A Simpler QPTAS for Scheduling Jobs with Precedence Constraints

Syamantak Das
Department of Comp Science and Engineering, IIIT Delhi, India

Andreas Wiese
Department of Mathematics, Technical University of Munich, Germany

Abstract

We study the classical scheduling problem of minimizing the makespan of a set of unit size jobs with precedence constraints on parallel identical machines. Research on the problem dates back to the landmark paper by Graham from 1966 who showed that the simple List Scheduling algorithm is a $(2-\frac{1}{m})$-approximation. Interestingly, it is open whether the problem is NP-hard if $m=3$ which is one of the few remaining open problems in the seminal book by Garey and Johnson. Recently, quite some progress has been made for the setting that m is a constant. In a breakthrough paper, Levey and Rothvoss presented a $(1+\epsilon)$-approximation with a running time of $n^{(\log n)^{O((m^2/\epsilon^2)\log\log n)}}$ [STOC 2016, SICOMP 2019] and this running time was improved to quasi-polynomial by Garg [ICALP 2018] and to even $n^{O_{m,\epsilon}(\log^3 \log n)}$ by Li [SODA 2021]. These results use techniques like LP-hierarchies, conditioning on certain well-selected jobs, and abstractions like (partial) dyadic systems and virtually valid schedules.

In this paper, we present a QPTAS for the problem which is arguably simpler than the previous algorithms. We just guess the positions of certain jobs in the optimal solution, recurse on a set of guessed subintervals, and fill in the remaining jobs with greedy routines. We believe that also our analysis is more accessible, in particular since we do not use (LP-)hierarchies or abstractions of the problem like the ones above, but we guess properties of the optimal solution directly.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases makespan minimization, precedence constraints, QPTAS

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.40

1 Introduction

A classical problem in scheduling theory is the problem to schedule jobs on parallel machines in order to minimize the makespan, while obeying precedence constraints between the jobs. It goes back to the 1966 when Graham proved in his seminal paper [5] that the simple List Scheduling algorithm yields a is a $(2-\frac{1}{m})$-approximation algorithm. Formally, the input consists of a set J of n jobs, a number of machines $m \in \mathbb{N}$, and each job $j \in J$ is characterized by a processing time $p_j \in \mathbb{N}$. We seek to schedule them non-preemptively on m machines in order to minimize the time when the last job finishes, i.e., to minimize the makespan. Additionally, there is a precedence order $\prec$ which is a partial order between the jobs. Whenever $j \prec j'$ for two jobs $j, j' \in J$ then job j' can only be started when j has already finished. Given that the List Scheduling algorithm is essentially a simple greedy routine, one may imagine that one can achieve a better approximation ratio with more sophisticated algorithmic techniques. However, Svensson showed that even for unit size jobs (i.e., $p_j = 1$ for each $j \in J$) there can be no $(2-\epsilon)$-approximation algorithm for any $\epsilon > 0$ [9], assuming a variant of the Unique Games Conjecture. Hence, under this conjecture List Scheduling is the essentially best possible algorithm. Slight improvements are known for unit size jobs: there is an algorithm by Coffman and Graham [1] which computes an optimal solution when $m=2$, and a which is a $(2-\frac{2}{m})$-approximation algorithm for general m, as shown by Lam and Sethi [6]. Also, there is an $(2-\frac{7}{3m+1})$-approximation algorithm known

due to Gangal and Ranade [2]. In fact, the setting of unit-size jobs is interesting since then the complexity of the problem stems purely from the precedence constraints and not from the processing times of the jobs (which might encode problems like PARTITION). We assume this case from now.

In practical settings m might be small, e.g., m might be the number of processors in a system, or the number of cores of a CPU. Thus, it is a natural question whether better approximation ratios are possible when m is a constant. Note that the mentioned lower bound of $2-\epsilon$ [9] does not hold if $m = O(1)$. For $m = 2$ the mentioned algorithm by Coffman and Graham [1] computes an optimal solution; however, even if $m = 3$ it is not known whether the problem is NP-hard! In fact, it is one of the few remaining open problems in the book by Garey and Johnson [3].

In a break-through result, Levey and Rothvoss presented a $(1+\epsilon)$-approximation with a running time of $n^{(\log n)^{O((m^2/\epsilon^2)\log\log n)}}$ [7]. Subsequently, the running time was improved by Garg [4] to $n^{O_{m,\epsilon}(\log^{O(m^2/\epsilon^2)} n)}$ which is quasi-polynomial. Both algorithms are based on the natural-LP relaxation of the problem, (essentially) lifted by a certain number r of rounds of the Sherali-Adams hierarchy, and it can be solved in time $n^{O(r)}$. Given the optimal LP-solution x^*, they *condition* on certain variables in the support of x^*, which effectively fixes time slots for the corresponding jobs. Each conditioning operation changes not only the variable that one conditions on, but possibly also other variables in the support. After a well-chosen set of conditioning operations, they recurse into smaller subintervals and give each of them a copy of the current LP-solution (which might be different from x^* due to the conditioning operations). Intuitively, in each recursive call for some subinterval I they seek to schedule jobs that can only be scheduled during I according to the previous conditionings and the precedence constraints; they call these jobs *bottom jobs* and the other jobs *top* and *middle jobs*. The middle jobs can be discarded. For the top jobs, they first use a matching argument to show that most of the top jobs can be inserted if one can ignore the precedence constraints between top jobs. Knowing this, they insert most of the top jobs with a variation of Earliest-Deadline-First (EDF), such that all precedence constraints are satisfied; some of the top jobs are discarded in the process. The discarded jobs are later inserted in a greedy manner which is affordable since they are very few.

