} && \text{by construction} \\
&= \rho(1 - 1/m^{1/4}) + \rho m^{1/3}/m^{1/4} && \text{by assignment} \\
&= m^{1/2} - m^{1/4} + m^{7/12} && \text{by definition of } \rho.
\end{aligned}$$

Since any v has at most $\sqrt{m} \cdot m^{1/12} = m^{7/12}$ predecessors u such that $y_{u,v} = 1$, this shows the constraints are satisfied.

By assignment of C, we have that $C = \rho\sqrt{m}/m^{1/12} = m^{11/12}$. Since the optimal makespan is at least m this proves the lemma. □

References

[1] Ishfaq Ahmad and Yu-Kwong Kwok. On exploiting task duplication in parallel program scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):872–892, Sep. 1998.

[2] B. Awerbuch and D. Peleg. Sparse partitions. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 503–513, 1990.

[3] Evripidis Bampis, Aristotelis Giannakos, and Jean-Claude König. On the complexity of scheduling with large communication delays. *European Journal of Operational Research*, 94:252–260, 1996.

[4] N. Bansal. Scheduling open problems: Old and new. MAPSP 2017, 2017.

[5] Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, Oct 2009.

[6] Abbas Bazzi and Ashkan Norouzi-Fard. Towards tight lower bounds for scheduling problems. *Lecture Notes in Computer Science*, page 118129, 2015.

[7] D. Bozdag, F. Ozguner, and U. V. Catalyurek. Compaction of schedules and a two-stage approach for duplication-based dag scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):857–871, 2009.

[8] I. Casas, J. Taheri, R. Ranjan, L. Wang, and A. Zomaya. A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems. *Future Generation Computer Systems*, 74, September 2017.

[9] Chandra Chekuri and Rajeev Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98(1-2):29–38, 1999.

[10] Fabián A. Chudak and David B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

[11] S. Darbha and D. P. Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 9:87–95, 1998.

[12] Yuanxiang Gao, Li Chen, and Baochun Li. Optimizing device placement for training deep neural networks. In *International Conference on Machine Learning*, 2018.

[13] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:416429, 1969.

[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. *Mathematics of Operations Research*, 22(3):513544, Aug 1997.

[15] 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.

[16] D. Hu and B. Krishnamachari. Throughput optimized scheduler for dispersed computing systems. In *2019 7th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 76–84, 2019.

[17] Jeffrey M. Jaffe. Efficient scheduling of tasks without full use of processor resources. *Theoretical Computer Science*, 12(1):117, Sep 1980.

[18] Janardhan Kulkarni, Shi Li, Jakub Tarnawski, and Minwei Ye. Hierarchy-based algorithms for minimizing makespan under precedence and communication constraints. In *Proceedings of the Fortieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, to appear, 2020.

[19] Jan Karel Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.

[20] Renaud Lepere and Christophe Rapine. An asymptotic $\mathcal{O}(\ln \rho / \ln \ln \rho)$-approximation algorithm for the scheduling problem with duplication on large communication delay graphs. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 154–165. Springer, 2002.

[21] Shi Li. Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 283–294, 2017.

[22] N. Linial and M. Saks. Low diameter graph decompositions. *Combinatorica*, 13:441–454, 1993.

[23] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. Hierarchical planning for device placement. In *International Conference on Learning Representations*, 2018.

[24] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, pages 2430–2439, 2017.

[25] A. Munier and C. Hanen. Using duplication for scheduling unitary tasks on m processors with unit communication delays. *Theoretical Computer Science*, 178(1):119 – 127, 1997.

[26] Alix Munier. Approximation algorithms for scheduling trees with general communication delays. *Parallel Computing*, 25(1):41–48, 1999.

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

[28] Michael A. Palis, Jing-Chiou Liou, and David S. L. Wei. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):46–55, 1996.

[29] Christos H. Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM journal on computing*, 19(2):322–328, 1990.

[30] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, Philadelphia, PA, 2000.

[31] C Picouleau. *Two new NP-complete scheduling problems with communication delays and unlimited number of processors*. Inst. Blaise Pascal, Univ., 1991.

[32] Maurice Queyranne and Maxim Sviridenko. Approximation algorithms for shop scheduling problems with minsum objective. *Journal of Scheduling*, 5(4):287305, 2002.

[33] Victor J Rayward-Smith. Uet scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 18(1):55–71, 1987.

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

[35] Inseong Song, Wanoh Yoon, Eunmee Jang, and Sangbang Choi. Task scheduling algorithm with minimal redundant duplications in homogeneous multiprocessor system. In Tai-hoon Kim, Hojjat Adeli, Hyun-seob Cho, Osvaldo Gervasi, Stephen S. Yau, Byeong-Ho Kang, and Javier García Villalba, editors, *Grid and Distributed Computing*, pages 238–245, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[36] Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. *Proceedings of the 42nd ACM symposium on Theory of computing - STOC 10*, 2010.

[37] Bart Veltman, B. J. Lageweg, and Jan Lenstra. Multiprocessor scheduling with com-munication delays.parallel computing. *Parallel Computing*, 16:173–182, 1990.