Quasi-PTAS for Scheduling with Precedences using LP Hierarchies

Shashwat Garg[1]
Eindhoven University of Technology, Netherlands
s.garg@tue.nl

Abstract

A central problem in scheduling is to schedule n unit size jobs with *precedence constraints* on m identical machines so as to minimize the makespan. For $m = 3$, it is not even known if the problem is NP-hard and this is one of the last open problems from the book of Garey and Johnson.

We show that for fixed m and ϵ, polylog(n) rounds of Sherali-Adams hierarchy applied to a natural LP of the problem provides a $(1+\epsilon)$-approximation algorithm running in quasi-polynomial time. This improves over the recent result of Levey and Rothvoss, who used $r = (\log n)^{O(\log \log n)}$ rounds of Sherali-Adams in order to get a $(1 + \epsilon)$-approximation algorithm with a running time of $n^{O(r)}$.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Approximation algorithms, hierarchies, scheduling, rounding techniques

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.59

Related Version A full version of the paper is available at `https://arxiv.org/pdf/1708.04369.pdf`.

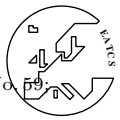
Acknowledgements We would like to thank Seeun William Umboh, Martin Böhm and Nikhil Bansal for helpful discussions throughout this work.

1 Introduction

A central problem in scheduling is the following: suppose we are given n unit jobs which have to be processed non-preemptively on m identical machines. There is also a precedence order among the jobs: if $i \prec j$, then job i has to be completed before j can begin. The goal is to find a schedule of the jobs with the minimum makespan, which is defined as the time by which all the jobs have finished.

This problem admits an easy $(2 - \frac{1}{m})$ approximation algorithm which was given by Graham [5] in the 60's and is one of the landmark results in scheduling. This algorithm is known as the *list-scheduling* algorithm and works as follows: at every time $t = 1, 2, \ldots$, if there is an empty slot on any of the m machines, schedule any *available* job there, where a job is available if it is not yet scheduled and all the jobs which must precede it have already been scheduled. This simple greedy algorithm is essentially the best algorithm for the problem and for almost half a century it was an open problem whether one can get a better approximation algorithm. In fact, this was one of the ten open problems in Schuurman and Woeginger's influential list of open problems in scheduling [11]. It was known since the 70's that it is NP-hard to get an approximation factor better than 4/3 [8]. Slight improvements were given

[1] Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 022.005.025.



by Lam and Sethi [6] who gave a $2 - \frac{2}{m}$ approximation algorithm, and Gengal and Ranade [3] who gave a $2 - \frac{7}{3m+1}$ approximation algorithm for $m \geq 4$. Finally in 2010, Svensson [13] showed that assuming a variant of Unique Games conjecture due to Bansal and Khot [1], for any constant $\epsilon > 0$ there is no $(2 - \epsilon)$ approximation algorithm for the problem.

However, this still leaves open the problem for the important case when m is a constant. In fact in practice, usually the number of jobs are very large but there are only a few machines. Surprisingly, for $m = 3$, it is not even known if the problem is NP-hard. This is one of the four problems from the book of Garey and Johnson [4] whose computational complexity is still unresolved.

In order to get a better algorithm for the case when m is a constant, a natural strategy is to write a linear program (LP), and for this problem, one such LP is the time-indexed LP (1), in which we first make a guess T of the makespan and then solve the LP. The value of the LP is the smallest T for which the LP is feasible, and the worst case ratio of the optimal makespan and the value of the LP is known as the integrality gap of the LP. It is well known that LP (1) has an integrality gap of at least $2 - \frac{2}{m+1}$ (see e.g. [9]), which suggests that one needs to look at stronger convex relaxations in order to get a better algorithm. Such a stronger convex relaxation can be obtained by applying a few rounds of a hierarchy to the LP, and in this paper, we will use the Sherali-Adams hierarchy [12]. It is known that just one round of Sherali-Adams hierarchy reduces the integrality gap to 1 for $m = 2$ and thus, the problem can be solved exactly in this case (credited to Svensson in [10]). Claire Mathieu in a workshop in Dagstuhl 2010 asked if one can get a $(1 + \epsilon)$-approximation algorithm using $f(\epsilon, m)$ rounds of Sherali-Adams hierarchy for some function f independent of n, which would imply a PTAS for the problem when m is a constant. This is also Open Problem 1 in Bansal's recent list of open problems in scheduling presented at MAPSP 2017.

To get some intuition behind why hierarchies should help in this problem, let us first look at the analysis for Graham's list-scheduling algorithm. At the end of this algorithm, the number of time slots which are busy, that is where all the m machines have some job scheduled on them, is a lower bound on the optimum. Also, the number of non-busy time slots is a lower bound on the optimum. This is because there must be a *chain* of jobs $j_1 \prec j_2 \prec \cdots \prec j_k$ such that one job from this chain is scheduled at each non-busy time, and the length of any chain in the instance is clearly a lower bound on the optimum. This implies that the makespan given by the algorithm, which is the sum of the number of busy and non-busy time slots, is a 2-approximation of the optimum makespan, and a slightly more careful argument gives the guarantee of $(2 - 1/m)$. Now the key idea is that if the instance given to us has a maximum chain length of at most ϵ times the optimal makespan, then Graham's list-scheduling algorithm already gives a $(1 + \epsilon)$-approximation, and hierarchies provide, via conditionings, a good way to "effectively" reduce the length of the chains in any given instance.

