An Efficient Approximation Algorithm for Minimizing Makespan on Uniformly Related Machines*

Chandra Chekuri† Michael Bender‡

October 3, 2001

Abstract

We give a new and efficient approximation algorithm for scheduling precedence constrained jobs on machines with different speeds. The problem is as follows. We are given n jobs to be scheduled on a set of m machines. Jobs have processing times and machines have speeds. It takes p_j/s_i units of time for machine i with speed s_i to process job j with processing requirement p_j. Precedence constraints between jobs are given in the form of a partial order. If $j \prec k$, processing of job k cannot start until job j's execution is completed. The objective is to find a non-preemptive schedule to minimize the *makespan* of the schedule. Chudak and Shmoys [2] gave an algorithm with an approximation ratio of $O(\log m)$, significantly improving the earlier ratio of $O(\sqrt{m})$ due to Jaffe [6]. Their algorithm is based on solving a linear programming relaxation. Building on some of their ideas we present a combinatorial algorithm that achieves a similar approximation ratio but runs in $O(n^3)$ time. Our algorithm is based on a new and simple lower bound which we believe is of independent interest.

1 Introduction

The problem of scheduling precedence constrained jobs on a set of identical parallel machines to minimize makespan is one of the first problems for which approximation algorithms have been devised. Graham [4] showed that list scheduling gives a $(2 - 1/m)$ approximation when scheduling on m machines. In *list scheduling*, jobs are ordered according to a given list and whenever a machine

*A preliminary version of this paper appeared in the *Proceedings of Integer Programming and Combinatorial Optimization (IPCO)*, 1998 [1].

†Department of Computing Principles Research, Bell Labs, 600-700 Mountain Ave, Murray Hill, NJ 07974. Email: chekuri@research.bell-labs.com. This work was done while the author was a graduate student at Stanford University where he was supported by an IBM Cooperative Fellowship.

‡Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400. Email: bender@cs.sunysb.edu. This work was done while the author was a graduate student at Harvard University where he was supported by NSF grant CCR-95-04436.

is free, the earliest unscheduled job from the list that is ready (all its predecessors have finished), is scheduled on the free machine. Graham's analysis shows that the $(2 - 1/m)$ approximation holds irrespective of the list. To date this ratio has not been improved. We consider a generalization of this problem in which machines have different speeds. In the scheduling literature such machines are called *uniformly related*. Formally, we are given n jobs that are to be scheduled on a set of m machines. Job j has a processing time p_j and machine i has a speed s_i and it takes p_j/s_i time units to process job j on machine i. Precedence constraints between jobs impose a partial order on their execution and are given in the form of a directed acyclic graph (DAG). If i precedes j, denoted by $i \prec j$, execution of job j cannot start until job i is finished. Let C_j denote the completion time of job j. The objective is to find a schedule that minimizes the time to finish all the jobs, or in other words to minimize the quantity $C_{\max} = \max_j C_j$, called the makespan of the schedule. We restrict ourselves to non-preemptive schedules where a job once started on a machine has to run to completion on the same machine. Our results carry over to the preemptive case as well. In the scheduling literature where problems are classified in the $\alpha|\beta|\gamma$ notation [7], this problem is referred to as $Q|prec|C_{\max}$.

Liu and Liu [9] analyzed the performance of Graham's list scheduling algorithm for the case of different speeds and showed that it has an approximation ratio of $(1 + \max_i s_i/\min_i s_i - \max_i s_i/\sum_i s_i)$. This approximation ratio depends on the ratio of the largest to the smallest speed and could be arbitrarily large even for a small number of machines. The first algorithm to have an approximation ratio independent of the speeds was given by Jaffe [6]. By generalizing the analysis of Liu and Liu, he showed that list scheduling when restricted to the set of machines with speeds that are at least $1/\sqrt{m}$ times the fastest machine speed, results in an $O(\sqrt{m})$ approximation ratio. Recently Chudak and Shymoys [2] improved the ratio significantly by giving an $O(\log m)$ approximation algorithm. Their algorithm relies on solving a linear programming relaxation of the problem. We obtain a new algorithm for this problem. The approximation ratio of our algorithm is also $O(\log m)$ but is advantageous for the following reasons. Our algorithm is combinatorial and runs in $O(n^3)$ time, hence is more efficient than the algorithm in [2]. Further, the analysis of our algorithm relies on a new and natural lower that might be useful in other contexts. We also show that our algorithm achieves a constant factor approximation when the precedence constraints are induced by a collection of chains. By a result of Shmoys, Wein, and Williamson [10], our algorithm can be extended to obtain similar ratios when jobs have release dates. Our work uses some basic ideas of Chudak and Shmoys [2]. A linear programming relaxation that gives an $O(\log m)$ approximation ratio for the more general problem of minimizing the sum of weighted completion times $(Q|prec|\sum_j w_j C_j)$ is also presented in[2]. Our ideas do not generalize for that problem.

