

Scheduling on Unrelated Machines under Tree-Like Precedence Constraints

V.S. ANIL KUMAR ¹ MADHAV V. MARATHE ² SRINIVASAN PARTHASARATHY ³
ARAVIND SRINIVASAN ⁴

Abstract

We present polylogarithmic approximations for the $R|prec|C_{max}$ and $R|prec|\sum_j w_j C_j$ problems, when the precedence constraints are “treelike” - i.e., when the undirected graph underlying the precedences is a forest. These are the first non-trivial generalizations of the job shop scheduling problem to scheduling with precedence constraints that are not just chains. These are also the first non-trivial results for the weighted completion time objective on unrelated machines with precedence constraints of *any kind*. We obtain improved bounds for the weighted completion time and flow time for the case of chains with restricted assignment - this generalizes the job shop problem to these objective functions. We use the same lower bound of “congestion+dilation”, as in other job shop scheduling approaches (e.g. [21]). The first step in our algorithm for the $R|prec|C_{max}$ problem with treelike precedences involves using the algorithm of Lenstra, Shmoys and Tardos [13] to obtain a processor assignment with the congestion + dilation value within a constant factor of the optimal. We then show how to generalize the random delays technique of Leighton, Maggs and Rao [14] to the case of trees. For the weighted completion time, we show a certain type of reduction to the makespan problem, which dovetails well with the lower bound we employ for the makespan problem. For the special case of chains, we show a dependent rounding technique which leads to improved bounds on the weighted completion time and new bicriteria bounds for the flow time.

1 Introduction

The most general scheduling problem involves unrelated parallel machines and precedence constraints, i.e., we are given: (i) a set of n jobs with precedence constraints that induce a partial order on the jobs; (ii) a set of m machines, each of which can process at most one job at any time, and (iii) an arbitrary set of integer values $\{p_{i,j}\}$, where $p_{i,j}$ denotes the time to process job j on machine i . Let C_j denote the completion time of job j . Subject to the above constraints, two commonly studied versions are (i) minimize the *makespan*, or the maximum time any job takes, i.e. $\max_j \{C_j\}$ - this is denoted by $R|prec|C_{max}$, and (ii) minimize the weighted completion time - this is denoted by $R|prec|\sum_j w_j C_j$. Numerous other variants, involving release dates or other objectives have been studied (see e.g. [7]).

¹Basic and Applied Simulation Science (CCS-DSS), Los Alamos National Laboratory, MS M997, P.O. Box 1663, Los Alamos, NM 87545. Email: {anil,marathe}@lanl.gov. The work is supported by the Department of Energy under Contract W-7405-ENG-36.

²Virginia Bio-informatics Institute and Department of Computer Science Virginia Tech, Blacksburg 24061. Email: mmarathe@vbi.vt.edu.

³Department of Computer Science, University of Maryland, College Park, MD 20742. Most of the work was done while visiting Los Alamos National Laboratory; research supported in part by NSF Award CCR-0208005. Email: sri@cs.umd.edu.

⁴Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported in part by NSF Award CCR-0208005. Email: srin@cs.umd.edu.

Almost optimal upper and lower bounds are known for the versions of the above problems without precedence constraints (i.e., the $R||C_{max}$ and $R||\sum_j w_j C_j$ problems) [3, 13, 22], but very little is known in the presence of precedence constraints. The only case of the general $R|prec|C_{max}$ problem for which non-trivial approximations are known is the case where the precedence constraints are chains - this is the job shop scheduling problem [21], which itself has a long history. The first result for job shop scheduling was the breakthrough work of Leighton et al. [14, 15] for packet scheduling, which implied an $O(\log n)$ approximation for the case of unit processing costs. Leighton et al. [14, 15] introduced the “random delays” technique, and almost all the results on the job shop scheduling problem are based on variants of this technique. The result of [14, 15] was generalized to non uniform processing costs by Shmoys et al. [21], who obtained an approximation factor of $O(\log(m\mu) \log(\min\{m\mu, p_{max}\}) / \log \log(m\mu))$, where p_{max} is the maximum processing time of any job, and μ is the maximum length of any chain in the given precedence constraints. These bounds were improved by an additional $\log \log(m\mu)$ factor by Goldberg et al. [6]; see [5] for additional relevant work. Shmoys et al. [21] also generalize job-shop scheduling to DAG-shop scheduling, where the operations of each job form a DAG, instead of a chain, with the additional constraint that *the operations within a job can be done only one at a time*. They show how the results for the case of a chain extend to this case also.

The only results known for the case of arbitrary number of processors with more general precedence constraints are for identical parallel machines (denoted by $P|prec|C_{max}$) [7], or for related parallel machines (denoted by $Q|prec|C_{max}$) [4, 2]. The weighted completion time objective has also been studied for these variants [3, 8]. When the number of machines is constant, polynomial time approximation schemes are known [9, 11]. Note that all of the above discussion relates to *non-preemptive* schedules, i.e., once the processing of a job is started, it cannot be stopped until it is completely processed; preemptive variants of these problems have also been well studied (see e.g. [19]).

Far less is known for the weighted completion time objective in the same setting, instead of the makespan. The known approximations are either for the case of no precedence constraints [22], or for precedence constraints with parallel/related processors [8]. To the best of our knowledge, no non-trivial bound is known on the weighted completion time on unrelated machines, in the presence of precedence constraints of *any kind*.

Here, motivated by applications such as evaluating large expression-trees and tree-shaped parallel processes, we consider the special case of the $R|prec|C_{max}$ and $R|prec|\sum_j w_j C_j$ problems, where the precedences form a forest, i.e., the undirected graph underlying the precedences is a forest. Thus, this naturally generalizes the job shop scheduling problem, where the precedence constraints form a collection of disjoint directed chains.

Summary of results We present the first polynomial time approximation algorithms for the $R|prec|C_{max}$ and $R|prec|\sum_j w_j C_j$ problems, under “treelike” precedences. As mentioned earlier, these are the first non-trivial generalizations of the job shop scheduling problems to precedence constraints which are not chains. Since most of our results hold in the cases where the precedences form a forest (i.e., the undirected graph underlying the DAG is a forest), we will denote the problems by $R|forest|C_{max}$, and $R|forest|\sum_j w_j C_j$, respectively, to simplify the description - this generalizes the notation used by [10] for the case of chains.

1. The $R|forest|C_{max}$ problem. We obtain a polylogarithmic approximation for this problem. We employ the same lower bound used in [14, 21, 6, 5]: $LB \doteq \max\{P_{max}, \Pi_{max}\}$, where P_{max} is the maximum processing time along any directed path and Π_{max} is the maximum processing time needed by any machine. Let $p_{max} = \max_{i,j} p_{i,j}$ be the maximum processing time of any job on any machine. We obtain an $O(\frac{\log^2 n}{\log \log n} \lceil \frac{\log \min(p_{max}, n)}{\log \log n} \rceil)$ approximation to the $R|forest|C_{max}$ problem. When the forests are out-trees or in-trees, we show that this polylogarithmic factor can be improved to $O(\log n \cdot \lceil \log(\min\{p_{max}, n\}) / \log \log n \rceil)$; for the special case of unit processing times, this actually be-

comes $O(\log n)$. We also show that the lower-bound LB cannot be put to much better use, even in the case of trees - for unit processing costs, we show instances whose optimal schedule is $\Omega(LB \cdot \log n)$.