A different approach is used by Li [8] who improved the running time further to $n^{O_{m,\epsilon}(\log^3 \log n)}$. Instead of working with an LP, he guesses directly certain properties of the optimal solution. While in the above argumentation each conditioning step costs a factor $n^{O(1)}$ in the running time, he argues that – roughly speaking – most of the time the information he guesses is binary and hence costs only a factor of 2 in the running time. More precisely, he guesses properties of a technical abstraction based on *dyadic systems*, *partial dyadic systems*, and *virtually valid schedules*. On a high level, he shows that based on the optimal schedule one can define a corresponding dyadic system and a virtually valid schedule for it, at the cost of discarding a few jobs. His algorithm then searches for the dyadic system and a virtually valid schedule for it that discards as few jobs as possible. Then, he shows that based on them, he can construct an actually valid schedule, which discards only few additional jobs.

1.1 Our Contribution

In this paper we present a QPTAS for the makespan minimization problem with unit size jobs on a constant number of machines along with precedence constraints ($Pm|prec, p_j = 1|C_{\max}$ in the three-field notation) which is arguably simpler than the $(1+\epsilon)$-approximation algorithms

sketched above. We do not use an LP-formulation and in particular no LP-hierarchy or a similar approach based on conditioning on variables. Instead, we guess properties of the optimal solution directly, similarly as Li [8]. However, we do not use the reduction to dyadic systems or a similar abstraction but work with the optimal solution directly. We believe that this makes the algorithm and the analysis easier to understand. Our running time is $n^{O_{m,\varepsilon}((\log n)^{m/\varepsilon})}$ so it is asymptotically better than the running time by Garg [4] up to hidden constants.

Our algorithm is actually pretty simple. Let T be the optimal makespan (which we guess). For a parameter $k = (\log n)^{O(1)}$ we guess the placement of k jobs in OPT and a partition of $[0, T)$ into at most k intervals. For each interval I, there are some jobs that need to be scheduled during I according to our guesses and the precedence constraints. We recurse on each interval I and its corresponding jobs. Then, we add all remaining jobs with a simple variant of EDF where the release dates and deadlines are defined such that the precedence constraints between these jobs and the other jobs (that we recursed on) are satisfied. For the correct guesses, we show that the resulting schedule discards at most $O(\epsilon T)$ jobs, and we add those jobs at the end with a simple greedy routine.

In our analysis, we use a hierarchical decomposition of intervals, like the previous results [8, 7, 4]. In contrast to those, we define the decomposition such that each interval is subdivided into $O(\log n/\varepsilon)$ subintervals (instead of 2 subintervals) since this makes the analysis easier. Based on the optimal solution, for each level we identify certain jobs that we want to guess in that level later. Also, we assign a level to each job. Via a shifting step, we show that we can discard the jobs from intuitively every (m/ϵ)-th level. Based on these levels, we argue that there are guesses for our algorithm that yield a small total number of discarded jobs. Intuitively, we guess the jobs from the first m/ϵ levels that we identified above and the intervals of the $(m/\epsilon + 1)$-th level. We recurse on the latter intervals. When we insert the jobs of the first m/ϵ levels via EDF (those jobs that we did not guess already), the jobs from our recursion and their precedence constraints dictate for each inserted job j a time window $[r_j, d_j)$. It might happen that the position of j in the optimal solution is not contained in $[r_j, d_j)$, but we show that it is always contained in the larger time window of $[r_j - \lambda, d_j + \lambda]$ for a small value $\lambda > 0$. We show that we need to discard at most $O(mT\frac{\epsilon}{\log n})$ jobs to compensate for this error and due to the precedence constraints between the inserted jobs. We analyze EDF directly and do not need to go via a matching argumentation as done in [7]. It turns out that our algorithm recurses for at most $O(\frac{\varepsilon}{m}\log n)$ levels and hence this yields $O(\frac{\varepsilon \log n}{m} \cdot mT\frac{\epsilon}{\log n}) = O(\epsilon^2 T)$ discarded jobs in total.

While our analysis borrows ideas from the mentioned previous results [8, 7, 4] we believe that our algorithm and our analysis are simpler and more accessible.

2 Algorithm

We present a simple QPTAS for the problem $Pm|prec, p_j = 1|C_{\max}$. Let $\epsilon > 0$ and assume w.l.o.g. that $1/\epsilon \in \mathbb{N}$. We guess $T := \mathrm{OPT}$ and assume w.l.o.g. that T is a power of 2 (if this is not the case, then we can add some dummy jobs that need to be processed after all other jobs; note that a $(1+\epsilon)$-approximate solution for this larger instance is a $(1+2\epsilon)$-approximate solution for the original instance). Also, w.l.o.g. we assume that in OPT each job starts and ends at an integral time point, i.e., during a time interval of the form $[t, t+1)$ for some $t \in \mathbb{N}$. We will refer to such a time interval as a *time slot*. Furthermore, we assume that w.l.o.g. that the precendence constraints are transitive, i.e., if $j \prec j'$ and $j' \prec j''$ then also $j \prec j''$.

Our algorithm works on a "guess and recurse" framework. Define a parameter $k := (m \log n/\varepsilon)^{m/\epsilon+1}$. The reader can think of the parameter k as $O_{m,\varepsilon}((\log n)^{m/\varepsilon+1})$, where $O_{m,\varepsilon}()$ hides constants that are only dependent on m, ε. Our algorithm has three steps. First, we guess up to k jobs from OPT and their time slots in OPT, i.e., we try all combinations of up to k input jobs and all combinations for their time slots to schedule them. Then, we guess a partition of $[0, T)$ into at most k intervals, i.e., we try all combinations of partitioning $[0, T)$ into at most k intervals. By allowing empty intervals, we assume w.l.o.g. that we guess exactly k intervals and denote them by $I_1, ..., I_k$. Let J_{guess} denote the guessed jobs. There might be precedence constraints between jobs in J_{guess} and jobs in $J \setminus J_{guess}$. In particular, these precedence constraints might dictate that some job $j \in J \setminus J_{guess}$ needs to be scheduled within some interval I_j. In this case, we say that j is a *bottom job*. Let $J_{bottom} \subseteq J \setminus J_{guess}$ denote the set of all bottom jobs. We call each job $j \in J \setminus (J_{guess} \cup J_{bottom}) =: J_{top}$ a *top job*.

