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. [23]). The first step in our algorithm for the $R|prec|C_{max}$ problem with treelike precedences involves using the algorithm of Lenstra, Shmoys & Tardos 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 & Rao to the case of trees. For the special case of chains, we show a dependent rounding technique which leads to a bicriteria approximation algorithm for minimizing the flow time, a notoriously hard objective function.

1 Introduction

A very general type of 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. Thus, we need to decide which machine to schedule each job on, and then run the jobs in some order consistent with the precedence constraints. 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. $\max_j\{C_j\}$ -

*A preliminary version of this paper appeared in the *Proc. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems* (APPROX), pages 146–157, 2005.

[1] Virginia Bio-informatics Institute and Department of Computer Science Virginia Tech., Blacksburg, VT 24061. Part of this work was done while at the Los Alamos National Laboratory, and supported in part by the Department of Energy under Contract W-7405-ENG-36. Email: akumar@vbi.vt.edu.

[2] Virginia Bio-informatics Institute and Department of Computer Science Virginia Tech., Blacksburg, VT 24061. Part of this work was done while at the Los Alamos National Laboratory, and supported in part by the Department of Energy under Contract W-7405-ENG-36. Email: mmarathe@vbi.vt.edu.

[3] Department of Computer Science, University of Maryland, College Park, MD 20742. Part of this work was done while visiting Los Alamos National Laboratory; research supported in part by NSF Award CCR-0208005 and NSF ITR Award CNS-0426683. Email: sri@cs.umd.edu.

[4] Department 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 and NSF ITR Award CNS-0426683. Email: srin@cs.umd.edu.

this is denoted by $R|prec|C_{max}$, and (ii) minimize the weighted completion time - this is denoted by $R|prec|\sum_j w_jC_j$. Numerous other variants, involving release dates or other objectives have been studied (see e.g. [8]). Most such variants are NP-hard; thus, we will focus throughout on approximation algorithms with improved approximation guarantees. Recall that a ρ-approximation algorithm for a minimization problem is a polynomial-time algorithm which always produces a feasible solution with objective function value at most ρ times optimal; ρ is called the *approximation guarantee* or *approximation ratio* of such an algorithm.

Almost-optimal upper and lower bounds on the approximation ratio are known for the versions of the above problems without precedence constraints (i.e., the $R||C_{max}$ and $R||\sum_j w_jC_j$ problems) [4, 14, 24], 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 a collection of node-disjoint chains - this is the job shop scheduling problem [23], which itself has a long history. The first result for job shop scheduling was the breakthrough work of Leighton et al. [15, 16] for packet scheduling, which implied a logarithmic approximation for the case of unit processing costs. Leighton et al. [15, 16] 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 [15, 16] was generalized to nonuniform processing costs by Shmoys et al. [23], 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. [7]; see [6] for additional relevant work. Shmoys et al. [23] 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 (i.e., machines) with more general precedence constraints are for identical parallel machines (denoted by $P|prec|C_{max}$) [8], or for uniformly-related parallel machines (denoted by $Q|prec|C_{max}$) [5, 2]. The weighted completion time objective has also been studied for these variants [4, 5, 9]. When the number of machines is constant, polynomial-time approximation schemes are known [10, 12]. Note that all the discussion here 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. [21]). Less is known for the weighted completion time objective in the same setting, as compared to the makespan. The known approximations are either for the case of no precedence constraints [24], or for precedence constraints with parallel/related processors [5, 9, 20]. 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 $R|prec|\sum_j w_jC_j$ 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 chains.

Summary of results. We present the first polylogarithmic approximation algorithms for the $R|prec|C_{max}$ and $R|prec|\sum_j w_jC_j$ problems, under "treelike" precedences. 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 $R|forest|\sum_j w_jC_j$, respectively, to simplify the description - this generalizes the notation used by [11] for the case of chains.

(a). The $R|forest|C_{max}$ problem. We obtain a polylogarithmic approximation for this problem. We employ the same lower bound LB (described shortly) used in [15, 23, 7, 6], except that we are dealing with the more general situation where jobs have not yet been assigned to machines. Given an assignment of jobs to machines, let $P_{\max}$ denote the maximum total processing time along any directed path, and $\Pi_{\max}$ be the maximum total processing time needed on any machine. It is immediate that given such an assignment, $\max\{P_{\max}, \Pi_{\max}\}$ is a lower bound on the makespan of any schedule. Let LB denote the minimum possible value of $\max\{P_{\max}, \Pi_{\max}\}$, taken over all possible legal assignments of jobs to machines; LB is thus a lower bound on the makespan. 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 becomes $O(\log n)$. 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. [23]) and involves two steps: (i) we show how to compute a processor assignment whose LB value is within a $(\frac{3+\sqrt{5}}{2})$–factor of optimal, by extending the approach of [14], and (ii) 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 (ii) 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 [23].[5] Our algorithm for treelike instances of GDSS is similar to one used in [15, 23, 7], namely injecting random delays to the start times of the jobs; this allows for contention resolution. However, unlike [15, 23, 7], it is not adequate to insert random delays only at the head of the trees - we actually insert random delays throughout the schedule (here and in the rest of the paper, the head node or a start node amongst a set of nodes refers to the minimum node / element in that set, as determined by the partial order). 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 [15], 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 [19] that works in the presence of correlations.

(b). 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 similar in spirit to that of [5, 20]: using geometric time windows and appropriate linear constraints. We employ additional ideas here in order to handle our specific situation (e.g., the reduction in [20] is meant for *identical* parallel machines while ours is for *unrelated* machines).