Our algorithm for solving $R|forest|C_{max}$ follows the overall approach used to solve the job shop scheduling problem (see, e.g. [21]) and involves two steps: (1) We show how to compute a processor assignment within a $(\frac{3+\sqrt{5}}{2})$ -factor of LB , by extending the approach of [13], and, (2) We design a poly-logarithmic approximation algorithm for the resulting variant of the $R|prec|C_{max}$ problem with pre-specified processor assignment, and forest shaped precedences.

We call the variant of the $R|prec|C_{max}$ problem arising in step (2) above (i.e., when the processor assignment is pre-specified), the *Generalized DAG-Shop Scheduling* or the GDSS problem, for brevity. Note that the job shop scheduling problem is a special case of GDSS, and this problem is different from the Dagshop scheduling problem defined by [21].⁵ Our algorithm for treelike instances of GDSS is similar to one used in [14, 21, 6], namely injecting random delays to the start times of the jobs; this allows for contention resolution. However, unlike [14, 21, 6], it is not adequate to insert random delays only at the head of the trees - we actually insert random delays throughout the schedule. Our algorithm partitions the forest into blocks of chains suitably, and the problem restricted to a block of chains is simply a job shop problem; also, the decomposition guarantees that the solutions to these job shop problems can be pasted together to get a complete schedule - this immediately gives us a reduction from the $R|forest|C_{max}$ problem to the job shop problem, with the quality depending on the number of blocks. We can remove a logarithmic factor when the DAG is an in-/out-tree, by a different analysis, which does not reduce this problem to a collection of job shop problems. As in the original approach of [14], we bound the contention by a Chernoff bound. However, the events we need to consider are not independent, and we need to exploit the variant of this bound from [18], that works in the presence of correlations.

2. The $R|forest|\sum_j w_j C_j$ problem. We show a reduction from $R|prec|\sum_j w_j C_j$ to $R|prec|C_{max}$ of the following form: if there is a schedule of makespan $(P_{max} + \Pi_{max}) \cdot \rho$ for the latter, then there is an $O(\rho)$ -approximation algorithm for the former. We exploit this, along with the fact that our approximation guarantee for $R|forest|C_{max}$ is of the form “ $(P_{max} + \Pi_{max})$ times polylog”, to get a polylogarithmic approximation for the $R|forest|\sum_j w_j C_j$ problem. Our reduction is based on an LP rounding approach, and reduces the problem to a collection of $R|forest|C_{max}$ problems; we use the result of [13] in this rounding. The LP we use has exponentially many constraints with a polytime separation oracle, and works for any set of precedence constraints, not just trees. However, the specific lower bound we use for the $R|forest|C_{max}$ problem is crucial, since the overall approximation factor is small only in the cases where ρ is small.

3. Minimizing weighted completion time and flow time on chains. For a variant of the $R|forest|\sum_j w_j C_j$ problem where (i) the forest is a collection of chains (i.e., the weighted completion time variant of the job shop scheduling problem), and (ii) for each machine i and operation v , $p_{i,v} \in \{p_v, \infty\}$ (i.e., the *restricted-assignment variant*), we show a better approximation of $O(\log n / \log \log n)$ to the weighted completion time. Our result ensures that (i) the precedence constraints are satisfied with *probability* 1, and (ii) for any (v, t) , the probability of scheduling v at time t equals its fractional (LP) value $x_{v,t}$. This result also leads to a bicriteria $(1 + o(1))$ -approximation for weighted flow time variant of this problem, using $O(\log n / \log \log n)$ copies of each machine.

Organization. We discuss the algorithms for the $R|forest|C_{max}$ and $R|forest|\sum_j w_j C_j$ problems in Sections 2 and 3, respectively. We discuss improved bounds on the weighted completion time for the case of chains in Section 4. Because of space limitations, several proofs and algorithm details are presented in the Appendix.

⁵In the Dagshop problem [21], the input is a collection of DAGs, but in each DAG, at most one operation could be done at a time.

2 The $R|forest|C_{max}$ problem

Let x denote any processor assignment, i.e., $x_{i,j}$ is the fraction of job j assigned to machine i . In the description below, we will use the terms “node” and “job” interchangeably; we will not use the term “operation” to refer to nodes of a DAG, because we do not have the job shop or dag shop constraints that at most one node in a DAG can be processed at a time. As before, P_{max} denotes the maximum processing time along any directed path, i.e., $P_{max} = \max_{\text{path}} P\{\sum_{j \in P} \sum_i x_{i,j} p_{i,j}\}$. Also, Π_{max} denotes the maximum load on any machine, i.e., $\Pi_{max} = \max_i \{\sum_j x_{i,j} p_{i,j}\}$. Our algorithm for the $R|prec, Forest|C_{max}$ problem involves the following two steps: **Step 1:** We first construct a processor assignment for which the value of $\max\{P_{max}, \Pi_{max}\}$ is within a constant factor $((3 + \sqrt{5})/2)$ of the smallest-possible. This is described in Section 2.1. **Step 2:** Solve the GDSS problem we get from the previous step to get a schedule of length polylogarithmically more than $\max\{P_{max}, \Pi_{max}\}$. This is described in Section 2.2.

2.1 Step 1: A processor assignment within a constant factor of $\max\{P_{max}, \Pi_{max}\}$

We now describe the algorithm for processor assignment, using some of the ideas from [13]. Let T be our “guess” for the optimal value of $LB = \max\{P_{max}, \Pi_{max}\}$. Define $S_T = \{(i, j) \mid p_{ij} \leq T\}$. Let J and M denote the set of jobs and machines, respectively. We now define a family of linear programs $LP(T)$, one for each value of $T \in \mathbf{Z}^+$, as follows: **(A1)** $\forall j \in J \sum_i x_{ij} = 1$, **(A2)** $\forall i \in M \sum_j x_{ij} p_{ij} \leq T$, **(A3)** $\forall j \in J z_j = \sum_i p_{ij} x_{ij}$, **(A4)** $\forall (j' \prec j) c_j \geq c_{j'} + z_j$, **(A5)** $\forall j \in J c_j \leq T$. The constraints (A1) ensure that each job is assigned a machine, constraints (A2) ensure that the maximum fractional load on any machine (Π_{max}) is at most T . Constraints (A3) define the fractional processing time z_j for a job j and (A4) capture the precedence constraints amongst jobs (c_j denotes the fractional completion of time of job j). We note that $\max_j c_j$ is the fractional P_{max} . Constraints (A5) state that the fractional P_{max} value is at most T .

Let T^* be the smallest value of T for which $LP(T)$ has a feasible solution. It is easy to see that T^* is a lower bound on LB . We now present a rounding scheme which rounds a feasible fractional solution to $LP(T^*)$ to an integral solution. Let X_{ij} denote the indicator variable which denotes if job j was assigned to machine i in the integral solution, and let C_j be the integer analog of c_j and z_j . We first modify the x_{ij} values using *filtering* [16]. Let $\mu = \frac{3+\sqrt{5}}{2}$. For any (i, j) , if $p_{ij} > \mu z_j$, then set x_{ij} to zero. This step could result in a situation where, for a job j , the fractional assignment $\sum_i x_{ij}$ drops to a value r such that $r \in [1 - \frac{1}{\mu}, 1)$. So, we scale the (modified) values of x_{ij} by a factor of at most $\gamma = \frac{\mu}{\mu-1}$. Let \mathcal{A} denote this fractional solution. Crucially, we note that any rounding of \mathcal{A} , which ensures that only non-zero variables in \mathcal{A} are set to non-zero values in the integral solution, has an integral P_{max} value which is at most μT^* . This follows from the fact that if $X_{ij} = 1$ in the rounded solution, then $p_{ij} \leq \mu z_j$. Hence, it is easy to see that by induction, for any job j , C_j is at most $\mu c_j \leq \mu T^*$.