Given our guess for the jobs J_{guess} and their time slots and our guessed partitioning of $[0, T)$, we make k recursive calls: one for each interval I_i with $i = 1, 2, \cdots, k$. Let us denote by $J^{(i)}_{bottom}$ the subset of jobs in J_{bottom} that need to be scheduled within I_i according to our guesses above, and let $J^{(i)}_{guess}$ denote the jobs in J_{guess} for which we guessed a time slot within I_i. We make a recursive call on the interval I_i whose input are the jobs $J^{(i)}_{bottom} \cup J^{(i)}_{guess}$ and the guessed time slots for the jobs in $J^{(i)}_{guess}$. In this recursive call, we want to compute a schedule in which

- all jobs in $J^{(i)}_{guess}$ are scheduled in the time slots that we had guessed for them,
- a (hopefully very large) subset of the jobs in $J^{(i)}_{bottom}$ are scheduled; we denote by $J^{(i)}_{disc} \subseteq J^{(i)}_{bottom}$ the jobs in $J^{(i)}_{bottom}$ that were not scheduled, we call them the *discarded* jobs, and
- we obey the precedence constraints between the jobs in $J^{(i)}_{bottom} \cup J^{(i)}_{guess}$. We ignore all precedence constraints that involve the jobs in $J^{(i)}_{disc}$.

Suppose that we are given a solution from each recursive call. We define $J_{disc} := \bigcup_{i=1}^{k} J^{(i)}_{disc}$. We ignore these discarded jobs for now. We want to schedule the jobs in J_{top}. Recall that these are jobs in $J \setminus J_{bottom} \cup J_{guess}$. To this end, for each job $j \in J_{top}$ we define an artificial release date r_j and an artificial deadline d_j. Let $j \in J_{top}$. We define r_j to be the earliest start time of an interval I_i at which each job $j' \in J_{guess} \cup J_{bottom}$ with $j' \prec j$ has completed; we define $r_j := 0$ if there is no such job j'. Similarly, we define d_j to be the latest end time of an interval I_i at which no job $j'' \in J_{guess} \cup J_{bottom}$ with $j \prec j''$ has already started; again, if there is no such job j'' we define $d_j := T$. In order to find slots for the jobs in J_{top} we use the following variation of the Earliest Deadline First (EDF) algorithm. We sweep the time axis from left to right. For each time $t = 0, 1, 2, ...$ we consider the not yet scheduled jobs $j \in J_{top}$ with $r_j \leq t$ whose predecessors in $J_{guess} \cup J_{bottom} \setminus J_{disc}$ we have already scheduled before time t. We sort these jobs non-decreasingly by their deadlines (breaking ties arbitrarily) and add them in this order to the machines that are idle during $[t, t + 1)$. We do this until no more machine is idle during $[t, t + 1)$. It might happen that a job $j \in J_{top}$ misses its deadline at the current time t, i.e., it holds that $t = d_j$ but j has not been scheduled by us at any slot $r_j \leq t' \leq t$ and during $[t', t' + 1)$ all machines are busy. In this case we add j to the set J_{disc}. Among all our guesses for the jobs and the partition into intervals, we output the solution in which at the very end the smallest number of jobs is in the set J_{disc} (breaking ties arbitrarily).

Each recursive call for an interval works similarly as the main call of the recursion described above. The (straightforward) differences are the following: the input of each recursive call consists of the interval $\bar{I}$, a set of jobs $\bar{J}_{guess}$ which were guessed in previous levels in the recursion, together with their guessed time slots, and a set of (not yet scheduled)

jobs $\bar{J}$. We guess the time slots for up to k guessed jobs $J_{guess} \subseteq \bar{J}$, and we require that these guesses do not violate the precedence constraints with the jobs in $\bar{J}_{guess}$. Also, we guess a partition of I into k intervals (rather than a partition of $[0, T)$). The input for each recursive call for a subinterval $\bar{I}_i$ of $\bar{I}$ consists of $\bar{I}_i$, $\bar{J}_{guess} \cup J_{guess}$, and the jobs in $\bar{J}$ that need to be scheduled within $\bar{I}_i$ according to our guesses for $\bar{J}_{guess} \cup J_{guess}$. We return the computed schedule for $\bar{I}$ for a subset of the jobs $\bar{J}_{guess} \cup \bar{J}$ and the discarded jobs in $\bar{J}_{guess} \cup \bar{J}$.

If the algorithm is called on an interval of length 1 then we skip the step of partitioning the interval further into at most k subintervals (and in particular we do not recurse anymore). We will show later that there are guesses that lead to an $(1 + \epsilon)$-approximate solution such that the recursion depth is $\frac{\varepsilon}{m} \log n$. In order to enforce that the recursion depth is $\frac{\varepsilon}{m} \log n$ (limiting the running time of the algorithm), we define that a recursive call at recursion depth $\frac{\varepsilon}{m} \log n + 1$ simply outputs a solution in which all of the jobs in $\bar{J}$ are discarded and the algorithm does not recurse further.

After running the recursive algorithm described above, we need to schedule the jobs in J_{disc}. Intuitively, for each such job $j \in J_{disc}$ we create an empty time slot that we add into our schedule and inside which we schedule j. This will increase our makespan by $|J_{disc}|$. Formally, we consider the jobs $j \in J_{disc}$ in an arbitrary order. For each job $j \in J_{disc}$ we determine a time t such that all its predecessors in our current schedule have finished by time t, but none of its successors in our current schedule have started yet. Such a time t always exists since we show later that we obtain a feasible schedule for all jobs in $J \setminus J_{disc}$ and we assumed that the precedence constraints are transitive. We insert an empty time slot $[t, t + 1]$ into our schedule, i.e., we move all jobs scheduled during $[t, \infty)$ by one unit to the right, and we schedule j during $[t, t + 1]$. This completes the description of our algorithm.