Though the question of whether one can get a $(1 + \epsilon)$-approximation algorithm using $f(\epsilon, m)$ rounds of Sherali-Adams hierarchy is still unresolved, a major breakthrough was made recently by Levey and Rothvoss [9], who gave a $(1 + \epsilon)$-approximation algorithm using $r = (\log n)^{O(m^2 \log\log n/\epsilon^2)}$ rounds of Sherali-Adams. This gives an algorithm with a running time of $n^{O(r)}$, which is faster than exponential time but worse than quasi-polynomial time.

1.1 Our Result

In this paper, we improve over the result of Levey and Rothvoss [9] by giving a $(1 + \epsilon)$-approximation algorithm which runs in quasi-polynomial time. Formally, we show the following:

▶ **Theorem 1.** *The natural LP* (1) *for the problem augmented with r rounds of Sherali-Adams hierarchy has an integrality gap of at most $(1+\epsilon)$, where $r = O_{m,\epsilon}(\log^{O(m^2/\epsilon^2)} n)$. Moreover, there is a $(1+\epsilon)$-approximation algorithm for this problem running in time $n^{O(r)}$.*

Throughout the paper, we use the notation $O_{m,\epsilon}(.)$ to hide factors depending only on m and ϵ. The natural LP for the problem is the following:

$$\begin{aligned}
\sum_{t=1}^{T} y_{jt} &= 1 && \forall j \in [n] \\
\sum_{j} y_{jt} &\leq m && \forall t \leq T \\
\sum_{t' \leq t} y_{jt'} &\geq \sum_{t' \leq t+1} y_{it'} && \forall t \leq T,\ \forall j \prec i \\
y_{jt} &\geq 0 && \forall t \leq T,\ \forall j \in [n]
\end{aligned} \tag{1}$$

Here T is our guess on the optimum makespan. In an integral solution, $y_{jt} = 1$ if job j is scheduled at time t, and 0 otherwise. The first constraint ensures that each job is scheduled at exactly one time and the second constraint ensures that no more than m jobs are scheduled at any time. The third constraint ensures that if $j \prec i$, then job i can only be scheduled at a time strictly later than job j.

1.2 Overview of Our Algorithm

Let us first give an overview of the algorithm of Levey-Rothvoss [9] since our algorithm builds up on it.

Previous approach

At a high-level, the algorithm of Levey-Rothvoss [9] works by constructing a laminar family of intervals, where the topmost level has one interval $[1, T]$ and each succeeding level is constructed by dividing each interval of the previous level into two equal sized intervals, as shown in the figure below. Thus, there are $(1 + \log T)$ levels where level ℓ contains 2^ℓ intervals, each of size $\frac{T}{2^\ell}$ for $\ell = 0, 1, \ldots, \log T$. This laminar family can be thought of as being a binary tree of depth $\log T$ with the interval $[1, T]$ as the root and the level ℓ intervals being vertices at depth ℓ.

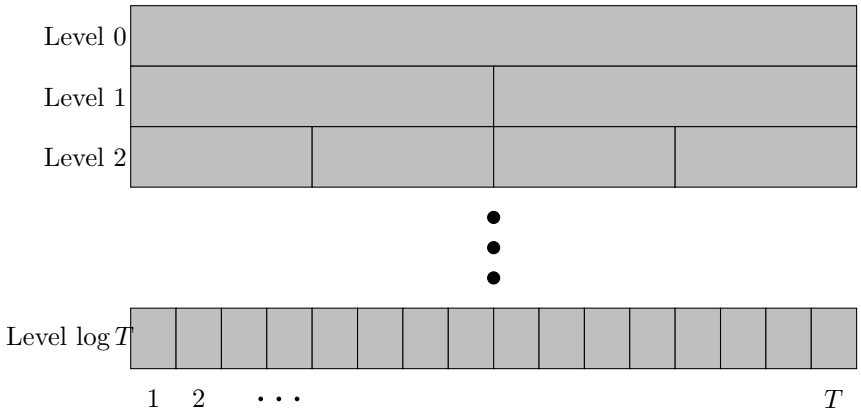


Figure 1 Construction of the laminar family used in the algorithm.

Each job j is first assigned to the smallest interval in this laminar family which fully contains the fractional support of j as per the solution of the LP. Let $k = O(\log \log n)$ and let us call the top k^2 levels in the laminar family as the *top levels* and the level succeeding it, that is the level k^2 as the *bottom level*. Their algorithm conditions (see Section 2 for the definition of conditioning) roughly 2^{k^2} times in order to reduce the maximum chain length among the jobs assigned to the top levels. Once the length of the chains in the top levels is reduced, the last k of the top levels are discarded from the instance, and the sub-instances corresponding to each interval in the bottom level is recursively solved in order to get a partial schedule for all the jobs except those assigned to the top levels. The discarding of the k levels is done in order to create a large gap between the top levels and the bottom level. Having such a gap makes it easier to schedule the remaining jobs in the top levels in the gaps of the partial schedule and Levey-Rothvoss [9] give an elegant algorithm to do this, provided that the maximum chain length among the jobs in the top levels is small. This step increases the makespan by at most a $(1 + \epsilon/\log n)$ factor, which adds up to a loss of a $(1 + \epsilon)$ factor in total over the at most $\log n$ depth of the recursion. Finally, one must also schedule the jobs in the k levels which were discarded; to do this without increasing the makespan by more than a $(1 + \epsilon)$ factor, it suffices to ensure that these k discarded levels contain at most an ϵ fraction of the jobs contained in the top levels. Let us call such a set of k consecutive levels, which contains at most an ϵ fraction of the number of jobs in the levels above it, as a *good batch*.