We now show how to round \mathcal{A} . Recall that [13] presents a rounding algorithm for unrelated parallel machines scheduling *without* precedence constraints with the following guarantee: if the input fractional solution has a fractional Π_{max} value of x , then the output integral solution has an integral Π_{max} value of at most $x + \max_{x_{ij} > 0} p_{ij}$. We use \mathcal{A} as the input instance for the rounding algorithm in [13]. Note that \mathcal{A} has a fractional Π_{max} value of at most γT^* . Further, $\max_{x_{ij} > 0} p_{ij} \leq T^*$. This results in an integral solution I whose P_{max} value is at most μT^* , and whose Π_{max} value is at most $(\gamma + 1)T^*$. Observe that, setting $\mu = \frac{3+\sqrt{5}}{2}$ results in $\mu = \gamma + 1$. Finally, we note that the optimal value of T can be arrived at by a bisection search in the range $[0, np_{max}]$, where $n = |J|$ and $p_{max} = \max_{i,j} p_{ij}$. Since T^* is a lower bound on LB , we have the following result.

Theorem 1 *The above algorithm computes a processor assignment for each job such that the value of $\max\{P_{max}, \Pi_{max}\}$ for the resulting assignment is within a $(\frac{3+\sqrt{5}}{2})$ -factor of the optimal.*

2.2 Step 2: Solving the GDSS problem under treelike precedences

We first consider the case when the precedences are a collection of directed in-trees or out-trees in section 2.2.1. We then extend this to the case where the precedences form an arbitrary forest (i.e., the underlying undirected graph is a forest) in Section 2.2.2. Since the processor assignment is already specified in the GDSS problem, we will use the notation $m(v)$ to denote the machine to which node v is assigned. Also, since the machine is already fixed, the processing time for node v is also fixed, and is denoted by p_v .

2.2.1 GDSS on Out-/In-Arborescences

An out-tree is a tree rooted at some node, say r , with all edges directed away from r ; an in-tree is a tree obtained by reversing all the directions in an out-tree. In the discussion below, we only focus on out-trees; the same results can be obtained for in-trees. The algorithm for out-trees requires a careful partitioning the tree into blocks of chains, and giving random delays at the start of each chain in each of the blocks - thus the delays are spread all over the tree. The head of the chain waits for all its ancestors to finish running, after which it waits for an amount of time equal to its random delay. After this, the entire chain is allowed to run without interruption. Of course, this may result in an infeasible schedule where multiple jobs simultaneously contend for the same machine (at the same time). We show that this contention is low and can be resolved by expanding the infeasible schedule produced above.

Chain Decomposition We define the notions of *chain decomposition* of a graph and its *chain width*. Given a DAG $G(V, E)$, let $d_{in}(u)$ and $d_{out}(u)$ denote the in-degree and out-degree, respectively, of u in G . A *chain decomposition* of $G(V, E)$ is a partition of its vertex set into subsets B_1, \dots, B_λ (called blocks) such that the following properties hold: (i) The subgraph induced by each block B_i is a collection of vertex-disjoint directed chains, (ii) For any $u, v \in V$, let $u \in B_i$ be an ancestor of $v \in B_j$. Then, either $i < j$, or $i = j$ and u and v belong to the same directed chain of B_i , (iii) If $d_{out}(u) > 1$, then none of u 's out-neighbors are in the same block as u . The *chain-width* of a DAG is the minimum value λ such that there is a chain decomposition of the DAG into λ blocks.

Well structured schedules. We now state some definitions motivated by those in [6]. Given a GDSS instance with a DAG $G(V, E)$ and given a chain decomposition of G into λ blocks, we construct a *B-delayed schedule* for it as follows; B is an integer that will be chosen later. Each job v which is the head of a chain in a block is assigned a delay $d(v)$ in $\{0, 1, \dots, B - 1\}$. Let v belong to the chain C_i . Job v waits for $d(v)$ amount of time after all its predecessors have finished running, after which the jobs of C_i are scheduled consecutively (of course, the resulting schedule might be infeasible). A *random B-delayed schedule* is a B -delayed schedule in which all the delays have been chosen independently and uniformly at random from $\{0, 1, \dots, B - 1\}$. For a B -delayed schedule S , the *contention* $C(M_i, t)$ is the number of jobs scheduled on machine M_i in the time interval $[t, t + 1)$. As in [6, 21], we assume w.l.o.g. that all job lengths are powers of two. This can be achieved by multiplying each job length by at most a factor of two (which affects our approximation ratios only by a constant factor). A delayed scheduled S is *well-structured* if for each k , all jobs with length 2^k begin in S at a time instant that is an integral multiple of 2^k . Such schedules can be constructed from randomly delayed schedules as follows. First create a new GDSS instance by replacing each job $v = (m(v), p_v)$ by the job $v = (m(v), 2p_v)$. Let S be a random B -delayed schedule for this modified instance, for some B ; we call S a *padded random B-delayed schedule*. From S , we can construct a well-structured delayed schedule, S' , for the original GDSS instance as follows: insert v with the correct boundary in the slot assigned to \hat{v} by S . S' will be called a *well-structured random B-delayed schedule* for the original GDSS instance.

Our algorithm. We now describe our algorithm; for the sake of clarity, we occasionally omit floor and ceiling symbols (e.g., “ $B = \lceil 2\Pi_{\max} / \log(np_{\max}) \rceil$ ” is written as “ $B = 2\Pi_{\max} / \log(np_{\max})$ ”). As before let $p_{\max} = \max_v p_v$.

1. Construct a chain decomposition of the DAG $G(V, E)$ and let λ be its chain width.
2. Let $B = 2\Pi_{\max}/\log(np_{\max})$. Construct a padded random B -delayed schedule S by first increasing the processing time of each job v by a factor of 2 (as described above), and then choosing a delay $d(v) \in \{0, \dots, B - 1\}$ independently and uniformly at random for each v .
3. Construct a well-structured random B -delayed schedule S' as described above.
4. Construct a valid schedule S'' using the technique from [6] as follows:
 - (a) Let the makespan of S' be L .
 - (b) Partition the schedule S' into *frames* of length p_{\max} ; i.e., into the set of time-intervals $\{[ip_{\max}, (i+1)p_{\max}), i = 0, 1, \dots, \lceil L/p_{\max} \rceil - 1\}$.
 - (c) For each frame, use the frame-scheduling technique from [6] to produce a feasible schedule for that frame. Concatenate the schedules of all frames to obtain the final schedule.

The following theorem shows the performance guarantee of the above algorithm, when given a chain decomposition. The proof appears in the Appendix.