3 Analysis

In this section, we prove that the above algorithm is a QPTAS.

▶ **Lemma 1.** *The above algorithm runs in time* $n^{(\log n)^{O_{m,\varepsilon}(m/\epsilon)}}$.

Proof. For the running time, we observe that there are at most $n^{O(k)}$ choices for the guessed k jobs, at most $T^{O(k)} = (n)^{O(k)} = n^{O(k)}$ choices for their time slots, and similarly $n^{O(k)}$ choices for the at most k intervals that we guess. Since we bound the recursion depth to be at most $\log n + 1$, this yields a running time of $n^{O(k \log n)} = n^{(\log n)^{O_{m,\varepsilon}(m/\epsilon)}}$. ◀

Now we prove the approximation ratio of our algorithm. To this end, we define guesses for jobs and their time slots and intervals in each call of our recursion, such that for these guesses our algorithm outputs a schedule with makespan at most $1 + 6\epsilon$.

3.1 Laminar family of intervals

For this, we define a laminar family $\mathcal{L}$ of intervals. Recall that we assumed that T, the guessed makespan, is a power of 2. We define that the entire interval $[0, T)$ forms the (only) interval of level 0. Let us define $\rho = \lceil \log(\log n/\varepsilon) \rceil$. Consider an interval I of some level $\ell = 0, 1, 2, \cdots$ (we will argue the number of levels later). If $|I| \geq 2^\rho$ then I is partitioned into $2^\rho = \Theta(\log n/\varepsilon)$ equal-sized intervals of length $|I|/2^\rho$. These intervals constitute the level $\ell + 1$ of the family $\mathcal{L}$. For each interval $I \in \mathcal{L}$, we denote by $\ell(I)$ the level of I.

If $1 < |I| < \rho$ then I is partitioned into $|I|$ intervals of level $\ell + 1$ of length 1 each. If $|I| = 1$ then I is not partitioned further.

▶ **Lemma 2.** *The total number of levels in the laminar family $\mathcal{L}$ is at most* $\log n/\log(\log n/\varepsilon)+1$.

Proof. By construction, each interval at a particular level $\ell = 0, 1, 2, \cdots$ is of equal length. Hence the length of an interval at level ℓ is at most $T/(\log n/\varepsilon)^\ell$. Further, once the length of intervals of a level becomes less than $2^\rho \leq 2\log n/\varepsilon$, there could only be one additional level where every interval is of length 1. Hence the total number of levels could be at most $\log T/\log(\log n/\varepsilon)+1 \leq \log n/\log(\log n/\varepsilon)+1$ ◀

3.2 Guessed, top, and bottom jobs

Next, we assign the jobs to levels. More precisely, for each level ℓ we define a set of *guessed jobs* $J^{(\ell)}_{guess}$ and a set of *top jobs* $J^{(\ell)}_{top}$. The intuition is that later we want to guess the jobs in $\bigcup_{\ell'=0}^{\ell} J^{(\ell')}_{guess}$ and the jobs in $\bigcup_{\ell'=0}^{\ell} J^{(\ell')}_{top}$ will form top jobs. We say that a *chain of jobs* is a set of jobs $J' = \{j_1, j_2, ..., j_c\}$ for some $s \in \mathbb{N}$ such that $j_i \prec j_{i+1}$ for each $i \in \{1, ..., c-1\}$, and we say that c is the *length* of the chain.

We define these sets $J^{(\ell)}_{guess}$ and $J^{(\ell)}_{top}$ level by level in the order $\ell = 0, 1, 2, ...$. Consider a level ℓ. Let I be an interval of level ℓ. We initialize $J^{(\ell)}_{guess} = J^{(\ell)}_{top} = \emptyset$. Our plan is that we add jobs to $J^{(\ell)}_{guess}$ step by step. Let J_I denote the jobs that can only be scheduled during I, assuming that we schedule the jobs in $J^{(0)}_{guess} \cup \cdots \cup J^{(\ell-1)}_{guess}$ exactly as in OPT. We say that a job $j \in J_I$ is *flexible* if it can still be scheduled in more than one subinterval of level $\ell+1$ of I, assuming that we schedule the jobs in $J^{(0)}_{guess} \cup ... \cup J^{(\ell)}_{guess}$ exactly as in OPT. Suppose that there is a chain $J' \subseteq J_I \setminus J^{(\ell)}_{guess}$ of length at least $\varepsilon|I|/2^{\lceil \log\log n\rceil}$ that contains only flexible jobs. Then for each interval I' of level $\ell+1$ we add to $J^{(\ell)}_{guess}$ the first and the last job from J' that is scheduled during I' in OPT. If we guess these jobs in our algorithm, the effect is that each job in J' that we did *not* add to $J^{(\ell)}_{guess}$ can be scheduled only during one interval of level $\ell+1$. Hence, one way to think of this procedure is that we push these jobs one level down. We do this operation until there is no more chain J' of length at least $\varepsilon|I|/2^{\lceil \log\log n\rceil}$ that contains only flexible jobs. We define that $J^{(\ell)}_{top,I}$ contains all remaining flexible jobs in J_I. We do this procedure for each interval I of level ℓ, and define at the end $J^{(\ell)}_{top} := \bigcup_I J^{(\ell)}_{top,I}$.

▶ **Proposition 3.** *For every job $j \in J$, there exists a unique $\ell \in \{0, 1, 2, \cdots, \rho\}$ such that $j \in J^{\ell}_{top}$.*

3.3 Few rejected jobs

With the preparation above, we will show that there are guesses of our algorithm for the guessed jobs and the intervals that lead to few discarded jobs overall, at most $O(\epsilon T)$ many. Since the algorithm selects the guesses that lead to the minimum total number of discarded jobs, we will show that it computes a solution with at most $O(\epsilon T)$ discarded jobs. We need some preparation for this. First, we establish that we can afford to discard all jobs in sets $J^{(a+r\cdot m/\epsilon)}_{top}$ for $r \in \mathbb{N}_0$, for some offset a.