Now the reason they had to condition $2^{k^2} = 2^{O((\log \log n)^2)}$ times, which leads to the running time of $n^{O(2^{k^2})}$, comes from the fact that they condition on every interval in the top k^2 levels. And this is necessary to ensure that a good batch exists. For example, the number of jobs contained in the levels $[pk, (p + 1)k)$ may be about $e^{\epsilon p} \cdot (\epsilon T/\log n)$ for all $p < (1/\epsilon) \ln(m \log n) \approx k$, in which case there is no good batch in the first $o(k^2)$ levels.

Our approach

To get around the above issue, we observe the following: if, after conditioning on only the top Ck levels, where $C = O(1/\epsilon^2)$ is a big enough constant, there does not exist a good batch in the top Ck levels, then in fact a $(1 - \epsilon)$ fraction of the jobs in the top Ck levels must lie in the last $(1/\epsilon^2)k$ levels, that is, in levels from $Ck - (1/\epsilon^2)k$ to $Ck - 1$. This implies that we can discard the jobs in the first $Ck - (1/\epsilon^2)k$ levels by charging them to the jobs in the levels from $Ck - (1/\epsilon^2)k$ to $Ck - 1$, and in doing so we only discard an ϵ fraction of the total number of jobs.

Notice that we have only conditioned about $2^{Ck} = \text{polylog}(n)$ times till now as there are these many intervals in the top Ck levels. The next crucial observation is that after deleting the top $Ck - (1/\epsilon^2)k$ levels, the sub-instances defined by each of the subtrees rooted at the intervals on the level $Ck - (1/\epsilon^2)k$ can be solved independently of each other. This means that we can perform conditioning in parallel on each such sub-instance, and thus in total, we will condition at most $2^{Ck} \cdot \log T = \text{polylog}(n)$ times, since the depth of the recursion is at most the height of the tree.

Now it might also happen that we already find a good batch in the top Ck levels and in this case, we follow a strategy similar to Levey-Rothvoss [9] by recursing on the bottom intervals to find a partial schedule and fitting the jobs in the top levels in this partial schedule. This step might discard an ϵ fraction of the jobs in the top levels. These two cases, one where we recurse because there is no good batch in the top Ck levels and one where we recurse because there is a good batch in the top Ck levels, might interleave in a complicated manner. We show that the number of jobs ever discarded in the algorithm due to each type of recursion is at most an $O(\epsilon)$ fraction of the total number of jobs, which implies that we can achieve a makespan of $(1 + \epsilon)T$.

The above high-level description skims over a few important issues. One big challenge in the above approach is to ensure that the number of jobs discarded in the cases where we do not find a good batch stays small during the whole algorithm. Even though this is the case in one such recursive call, this might not happen over all the recursive calls taken together and we might end up discarding a constant fraction of the jobs. To get over this obstacle, we carefully control which interval each job is assigned to: if a job j is assigned to an interval I but after conditioning on some job $i \neq j$ which is assigned to a level lower than j, the fractional support of j shrinks to a sub-interval of I, then we will still keep j assigned to I, rather than moving it down the laminar family. This ensures that each job is not charged more than once for discarded jobs and thus the total number of discarded jobs is at most ϵn. This however slightly changes the way jobs are assigned to intervals and the techniques developed by Levey and Rothvoss [9] cannot be immediately applied to fit the jobs of the top levels in the partial schedule of the bottom levels in the case when a good batch exists. To tackle this issue, we will allow each job in the top levels to be scheduled outside of its (current) fractional support as long as it doesn't violate the precedence constraints with the jobs in the bottom levels. With this modification, we will be able to fit the jobs in the top levels in the partial schedule of the bottom levels without discarding more than an ϵ fraction of the top jobs. This implies that in both types of recursions, we only discard an $O(\epsilon)$ fraction of the jobs.

2 Preliminaries on Sherali-Adams Hierarchy

In this section, we state the basic facts about Sherali-Adams hierarchy which we will need. We refer the reader to the excellent surveys [7, 2, 10] for a more extensive introduction to hierarchies.

Consider a linear program with n variables $y_1, \ldots, y_n$ where for each $i \in [n]$, $0 \leq y_i \leq 1$. For $s \geq 0$, the s^{th}-round Sherali-Adams lift of this linear program is another linear program with variables $y_S^{(s)}$ for each $S \subseteq [n]$ satisfying $|S| \leq s+1$, and some additional constraints. We will often denote $y_{\{i\}}^{(s)}$ by $y_i^{(s)}$ for simplicity.

If we think of y_i as the probability that $y_i = 1$, intuitively the variables $y_S^{(s)}$ should equal the probability that each $i \in S$ has $y_i = 1$, that is we would like to have that $y_S^{(s)} = \Pi_{i \in S} y_i$. As these constraints are not convex, we can only impose some linear conditions implied by them. In particular, for every constraint $a^T y \leq b$ of the starting LP, we add, for every $S, T \subseteq [n]$ such that $|S| + |T| \leq s$, a new constraint given by

$$\sum_{T' \subseteq T} (-1)^{|T'|} \left(\sum_{i=1}^{n} a_i y_{S \cup T' \cup \{i\}}^{(s)} - b y_{S \cup T'}^{(s)} \right) \leq 0. \tag{2}$$

If $y_S^{(s)} = \Pi_{i \in S} y_i$ was indeed true for all $|S| \leq s+1$, then the above inequality can be succinctly written as $(a^T y - b) \cdot \Pi_{i \in S} y_i \cdot \Pi_{i \in T}(1 - y_i) \leq 0$, and are thus valid constraints for all $0-1$ solutions.