Theorem 2 *Let $p_{\max} = \max_{i,j} p_{ij}$. Given an instance of treelike GDSS and a chain decomposition of its DAG $G(V, E)$ into λ blocks, the schedule S'' produced by the above algorithm has makespan $O(\rho \cdot (P_{\max} + \Pi_{\max}))$ with high probability, where $\rho = \max\{\lambda, \log n\} \cdot \lceil \log(\min\{p_{\max}, n\}) / \log \log n \rceil$. Furthermore, the algorithm can be derandomized.*

The proof of Theorem 3 demonstrates a chain decomposition of width $O(\log n)$ for any out-tree: this completes the algorithm for an out-tree. An identical argument would work for the case of a directed in-tree. We note that the notions of chain decomposition and chain-width for the out-directed arborescences are similar to those of caterpillar decomposition and caterpillar dimension for trees [17]. However, in general, a caterpillar decomposition for an arborescence need not be a chain-decomposition and vice-versa. The proof of Theorem 3 appears in the Appendix.

Theorem 3 *There is a deterministic polynomial-time approximation algorithm for solving the GDSS problem when the underlying DAG is restricted to be an in/out tree. The algorithm computes a schedule with makespan $O((P_{\max} + \Pi_{\max}) \cdot \rho)$, where $\rho = \log n \cdot \lceil \log(\min\{p_{\max}, n\}) / \log \log n \rceil$. In particular, we get an $O(\log n)$ -approximation in the case of unit-length jobs.*

2.2.2 GDSS on arbitrary forest-shaped DAGs

We now consider the case where the undirected graph underlying the DAG is a forest. The chain decomposition algorithm described in Theorem 3 does not work for arbitrary forests, and it is not clear how to make the Lemma 9 work with chain decompositions of arbitrary forests. Instead of following the approach of Section 2.2.1, we observe that once we have a chain decomposition, the problem restricted to a block of chains is precisely the job shop scheduling problem. This allows us to reduce the $R|forest|C_{\max}$ problem to a set of job shop problems, for which we use the algorithm of [6]. While this is simpler than the algorithm in Section 2.2.1 for in-/out-trees, we incur another logarithmic factor in the approximation guarantee.

The following lemmas show that a good decomposition can be computed for forests which can be exploited to yield a good approximation ratio.

Lemma 4 *Every DAG T whose underlying undirected graph is a forest, has a chain decomposition into γ blocks, where $\gamma \leq 2(\lceil \lg n \rceil + 1)$.*

Theorem 5 *Given a GDSS instance and a chain decomposition of its DAG $G(V, E)$ into γ blocks, there is a deterministic polynomial-time algorithm which delivers a schedule of makespan $O((P_{\max} + \Pi_{\max}) \cdot \rho)$, where $\rho = \frac{\gamma \log n}{\log \log n} \lceil \frac{\log \min(p_{\max}, n)}{\log \log n} \rceil$. Thus, Lemma 4 implies that $\rho = O(\frac{\log^2 n}{\log \log n} \lceil \frac{\log \min(p_{\max}, n)}{\log \log n} \rceil)$ is achievable.*

3 The $R|forest|\sum_j w_j C_j$ problem.

We now consider the objective of minimizing weighted completion time, where the given weight for each job j is $w_j \geq 0$. Given an instance of $R|prec|\sum_j w_j C_j$ where the jobs have not been assigned their processors, we now reduce it to instances of $R|prec|C_{\max}$ with processor assignment. More precisely, we show the following: let P_{\max} and Π_{\max} denote the “dilation” and “congestion” as usual; if there exists a schedule of makespan $\rho \cdot (P_{\max} + \Pi_{\max})$ for the latter, then there is a $O(\rho)$ -approximation algorithm for the former. Let the machines and jobs be indexed by i and j ; $p_{i,j}$ is the (integral) time for processing job j on machine i , if we choose to process j on i . We now present an LP-formulation for $R|prec|\sum_j w_j C_j$ which has the following variables: for $\ell = 0, 1, \dots$, variable $x_{i,j,\ell}$ is the indicator variable which denotes if “job j is processed on machine i , and completes in the time interval $(2^{\ell-1}, 2^\ell]$ ”; for job j , C_j is its completion time, and z_j is the time spent on processing it. The objective is to minimize $\sum_j w_j C_j$ subject to: **(1)** $\forall j, \sum_{i,\ell} x_{i,j,\ell} = 1$, **(2)** $\forall j, z_j = \sum_i p_{i,j} \sum_{\ell} x_{i,j,\ell}$, **(3)** $\forall (j \prec k), C_k \geq C_j + z_j$, **(4)** $\forall j, \sum_{i,\ell} 2^{\ell-1} x_{i,j,\ell} < C_j \leq \sum_{i,\ell} 2^\ell x_{i,j,\ell}$, **(5)** $\forall (i, \ell), \sum_j p_{i,j} \sum_{t \leq \ell} x_{i,j,t} \leq 2^\ell$, **(6)** $\forall \ell \forall \text{maximal chains } \mathcal{P}, \sum_{j \in \mathcal{P}} \sum_i p_{i,j} \sum_{t \leq \ell} x_{i,j,t} \leq 2^\ell$, **(7)** $\forall (i, j, \ell), (p_{i,j} > 2^\ell) \Rightarrow x_{i,j,\ell} = 0$, **(8)** $\forall (i, j, \ell), x_{i,j,\ell} \geq 0$

Note that (5) and (6) are “congestion” and “dilation” constraints respectively. Our reduction proceeds as follows. Solve the LP, and let the optimal fractional solution be denoted by variables $x_{i,j,\ell}^*$, C_j^* , and z_j^* . We do the following filtering, followed by an assignment of jobs to (machine, time-frame) pairs.

Filtering: For each job j , note from the first inequality in (4) that the total “mass” (sum of $x_{i,j,\ell}$ values) for the values $2^\ell \geq 4C_j^*$, is at most $1/2$. We first set $x_{i,j,\ell} = 0$ if $2^\ell \geq 4C_j^*$, and scale each $x_{i,j,\ell}$ to $x_{i,j,\ell} / (1 - \sum_{\ell' \geq 4C_j^*} \sum_i x_{i,j,\ell'})$, if ℓ is such that $2^\ell < 4C_j^*$ - this ensures that equation (1) still holds. After the filtering, each non-zero variable increases by at most a factor of 2. Additionally, for any fixed j , the following property is satisfied: consider the largest value of ℓ such that $x_{i,j,\ell}$ is non-zero; let this value be ℓ' ; then, $2^{\ell'} = O(C_j^*)$. The right-hand-sides of (5) and (6) become at most $2^{\ell'+1}$ in the process and the C_j values increase by at most a factor of two.

Assigning jobs to machines and frames For each j , set $F(j)$ to be the frame $(2^{\ell'-1}, 2^{\ell'}]$, where ℓ' is the index such that $4C_j^* \in F(j)$. Let $G[\ell]$ denote the sub-problem which is restricted to the jobs in this frame. Let $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ be the fractional congestion and dilation, respectively, for the sub-problem restricted to $G[\ell]$. From constraints (5) and (6), and due to our filtering step, which at most doubles any non-zero variable, it follows that both $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ are $O(2^\ell)$. We now perform a processor assignment as follows: for each $G[\ell]$, we use the processor assignment scheme in Section 2.1 to assign processors to jobs. This ensures that the integral $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ values are at most a constant times their fractional values.