(c). Minimizing weighted flow time on chains. Given a "release time" for each job (the time at which it enters the system) and a schedule, the *flow time* of a job is the time elapsed from its release time to its completion time. Minimizing the weighted flow time of the jobs is a notoriously hard problem, and no reasonable approximation algorithm is known even for the special case of job-shop scheduling. We consider the case of this problem where (i) the forest is a collection of node-disjoint chains, (ii) for each

[5] In the Dagshop problem [23], the input is a collection of DAGs, but in each DAG, at most one operation can be done at a time.

machine i and operation v, $p_{i,v} \in \{p_v, \infty\}$ (i.e., the *restricted-assignment variant*), and (iii) all processing times p_v are polynomially-bounded in the input length N. (Note that job shop scheduling, where we have a collection of node-disjoint chains and where the jobs are pre-assigned to machines, is a special case of what we consider; however, we also assume that the processing times are polynomially-bounded.) We describe a natural LP-relaxation and a dependent randomized rounding scheme for this problem. Our rounding ensures that (i) the precedence constraints are satisfied with *probability* 1, and (ii) for any (v,t), the probability of starting v at time t equals its fractional (LP) value $z_{v,t}$. This result also leads to a bicriteria $(1+o(1))$–approximation for the weighted flow time, using $O(\log N/\log\log N)$ copies of each machine.

Organization. We develop 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 then study the weighted flow time for the case of chains in Section 4. Section 5 concludes with some open problems.

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

We now present approximation algorithms for the $R|forest|C_{max}$ problem, and also study the limitations of our approach. 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.

Our algorithm for the $R|forest|C_{max}$ problem involves the following two steps:

Step 1: 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 value, LB. This is described in Section 2.1.

Step 2: Solve the GDSS problem we get from the previous step to get a schedule whose length is at most a polylogarithmic factor away from $\max\{P_{\max}, \Pi_{\max}\}$. This is described in Section 2.2.

2.1 Step 1: A processor assignment whose $\max\{P_{\max}, \Pi_{\max}\}$ value is $O(LB)$

We now describe the algorithm for processor assignment, using some of the ideas from [14]. Let T be our "guess" for the optimal value $LB = \min\{\max\{P_{\max}, \Pi_{\max}\}\}$. Let J and M denote the set of jobs and machines, respectively. Let x denote any fractional processor assignment, i.e., the non-negative value $x_{i,j}$ is the fraction of job j assigned to machine i; we have for all j that $\sum_i x_{i,j} = 1$. As mentioned before, P_{max} denotes the maximum total processing time along any directed path, i.e., $P_{max} = \max_{path\ P}\{\sum_{j\in P}\sum_i x_{i,j}p_{i,j}\}$. Also, Π_{max} denotes the maximum total load on any machine, i.e., $\Pi_{max} = \max_i\{\sum_j x_{i,j}p_{i,j}\}$. We now define a family of linear programs $LP(T)$, one for each value of $T \in \mathbf{Z}^+$, as follows:

$$\forall j \in J,\ \sum_i x_{ij} = 1 \tag{1}$$

$$\forall i \in M,\ \sum_j x_{ij}p_{ij} \leq T \tag{2}$$

$$\forall j \in J,\ z_j = \sum_i p_{ij}x_{ij} \tag{3}$$

$$\forall (j' \prec j)\ c_j \geq c_{j'} + z_j \tag{4}$$

$$\forall j \in J,\ c_j \leq T \tag{5}$$

$$\forall (i,j),\ (p_{i,j} > T) \Rightarrow x_{i,j} = 0 \tag{6}$$

$$\forall (i,j),\ x_{i,j} \geq 0$$

$$\forall j,\ c_j \geq 0$$

The constraints (1) ensure that each job is assigned a machine, and (2) ensures that the maximum fractional load on any machine ($\Pi_{\max}$) is at most T. Constraints (3) define the fractional processing time z_j for a job j, and (4) captures 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 (5) state that the fractional $P_{\max}$ value is at most T, and those of (6) are the valid constraints that if it takes more than T steps to process job j on machine i, then j should not be scheduled on i.

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 η_j be the integer analog of c_j. We first modify the x_{ij} values using *filtering* [17]. Let $K_1 = \frac{3+\sqrt{5}}{2}$. For any (i,j), if $p_{ij} > K_1 z_j$, 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 $r \in [1 - \frac{1}{K_1}, 1)$. So, we scale the (modified) values of x_{ij} by a factor of at most $K_2 = \frac{K_1}{K_1 - 1}$. 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 $K_1 T^*$. This follows from the fact that if $X_{ij} = 1$ in the rounded solution, then $p_{ij} \leq K_1 z_j$. Hence, it is easy to see that by induction, for any job j, η_j is at most $K_1 c_j \leq K_1 T^*$.

We now show how to round $\mathcal{A}$. Recall that [14] presents a rounding algorithm in the "unrelated parallel machines and *no* precedence constraints" context with the following guarantee: if the input fractional solution has a fractional $\Pi_{\max}$ value of α, then the output integral solution has an integral $\Pi_{\max}$ value of at most $\alpha + \max_{(i,j):\, x_{ij}>0} p_{ij}$. We use $\mathcal{A}$ as the input instance for the rounding algorithm in [14]. Note that $\mathcal{A}$ has a fractional $\Pi_{\max}$ value of at most $K_2 T^*$. Further, $\max_{(i,j):\, x_{ij}>0} p_{ij} \leq T^*$ by (6). Thus, the algorithm of [14] yields an integral solution I whose $P_{\max}$ value is at most $K_1 T^*$, and whose $\Pi_{\max}$ value is at most $(K_2 + 1)T^*$. Observe that setting $K_1 = \frac{3+\sqrt{5}}{2}$ results in $K_1 = K_2 + 1$. 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 *Given an arbitrary (not necessarily forest-shaped) DAG, the above algorithm computes a processor assignment for each job in which the value of* $\max\{P_{\max}, \Pi_{\max}\}$ *is within a* $(\frac{3+\sqrt{5}}{2})$*-factor away from LB.*

2.2 Step 2: Solving the GDSS problem under treelike precedences

We can now assume that the assignment of jobs to machines is given. 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. We will use the notation $m(v)$ to denote the machine to which node v is assigned, and the processing time for node v will be 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 the directions of all the arcs in an out-tree. In the discussion below in Section 2.2.1, we only focus on out-trees; the same results can be obtained similarly for in-trees.