Observe that an s^{th}-round Sherali-Adams lift of an LP with n variables and m constraints is just another LP with $n^{O(s)}$ variables and $m \cdot n^{O(s)}$ constraints. Letting $y^{(s)}$ denote a feasible solution of the s^{th}-round Sherali-Adams lift, $y^{(s)}$ is also feasible for all $s' \leq s$ rounds of Sherali-Adams and in particular is a feasible solution of the starting LP.

Conditioning

Given a feasible solution $y^{(s)}$ of the s^{th}-round Sherali-Adams lift and $i \in [n]$ such that $y_i^{(s)} > 0$, then we can *condition on the event* $y_i = 1$ to get a feasible solution $z^{(s-1)}$ of the $(s-1)^{th}$-round Sherali-Adams lift defined as

$$z_S^{(s-1)} = \frac{y_{S\cup\{i\}}^{(s)}}{y_i^{(s)}} \qquad \forall S : |S| \leq s.$$

The fact that $z^{(s-1)}$ is a feasible solution of the $(s-1)^{th}$-round Sherali-Adams lift follows easily from (2). Moreover, z satisfies $z_i^{(s-1)} = 1$ and the following useful property:

▶ **Observation 2.** *If for some $j \in [n]$, $y_j^{(s)} = 0$ and we condition on $y_i = 1$ for any $i \in [n]$, then $z_j^{(s-1)} = 0$.*

Proof. Using the Sherali-Adams lift (2) of the constraint $y_i \leq 1$ with $S = \{j\}$ and $T = \phi$, we get $y_{\{i,j\}}^{(s)} \leq y_j^{(s)}$. This gives $z_j^{(s-1)} = \frac{y_{\{i,j\}}^{(s)}}{y_i^{(s)}} \leq \frac{y_j^{(s)}}{y_i^{(s)}} = 0$. ◀

One can think of the solution $z_S^{(s-1)}$ as giving the conditional probability of $y_S^{(s)} = 1$ given $y_i^{(s)} = 1$. By conditioning on a variable y_i to be 1, we will mean that we replace the current fractional solution $y^{(s)}$ with the fractional solution $z^{(s-1)}$ as in above. Observation 2 implies that conditioning can never increase the support of any variable, or in other words, if the probability that $y_j = 1$ is zero, then the conditional probability that $y_j = 1$ conditioned on $y_i = 1$, is also zero.

3 Algorithm

Before we describe our algorithm, we first develop some notation. Let T denote the value of the LP (1) and let $y^{(s)}$ denote the feasible solution of the s^{th}-round Sherali-Adams lift of the LP we get after we condition $r - s$ times in the algorithm. We will say that we are in *round* s of the algorithm if we have conditioned $r - s$ times so far. So we will start the algorithm in round r with solution $y^{(r)}$, and if we condition in round s, we go to round $s-1$ with solution $y^{(s-1)}$.

For each job j, define the *fractional support interval of j in round s* as $F_j^{(s)} := [r_j^s, d_j^s]$, where r_j^s is the smallest time t for which $y_{jt}^{(s)} > 0$ and d_j^s is the largest time for which $y_{jt}^{(s)} > 0$ (r_j and d_j are used to symbolize release time and deadline). In other words, $F_j^{(s)}$ is the minimal interval which fully contains the fractional support of job j in $y^{(s)}$. By Observation 2, upon conditioning, the fractional support interval can only shrink, that is $F_j^{(s-1)} \subseteq F_j^{(s)}$.

For each job j, we also define a *support interval* $S_j^{(s)}$. We will initially set $S_j^{(r)} := F_j^{(r)}$. In later rounds, we will update $S_j^{(s)}$ in such a way that $F_j^{(s)} \subseteq S_j^{(s)} \subseteq F_j^{(r)}$. Intuitively, $S_j^{(s)}$ reflects our knowledge in round s of where j can be scheduled. Notice that we might schedule j outside of the fractional support interval $F_j^{(s)}$.

A schedule of jobs is called a *feasible schedule* if it satisfies the precedence constraints among all the jobs and a *partial feasible schedule* if it schedules some of the jobs and satisfies the precedence constraints among them. In order to get a feasible schedule of all the jobs with a makespan of at most $(1+\epsilon)T$, it suffices to show the following:

▶ **Theorem 3.** *We can find a partial feasible schedule $\sigma : [n] \to [T] \cup \{DISCARDED\}$ such that $\sigma(j) = DISCARDED$ for at most ϵT jobs.*

Clearly a schedule σ as in Theorem 3 has makespan at most T. Having such a partial feasible schedule, we can easily convert it to a feasible schedule of all the n jobs with a makespan of at most $(1+\epsilon)T$: iterate through every job j discarded in σ and find the earliest time t by when all the jobs which must precede j have either already been scheduled or are as of yet discarded. Create a new time slot between times t and $t+1$ containing only job j. This increases the makespan by one for every job discarded in σ.