Scheduling: First schedule all jobs in $G[1]$; then schedule all jobs in $G[2]$, and so on. We can use any approximation algorithm for makespan-minimization, for each of these scheduling steps. It is easy to see that we get a feasible solution: for any two jobs j_1, j_2 , if $j_1 \prec j_2$, then $C_{j_1}^* \leq C_{j_2}^*$ and frame $F(j_1)$ occurs before $F(j_2)$ and hence gets scheduled first.

Theorem 6 *If there exists an approximation algorithm which yields a schedule whose makespan is $O((P_{\max} + \Pi_{\max}) \cdot \rho)$, then there is also an $O(\rho)$ -approximation algorithm for minimizing weighted completion time. Thus, Theorem 5 implies that $\rho = O\left(\frac{\log^2 n}{\log \log n} \lceil \frac{\log \min(p_{\max}, n)}{\log \log n} \rceil\right)$ is achievable.*

4 Weighted completion time and flow time on Chains

We now consider the case of $R|prec|\sum_j w_j C_j$, where the processor-assignment is *not* prespecified. We consider the *restricted-assignment variant* of this problem [13, 20], where for every job v , there is a value p_v such that for all machines i , $p_{i,v} \in \{p_v, \infty\}$. Let $S(v)$ denote the set of machines such that $p_{i,v} = p_v$. We focus on the case where the precedence DAG is a disjoint union of chains with all p_u being polynomially-bounded positive integers in the input-size N . We now present our approximation algorithms for weighted completion time and flow time.

Weighted Completion Time Recall that $N = \max_v \{n, m, p_v\}$ denote the “input size”. In this section, we obtain an $O(\log N / \log \log N)$ -approximation algorithm for the minimum weighted completion time problem. We first describe an LP relaxation. Let $T = \sum_v p_v$. In the following LP, for ease of exposition, we assume that all the p_v values are equal to one. Our algorithm easily generalizes to the case where the p_v values are arbitrary positive integers, and the LP is polynomial-sized if all p_v values are polynomial in N . Let \prec denote the immediate predecessor relation, i.e., if $u \prec v$, then they both belong to the same chain and u is an immediate predecessor of v in this chain. Note that if v is the first job in its chain, then it has no predecessor. In the time-indexed LP formulation below, the variable $x_{v,i,t}$ denotes the fractional amount of job v that is processed on machine i at time t . The objective is $\min \sum_v w(v)C(v)$, subject to: **(B1)** $\forall v, \sum_{i \in S(v)} \sum_{t \in [1, \dots, T]} x_{v,i,t} = 1$, **(B2)** $\forall i \in [1, \dots, m], \forall t \in [1, \dots, T], \sum_v x_{v,i,t} \leq 1$, **(B3)** $\forall v, \forall t \in [1, \dots, T], z_{v,t} = \sum_{i \in [1, \dots, m]} x_{v,i,t}$, **(B4)** $\forall u \prec v, \forall t \in [1, \dots, T], \sum_{t' \in [1, \dots, t]} z_{v,t'} \leq \sum_{t' \in [1, \dots, t-1]} z_{u,t'}$, **(B5)** $\forall v, C(v) = \sum_{t \in [1, \dots, T]} t \cdot z_{v,t}$, **(B6)** $\forall v, \forall i \in S(v), \forall t \in [1, \dots, T], x_{v,i,t} \geq 0$.

The constraints (B1) ensure that all jobs are processed completely, and (B2) ensure that at most one job is (fractionally) assigned to any machine at any time. The variable $z_{v,t}$ denotes the fractional amount of job v that has been processed on all machines at time t . Constraints (B3) and (B4) are the precedence constraints and (B6) define the completion time $C(v)$ for job v .

Our algorithm proceeds as follows. We first solve the above LP optimally. Let OPT be the optimal value of the LP and let x and z denote the optimal solution values in the rest of the discussion. We define a rounding procedure for each chain such that the following hold:

1. Let $Z_{v,t}$ be the indicator random variable which denotes if v is executed at time t in the rounded solution. Let $X_{v,i,t}$ be the indicator random variable which denotes if v is executed at time t on machine i in the rounded solution. Then $\mathbf{E}[Z_{v,t}] = z_{v,t}$ and $\mathbf{E}[X_{v,i,t}] = x_{v,i,t}$.
2. All precedence constraints are satisfied in the rounded solution.
3. Jobs in different chains are rounded independently.

After the $Z_{v,t}$ values have been determined, we do the machine assignment as follows: if $Z_{v,t} = 1$, then job v is assigned to machine i with probability $(x_{v,i,t}/z_{v,t})$. In general, this assignment strategy might result in jobs from *different* chains executing on the same machine at the same time, and hence an *infeasible* schedule. Let C_1 denote the cost of this infeasible solution. Property 1 above ensures that $\mathbf{E}[C_1] = OPT$. Let Y be the random variable which denotes the maximum contention of any machine at any time. We obtain a feasible solution by “expanding” each time slot by a factor of Y .

We now show our rounding procedure for the jobs of a specific chain such that properties 1 and 2 hold; different chains are handled independently as follows: for each chain Γ , we choose a value $r(\Gamma) \in [0, 1]$ uniformly and independently at random. For each job v belonging to chain Γ , $Z_{v,t'} = 1$ iff $\sum_{t=1}^{t'-1} z_{v,t} <$

$r(\Gamma) \leq \sum_{t=1}^{t'} z_{v,t}$. Bertsimas *et al.* [1] show other applications for such rounding techniques. A moment's reflection shows that property **1** holds due to the randomized rounding and property **2** holds due to Equation (B4). A straight forward application of the Chernoff-type bound from Fact 8 yields the following lemma:

Lemma 7 *Let \mathcal{E} denote the event that $Y \leq (\alpha \log N / \log \log N)$, where $\alpha > 0$ is a suitably large constant. Event \mathcal{E} occurs after the randomized machine assignment with high probability: this probability can be made at least $1 - 1/N^\beta$ for any desired constant $\beta > 0$, by letting the constant α be suitably large.*

Finally, we note that we expand an infeasible schedule only if the event \mathcal{E} occurs. Otherwise, we can repeat the randomized machine assignment until event \mathcal{E} occurs and expand the resultant infeasible schedule. Let the final cost of our solution be C . We now have an $O(\log N / \log \log N)$ -approximation from:

$$\mathbf{E}[C \mid \mathcal{E}] \leq \mathbf{E}[Y \cdot C_1 \mid \mathcal{E}] \leq \mathbf{E}\left[O\left(\frac{\log N}{\log \log N}\right) \cdot C_1 \mid \mathcal{E}\right] \leq O\left(\frac{\log N}{\log \log N}\right) \cdot \frac{\mathbf{E}[C_1]}{\Pr[\mathcal{E}]} \leq O\left(\frac{\log N}{\log \log N}\right) \cdot OPT.$$