We will need Fact 2, a generalization of the Chernoff bound from [19]. Note that the *ordering* of the X_i is important in Fact 2; we make a careful choice of such an ordering in the proof of Lemma 4.

Fact 2 *([19]) Let $X_1, X_2, \ldots, X_l \in \{0,1\}$ be random variables such that for all i, and for any $S \subseteq \{X_1, \ldots, X_{i-1}\}$, $\Pr[X_i = 1 | \bigwedge_{j \in S} X_j = 1] \leq q_i$. (In particular, $\Pr[X_i = 1] \leq q_i$.) Let $X \doteq \sum_i X_i$; note that $\boldsymbol{E}[X] \leq \sum_i q_i$. Then for any $\delta > 0$, $\Pr[X \geq (1+\delta) \cdot \sum_i q_i] \leq (e^\delta/(1+\delta)^{1+\delta})^{\sum_i q_i}$.*

Our algorithm for out-trees requires a careful partitioning of 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 $B_1, \ldots, B_\lambda$ (called blocks) such that the following properties hold:

(P1) The subgraph induced by each block B_i is a collection of vertex-disjoint directed chains, i.e., the in-degree and out-degree of each node in the induced subgraph is at most one (and there are of course no cycles); and

(P2) 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.

The *chain-width* of a DAG is the minimum value λ such that there is a chain decomposition of the DAG into λ blocks. (Such a decomposition always exists: trivially, we could take each block to be a singleton vertex. We also note that the notions of chain decomposition and chain-width are similar to those of caterpillar decomposition and caterpillar dimension for trees [18]. However, in general, a caterpillar decomposition need not be a chain-decomposition and vice-versa.)

Well-structured schedules. We now state some definitions motivated by those in [7]. 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 $\{0, 1, \ldots, B-1\}$. 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 $\{0, 1, \ldots, B-1\}$. 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 [7, 23], 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 schedule 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 $d(v) \in \{0, \ldots, B-1\}$ independently and uniformly at random for each job v which is the head of its chain in a block.

3. Construct a well-structured random B-delayed schedule S' as described above.

4. Construct a valid schedule S'' using the technique from [7] 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 $\{[ip_{\max}, (i+1)p_{\max}),\ i = 0, 1, \ldots, \lceil L/p_{\max} \rceil - 1\}$.

   (c) For each frame, use the frame-scheduling technique from [7] to produce a feasible schedule for that frame. Concatenate the schedules of all frames to obtain the final schedule.

For the sake of completeness, we now describe the frame-scheduling algorithm; the following description (with the exception of the symbols used) is paraphrased from [7] and shows how to schedule the jobs within a given frame of length $p_{\max}$.

"Let Υ be a rooted complete binary tree with $p_{\max}$ leaves. For every node u of Υ, let $l(u)$ and $r(u)$ be the labels, respectively, of the leftmost and rightmost leaves of the subtree rooted at u. We shall associate the jobs scheduled during the frame with the nodes of Υ in a natural way. For $i = 1, \ldots, m$ we define $\xi_i(u)$ to be those jobs that are scheduled on machine i by Υ for precisely the time interval $[l(u), r(u) + 1)$; each job scheduled by Υ in the frame is in exactly one $\xi_i(u)$. Let $p(u) = (r(u) - l(u) + 1) \cdot max_i ||\xi_i(u)||$, where $||\xi_i(u)||$ denotes the cardinality of $\xi_i(u)$. $p(u)$ is the amount of time needed to perform the jobs associated with u. Let the nodes of Υ be numbered as u_1, u_2, ... in the preorder traversal of Υ. Define $f(u_1) = 0$ and for $j \geq 2$, let $f(u_j) = \sum_{k<j} p(u_k)$. The algorithm simply schedules the jobs in $\xi_i(u)$ on machine i consecutively beginning at time $f(u) + 1$ and concluding by the end of timestep $f(u) + p(u)$."

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

Theorem 3 *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 in polynomial time.*

Proof We only analyze the above 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.

The following key lemma bounds the contentions:

Lemma 4 *There exists a constant $c_1 > 0$ such that $\forall i \in \{1, \ldots, m\}$,$\forall t \in \{1, \ldots, L\}$, $C(M_i, t) \leq \alpha$ 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 $\{0,1,\ldots,B-1\}$ 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 $B_1,\ldots,B_\lambda$ 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 $\pi(1),\ldots,\pi(n)$ 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 $\Pr[X(v,i,t) = 1 \mid \bigwedge_{v' \in W} X(v',i,t) = 1] \leq p_v/B$ 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, $\Pr[X(v,i,t) = 1 \mid \bigwedge_{v' \in W} X(v',i,t) = 1] \leq p_v/B$ continues to hold. Thus, Fact 2 can now be applied to bound $\Pr[C(M_i,t) \geq \alpha]$, with $\sum_i q_i = \log(np_{\max})$ and $\delta = c_1 - 1$. Since $e^\delta/(1+\delta)^{1+\delta}$ decreases with δ for $\delta \geq 0$ and tends to 0 as $\delta \to \infty$, we thus get $\Pr[C(M_i,t) \geq \alpha] \leq 1/(np_{\max})^c$, where the constant c can be made arbitrarily large by taking c_1 large enough. 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 Step 4 of our algorithm 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 [7] gives a feasible schedule with the following bounds.