Laminar Family.

A laminar family of intervals is defined in the following manner. The topmost level, level 0, has one interval $[1, T]$. Each succeeding level is constructed by dividing each interval of the previous level into two equal sized intervals[2]. Thus there are $(1 + \log T)$ levels, where level ℓ contains 2^ℓ intervals each of size $\frac{T}{2^\ell}$ for $\ell = 0, 1, \ldots, \log T$. This laminar family can be thought of as being a binary tree of depth $\log T$ with the interval $[1, T]$ as the root, and the level ℓ intervals as being vertices at depth ℓ.

Let $\mathcal{I}_\ell$ denote the set of intervals at level ℓ of the laminar family. For an interval $I \in \mathcal{I}_\ell$, a *sub-interval of* I is any interval $I' \subseteq I$ of the laminar family, including I itself; and $I_{\text{left}}, I_{\text{right}}$ will denote the left and right sub-intervals respectively of I in $\mathcal{I}_{\ell+1}$. By the midpoint of I, we will mean the right boundary of I_{left}.

Job j is *assigned* to interval I in round s if I is the smallest interval in the laminar family such that $S_j^{(s)} \subseteq I$. This assignment of jobs to intervals depends on $S_j^{(s)}$ and will change as s and $S_j^{(s)}$ change during the algorithm. Let $I^{(s)}(j)$ denote the interval to which j is assigned in round s of the algorithm. For an interval I in the laminar family, let $\mathcal{J}^{(s)}(I)$ denote the set of jobs assigned to I, and let $\mathcal{J}^{(s)}(\mathcal{I}_\ell)$ denote the set of jobs assigned to intervals in $\mathcal{I}_\ell$ in round s of the algorithm.

Batches.

Let $k = \log(\frac{32m}{\epsilon} \cdot \log n)$. For $p \geq 0$, define the p^{th} batch as

$$\mathcal{B}_p = \{\mathcal{I}_{pk}, \mathcal{I}_{pk+1}, \ldots, \mathcal{I}_{(p+1)k-1}\}.$$

That is, it denotes the set of k consecutive levels starting from level pk till level $(p+1)k-1$. Let $\mathcal{J}^{(s)}(\mathcal{B}_p)$ denote the set of jobs assigned to intervals in batch p in round s. Batch $\mathcal{B}_p$ for $p \geq 1$ is called a *good batch with respect to* $[T]$ *in round* s if

$$|\mathcal{J}^{(s)}(\mathcal{B}_p)| \leq \frac{\epsilon}{4m} \sum_{i=0}^{p-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)|. \tag{3}$$

We will omit the "with respect to $[T]$" if it is clear from the context that we start the summation in the right hand side of (3) from the first batch in $[T]$. Similarly, we will omit the "in round s" if s is clear from the context.

Algorithm.

We can now describe our algorithm and split its description in two steps for clearer exposition. Let $C = 2(4m/\epsilon)^2 + 1, k = \log(\frac{32m}{\epsilon} \cdot \log n)$ and $\delta = \frac{\epsilon}{8mCk2^{Ck}\log n}$. The reader can think of

[2] Without loss of generality, T is a power of 2. Otherwise, we can add a few dummy jobs at the end which must succeed all other jobs and which make T a power of 2.

these parameters as being $k = \Theta_{m,\epsilon}(\log\log n)$ and $\delta = \Theta_{m,\epsilon}(1/\text{polylog}(n))$. s will always denote the current round of the algorithm, unless otherwise specified. We initialise $s := r$ and for each job j, $S_j^{(r)} := F_j^{(r)}$.

Schedule($y^{(r)}, T$):[3]

1. **Step 1: Reducing chain length in the top $qk \leq Ck$ levels**

   In this step, we will reduce the length of the chains in each interval I in the top qk levels of the laminar family to at most $\delta|I|$, for some $q \leq C$. This is done by going down the levels, starting from level 0 till level $qk-1$, where q is chosen such that

   a. after having conditioned on all the levels from 0 to $qk-1$, $\mathcal{B}_{q-1}$ is a good batch, or

   b. we have already conditioned on the top Ck levels and found no good batch, in which case we set $q = C$.

   The conditioning on the levels and update of S_j's is done as follows. For $\ell = 0, 1, \ldots, qk-1$:

   - Let $s_{\text{old}} = s$ and for each j, let $\ell(j)$ denote the level of the interval $I^{(s_{\text{old}})}(j)$, that is the level to which j is assigned at the beginning of this iteration of the loop.
   - We go over every interval $I \in \mathcal{I}_\ell$ and do the following: if $\mathcal{J}^{(s)}(I)$ has a chain of length more than $\delta|I|$, let j be the first job in this chain. We condition on j lying in I_{right}. After every conditioning, update $s := s-1$ and set $S_j^{(s)}$ for every job j as follows:
     - if $\ell(j) < \ell$, let m_j denote the midpoint of $I^{(s_{\text{old}})}(j)$ and $[t_r, t_d] := F_j^{(s)}$. If $F_j^{(s)} \subseteq I_{\text{left}}$, then we set $S_j^{(s)} := [t_r, m_j + 1]$, and if $F_j^{(s)} \subseteq I_{\text{right}}$, then we set $S_j^{(s)} := [m_j, t_d]$. Otherwise, set $S_j^{(s)} := F_j^{(s)}$.
     - if $\ell(j) \geq \ell$, set $S_j^{(s)} := F_j^{(s)}$.