Minimizing Weighted Flow Time with Resource Augmentation Consider the following extension to the problem of weighted completion time discussed above: in addition to the chain-shaped precedence constraints and the machine assignment constraints, each job v also has a “release-time” r_v and a deadline l_v . The release time specifies the time at which the job v was created and hence it can be scheduled only at times which are $\geq r_v$. The deadline l_v specifies the last time slot by which this job can be scheduled. Our goal is to find a schedule which minimizes the weighted flow time $\sum_v w(v)(C(v) - r_v)$ subject to the above constraints; again, we focus on the case “ $p_v \equiv 1$ ”, but the results here hold for the case where all p_v are polynomially bounded positive integers. In general, minimizing weighted flow time is hard to approximate. Leonardi & Raz [12] provide a $O(\sqrt{n} \log n)$ -approximation algorithm for this problem and show that it cannot be approximated within a factor of $O(n^{\frac{1}{3}-\delta})$ for any constant $\delta > 0$, unless $P = NP$. One way of dealing with this is through resource augmentation, where we allow our solution to use ψ copies of a machine. In this case, we say that the solution is an ψ -speed solution. Let OPT be the cost of the optimal schedule. We say that a solution is (ψ, γ) -approximate, if its speed is ψ and the cost of the solution is at most $\gamma \cdot OPT$. Note that we compare the cost of our ψ -speed solution with that of a 1-speed optimal solution. We now present an algorithm, which either proves that input instance is has no feasible solution, or outputs a (ψ, γ) -approximate solution where $\psi = O\left(\frac{\log N}{\log \log N}\right)$ and $\gamma = 1 + o(1)$ with high probability.

Our algorithm proceeds as follows. We first formulate this problem as an LP . This LP is similar to that considered for the weighted completion time earlier, except that the LP considers only those $x_{v,i,t}$ variables such that $r_v \leq t \leq l_v$. We solve this LP optimally. Note that any integer solution is also a feasible solution for this LP formulation. Hence, if the LP is infeasible, then this instance is not feasible either. Assume that the LP has an optimal fractional solution whose value is OPT . We now round this fractional schedule to obtain an integral schedule and a machine assignment for the jobs as in the case of the weighted completion time problem earlier. As before, this will result in an infeasible schedule of cost C_1 , where the maximum contention for any machine at any time slot is denote by the random variable Y . We note that by replacing each machine with a set of Y machines, we obtain a Y -speed solution whose cost remains the same as C_1 . Recall Lemma 7 and its notation, and also note that for any job v , its expected completion time equals its fractional completion time $C^*(v)$. Now, by Lemma 7, we can take β large enough so that even conditional on the event \mathcal{E} , the expected completion time of v is at most $C^*(v) + o(1)$ (since $C^*(v)$ is bounded by a fixed polynomial of N , and since β is sufficiently large). Since the fractional flow time $C^*(v) - r_v$ of v is at least 1, this addition of $o(1)$ is negligible. Thus, even conditional on \mathcal{E} (which happens with high probability), the expected cost of our solution is at most $(1 + o(1))OPT$.

Acknowledgments. We are thankful to David Shmoys for valuable comments.

References

- [1] D. Bertsimas, C.-P. Teo and R. Vohra. *On Dependent Randomized Rounding Algorithms*. Operations Research Letters, 24(3):105–114, 1999.
- [2] C. Chekuri and M. Bender. *An Efficient Approximation Algorithm for Minimizing Makespan on Uniformly Related Machines*. Journal of Algorithms, 41:212–224, 2001.
- [3] C. Chekuri and S. Khanna. *Approximation algorithms for minimizing weighted completion time*. Handbook of Scheduling, 2004.
- [4] F. A. Chudak and D. 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.
- [5] U. Feige and C. Scheideler. *Improved bounds for acyclic job shop scheduling*. Combinatorica, 22:361–399, 2002.
- [6] L.A. Goldberg, M. Paterson, A. Srinivasan and E. Sweedyk. *Better approximation guarantees for job-shop scheduling*. SIAM Journal on Discrete Mathematics, Vol. 14, 67-92, 2001.
- [7] L. Hall. *Approximation Algorithms for Scheduling*. in *Approximation Algorithms for NP-Hard Problems*, Edited by D. S. Hochbaum. PWS Press, 1997.
- [8] L. Hall, A. Schulz, D.B. Shmoys, and J. Wein. *Scheduling to minimize average completion time: Offline and online algorithms*. Mathematics of Operations Research, 22:513–544, 1997.
- [9] K. Jansen and L. Porkolab, *Improved Approximation Schemes for Scheduling Unrelated Parallel Machines*. Proc. ACM Symposium on Theory of Computing (STOC), pp. 408-417, 1999.
- [10] K. Jansen and R. Solis-oba. *Scheduling jobs with chain precedence constraints*.
- [11] K. Jansen, R. Solis-Oba and M. Sviridenko. *Makespan Minimization in Job Shops: A Polynomial Time Approximation Scheme*. Proc. ACM Symposium on Theory of Computing (STOC), pp. 394-399, 1999.
- [12] S. Leonardi and D. Raz. In *Approximating total flow time on parallel machines*. In Proc. ACM Symposium on Theory of Computing, 110-119, 1997.
- [13] J. K. Lenstra, D. B. Shmoys and É. Tardos. *Approximation algorithms for scheduling unrelated parallel machines*. Mathematical Programming, Vol. 46, 259-271, 1990.
- [14] F.T. Leighton, B. Maggs and S. Rao. *Packet routing and jobshop scheduling in $O(\text{congestion} + \text{dilation})$ Steps*, Combinatorica, Vol. 14, 167-186, 1994.
- [15] F. T. Leighton, B. Maggs, and A. Richa, *Fast algorithms for finding $O(\text{congestion} + \text{dilation})$ packet routing schedules*. Combinatorica, Vol. 19, 375-401, 1999.
- [16] J. H. Lin and J. S. Vitter. *ϵ -approximations with minimum packing constraint violation*. In Proceedings of the ACM Symposium on Theory of Computing, 1992, pp. 771–782.
- [17] N. Linial, A. Magen, and M.E. Saks. *Trees and Euclidean Metrics*. In Proceedings of the ACM Symposium on Theory of Computing, 169-175, 1998.
- [18] A. Panconesi and A. Srinivasan. *Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds*, SIAM Journal on Computing, Vol. 26, 350-368, 1997.
- [19] A. Schulz and M. Skutella. *The power of α -points in preemptive single machine scheduling*. Journal of Scheduling 5(2): 121 - 133, 2002.
- [20] P. Schuurman and G. J. Woeginger. *Polynomial time approximation algorithms for machine scheduling: Ten open problems*. Journal of Scheduling 2:203-213, 1999.
- [21] D.B. Shmoys, C. Stein and J. Wein. *Improved approximation algorithms for shop scheduling problems*, SIAM Journal on Computing, Vol. 23, 617-632, 1994.
- [22] M. Skutella. *Convex quadratic and semidefinite relaxations in scheduling*. Journal of the ACM, 46(2):206–242, 2001.

Appendix

A Proofs

A.1 Proofs from Section 2.2

Proof (of Theorem 2:) We only analyze the randomized algorithm. The delays can then be easily seen to be computable deterministically by the method of conditional probabilities.

First, observe that S has a makespan of at most $L \doteq 2(P_{\max} + \lambda \Pi_{\max} / \log(np_{\max}))$: this is because the maximum processing time along any directed path is at most $2P_{\max}$, and since there are λ points along any path which have been delayed, the additional delay is at most $2\lambda \Pi_{\max} / \log(np_{\max})$. Clearly, S' has no larger makespan. Let $C(M_i, t)$ be the contention of machine M_i at time t under S . The contention on any machine at any time under S' is no more than under S . We will need the following variation on the Chernoff bound.