Fact 5 *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 3 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 5 implies that we can obtain a schedule $\mathcal{S}'$ 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 $\mathcal{S}'$ valid schedule $\mathcal{S}$ for I such the makespan increases by at most $P_{\max}/n$, and hence schedule $\mathcal{S}$ is also of length $\rho(P_{\max} + \Pi_{\max})$.

This completes the proof of Theorem 3. ■

Theorem 6 demonstrates a chain decomposition of width $O(\log n)$ for any out-tree: this completes the algorithm for an out-tree. An identical argument works for the case of a directed in-tree.

Theorem 6 *Given an out-tree, we can construct a chain decomposition of it with chain-width at most $\lceil \lg n \rceil + 1$, in deterministic polynomial-time.*

Proof 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 remaining tree 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; we then remove all vertices of this block from T_i. 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 on T_{i+1}; therefore, we do obtain a valid chain decomposition.

Claim 7 *Let $\beta_{\lambda+1-i}$ denote the number of chains induced by $B_{\lambda+1-i}$, in the ith 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. ■

Claim 7 implies that $\beta_\lambda \geq 2^{\lambda-1}$. Since $\beta_\lambda \leq n$, Theorem 6 follows. ■

Thus we get:

Theorem 8 *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 6 does not work for arbitrary forests, and it is not clear how to make the Lemma 4 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 [7]. 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.

We now show how a good decomposition can be computed for forest-shaped DAGs:

Lemma 9 *Given an arbitrary DAG T whose underlying undirected graph is a forest, we can efficiently construct a chain decomposition of it into γ blocks, where $\gamma \leq 2(\lceil \lg n \rceil + 1)$.*

Proof Add an artificial "root" r to T and add an arc from r to some nodes in T, so that the underlying undirected graph becomes a tree. If we imagine T hanging down from r, some edges in T will be pointing away from the root (down) and others will be pointing toward the root (up). Imagine that T is an out-tree and perform a chain decomposition as in the proof of Theorem 6 which will result in a decomposition $\mathcal{B}$ with *blocks* $B_1, \ldots, B_\lambda$ and intermediate trees $T_1, \ldots, T_\lambda$. Recall that we process the subtree $T_{\lambda+1-i}$ to obtain block B_i, and then delete the vertices in B_i from $T_{\lambda+1-i}$ to obtain $T_{\lambda+2-i}$. We now re-partition the blocks in $\mathcal{B}$ into *partitions* $P_1, \ldots, P_{2\lambda}$ of the chain decomposition $\mathcal{P}$ (refer to Figure 1 for an illustration; note that we use the term *blocks* for the intermediate decomposition and *partitions* for the final decomposition).

Consider a "chain" $\mathcal{C}$ in B_i. In general, some edges of $\mathcal{C}$ will point down and others will point up. For instance, in Figure 1, nodes $a, b, \ldots, f$ form a chain in B_i and so do nodes $g, \ldots, m$; the edges (a, b) and (c, b) point down and up respectively. Each node $u \in B_i$ can be classified into the two following types and is put into $P_{\lambda+1-i}$ or $P_{\lambda+i}$ accordingly. Imagine that the tree is undirected, and consider the sequence of edges which connect node u to the tree $T_{\lambda+2-i}$. If the first edge e in this sequence (i.e., the edge e which is incident on u) points up, then u is a *type 1* node and put into $P_{\lambda+1-i}$. Otherwise, if e points down, then u is a *type 2* node and put into $P_{\lambda+i}$. This classification is motivated by the following observation: no node in $T_{\lambda+2-i}$ can be the ancestor of a type 1 node or a descendant of a type 2 node; since type 1 nodes belong to $P_{\lambda+1-i}$, type 2 nodes belong to $P_{\lambda+i}$, and nodes in $T_{\lambda+2-i}$ belong to partitions P_j where $\lambda + 1 - i < j < \lambda + i$, the precedence conditions in the chain decomposition (as required by property **(P2)**) are satisfied. We now formally argue that our construction results in a valid chain decomposition.

We first show that Property **(P1)** of the chain decomposition holds, i.e., in the induced subgraph of a partition, each node has in and out-degrees of at most one, and there are no cycles. Since T has a tree structure, the cycle-free property follows immediately. Consider the stage of the decomposition in which block B_i was created. Let H_i be the induced subgraph of the nodes in block B_i. The total degree (in-degree + out-degree) of any node in H_i is at most 2. If a node u has two out-neighbors in H_i, then u must be of type 1 and at least one of its out-neighbors in H_i must be of type 2 (see node d in Figure 1 for instance). Hence, node u is in partition $P_{\lambda+1-i}$ and one of its out-neighbors in H_i is in partition $P_{\lambda+i}$. It follows from a similar argument that if u has two in-neighbors in H_i, then u will be in $P_{\lambda+i}$ and one of its in-neighbors in H_i will be in $P_{\lambda+1-i}$. All other nodes nodes have an in-degree and out-degree of at most one in H_i. Hence, in the induced subgraphs of $P_{\lambda+1-i}$ and $P_{\lambda+i}$, each node has an in-degree and out-degree of at most one, and property **(P1)** holds.

We now show that property **(P2)** holds. Let node v be a descendant of node u. We consider the following cases:

Case 1: Both u and v belong to the same chain in block B_i. If u and v are of the same type, then it is easy to see that all the nodes in the directed path from u to v are also of the same type as u and v. Hence, all these nodes will be put in the same partition, and property **(P2)** holds. If u and v are of different types, the only possibility is that u is of type 1 and v is of type 2. In this case, $u \in P_{\lambda+1-i}$ and $v \in P_{\lambda+i}$; since $\lambda + 1 - i < \lambda + i$, property **(P2)** follows.

Case 2: Nodes u and v belong to different chains in block B_i. In this case, there exists a directed path from u to a node x in $T_{\lambda+2-i}$, a directed path from node y in $T_{\lambda+2-i}$ to v, and a directed path from x to y in $T_{\lambda+2-i}$. Clearly, node u will be of type 1 and node v will be of type 2 and property **(P2)** follows due to the same argument as in Case 1.