   That is, the support intervals $S_j^{(s)}$ are set such that if we condition on jobs in level ℓ, then the jobs assigned to a level $\ell' < \ell$ before the conditionings stay assigned to level ℓ', and for all other jobs, $S_j^{(s)}$ equals the fractional support interval $F_j^{(s)}$.

2. **Step 2: Recursion**

   There are two cases to consider here, depending on which of (a) or (b) took place in the previous step.

   (i) If (a) occurred, perform a recursion of type 1.

   This step is similar to the algorithm of [9]. We discard all the jobs in the good batch $\mathcal{B}_{q-1}$. Then for each interval $I \in \mathcal{I}_{qk}$, we recursively call Schedule$(y^{(s)}, I)$ to obtain a schedule $\tilde{\sigma}_I$, which are put together to form a partial feasible schedule $\tilde{\sigma}$ for the jobs assigned to a level $\ell \geq qk$.

   Then we fit the jobs in the top levels, that is the jobs in $\mathcal{J}^{(s)}(\mathcal{B}_0) \cup \cdots \cup \mathcal{J}^{(s)}(\mathcal{B}_{q-2})$ in the empty slots in $\tilde{\sigma}$. We give more details of how this is done in Section 4.2.1. Some jobs in the top levels will be discarded while doing this.

   Call this step a *recursion of type* 1. The number of jobs discarded in this step, that is the jobs in batch $\mathcal{B}_{q-1}$ along with the jobs in the top levels which are discarded, will be referred to as the jobs discarded due to this step. Notice that this does not include the jobs discarded in each recursive call to the intervals in $\mathcal{I}_{qk}$.

   (ii) If (b) occurred, perform a recursion of type 2.

   In this case, we discard all the jobs in $\mathcal{J}^{(s)}(\mathcal{B}_0) \cup \cdots \cup \mathcal{J}^{(s)}(\mathcal{B}_{C-(4m/\epsilon)^2-1})$. Then for each interval $I \in \mathcal{I}_{(C-(4m/\epsilon)^2)k}$, we recursively call Schedule$(y^{(s)}, I)$ to get a

[3] When the algorithm is called on an interval of length $L \leq 2^{Ck}$, we can just "brute force" by conditioning mL times to find an exact solution. We avoid writing this explicitly in the algorithm for simplicity.

schedule σ_I which are put together to form a partial feasible schedule σ for all the jobs assigned to a level $\ell \geq (C - (4m/\epsilon)^2)k$.

Call this step a *recursion of type 2*. The number of jobs discarded in this step, that is the jobs in batches $\mathcal{B}_0, \ldots, \mathcal{B}_{C-(4m/\epsilon)^2-1}$ will be referred to as the jobs discarded due to this step. Just like before, this does not include the jobs discarded in each recursive call to the intervals in $\mathcal{I}_{(C-(4m/\epsilon)^2)k}$.

In each type of recursion, we recurse on multiple sub-instances defined by intervals of some level. It is important that the recursions on these sub-instances are done independently of each other. That is, we pass the same (current) Sherali-Adams solution to each recursive call, and conditionings done in one recursive call are independent of conditionings done in any other recursive call, and thus do not affect the fractional solution of any other recursive call.

4 Analysis

In this section, we prove Theorem 3 which will imply Theorem 1. We split the analysis into two parts: in the first part, we give a bound on the number of rounds of Sherali-Adams needed in the algorithm. In the second part, we show that we discard at most ϵT jobs during the algorithm and schedule all other jobs by time T, thus proving Theorem 3.

But first, we need to show that the algorithm is well-defined.

▶ **Observation 4.** *In step* 1*, when we condition on an interval I by finding a chain $\mathcal{C}$ in I and conditioning the first job j in this chain to lie in I_{right}, this is possible to do. Moreover, this assigns every job in $\mathcal{C}$ to a sub-interval of I_{right}.*

Proof. For the first part of the observation, we need to show that $F_j^{(s)} \cap I_{\text{right}} \neq \phi$, where s is the round of the algorithm just before we condition on j in I. As j is assigned to I in round s, it must be that $S_j^{(s)} \cap I_{\text{right}} \neq \phi$. The support intervals are updated in a way such that we can only have $S_j^{(s)} \neq F_j^{(s)}$ after we condition on a level below that of j. But because we always condition on the levels from top to bottom, we must have $S_j^{(s)} = F_j^{(s)}$. This proves the first part of the observation.

The moreover part follows easily now since every other job $i \in \mathcal{C}$ satisfies $j \prec i$ and must start scheduling only after j. ◀

4.1 Bounding number of rounds of Sherali-Adams

Let $r(|I|)$ denote the number of rounds of Sherali-Adams the algorithm uses when run on the instance defined by the subtree rooted at interval I of the laminar family. Our goal in this subsection is to show $r(T) \leq r$ for $r = O_{m,\epsilon}(\log^{O(m^2/\epsilon^2)} n)$.

We first give an upper bound on the number of conditionings done in one interval I.

▶ **Lemma 5.** *The algorithm conditions at most m/δ times on any interval I in step* 1*.*

Proof. Let s_{old} denote the round of the algorithm just before we start to condition in I, and let $\ell \geq 0$ be such that $I \in \mathcal{I}_\ell$. Each time we condition in I, we assign at least $\delta|I|$ jobs in I to a sub-interval of I_{right} (by Observation 4).