Fact 8 ([18]) *Let $X_1, X_2, \dots, X_l \in \{0, 1\}$ be random variables such that for all i , and for any $S \subseteq \{X_1, \dots, X_{i-1}\}$, $\Pr[X_i = 1 | \bigwedge_{j \in S} X_j = 1] \leq p_i$. Let $X \doteq \sum_i X_i$ and $\mu \doteq \sum_i p_i$. Then for any $\delta > 0$, $\Pr[X \geq \mu(1 + \delta)] \leq G(\mu, \delta)$.*

The following key lemma bounds the contentions:

Lemma 9 *There exists a constant $c_1 > 0$ such that $\forall i \in \{1, \dots, m\}, \forall t \in \{1, \dots, L\}, C(M_i, t) \leq \alpha$ with high probability, where $\alpha = c_1 \log(np_{\max})$.*

Proof For any job v , define the random variable $X(v, i, t)$ to be 1 if v is scheduled on M_i during the time interval $[t, t + 1)$ by S , and 0 otherwise. Note that $C(M_i, t) = \sum_{v: m(v)=M_i} X(v, i, t)$. Let $d(v)$ be the random delay given to the chain to which v belongs. Conditioning on all other delays, $d(v)$ can take at most p_v values in the range $\{0, 1, \dots, B - 1\}$ that will lead to v being scheduled on M_i during $[t, t + 1)$. Hence, $\mathbf{E}[X(v, i, t)] = \Pr[X(v, i, t) = 1] \leq \frac{p_v}{B}$. Hence $\mathbf{E}[C(M_i, t)] \leq \frac{\Pi_{\max}}{B} \leq \log(np_{\max})$. Although the random variables $X(v, i, t)$ are not independent, we will now present an upper-tail bound for $C(M_i, t)$.

Let B_1, \dots, B_λ be the blocks in the chain decomposition. Consider the following ordering of nodes in V : nodes within each B_i are ordered so that if v is an ancestor of w , then v precedes w , and nodes in B_i are ordered before nodes in B_{i+1} , for each i . Let $\pi(1), \dots, \pi(n)$ be the resulting ordering of nodes. For any node v , and for any subset $W \subset V$ such that $\forall v' \in W, \pi(v') < \pi(v)$, we will argue that $\Pr[X(v, i, t) = 1 | \bigwedge_{v' \in W} X(v', i, t) = 1] \leq p_v/B$ in such a case. First, observe that if there is a node $v' \in W$ such that v' is an ancestor or descendant of v , then $X(v, i, t) = 0$, since the schedule S' preserves precedences. Therefore, assume that for each $v' \in W$, it is neither an ancestor nor a descendant of v . Let A be the chain containing v in the chain decomposition. Then, the random delay given at the start node of A does not affect any of the nodes in W , and conditioned on all other delays, $\Pr[X(v, i, t) = 1 | \bigwedge_{v' \in W} X(v', i, t) = 1] \leq p_v/B$ continues to hold. Thus, Fact 8 can now be applied, to get $\Pr[C(M_i, t) \geq \alpha \log(np_{\max})] \leq 1/(np_{\max})^c$, for a suitable constant c . Since the number of events “ $C(M_i, t) \geq \alpha \log(np_{\max})$ ” is $O((np_{\max})^{c'})$, for a constant c' , the lemma now follows via a union bound. ■

The above lemma implies that schedule S' has a low contention for each machine at each time instant, with high probability. Our final task is to verify that the last step, using the technique from [6] gives the desired bounds. From the observation earlier, S' has a makespan at most L . By the definition of p_{\max}

and the fact that S' is well-structured, no job crosses over a frame. Given such a well-structured frame of length p_{\max} where the maximum contention on any machine is at most α , the frame scheduling algorithm of [6] gives a feasible schedule with the following bounds.

Fact 10 *Let $p_{\max} = \max_{i,j} p_{ij}$. Given a well-structured frame of length p_{\max} where the maximum contention on any machine is at most α , there exists a deterministic algorithm which delivers a schedule for this frame with makespan $O(p_{\max}\alpha \lceil \log p_{\max} / \log \log \alpha \rceil)$. Hence, concatenating the frames yields a schedule of length $O(\rho'(P_{\max} + \Pi_{\max}))$, where $\rho' = \max\{\lambda, \log(np_{\max})\} \lceil \frac{\log p_{\max}}{\log \log(np_{\max})} \rceil$.*

Note that if p_{\max} is polynomially bounded in n , then Theorem 2 holds immediately. We now propose a simple reduction for the case where $p_{\max} \gg n$ to the case where p_{\max} is polynomial in n . Assume that in the given instance I , $p_{\max} \geq n^{10}$. Create a new instance I' which retains only those vertices in I whose processing times are greater than p_{\max}/n^2 . Vertices in the new instance I' inherit the same precedence constraints amongst themselves which they were subject to in I . However, all these vertices have processing times in the range $[p_{\max}/n^2, p_{\max}]$. Equivalently, all processing times can be scaled down such that they are in the range $[1, n^2]$. Hence, Fact 10 implies that we can obtain a schedule S' for instance I' whose length is $\rho(P_{\max} + \Pi_{\max})$, where $\rho = \max\{\lambda, \log n\} \cdot (\log n / \log \log n)$. We note that the total processing time of all the vertices in $I \setminus I'$ is at most $n \frac{P_{\max}}{n^2} = P_{\max}/n$. Hence, these vertices can be inserted into S' valid schedule S for I such the makespan increases by at most P_{\max}/n , and hence schedule S is also of length $\rho(P_{\max} + \Pi_{\max})$. ■

Proof (of Theorem 3) We show that any out-directed arborescence $T(V, E)$ has chain-width at most $\lambda = \lceil \lg n \rceil + 1$ by constructing such a chain decomposition. The construction proceeds in iterations, each of which creates a block of the decomposition. Define $T_1(V_1, E_1) = T(V, E)$. Let $T_i(V_i, E_i)$ be the subtree at the beginning the i^{th} iteration. Let $S_i \subseteq V_i$ be the set of vertices u such that: (i) the subtree rooted at u in T_i is a directed chain, and (ii) the parent (if any) of u in T_i has out-degree at least two. During the i^{th} iteration, we create a block $B_{\lambda+1-i}$ which contains each $u \in S_i$ along with its subtree. It is easy to see that the graph induced by V_{i+1} is an out-tree T_{i+1} , and this procedure can be run recursively; therefore, we do obtain a valid chain decomposition.