The rest of this paper is organized as follows. In Section 2 we describe some ideas from the paper of Chudak and Shmoys [2] that are useful to us. We present our lower bound in Section 3, and give the approximation algorithm and its analysis in Section 4.

2 Preliminaries

We summarize below the basic ideas in the work of Chudak and Shmoys [2]. Their main result is an algorithm that gives an approximation ratio of $O(K)$ for the problem of $Q|prec|C_{\max}$ where K is the number of distinct speeds. They show how to reduce an instance with arbitrary speeds to one in which there are only $O(\log m)$ distinct speeds, as follows.

- Ignore all machines with speed less than $1/m$ times the speed of the fastest machine.
- Round down all speeds to the nearest power of 2.

They observe that the above transformation can be done with only a constant factor loss in the approximation ratio. We will therefore restrict ourselves to instances with K distinct speeds with the implicit assumption that $K = O(\log m)$. The above ideas also imply an $O(\log(\max_i s_i / \min_i s_i))$ approximation ratio.

When all machines have the same speed ($K = 1$), Graham [4] showed that list scheduling gives a 2 approximation. His analysis shows that in any schedule produced by list scheduling, we can identify a chain of jobs $j_1 \prec j_2 \ldots \prec j_r$ with the following property: during the schedule, either all machines are busy processing jobs, or some machine is processing a job from the chain. The time spent processing the chain is clearly a lower bound on the optimum makespan. In addition, the total measure of time during which all machines are busy is also a lower bound via arguments about the average load. These two bounds provide an upper bound of 2 on the approximation ratio of list scheduling. One can apply a similar analysis for the multiple speed case. As observed in [2], the difficulty is that the time spent in processing the chain identified from the list scheduling analysis, is not a lower bound when speeds are different. We can claim that the processing time of any chain on the fastest machine is a lower bound. However, the jobs in the chain identified by the list scheduling analysis do not necessarily run on the fastest machine. Based on this observation, the algorithm in [2] finds an assignment of jobs to speeds (machines) that ensures that the processing time of any chain is bounded by some factor of the optimal.

We will follow the notation of [2] for sake of continuity and convenience. Recall that our instance has K distinct speeds. Let m_k be the number of machines with speed $\overline{s}_k$, $k = 1, \ldots, K$, where $\overline{s}_1 > \ldots > \overline{s}_K$. In the sequel we will be interested in assigning jobs to speeds. For a given assignment, let $k(j)$ denote the speed at which job j is assigned to be processed. The average processing allocated to a machine of a specific speed k, denoted by D_k, is the following.

$$D_k = \frac{1}{m_k \overline{s}_k} \sum_{j : k(j) = k} p_j .$$

A chain is simply a subset of jobs that are totally ordered by the precedence constraints. A formal definition follows.

Definition 1 *A* chain *P is a set of jobs $j_1, \ldots, j_r$ such that for all $1 \leq i < r$, $j_i \prec j_{i+1}$. The* length *of a chain P, denoted by $|P|$, is the sum of the processing times of the jobs in P.*

Let $\mathcal{P}$ be the set of all chains induced by the precedence constraints. For a chain P, the quantity $\sum_{j \in P} \frac{p_j}{\overline{s}_{k(j)}}$ denotes the minimum time required to finish the jobs in the chain *if* they are scheduled according to the given assignment of jobs to speeds. For a given job assignment we can compute a quantity C defined by the following equation.

$$C = \max_{P \in \mathcal{P}} \sum_{j \in P} \frac{p_j}{\overline{s}_{k(j)}}$$

Thus C represents a lower bound on the makespan for the *given* assignment of jobs to speeds.

A natural variant of list scheduling called speed based list scheduling, developed in [2], is constrained to schedule according to the speed assignments of the jobs. In classical list scheduling, whenever a machine is free the first available job from the list is scheduled on it. In speed based list scheduling, an available job is scheduled on a free machine provided the speed of the free machine matches the speed assignment of the job. The proof of the following theorem follows from a straight forward generalization of Graham's analysis of list scheduling.

Theorem 1 (Chudak & Shmoys [2]) *For any job assignment $k(j)$, $j = 1, \ldots, n$, the speed-based list scheduling algorithm produces a schedule of length*

$$C_{\max} \leq C + \sum_{k=1}^{K} D_k.$$

Let $C^*_{\max}$ denote the optimum makespan for the given instance. In [2] a linear programming relaxation of the problem is used to obtain a job assignment that satisfies the following two conditions: $\sum_{k=1}^{K} D_k \leq (K + \sqrt{K})C^*_{\max}$, and $C \leq (\sqrt{K} + 1)C^*_{\max}$. Plugging these bounds in Theorem 1 gives an $O(K)$ approximation ratio. We use an alternative method based on chain decompositions to obtain an assignment satisfying similar properties.

