

Scheduling on Unrelated Machines under Tree-Like Precedence Constraints

V.S. Anil Kumar¹, Madhav V. Marathe², Srinivasan Parthasarathy³, and Aravind Srinivasan³

¹ Basic and Applied Simulation Science (CCS-DSS), Los Alamos National Laboratory, MS M997, P.O. Box 1663, Los Alamos, NM 87545. Email: anil@lanl.gov*

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

³ Department of Computer Science, University of Maryland, College Park, MD 20742. Email: [sri,srin}@cs.umd.edu](mailto:{sri,srin}@cs.umd.edu) **

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. We also 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. 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 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 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,

* Research supported by the Department of Energy under Contract W-7405-ENG-36.

** Research supported in part by NSF Award CCR-0208005 and NSF ITR Award CNS-0426683.

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. Hall [7]).

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, 23], 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 (see Shmoys *et al.* [22]), 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 [22, 6, 5] are based on variants of this technique. Shmoys *et al.* [22] 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*.

The only results known for the case of arbitrary number of processors with more general precedence constraints are for identical parallel machines (denoted by $P|prec|C_{max}$; Hall [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. Schulz and Skutella [20]).

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 [23], or for precedence constraints with identical/related processors [8, 19]. 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 Jansen and Solis-oba [10] for the case of chains.

1. The $R|forest|C_{max}$ problem. We obtain a polylogarithmic approximation for this problem in Section 2. We employ the same lower bound used in [14, 22, 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, for a fixed assignment of jobs to machines. 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 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. Shmoys *et al.* [22]) 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 Lenstra *et al.* [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.

2. The $R|forest|\sum_j w_j C_j$ problem. In Section 3, 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 Queyranne and Sviridenko [19]: both reductions employ geometric time windows and linear constraints on completion times for bounding congestion and dilation. However, the reduction in [19] is meant for *identical* parallel machines while our reduction works for *unrelated* machines. Further, [19] works for job-shop and dag-shop problems under the assumption that *no two operations from the same DAG can be executed concurrently, although no precedence relation might exist between the two operations*; in contrast, we do not impose this restriction and allow concurrent processing subject to precedence and assignment constraints being satisfied.

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 in Section 4. Our result en-

sures 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.

Due to space limitations, several proofs and algorithm details are omitted here and are deferred to the full version this paper.

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

Consider a (fractional) assignment x of jobs to machines, where $x_{i,j}$ 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|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 Lenstra *et al.* [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 $LP(T^*)$ 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* (Lin and Vitter [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 Lenstra *et al.* [13] present 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 of Lenstra *et al.* [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. We then extend this to the case where the precedences form an arbitrary forest (i.e., the underlying undirected graph is a forest). 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 .

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 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*; the decomposition for an out-directed arborescence is illustrated in Figure 1. 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.

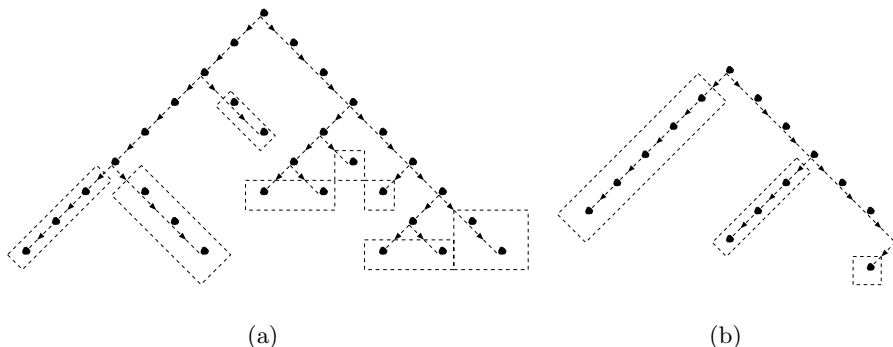


Figure 1. Chain decomposition of an out-directed arborescence. The vertices enclosed within the boxes in figures 1(a) and 1(b) are in blocks B_3 and B_2 respectively while the remaining vertices are in block B_1 .

Well structured schedules. We now state some definitions motivated by those in Goldberg *et al.* [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 be the head of 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, 22], 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 $\hat{v} = (m(v), 2p_v)$. 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 Goldberg *et al.* [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.

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 (see Linial *et al.* [17]). However, in general, a caterpillar decomposition for an arborescence need not be a chain-decomposition and vice-versa.

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.*

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 for in/out-trees does not work for arbitrary forests: instead of following the approach for in/out-trees, 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 Goldberg *et al.* [6]. While this is simpler than the algorithm for in/out-trees, we incur another logarithmic factor in the approximation guarantee.

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

Lemma 1. *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 4. *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*

$$\rho = \frac{\gamma \log n}{\log \log n} \lceil \frac{\log \min(p_{\max}, n)}{\log \log n} \rceil. \text{ Thus, Lemma 1 implies that}$$

$$\rho = O\left(\frac{\log^2 n}{\log \log n} \lceil \frac{\log \min(p_{\max}, n)}{\log \log n} \rceil\right) \text{ is achievable.}$$

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 5. *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 4 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.*

Proof. Consider any job j which belongs to $G[\ell]$; both $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ are $O(2^\ell)$, the final completion time of job j is $O(\rho 2^\ell)$. Since $2^\ell = O(C_j^*)$, the theorem follows.

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, 21], 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. The algorithm and proof techniques for flow time is similar to weighted completion time and is omitted from this version.

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 *dif-*

ferent 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 [18] yields the following lemma:

Lemma 2. *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 as follows: $\mathbf{E}[C \mid \mathcal{E}] \leq \mathbf{E}[O\left(\frac{\log N}{\log \log N}\right) \cdot C_1 \mid \mathcal{E}] \leq O\left(\frac{\log N}{\log \log N}\right) \cdot \frac{\mathbf{E}[C_1]}{\Pr[\mathcal{E}]} \leq O\left(\frac{\log N}{\log \log N}\right) \cdot OPT$.

Acknowledgments. We are thankful to David Shmoys and the anonymous APPROX 2005 referees 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*. *Parallel Processing and Applied Mathematics*, PPAM, LNCS 3019, pp. 105-112, 2003.
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. M. Queyranne and M. Sviridenko. *Approximation algorithms for shop scheduling problems with minsum objective*, *Journal of Scheduling*, Vol. 5, 287–305, 2002.
20. A. Schulz and M. Skutella. *The power of α -points in preemptive single machine scheduling*. *Journal of Scheduling* 5(2): 121 - 133, 2002.
21. P. Schuurman and G. J. Woeginger. *Polynomial time approximation algorithms for machine scheduling: Ten open problems*. *Journal of Scheduling* 2:203-213, 1999.
22. D.B. Shmoys, C. Stein and J. Wein. *Improved approximation algorithms for shop scheduling problems*, *SIAM Journal on Computing*, Vol. 23, 617-632, 1994.
23. M. Skutella. *Convex quadratic and semidefinite relaxations in scheduling*. *Journal of the ACM*, 46(2):206–242, 2001.