▶ **Lemma 4.** *There is an offset $a \in \{0, 1, ..., \frac{m}{\epsilon}-1\}$ such that $\left|\bigcup_{r\in\mathbb{N}_0} J^{(a+r\cdot m/\epsilon+1)}_{top}\right| \leq \epsilon T$.*

Proof. For every $a \in \{0, 1, ..., \frac{m}{\epsilon}-1\}$, we define $\mathcal{L}_a = \{\ell : \ell = (a + r\cdot m/\epsilon + 1), r \in \mathbb{N}_0\}$ and the set $\bigcup_{\ell\in\mathcal{L}_a} J^{\ell}_{top}$. Now Proposition 3 implies that the resulting sets $\bigcup_{\ell\in\mathcal{L}_a} J^{\ell}_{top}$ are pairwise disjoint. Since T is the optimal makespan, the total number of jobs cannot exceed mT. Hence, there exists some $a \in \{0, 1, ..., \frac{m}{\epsilon}-1\}$ such that $|\bigcup_{\ell\in\mathcal{L}_a} J^{\ell}_{top}| \leq \varepsilon T$. ◀

For the root problem of the recursion, we will show that the following guesses lead to few discarded jobs overall: the guessed subintervals are the subintervals of the laminar family at level $a+1$ where a which is the offset as identified by the above lemma. The guessed jobs are all jobs in $\bigcup_{\ell=0}^{a} J_{guess}^{(\ell)}$ and we guess their time slots in OPT. We recurse on the guessed intervals of level $a+1$ of the laminar family. Suppose that in some level $r \in \mathbb{N}$ of the recursion we are given as input an interval $\bar{I}$ of some level $\ell_r = a + (r-1) \cdot m/\epsilon$ of the laminar family, together with a set of jobs of the form $\bar{J} = \bar{J}_{bottom} \dot{\cup} \bar{J}_{guess}$ such that for each job $j \in \bar{J}_{guess}$ we are given a (guessed) time slot that equals the time slot in OPT during which j is executed, but we are not given a time slot for any job in $\bar{J}_{bottom}$. We will show that the following guesses lead to few discarded jobs overall: we guess the intervals of level $\ell_{r+1} = a + r \cdot m/\epsilon + 1$ of the laminar family; the guessed jobs are all jobs in $\bar{J}_{bottom} \cap \bigcup_{\ell=a+(r-1)\cdot m/\epsilon+1}^{a+r\cdot m/\epsilon} J_{guess}^{(\ell)}$ that are scheduled in $\bar{I}$ in OPT and we guess their time slots in OPT.

With the next lemma, we prove inductively that there are few discarded jobs. We define $r_{\max} := \lceil \epsilon(\log n / \log\log n + 1)/m \rceil$ which is an upper bound on the number of recursion levels that we need in this way.

▶ **Lemma 5.** *Consider a recursive call of our algorithm in which the input is of the following form:*

- *an interval $\bar{I}$ such that $\bar{I} = [0, T)$ (we define $r = 0$ in this case) or $\bar{I}$ is an interval of some level $\ell_r = a + (r-1) \cdot m/\epsilon$ of the laminar family, for some $r \in \mathbb{N}$,*
- *a set of jobs $\bar{J} = \bar{J}_{bottom} \dot{\cup} \bar{J}_{guess}$ such that*
  - *for each job $j \in \bar{J}_{guess}$ we are given a time slot that coincides with the time slot during which j is scheduled in* OPT*,*
  - *each job $j \in \bar{J}_{bottom}$ is scheduled during $\bar{I}$ in* OPT*.*

Then our algorithm returns a schedule for $\bar{J}$ in which at most

$$\sum_{\ell' \in \mathcal{L}_a} \sum_{\bar{I}' \in \mathcal{L}: \ell(\bar{I}') = \ell' \wedge \bar{I}' \subsetneq \bar{I}} \left| J_{top, \bar{I}'}^{(\ell')} \right| + \frac{5m\epsilon}{\log n} (r_{\max} - r) |\bar{I}| \tag{1}$$

jobs are discarded.

Our goal is now to prove Lemma 5. Consider a recursive call of our algorithm of the form specified in Lemma 5 for some $r \in \mathbb{N}$. If $r = r_{\max}$ then our algorithm simply enumerates over all possible schedules for $\bar{J}_{bottom}$ and thus finds a schedule in which no job is discarded (since this is the case in OPT). Suppose by induction that the claim is true for all $r \geq r^* + 1$ for some r^*. We want to prove that it is true also for $r = r^*$ so we consider such a recursive call. Let $\tilde{J}_{guess}$ denote the guessed jobs and let $\tilde{I}_1, ..., \tilde{I}_k$ denote the guessed partition of $\bar{I}$ into subintervals, according to our description right before Lemma 5. Let $\tilde{J}_{top} \subseteq \bar{J}$ and $\tilde{J}_{bottom} \subseteq \bar{J}$ denote the resulting set of top and bottom jobs, respectively (thus, the sets $\bar{J}, \bar{J}_{guess}, \bar{J}_{bottom}$ are part of the input, while the sets $\tilde{J}_{guess}, \tilde{J}_{top}, \tilde{J}_{bottom}$ and intervals $\tilde{I}_1, ..., \tilde{I}_k$ stem from our guesses). Recall that for each job $j \in \tilde{J}_{top}$ we define a release time r_j and a deadline d_j in our algorithm. Let λ denote the length of each interval $\tilde{I}_i$ (note that they all have the same length), i.e., the length of the intervals of level $\ell_{r+1} = a + r \cdot m/\epsilon + 1$ (where $r = 0$ in the root problem of the recursion). Note that $\lambda \leq |\bar{I}|/(\log n/\varepsilon)^{m/\varepsilon+1}$. We show in the next lemma that in OPT each job $j \in \tilde{J}_{top}$ is essentially scheduled during $[r_j, d_j)$ and thus r_j and d_j are almost consistent with OPT.