Case 3: Node $u \in B_i$ and $v \in T_{\lambda+2-i}$. In this case, there is a directed path from $u \in B_i$ to $v \in T_{\lambda+2-i}$, and hence u is of type 1 and $u \in P_{\lambda+1-i}$. Further, since $v \in T_{\lambda+2-i}$, v can only be in a partition P_j such that $\lambda + 1 - i < j < \lambda + i$. Since $T_{\lambda+2-i}$ is non-empty, we have $i > 1$; these facts together yield, $\lambda + 1 - i < j$, and **(P2)** is satisfied.

Case 4: Node $v \in B_i$ and $u \in T_{\lambda+2-i}$. Property **(P2)** is satisfied due to similar arguments as in Case 3.

This completes the proof of the Lemma. ■

Theorem 10 *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 $O((P_{\max} + \Pi_{\max}) \cdot \rho)$,*

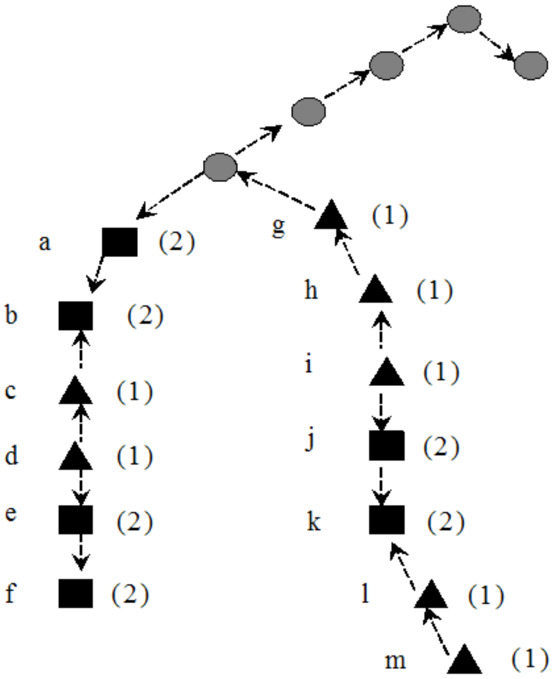

Figure 1: Chain-decomposition of a DAG whose underlying structure is a tree: the triangular and rectangular nodes are in block B_i while the circular nodes are in the subtree $T_{\lambda+2-i}$. The triangular nodes are of type 1 and belong to partition $P_{\lambda+1-i}$ while the rectangular nodes are of type 2 and belong to partition $P_{\lambda+i}$. The letters $a, \ldots, m$ to the left of the nodes denote their labels and the numbers to their right in parentheses denote their types.

where $\rho = \frac{\gamma \log n}{\log\log n} \cdot \left\lceil \frac{\log \min(p_{\max}, n)}{\log\log n} \right\rceil$*. Thus, Lemma 9 implies that* $\rho = O\left(\frac{\log^2 n}{\log\log n} \cdot \left\lceil \frac{\log \min(p_{\max}, n)}{\log\log n} \right\rceil\right)$ *is achievable for forest-shaped DAGs.*

Proof Consider the chain decomposition of the DAG with γ blocks $P_1, P_2, \ldots, P_\gamma$. 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 [7] 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 the proof of Theorem 3, we may assume without loss of generality that $p_{\max}$ is polynomially bounded in n). ■

2.3 The Limits of our Lower Bound

Our approach for the GDSS problem uses the fact that $\max\{P_{\max}, \Pi_{\max}\}$ is a lower-bound on the length of any schedule. Any attempt to improve our approximation guarantees must address the issue of how good a lower-bound is $\max\{P_{\max}, \Pi_{\max}\}$. We now show that in general DAGs, the $\mathcal{L} \doteq \max\{P_{\max}, \Pi_{\max}\}$ 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 $\Omega(\mathcal{L} \cdot \log n / \log \Pi_{\max})$.

We construct a rooted in-tree T for which the optimal makespan is $\Omega(\mathcal{L} \cdot \log n / \log \Pi_{\max})$, for any value of $\Pi_{\max}$ 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 11 *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. ■

2.3.1 An $\Omega(\sqrt{n})$ gap for general DAGs

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 $m = \sqrt{n}$ machines and m layers; each layer contains m nodes, each to be processed on a layer-specific machine with unit processing time (two nodes are assigned to the same machine if they are on the same layer, and different machines if they are on different layers). Let these layers be denoted by $V_1, \ldots, V_m$. For each $i = 1, \ldots, m-1$, 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 $n = P_{\max}\Pi_{\max}$. We show in Section 4 that the natural time-indexed integer program considered therein, also has an $\Omega(m)$ gap between the integral and fractional optima for this instance.

Finally, we remark that while the discussion in this Section (2.3) pertains to the GDSS problem (where processor assignment is pre-specified), our results can be easily generalized to the case where the processor assignment is not pre-specified.