Claim 11 *Let $\beta_{\lambda+1-i}$ denote the number of chains induced by $B_{\lambda+1-i}$, in the i^{th} iteration. Then for all i , $\beta_{\lambda+1-i} \geq 2\beta_{\lambda-i}$.*

Proof Consider a leaf vertex u in T_{i+1} (and hence belonging to $B_{\lambda-i}$). Vertex u has out-degree zero in T_{i+1} and out-degree of at least two in T_i (otherwise, u would have belonged to $B_{\lambda+1-i}$ leading to a contradiction). Hence, there are at least two chains induced by $B_{\lambda+1-i}$ for which u is an ancestor. Further, each chain in $B_{\lambda+1-i}$ has at most one ancestor in $B_{\lambda-i}$ which is a leaf. Since any directed chain has a unique leaf vertex, the claim follows. ■

The above claim implies that $\beta_{\lambda} \geq 2^{\lambda-1}$. Since $\beta_{\lambda} \leq n$, the theorem follows. ■

Proof (of Theorem 5): Consider the chain decomposition of the DAG with γ blocks P_1, \dots, P_{γ} . Each of these blocks P_i is an instance of job-shop scheduling, since it only consists of chains. These can be solved using the algorithm of [6] which, given a job-shop instance, produces a schedule with makespan at most $O(\frac{(P_{\max} + \Pi_{\max}) \log n}{\log \log n} \lceil \log p_{\max} / \log \log n \rceil)$. Also, by the properties of the chain decomposition, there are no precedence constraints from P_j to P_i , for $j > i$. Therefore, we can concatenate the schedules for each block, and this yields a schedule for the GDSS instance with the desired makespan (since, as argued in Section 2.2.1, we may assume without loss of generality that p_{\max} is polynomially bounded in n). ■

Proof (of Lemma 4:) We show how to construct such a decomposition. Fix a root for T . Some of the edges in T will be pointing away from the root (down) and others will be pointing towards the root (up). Imagine that T is an outward-arborescence and perform a chain decomposition as in the proof of Theorem 3 which will result in a decomposition \mathcal{B} with blocks B_1, \dots, B_λ and intermediate trees T_1, \dots, T_λ . We now re-partition these blocks into blocks $P_1, \dots, P_{2\lambda}$ of the chain decomposition \mathcal{P} as follows. Consider a chain in B_i . In general, some edges of this chain will point down and others will point up. Each vertex u in this chain will belong to one of the following types:

Type 1: u is the head of the chain and connected to T_i through an edge pointing up. In this case, $u \in P_i$.

Type 2: u is the head of the chain and connected to T_i through an edge pointing down. Then, $u \in P_{2\lambda+1-i}$.

Type 3: u is the tail of the chain with in-degree 1. Then, $u \in P_{2\lambda+1-i}$.

Type 4: u is the tail of the chain with out-degree 1. Then, $u \in P_i$.

Type 5: u has out-degree 2. Then, $u \in P_i$.

Type 6: u has in-degree 2. Then, $u \in P_{2\lambda+1-i}$.

Type 7: u has in-degree 1 and out-degree 1 with both the edges pointing up. Then, $u \in P_i$.

Type 8: u has in-degree 1 and out-degree 1 with both the edges pointing down. Then, $u \in P_{2\lambda+1-i}$.

We assume w.l.o.g. that none of the blocks in \mathcal{P} are empty (otherwise, we can delete such blocks). Since \mathcal{B} is a chain decomposition, it ensures that vertices in different chains of the same block are not ancestors or descendants of each other. It is easy to see that after repartitioning B_i , none of the vertices in P_i are descendants of any vertex in $P_{2\lambda+1-i}$. We now prove that \mathcal{P} is a leveled chain decomposition by showing that if $j \neq k$, there is an edge from a vertex in P_j to a vertex in P_k only if $j < k$. Let $i \in 1, \dots, \lambda$. From the construction, note that nodes in T_{i+1} only end up in the blocks $P_{i+1}, \dots, P_\lambda, P_{\lambda+1}, \dots, P_{2\lambda-i}$. No edge exists from any vertex v in T_{i+1} to a vertex in u in P_i . Indeed, if such an edge exists, then u must be the head of its chain in B_i and must be connected to T_i through an edge pointing down. In this case, u would have been in $P_{2\lambda+1-i}$, leading to a contradiction. Similarly, no edge exists from any u vertex in $P_{2\lambda+1-i}$ to a vertex in T_{i+1} . Again, if such an edge exists, then u must be the head of its chain in B_i and must be connected to T_i through an edge pointing up. In this case, u would have been in P_i leading to a contradiction. These two together imply that all edges across the partitions of P are from P_j to P_k such that $j < k$. ■

A.2 The Limits of our Lower Bound

Any attempt to improve our approximation guarantees must address the issue of how good the $P_{\max} + \Pi_{\max}$ lower bounds are. We show that in general DAGs, the $LB = \max\{P_{\max}, \Pi_{\max}\}$ lower bound is very weak: there are instances where the optimal makespan is $\Omega(P_{\max}\Pi_{\max})$. This leaves the question for forests- we show that even in this case, there are instances where the optimal makespan is $\Omega(LB \cdot \log n / \log \Pi_{\max})$.

We construct a rooted in-tree T for which the optimal makespan is $\Omega(LB \cdot \log n / \log \Pi_{\max})$, for any value of Π_{\max} that is $\Omega(\log n / \log \log n)$. All nodes (jobs) are of unit length. At level 0, we have the root which is assigned to a processor that is not used for any other nodes. Once the level- i nodes are fixed, level- $(i+1)$ nodes are fixed in the following manner. For each node v at level i , there are $C = \Pi_{\max}$ nodes in level $i+1$ that are immediate predecessors of v . All these C nodes are assigned to the same machine that is never used again. Since there are n nodes in T , it is clear that there are $\log n / \log C$ levels, and about n/C machines are used.

Lemma 12 *The optimal makespan for the above instance is $\Omega((\Pi_{\max} + P_{\max}) \log n / \log \Pi_{\max})$.*

Proof We have $C = \Pi_{\max}$. Let V_i denote the set of nodes in level i . Note that $D \doteq P_{\max} = \log n / \log C + 1$ is the number of levels in T . We will show by backward induction on i that the earliest time that nodes in V_i can start is $(D - i)C$. From this the lemma follows, since $C \geq \Omega(\log n / \log \log n)$. The base case $i = D$ is obvious. Now assume this claim is true for levels $j \geq i$. Consider $v \in V_{i-1}$. Let $P(v)$ denote the immediate predecessors of v in level i . By construction, $|P(v)| = C$, and by the induction hypothesis, the earliest time any node in $P(v)$ can start is $(D - i)C$. All the nodes in $P(v)$ are assigned to the same processor. Therefore, the earliest time all nodes in $P(v)$ are done is $(D - i)C + C = (D - i + 1)C$. Note that v can start only after all of $P(v)$ is completed. This completes the proof for forest-shaped instances. ■

An $\Omega(\sqrt{n})$ gap in general: In the above instance, the optimal makespan is also $\Omega(P_{\max}\Pi_{\max})$, but the ratio of this to $P_{\max} + \Pi_{\max}$ is only $O(\log n / \log \Pi_{\max})$, because $P_{\max} = \log n / \log \Pi_{\max}$. We now show an instance of the general GDSS problem where the optimal makespan is $\Omega(P_{\max}\Pi_{\max})$ and this quantity is $\Omega(\sqrt{n})$ times larger than $\Pi_{\max} + P_{\max}$. This instance has $m = \sqrt{n}$ layers, each layer containing m nodes. Let these layers be denoted by V_1, \dots, V_m . For each $i = 1, \dots, m - 1$, all edges in $V_i \times V_{i+1}$ are present. It is easy to see that $P_{\max} = \Pi_{\max} = m$ in this instance, but the optimal makespan is $n = P_{\max}\Pi_{\max}$. It can also be shown that a natural integer program based on time-indexed variables has an $\Omega(m)$ gap between the integral and fractional optima for this instance.