▶ **Lemma 6.** *For each job $j \in \tilde{J}_{top}$ it holds that in* OPT *the job j is scheduled during $[r_j - \lambda, d_j + \lambda)$.*

Proof. Let us recall that for each job $j \in \bar{J}_{top}$ the release time r_j is defined to be the earliest start time of an interval $\tilde{I}_i, i = 1, 2, 3, \cdots k$ such that every job $j' \in \bar{J}_{guess} \cup \bar{J}_{bottom}$ with $j' \prec j$ is completed before r_j. We want to prove in OPT the job j does not start before time $r_j - \lambda$. Let $\hat{I} = [t_1, t_2]$ denote the interval of level $\ell_{r+1} = a + r \cdot m/\epsilon + 1$ for which $t_2 = r_j$ (and observe that $t_2 = t_1 + \lambda$). By definition of r_j, there exists a job $j' \in \bar{J}_{guess} \cup \tilde{J}_{guess} \cup \tilde{J}_{bottom}$ with $j' \prec j$ that completes in $\hat{I}$ in the schedule that we obtained from the recursive call in $\hat{I}$. Since we assumed that our guessed time slots for the jobs in $\bar{J}_{guess} \cup \tilde{J}_{guess}$ are identical to the corresponding time slots in OPT, we conclude that also in OPT the job j' is scheduled during $\hat{I}$. Thus, in OPT the job j cannot start before time $t_1 = r_j - \lambda$. An analogous argument shows that j cannot be scheduled in OPT after $d_j + \lambda$. ◀

We want to show now that our variant of EDF discards only few jobs from $\tilde{J}_{top}$. To this end, we partition $\tilde{J}_{top}$ into the sets $\tilde{J}_{top,1} := \tilde{J}_{top} \cap \bigcup_{\ell=a+(r-1)\cdot m/\epsilon+1}^{a+r\cdot m/\epsilon} J_{top}^{(\ell)}$ and $\tilde{J}_{top,2} := \tilde{J}_{top} \cap J_{top}^{(a+r\cdot m/\epsilon+1)}$. We can afford to discard all jobs in $\tilde{J}_{top,2}$, see (1), but we need to bound the number of discarded jobs in $\tilde{J}_{top,1}$. For a job $j \in \tilde{J}_{top}$ it can happen that $r_j = d_j$. In this case we say that j is *degenerate*. Note that a degenerate job is always discarded.

▶ **Lemma 7.** *There are at most $2m\varepsilon|\bar{I}|/\log n$ degenerate jobs in $\tilde{J}_{top,1}$.*

Proof. Consider any job $j \in \tilde{J}_{top,1}$. By definition, there exist two adjacent intervals $I' = [t'_1, t'_2), I'' = [t'_2, t'_3)$ such that $\ell(I') = \ell(I'') = a + r \cdot m/\varepsilon$ such that j can be potentially scheduled during both the intervals as dictated by the guessed jobs $\bar{J}_{guess}$. Thus, if j is degenerate, then $r_j = d_j = t'_2$. Further, by Lemma 6, all jobs $j \in \tilde{J}_{top,1}, r_j = d_j = t'_2$ must be scheduled in OPT in the interval $[t'_2 - \lambda, t'_2 + \lambda]$. Hence the total number of such jobs is upper bounded by

$$
\begin{aligned}
& m \cdot 2\lambda |\{I' : I' \in \mathcal{L} \wedge \ell(I') = a + r \cdot m/\varepsilon \wedge I' \subset \bar{I}\}| \\
& \leq 2m \cdot (\log n/\varepsilon)^{m/\varepsilon} \cdot \frac{|\bar{I}|}{(\log n/\varepsilon)^{m/\varepsilon+1}} \\
& = 2m\varepsilon|\bar{I}|/\log n
\end{aligned}
$$

◀

Our goal now is to bound the number of discarded non-degenerate jobs in $\tilde{J}_{top,1}$. We partition $\bar{I}$ into *meta-intervals* $\hat{I}_1, \hat{I}_2, ...\hat{I}_{k'}$ with $k' \leq k$ with the properties that each interval $\hat{I}_i \in \{\hat{I}_1, \hat{I}_2, ...\}$ is of the form $\hat{I}_i = [t_1, t_2)$ for some $t_1, t_2 \in \mathbb{N}$ such that

- each value t_1, t_2 is the start or the end point of some interval in $\tilde{I}_1, ..., \tilde{I}_k$,
- at time t_2 there is no (non-degenerate) job $j \in \tilde{J}_{top}$ pending that was released before t_2,
- for each $t \in [t_1, t_2)$ such that t is the start or end point of some interval in $\tilde{I}_1, ..., \tilde{I}_k$, some non-degenerate job $j \in \bar{J}_{top}$ is pending at time t.

Now the intuition is that during each meta-interval $\hat{I}_i = [t_1, t_2)$ EDF tries to schedule only jobs that OPT schedules during $[t_1 - \lambda, t_2 + \lambda)$ (due to Lemma 6), so essentially we have enough space on our machines to schedule all these jobs. We might waste space due to the precedence constraints. However, this space is bounded via the following lemma.