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

We consider next 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 $\Pi_{\max}$ denote the "dilation" and "congestion" as usual; if there exists a schedule of makespan $\rho \cdot (P_{\max} + \Pi_{\max})$ for $R|prec|C_{\max}$, then there is a $O(\rho)$-approximation algorithm for $R|prec|\sum_j w_j C_j$. We adapt an approach of [5, 20] for this. Let the machines and jobs be indexed respectively 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 $\ell = 0, 1, \ldots$, 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 LP is to minimize $\sum_j w_j C_j$ subject to:

$$\begin{aligned}
\forall j,\ \sum_{i,\ell} x_{i,j,\ell} &= 1 && (7)\\
\forall j,\ z_j &= \sum_i p_{i,j} \sum_{\ell} x_{i,j,\ell} &&\\
\forall (j \prec k),\ C_k &\geq C_j + z_j &&\\
\forall j,\ \sum_{i,\ell} 2^{\ell-1} x_{i,j,\ell} &\leq C_j \leq \sum_{i,\ell} 2^{\ell} x_{i,j,\ell} && (8)\\
\forall (i,\ell),\ \sum_j p_{i,j} \sum_{t \leq \ell} x_{i,j,t} &\leq 2^{\ell} && (9)\\
\forall \ell\ \forall \text{maximal chains } \mathcal{P},\ \sum_{j \in \mathcal{P}} \sum_i p_{i,j} \sum_{t \leq \ell} x_{i,j,t} &\leq 2^{\ell} && (10)\\
\forall (i,j,\ell),\ (p_{i,j} > 2^{\ell}) &\Rightarrow (x_{i,j,\ell} = 0) && (11)\\
\forall (i,j,\ell),\ x_{i,j,\ell} &\geq 0 &&
\end{aligned}$$

Note that (9) and (10) 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 (8) that the total "mass" (sum of $x_{i,j,\ell}$ values) for the values ℓ such that $2^{\ell} \geq 4C^*_j$, 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 (7) still holds. After the filtering, each non-zero variable increases by at most a factor if 2. Additionally, for any fixed j, the following property is satisfied: if ℓ' is the largest integer such that $x_{i,j,\ell'}$ is non-zero, then $2^{\ell'} = O(C^*_j)$. The right-hand-sides of (9) and (10) 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 set of jobs whose frame has been set to ℓ. Let $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ be the maximum fractional congestion and dilation respectively for any job in $G[\ell]$. From constraints (9) and (10), 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 still at most $O(2^{\ell})$.

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 $C^*_{j_1} \leq C^*_{j_2}$ – either frame $F(j_1)$ precedes $F(j_2)$ or they coincide, and hence j_1 gets scheduled before j_2.

Theorem 12 *Consider any family $\mathcal{F}$ of precedence constraints that is closed under taking subsets: i.e., if a partial order σ is in $\mathcal{F}$, then any partial order obtained by removing arcs from σ is also in $\mathcal{F}$. If there exists an approximation algorithm for $R|prec|C_{\max}$ for all precedence constraints $\sigma \in \mathcal{F}$ that yields a schedule whose makespan is $O((P_{\max} + \Pi_{\max}) \cdot \rho)$, then there is also an $O(\rho)$–approximation algorithm for $R|prec|\sum_j w_j C_j$ for all $\sigma \in \mathcal{F}$. Thus, Theorem 10 implies that an $O(\frac{\log^2 n}{\log\log n}\lceil\frac{\log\min(p_{\max},n)}{\log\log n}\rceil)$-approximation is achievable for $R|forest|\sum_j w_j C_j$.*

Proof Consider any job j which belongs to $G[\ell]$; since both $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ are $O(2^\ell)$, the jobs in $G[\ell]$ take a total of $O(\rho 2^\ell)$ to complete. Thus, even given the wait for all jobs in $G[\ell']$ for $\ell' < \ell$, the completion time of job j is $O(\rho \cdot \sum_{\ell' \leq \ell} 2^{\ell'}) = O(\rho 2^\ell)$. Since $2^\ell = O(C_j^*)$, the theorem follows. ■

4 Minimizing the weighted flow time under precedence-chains

We now study the flow-time objective. We consider the *restricted-assignment variant* where for every job v, there is a value p_v such that for all machines i, $p_{i,v} \in \{p_v, \infty\}$. 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 present a bicriteria approximation algorithm for the weighted flow time.

Let us recall the flow-time objective. 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 started only at times which are $\geq r_v$. The deadline l_v specifies the last time slot at which this job can be started, say. 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. In general, minimizing the weighted flow time appears very hard to approximate. Leonardi & Raz [13] 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 such intractability 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 a ψ-count solution. Let OPT be the cost (total weighted flow time) of the optimal schedule. We say that a solution is (ψ, γ)-approximate, if its count is ψ and the cost of the solution is at most $\gamma \cdot OPT$. Note that we compare the cost of our ψ-count solution with that of a 1-count optimal solution. Chekuri *et al.* [3] study the minimum flow-time scheduling with resource augmentation for the setting without precedence constraints, and present online algorithms with approximation (competitive) ratio $(1 + \epsilon, O(\frac{1}{\epsilon}))$. Here we study flow-time scheduling with resource augmentation in the presence of precedences; we present an algorithm, which either proves that input instance is has no feasible solution, or outputs a (ψ, γ)-approximate solution where $\psi = O(\frac{\log N}{\log\log N})$ and $\gamma = 1 + o(1)$ with high probability. (Recall that N here denotes the input size.)

Let $T = \sum_u p_u$. The values r_v and l_v could trivially be 1 and ∞, respectively; if $l_v > T$, we reset l_v without loss of generality to T. Let $\vdash$ denote the immediate-predecessor relation, i.e., if $u \vdash v$, 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. Let $S(v)$ denote the set of machines on which v can be processed: i.e., the set $\{i : \; p_{i,v} = p_v\}$. In the time-indexed LP formulation below, we consider the LP relaxation of the integer program in which the variables have the following interpretation. For each job v and time t, $r_v \leq t \leq l_v$, $x_{v,i,t}$ is the indicator for job v being *started* on machine i at time t, $z_{v,t}$ is the indicator for job v being started at time t, and C_v is the completion time of v. (Henceforth, any variable $x_{v,i,t}$ or $z_{v,t}$ where t is not in the range $[r_v, l_v]$, is taken to be zero.) The objective is $\min \sum_v w_v(C_v - r_v)$, subject to:

$$\forall v, \quad \sum_{i \in S(v)} \sum_{t \in [r_v, \ldots, l_v]} x_{v,i,t} = 1 \tag{12}$$