Also, no job assigned to a level $\ell' < \ell$ in round s_{old} moves down the laminar family during conditionings done in I. And for all other jobs, they only get assigned to a sub-interval. Thus no new job is assigned to I while we are conditioning in I.

Using $F_j^{(s_{\text{old}})} \subseteq S_j^{(s_{\text{old}})}$ and the second constraint of LP (1), there can be at most $m|I|$ jobs in total assigned to I in round s_{old}. Thus, the number of times we condition in I is at most $\frac{m|I|}{\delta|I|} = \frac{m}{\delta}$. ◀

▶ **Lemma 6.** *The algorithm conditions at most $2^{Ck}m/\delta$ times in step 1 of the algorithm.*

Proof. By Lemma 5, we condition at most m/δ times per interval. As we condition on the topmost qk levels and hence on at most $2^{qk} \leq 2^{Ck}$ intervals, we condition at most $2^{Ck}m/\delta$ times in step 1. ◀

In step 2 of the algorithm, if we do a recursion of type 1 then we recurse on every interval at level $qk \geq k$. Otherwise, if we do a recursion of type 2 then we recurse on every interval at level $(C - (4m/\epsilon)^2)k \geq k$. In either case we recurse on every interval of some level $\ell \geq k$ and thus on an interval of size at most $T/2^k$. Because the conditionings done in one recursive call are done independently of the conditionings in any other recursive call, the total number of rounds of Sherali-Adams we need can be bounded by the following recurrence:

$$r(T) \leq \frac{2^{Ck}m}{\delta} + r(T/2^k)$$

where the base case is $r(2^{Ck}) = 2^{Ck}m$, and thus we get

$$r(T) \quad \leq \quad \frac{2^{Ck}m \log T}{\delta} \leq \frac{8m^2Ck(\log^2 n)2^{2Ck}}{\epsilon} = O_{m,\epsilon}((\log n)^{5+\frac{64m^2}{\epsilon^2}}) = r.$$

4.2 Bounding number of jobs discarded

In this subsection, we bound the number of jobs discarded in the algorithm and show that it is at most ϵT. We will separately bound the number of jobs discarded due to recursions of type 1 and recursions of type 2 and show that each is at most $\epsilon T/2$. The former uses a result proved by Levey and Rothvoss [9] but which needs to be heavily adapted to our algorithm. The latter uses a simple charging argument.

4.2.1 Jobs discarded due to recursions of type 1.

Suppose we perform a recursion of type 1 when the algorithm is called on the interval I of the laminar family. To be consistent with the notation of [9], we will call the set of jobs $\mathcal{J}^{(s)}(\mathcal{B}_{q-1})$ as $\mathcal{J}_{\text{middle}}$, the set of jobs $\mathcal{J}^{(s)}(\mathcal{B}_0) \cup \cdots \cup \mathcal{J}^{(s)}(\mathcal{B}_{q-2})$ as $\mathcal{J}_{\text{top}}$ and the jobs in the levels below these as $\mathcal{J}_{\text{bottom}}$ (here we are reindexing the batches such that the first level starts from interval I).

▶ **Claim 7.**

$$|\mathcal{J}_{middle}| \leq \frac{\epsilon}{4m}|\mathcal{J}_{top}|.$$

Proof. Follows from the fact that $\mathcal{B}_{q-1}$ is a good batch and (3). ◀

After discarding all the jobs in $\mathcal{J}_{\text{middle}}$, the algorithm recursively finds a partial feasible schedule $\tilde{\sigma}$ of the jobs in $\mathcal{J}_{\text{bottom}}$. Let $\mathcal{J}' \subseteq \mathcal{J}_{\text{bottom}}$ be the set of jobs scheduled by $\tilde{\sigma}$. The algorithm will then attempt to extend $\tilde{\sigma}$ to a schedule σ of the jobs in $\mathcal{J}_{\text{top}} \cup \mathcal{J}'$. We will be able to do this by discarding only a few jobs from $\mathcal{J}_{\text{top}}$. More formally:

▶ **Lemma 8.** *When the algorithm is called on an interval I, we can extend $\tilde{\sigma}$ to a feasible schedule σ of the jobs in $(\mathcal{J}_{top} \setminus \mathcal{J}_{discard}) \cup \mathcal{J}'$ where*

$$|\mathcal{J}_{discard}| \leq \frac{\epsilon |I|}{4 \log n}.$$

We defer the proof of Lemma 8 to the full version of the paper. We show below how Lemma 8 implies that we discard at most $\epsilon T/2$ jobs in all recursions of type 1.

▶ **Lemma 9.** *Total number of jobs discarded in all recursions of type 1 during the algorithm is at most $\epsilon T/2$.*

Proof. Using Claim 7 and Lemma 8, if we perform a recursion of type 1 when the algorithm is called on the interval I, the number of jobs discarded is at most

$$\frac{\epsilon}{4m} |\mathcal{J}_{\text{top}}| + \frac{\epsilon |I|}{4 \log n}.$$

Over all recursions of type 1, the first term sums up to at most $\epsilon n/4m \leq \epsilon T/4$. For any $\ell \geq 0$, the second term sums up to $\frac{\epsilon T}{4 \log n}$ over all intervals $I \in \mathcal{I}_\ell$. As there are at most $\log n$ levels, the second term also sums up to $\epsilon T/4$ over all recursions of type 1. ◀

4.2.2 Jobs discarded due to recursions of type 2.

Let $\epsilon' = \epsilon/4m$. Recall that in a recursion of type 2, we delete all the jobs assigned to levels 0 to $(C - (1/\epsilon')^2)k - 1$ and retain only the later $(1/\epsilon')^2$ batches. We show below that in such a case, at least a $(1 - \epsilon')$ fraction of the jobs in the top C batches are in the last $(1/\epsilon')^2$ batches and thus, by deleting the jobs in the first $C - (1/\epsilon')^2$ batches we only delete an ϵ' fraction of the jobs.