▶ **Lemma 8.** *Let $\tilde{I}_i$ be an interval such that at each time $t \in \tilde{I}_i$ some job $j \in \tilde{J}_{top}$ is pending. Then during $\tilde{I}_i$ there are at most $\left|\tilde{I}_i\right| \frac{\epsilon}{\log n}$ time slots $[t, t+1)$ with $t \in \mathbb{N}$ such that some machine is idle during $[t, t+1)$.*

Proof. For the interval $\hat{I}_i = [t_1, t_2)$, let $J_{\hat{I}_i}$ denote the subset of jobs in $j \in \bar{J}_{top,1}$ such that $r_j \leq t_1$. We now create a partition of $J_{\hat{I}_i}$ according to the precedence constraints. Let $J_0 \subseteq J_{\hat{I}_i}$ denote the jobs whose preceding jobs have been either scheduled or discarded

before t_1. For every $p = 1, 2, \cdots \eta$ (where η is some positive integer), let J_p denote the set of jobs $j \in J_{\hat{I}_i}$ for which there exists a job $j' \in J_{p-1}$ such that $j' \prec j$ and there is no job $j'' \in J_{\hat{I}_i} \setminus J_{p-1}$ such that $j'' \prec j$.

Let $\tau_1, \tau_2, \tau_3, \cdots, \tau_{\eta'}$ be defined such that the time slots $[\tau_1, \tau_1+1), [\tau_2, \tau_2+1), ..., [\tau_{\eta'}, \tau_{\eta'}+1)$ are exactly the time slots during $\hat{I}_i$ such that some machine is idle for the entire respective time slot. We claim that any job $j' \in \tilde{J}_{top}$ that is pending at the end of a time slot $[\tau_q, \tau_q + 1)$ with $q = \{1, 2, 3, \cdots\}$ must belong to J_p for some $p > q$. We prove this by induction. Since no jobs are released inside $\hat{I}_i$, no job in J_0 is pending at time $\tau_0 + 1$ since otherwise this would contradict the definition of our variant of EDF and the fact that a machine is idle during $[\tau_0, \tau_0 + 1)$. Now assume the hypothesis to be true for the time slots $[\tau_1, \tau_1 + 1), [\tau_2, \tau_2 + 1), ..., [\tau_q, \tau_q + 1)$ for some q and consider the time slot $[\tau_{q+1}, \tau_{q+1} + 1)$. If a job $j' \in \tilde{J}_{top}$ is pending at time $\tau_q + 1$ then $j' \in J_{p'}$ for some $p' > q$ by the induction hypothesis. This means that all jobs in $\bigcup_{p=0}^{q} J_p$ have completed before time $\tau_q + 1$. Hence, the jobs in J_{q+1} can be scheduled at any time after time $\tau_q + 1$. Suppose that there is a pending job j at time $\tau_{q+1} + 1$. Since a machine is idle during $[\tau_{q+1}, \tau_{q+1} + 1)$ we have that $j \in J_{p'}$ for some $p' > q+1$ since otherwise our variant of EDF would have scheduled j during $[\tau_{q+1}, \tau_{q+1} + 1)$. By our construction of the guessed, top and bottom jobs in Section 3.2 the length of the longest chain of the jobs in $\tilde{J}_{top}$ is at most $\varepsilon|\tilde{I}_i|/2^{\lceil \log \log n \rceil} \leq |\tilde{I}_i| \frac{\epsilon}{\log n}$. Therefore, $\eta' \leq \eta \leq |\tilde{I}_i| \frac{\epsilon}{\log n}$ which completes our proof. ◀

Also, we show that there are only few meta-intervals which – together with the argumentation above – yields the following lemma.

▶ **Lemma 9.** *The total number of discarded non-degenerate jobs in $\tilde{J}_{top,1}$ is at most $m|\bar{I}| \left(\frac{2\varepsilon}{\log^2 n} + \frac{\varepsilon}{\log n} \right)$.*

Proof. Consider a meta-interval $\hat{I}_i$. Let j be the last non-degenerate job in $\bar{J}_{top,1}$ that is discarded during $\hat{I}_i$. Let t_1 denote the beginning of the interval $\hat{I}_i$. We know that d_j is the end point of some interval $\tilde{I}_{i'}$. Since j is non-degenerate we know that $[r_j, d_j) \neq \emptyset$ and by definition of $\hat{I}_i$ this implies that at each time $t \in [t_1, d_j)$ there is some job from $\bar{J}_{top}$ pending. Using Lemma 8, the total number of (wasted) idle slots across all machines during $[t_1, d_j)$ is at most $m\varepsilon|[t_1, d_j)|/m \log n$.

We further invoke Lemma 6 to argue that all jobs in $\tilde{J}_{top}$ that we schedule or discard during $[t_1, d_j)$ are scheduled in OPT during $[t_1 - \lambda, d_j + \lambda)$. The maximum number of jobs that OPT could have scheduled during $[t_1 - \lambda, d_j + \lambda)$ is hence $m(d_j - t_1 + 2\lambda)$. Since our wasted space during $[t_1, d_j)$ is at most $m\varepsilon|[t_1, d_j)|/m \log n$, we conclude that we discard at most

$$m(d_j - t_1 + 2\lambda) - m(d_j - t_1) + \frac{\epsilon}{\log n}(d_j - t_1) = 2\lambda m + \frac{\epsilon}{\log n}(d_j - t_1)$$

jobs during $[t_1, d_j)$. By definition, for each job $j \in \bar{J}_{top,1}$ we have that $j \in J_{top}^{(\ell)}$ for some level ℓ with $\ell \leq a + (r+1) \cdot m/\epsilon - 1$. Thus, for such a job j we have that r_j and d_j lie in different intervals of level $a + (r+1) \cdot m/\epsilon$ of $\mathcal{L}$. Thus, if during a meta-interval $\hat{I}_i$ a job $j \in \bar{J}_{top,1}$ is discarded, then $\hat{I}_i$ has non-empty intersection with at least two different intervals of level $a + (r+1) \cdot m/\epsilon$ of $\mathcal{L}$. Hence, there can be at most $(\log n/\varepsilon)^{m/\epsilon - 1}$ such meta-intervals. We conclude that in total we discard at most

$$(\log n/\varepsilon)^{m/\epsilon-1} \cdot 2\lambda m + \frac{\epsilon}{\log n}|\bar{I}| \leq m\frac{2|\bar{I}|(\log n/\varepsilon)^{m/\epsilon-1}}{(\log n/\varepsilon)^{m/\varepsilon+1}} + \frac{\epsilon}{\log n}|\bar{I}| \leq m|\bar{I}| \left(\frac{2\varepsilon}{\log^2 n} + \frac{\varepsilon}{\log n} \right)$$

jobs in $\bar{J}_{top,1}$. ◀

Now we are ready to bound the total number of discarded jobs using Lemmas 7, 9, and the induction hypothesis.