$$\forall\, i \in [1, \ldots m]\ \forall t \in [1, \ldots T], \quad \sum_v \sum_{\max\{r_v, t-p_v+1\} \leq t' \leq t} x_{v,i,t'} \leq 1 \tag{13}$$

$$\forall v\ \forall t \in [r_v, \ldots, l_v], \quad z_{v,t} = \sum_{i \in [1, \ldots m]} x_{v,i,t} \tag{14}$$

$$\forall u \vdash v\ \forall t \in [r_v, \ldots, p_u], \quad z_{v,t} = 0 \tag{15}$$

$$\forall u \vdash v \ \forall t \in [p_u + 1, \dots T], \quad \sum_{t' \in [1,\dots,t]} z_{v,t'} \quad \le \quad \sum_{t' \in [1,\dots,t-p_u]} z_{u,t'} \tag{16}$$

$$\forall v, \ C_v \quad = \quad \sum_{t \in [1,\dots T]} (t + p_v - 1) \cdot z_{v,t} \tag{17}$$

$$\forall v \ \forall i \in S(v) \ \forall t \in [r_v, \dots, l_v], \ x_{v,i,t} \quad \ge \quad 0$$

The constraints (12) ensure that all jobs are processed completely, and (13) ensure that at most one job is (fractionally) assigned to any machine at any time. Constraints (15) and (16) are the precedence constraints and (17) defines the completion time C_v for job v.

Our algorithm proceeds as follows. We first solve the above LP optimally; if there is no feasible solution (e.g., if the deadlines l_v are too restrictive), we announce this and stop. Otherwise, let OPT be the optimal value of the LP and let x^*, z^* and C^* be the vectors denoting the optimal solution-values. We define a randomized rounding procedure for each chain such that the following three properties hold:

(A1) Let $Z_{v,t}$ be the indicator random variable which denotes if v is started at time t in the rounded solution. Let $X_{v,i,t}$ be the indicator random variable which denotes if v is started at time t on machine i in the rounded solution. Then $\mathbf{E}[Z_{v,t}] = z^*_{v,t}$ and $\mathbf{E}[X_{v,i,t}] = x^*_{v,i,t}$.
(A2) All precedence constraints are satisfied in the rounded solution with probability one.
(A3) Jobs in different chains are rounded independently.

Our rounding procedure to choose the $Z_{v,t}$ is 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,

$$Z_{v,t'} = 1 \text{ iff } \sum_{t=1}^{t'-1} z^*_{v,t} < R(\Gamma) \le \sum_{t=1}^{t'} z^*_{v,t}. \tag{18}$$

Bertsimas *et al.* [1] show other applications of such rounding techniques. 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 started on exactly one machine at time t, with the probability for machine i being $(x^*_{v,i,t}/z^*_{v,t})$. A moment's reflection shows that:

- property **(A1)** holds because the condition on $R(\Gamma)$ in (18) happens with probability $z^*_{v,t'}$;
- **(A2)** holds due to the precedence constraints (15) and (16); and
- **(A3)** is true since the different chains choose the values $R(\Gamma)$ independently.

In general, this assignment strategy might result in jobs from *different* chains executing on the same machine at the same time, and hence in an infeasible schedule. (Jobs from the same chain cannot contend for the same machine at the same time, since property **(A2)** holds.) Let Y be the random variable which denotes the maximum contention of any machine at any time. We obtain a feasible solution by resource augmentation: deploying Y copies of each machine.

An application of the Chernoff-type bound from Fact 2 yields the following bound on Y, which we state in general terms without assuming that all the p_v are bounded by a polynomial of N:

Lemma 13 *Let $p_{\max} \doteq \max_v p_v$, and define $M = \max\{N, p_{\max}\}$. Let $\mathcal{E}$ denote the event that $Y \le (\alpha \log M / \log\log M)$, where $\alpha > 0$ is a suitably large absolute constant. Event $\mathcal{E}$ occurs after the randomized machine assignment with high probability: this probability can be made at least $1 - 1/M^\beta$ for any desired constant $\beta > 0$, by letting the constant α be suitably large.*

Proof Let $L_{i,t}$ denote the contention on machine i at time-step t in the infeasible schedule. The number of such random variables $L_{i,t}$ is at most $m \cdot T \leq mn \cdot p_{\max}$, which is bounded by a fixed polynomial of M. So, it suffices to fix (i,t) arbitrarily, and to show that for any desired constant $\beta' > 0$, a large enough choice of the constant α ensures that

$$\Pr[L_{i,t} > \alpha \log M / \log\log M] \leq M^{-\beta'}; \tag{19}$$

a union bound over all (i,t) will then complete the proof.

Let us now prove (19). For each chain Γ, note that the total load imposed by Γ on machine i at time t in the infeasible schedule, is given by

$$U_{i,t}(\Gamma) \doteq \sum_{v \in \Gamma} \sum_{\max\{r_v, t-p_v+1\} \leq t' \leq t} X_{v,i,t'}.$$

So, $L_{i,t} = \sum_{\Gamma} U_{i,t}(\Gamma)$. We note some facts:

- By **(A1)** and by (13), $\mathbf{E}[L_{i,t}] \leq 1$;
- by **(A2)**, each $U_{i,t}(\Gamma)$ lies in $\{0,1\}$; and
- by **(A3)**, the random variables $U_{i,t}(\Gamma)$ are *independent* of each other.