▶ **Lemma 10.** *If case (b) occurs in step 1 of the algorithm, then*

$$\sum_{i=0}^{C-(1/\epsilon')^2-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)| \leq \epsilon' \sum_{i=C-(1/\epsilon')^2}^{C-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)|. \tag{4}$$

Proof. Let $S = \sum_{i=0}^{C-(1/\epsilon')^2-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)|$, the left hand side of (4). Case (b) occurs in step 1 of the algorithm if none of the batches $\mathcal{B}_p$ for $p \in [C - (1/\epsilon')^2, C - 1]$ are good. But then for $p \in [C - (1/\epsilon')^2, C - 1]$, we must have

$$|\mathcal{J}^{(s)}(\mathcal{B}_p)| > \epsilon' \sum_{i=0}^{p-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)| \geq \epsilon' \sum_{i=0}^{C-(1/\epsilon')^2-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)| = \epsilon' S.$$

This implies (4) as

$$\sum_{i=C-(1/\epsilon')^2}^{C-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)| \geq \sum_{i=C-(1/\epsilon')^2}^{C-1} \epsilon' S = \left(\frac{1}{\epsilon'}\right)^2 \epsilon' S = \frac{S}{\epsilon'}.$$ ◀

This implies that when we discard the top $C - (1/\epsilon')^2$ batches, we are only discarding at most an ϵ' fraction of the jobs in the next $(1/\epsilon')^2$ batches. We can imagine this as putting a charge of ϵ' on every job in the last $(1/\epsilon')^2$ batches. Thus the total charge on all the jobs at the end of the algorithm is an upper bound on the number of jobs discarded in recursions of type 2 during the algorithm.

▶ **Lemma 11.** *For every job j, we put a charge on j at most once.*

Proof. Fix a job j and suppose we put a charge on j at least once. When we put a charge on j for the first time, then in some recursion of type 2 it must have been assigned to the lowest $(1/\epsilon')^2$ batches among the top C batches. The algorithm will then recurse on every interval at level $(C-(1/\epsilon')^2)k$ and thus job j is now in the top $(1/\epsilon')^2$ batches in one of the recursive calls.

Let $I \in \mathcal{I}_{(C-(1/\epsilon')^2)k}$ be such that j is assigned to a sub-interval of I. When we recursively call the algorithm on I, the first $(1/\epsilon')^2$ batches already satisfy the property that any interval I' in them has maximum chain length at most $\delta|I'|$. Thus in step 1 of the algorithm, we will not condition on any interval in the top $(1/\epsilon')^2$ batches. This implies that job j always stays assigned to the top $(1/\epsilon')^2$ batches; this is because the assignment of a job to an interval can only change when we condition on an interval at the same level or at a level above that of the job.

Now suppose we put a charge on j again. Then we must have once again done a recursion of type 2 within the recursive call to I. But j is assigned to the topmost $(1/\epsilon')^2$ batches in this instance and in a recursion of type 2, we delete the topmost $C-(1/\epsilon')^2 > (1/\epsilon')^2$ batches and put a charge on only the later $(1/\epsilon')^2$ batches, which leads to a contradiction. ◀

▶ **Lemma 12.** *Number of jobs discarded in recursions of type 2 throughout the algorithm is at most $\epsilon T/2$.*

Proof. Because the number of jobs discarded in recursions of type 2 throughout the algorithm is at most the total charge on all the jobs and by Lemma 11, each job is charged at most once, we get that the number of jobs discarded in recursions of type 2 is at most

$$\epsilon' n = \epsilon n/4m \leq \epsilon T/4 \leq \epsilon T/2.$$ ◀

References

1. Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 453–462. IEEE, 2009.
2. Eden Chlamtac and Madhur Tulsiani. Convex relaxations and integrality gaps. In *Handbook on semidefinite, conic and polynomial optimization*, pages 139–169. Springer, 2012.
3. 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.
4. Michael R Garey. Ds johnson computers and intractability. *A Guide to the Theory of NP-Completeness*, 1979.
5. Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell Labs 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. Monique Laurent. A comparison of the sherali-adams, lovász-schrijver, and lasserre relaxations for 0–1 programming. *Mathematics of Operations Research*, 28(3):470–496, 2003.
8. Jan Karel Lenstra and AHG Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.
9. Elaine Levey and Thomas Rothvoss. A (1+epsilon)-approximation for makespan scheduling with precedence constraints using LP hierarchies. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 168–177. ACM, 2016. URL: `http://dl.acm.org/citation.cfm?id=2897518`, `doi:10.1145/2897518.2897532`.

10 Thomas Rothvoß. The lasserre hierarchy in approximation algorithms. *Lecture Notes for the MAPSP*, pages 1–25, 2013.

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

12 Hanif D Sherali and Warren P Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

13 Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 745–754. ACM, 2010.