Proof of Lemma 5. We shall prove the lemma for the input interval $\bar{I}$ at level $\ell_{r^*} = a + (r^* - 1) \cdot m/\varepsilon$. By induction hypothesis, suppose (1) is true for all $r \geq r^* + 1$. Hence, applying our recursive algorithm with the input intervals $\tilde{I}_i, i = 1, 2, \cdots k$ returns a schedule where number of discarded jobs is at most

$$\sum_{\ell' \in \mathcal{L}_a} \sum_{\bar{I}' \in \mathcal{L}: \ell(\bar{I}') = \ell' \wedge \bar{I}' \subsetneq \tilde{I}_i} \left| J^{(\ell')}_{top, \bar{I}'} \right| + \frac{5m\epsilon}{\log n} (r_{\max} - (r^* + 1)) |\tilde{I}_i| \tag{2}$$

Summing (2) over all $i = 1, 2, \cdots k$ and observing that $\bar{I} = \dot{\bigcup}_{i=1}^{k} \tilde{I}_i$ yields that the total number of discarded jobs is at most

$$\sum_{\ell' \in \mathcal{L}_a} \sum_{\bar{I}' \in \mathcal{L}: \ell(\bar{I}') = \ell' \wedge \bar{I}' \subsetneq \bar{I}} \left| J^{(\ell')}_{top, \bar{I}'} \right| + \frac{5m\epsilon}{\log n} (r_{\max} - (r^* + 1)) |\bar{I}| \tag{3}$$

We would now like to prove the statement for $r = r^*$. Now at the recursive call at level $r = r^*$ with the input subinterval $|\bar{I}|$ consider the guesses as described in the preceding discussion. Using Lemmas 9 and 7, the total number of jobs rejected from $\tilde{J}_{top,1}$ under these guesses is at most

$$2\varepsilon m |\bar{I}| / \log n + m |\bar{I}| \left(\frac{2\varepsilon}{\log^2 n} + \frac{\varepsilon}{\log n} \right) \leq 5m\varepsilon |\bar{I}| / \log n \tag{4}$$

Further, we could have potentially rejected all the jobs in $\tilde{J}_{top,2}$. The total number of such jobs is

$$|\tilde{J}_{top,2}| = \sum_{\bar{I}' \in \mathcal{L}: \ell(\bar{I}') = \ell', \ell' = a + r^* \cdot m/\varepsilon} \left| J^{(\ell')}_{top, \bar{I}} \right| \tag{5}$$

Adding the above two quantities (4) and (5) to the quantity (3) and observing that our recursive algorithm selects the guesses at a particular level that minimizes the number of discarded jobs, the lemma holds for a recursive call at level r^*. ◀

▶ **Lemma 10.** *Our algorithm computes a solution with a makespan of at most* $(1 + 6\epsilon)T$.

Proof. The input at the top level recursive call in our algorithm is an interval $\bar{I} = [0, T)$ and the entire set of jobs J. Plugging in $r = 0$ in Lemma 5 and using Lemma 4, the first term in (1) is bounded by $\sum_{\ell' \in \mathcal{L}_a} |J^{\ell}_{top}| \leq \varepsilon T$. Since $r_{\max} := \lceil \epsilon (\log n / \log \log n + 1)/m \rceil$, the second term for $r = 0$ in (1) bounded by $5\varepsilon^2 T / (\log \log n + 1)$.

Hence, the number of discarded jobs in our algorithm is upper bounded by $6\varepsilon T$. All the other jobs are scheduled within the interval $[0, T)$. We potentially need to introduce one additional time-slot of the form $[t, t + 1)$ for each discarded job. We observe that for each discarded job j we can find a position to insert a time-slot for j since we assumed the precendence constraints to be transitive and we obtained a feasible schedule for all non-discarded jobs. Hence, the total length of the schedule is at most $(1 + 6\varepsilon)T$. ◀

Combining Lemma 10 and Lemma 1 gives us the following theorem.

▶ **Theorem 11.** *There exists an algorithm for the precedence constrained scheduling on identical parallel machines that is a* $(1 + \varepsilon)$*-approximation and runs in time* $n^{O_{m,\varepsilon}((\log n)^{m/\varepsilon})}$.

References

1 Edward G Coffman and Ronald L Graham. Optimal scheduling for two-processor systems. *Acta informatica*, 1(3):200–213, 1972.

2 Devdatta Gangal and Abhiram Ranade. Precedence constrained scheduling in (2-7/3p+1)optimal. *Journal of Computer and System Sciences*, 74(7):1139–1146, 2008. doi:10.1016/j.jcss.2008.04.001.

3 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

4 Shashwat Garg. Quasi-PTAS for Scheduling with Precedences using LP Hierarchies. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 59:1–59:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.59.

5 Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.

6 Shui Lam and Ravi Sethi. Worst case analysis of two scheduling algorithms. *SIAM Journal on Computing*, 6(3):518–536, 1977.

7 Elaine Levey and Thomas Rothvo. A (1+ epsilon)-approximation for makespan scheduling with precedence constraints using LP hierarchies. *SIAM Journal on Computing*, 50(3):STOC16–201, 2019.

8 Shi Li. Towards PTAS for precedence constrained scheduling via combinatorial algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2991–3010. SIAM, 2021.

9 Ola Svensson. Hardness of precedence constrained scheduling on identical machines. *SIAM Journal on Computing*, 40(5):1258–1274, 2011.