Since Fact 2 directly applies to a sum of independent binary random variables, we get, by setting $\sum_i q_i = 1$ in Fact 2, that for any $\delta > 0$,

$$\Pr[L_{i,t} \geq 1 + \delta] \leq (e/(1+\delta))^{1+\delta}.$$

A simple calculation now shows that (19) is satisfied by choosing $1 + \delta = \alpha \log M / \log\log M$ for a large enough constant α. ■

Finally, we note that we construct a schedule only if the event $\mathcal{E}$ occurs. Otherwise, we can repeat the randomized machine assignment until event $\mathcal{E}$ occurs and resource-augment the resultant infeasible schedule. Since $p_{\max} \leq poly(N)$ by assumption, the event $\mathcal{E}$ implies that $Y = O(\log N / \log\log N)$. Note that for any job v, its completion time C_v equals $\sum_t (t + p_v - 1) \cdot Z_{v,t}$; so, $\mathbf{E}[C_v]$ equals the fractional completion time C_v^*, due to **(A1)** and the linearity of expectation. Thus, the expected value of the flow time F_v of v, also equals its fractional value $F_v^* = C_v^* - r_v$. Now, by Lemma 13, even conditional on the event $\mathcal{E}$,

$$\mathbf{E}[F_v \mid \mathcal{E}] \leq \frac{\mathbf{E}[F_v]}{\Pr[\mathcal{E}]} = \frac{F_v^*}{\Pr[\mathcal{E}]} = (1 + o(1)) \cdot F_v^*,$$

since $\Pr[\mathcal{E}] = 1 - o(1)$. So, even conditional on $\mathcal{E}$ (which happens with high probability), the expected cost of our solution is at most $(1 + o(1)) \cdot OPT$.

Integrality gap for the instance of Section 2.3.1. We now see that the relative of the above time-indexed formulation performs quite poorly for general DAGs, when applied to the GDSS problem (where we aim to minimize the makespan). Specifically, consider GDSS instances with all processing times being unity, as in Section 2.3.1. We first "guess" an upper bound T' for the makespan, as in Section 2.1. We write an LP such as the time-indexed one above, with the following modifications: (i) all the time variables t take values in $\{1, 2, \ldots, T'\}$; (ii) all the values r_v and l_v are trivial – i.e., $r_v \equiv 1$ and $l_v \equiv T'$, and (iii) the constraints (15) and (16) are included for *all* pairs of nodes (u,v) for which u is constrained to precede v. As mentioned in Section 2.3.1, the optimal (integral) solution for the instance therein has makespan $n = m^2$, but here is a fractional solution to this time-indexed formulation which makes $T' = 2m - 1$ feasible: if node v at level V_j has been pre-assigned to machine i, then $z_{v,t}^* = x_{v,i,t}^* = 1/m$ for $t = j, j+1, \ldots, j+m-1$ (and all other $z_{v,t}^*, x_{v,i,t}^*$ values are zero). Thus, even this time-indexed formulation has an integrality gap of $\Omega(\sqrt{n})$ for general DAGs.

5 Open Questions

The flow-time objective appears difficult in general, and merits further study. As one concrete problem, it would be interesting to know if we can get a "(poly)logarithmic resource augmentation" result for, say, job shop scheduling, without the assumption that the processing times are polynomially-bounded.

Can one improve our polylogarithmic approximations to constant, say? Finally, there is of course the problem of dag-shop scheduling with general DAGs. Semidefinite programming has found use in related contexts in scheduling; see, e.g., [24]. Can semidefinite programming be a useful approach in our contexts also?

Acknowledgments. We are thankful to Chandra Chekuri, David Shmoys, and the referees for their 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, A. Goel, S. Khanna, and A. kumar. *Multi-processor scheduling to minimize flow time with ϵ resource augmentation.* STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pages 363 – 372, 2004.

[4] C. Chekuri and S. Khanna. *Approximation algorithms for minimizing average weighted completion time.* Chapter 11, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, edited by Joseph Leung, CRC Press, pages 11-1 – 11-30, 2004.

[5] 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.

[6] U. Feige and C. Scheideler. *Improved bounds for acyclic job shop scheduling.* Combinatorica, 22:361–399, 2002.

[7] 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.

[8] L. Hall. *Approximation Algorithms for Scheduling.* in *Approximation Algorithms for NP-Hard Problems*, Edited by D. S. Hochbaum. PWS Press, 1997.

[9] 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.

[10] 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.

[11] K. Jansen and R. Solis-Oba. *Scheduling jobs with chain precedence constraints.* Parallel Processing and Applied Mathematics, PPAM, LNCS 3019, pp. 105-112, 2003.

[12] 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.

[13] S. Leonardi and D. Raz. In *Approximating total flow time on parallel machines.* In Proc. ACM Symposium on Theory of Computing, 110-119, 1997.

[14] J. K. Lenstra, D. B. Shmoys and É. Tardos. *Approximation algorithms for scheduling unrelated parallel machines.* Mathematical Programming, Vol. 46, 259-271, 1990.

[15] F.T. Leighton, B. Maggs and S. Rao. *Packet routing and jobshop scheduling in O(congestion + dilation) Steps,* Combinatorica, Vol. 14, 167-186, 1994.

[16] F. T. Leighton, B. Maggs, and A. Richa, *Fast algorithms for finding O(congestion + dilation) packet routing schedules.* Combinatorica, Vol. 19, 375-401, 1999.

[17] 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.

[18] 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.

[19] 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.

[20] M. Queyranne and M. Sviridenko. *Approximation algorithms for shop scheduling problems with minsum objective,* Journal of Scheduling, Vol. 5, 287–305, 2002.

[21] A. Schulz and M. Skutella. *The power of α-points in preemptive single machine scheduling.* Journal of Scheduling 5(2): 121 - 133, 2002.

[22] P. Schuurman and G. J. Woeginger. *Polynomial time approximation algorithms for machine scheduling: Ten open problems.* Journal of Scheduling 2:203-213, 1999.

[23] D.B. Shmoys, C. Stein and J. Wein. *Improved approximation algorithms for shop scheduling problems,* SIAM Journal on Computing, Vol. 23, 617-632, 1994.

[24] M. Skutella. *Convex quadratic and semidefinite relaxations in scheduling.* Journal of the ACM, 46(2):206–242, 2001.