3 A New Lower Bound

In this section we develop a simple and natural lower bound that will be used in the analysis of our algorithm. Before formally stating the lower bound we provide some intuition. The two lower bounds used in Graham's analysis for identical parallel machines are the maximum chain length and the average load. As discussed in the previous section, a naive generalization of the first lower bound is the maximum chain length divided by the fastest speed. However it is easy to show examples where the maximum of this bound and the average load is $O(1/m)$ times the optimal. We describe one such example to motivate our new bound. Suppose we have two speeds with $\overline{s}_1 = d$ for some $d > 1$ and $\overline{s}_2 = 1$. The precedence constraints between the jobs are induced by a collection of $\ell > 1$ disjoint chains, each of the same length d. Suppose $m_1 = 1$, and $m_2 = \ell \cdot d$. The average load for the instance (sum of the all the job processing times divided by the sum of all the machine speeds) is upper bounded by 1. In addition the time to process any chain on the fastest machine is 1. However if $d > \ell$ the optimum schedule value is $\Omega(\ell)$ since *at most* ℓ machines can be simulteneously busy. The key insight we obtain from the above example is that the amount of parallelism in an instance restricts the number of machines that can be used. We capture this insight in our lower bound in a simple way. We view the precedence relations between the jobs as a weighted poset where the weight of an element of the poset is the the processing time of the associated job. We will further assume that we have the transitive closure of the poset. We need a few definitions.

Definition 2 *A* chain decomposition *$\mathcal{P}$ of a set of precedence constrained jobs is a partition of the associated poset into an ordered collection of chains $(P_1, P_2, \ldots, P_r)$. A* maximal chain decomposition *is one in which P_1 is a longest chain and $(P_2, \ldots, P_r)$ is a maximal chain decomposition of the poset with elements of P_1 removed.*

Note that maximal chain decompositions are ordered. We will be using the fact that $|P_1| \geq |P_2| \geq \ldots \geq |P_r|$.

Definition 3 *Let $\mathcal{P} = (P_1, P_2, \ldots, P_r)$ be a chain decomposition of the given set of jobs. We define a quantity $L_{\mathcal{P}}$ associated with $\mathcal{P}$ and machine speeds $s_1 \geq s_2 \geq \ldots \geq s_m$ as follows.*

$$L_{\mathcal{P}} = \max_{1 \leq j \leq \min\{r,m\}} \frac{\sum_{i=1}^{j} |P_i|}{\sum_{i=1}^{j} s_i}.$$

Note that in Definition 3 the index of summation is over the machines and not the speed classes. For a given $j \leq m$, the quantity $\frac{\sum_{i=1}^{j} |P_i|}{\sum_{i=1}^{j} s_i}$ is the average load of the first j chains from $\mathcal{P}$ on the fastest j machines. With the above definitions in place we are ready to state and prove the new lower bound.

Theorem 2 *Let $\mathcal{P} = (P_1, \ldots, P_r)$ be any chain decomposition (in particular any* maximal *chain decomposition) of the precedence graph of the jobs. Let $AL = (\sum_{j=1}^{n} p_j)/(\sum_{i=1}^{m} s_i)$ denote the average load. Then*

$$C^*_{\max} \geq \max\{AL, L_{\mathcal{P}}\}.$$

Moreover the lower bound is valid for preemptive schedules as well.

Proof. The maximum *rate* at which the machines can process jobs is seen to be at most $\sum_{i=1}^{m} s_i$, which is achieved if and only if all machines are busy. Therefore, to finish all the jobs requires at least a time of $AL = (\sum_{j=1}^{n} p_j)/(\sum_{i=1}^{m} s_i)$. Hence we obtain that $C^*_{\max} \geq AL$.

We will show that for $1 \leq j \leq m$ the following is true

$$C^*_{\max} \geq \frac{\sum_{i=1}^{j} |P_i|}{\sum_{i=1}^{j} s_i}.$$

This will prove the theorem. Consider the first j chains. Suppose our input instance was modified to have only the jobs in the first j chains. It is easy to see that a lower bound for this modified instance is a lower bound for the original instance. Since it is possible to execute only one job from each chain at any time instant, only the fastest j machines are relevant for this modified instance. The expression $(\sum_{i=1}^{j} |P_i|)/(\sum_{i=1}^{j} s_i)$ is simply the average load for the modified instance, which as we observed before, is a lower bound. Since the average load is also a lower bound for preemptive schedules, the claimed lower bound applies even if preemptions are allowed. □

Horvath, Lam, and Sethi [5] proved that the above lower bound gives the optimal schedule length for preemptive scheduling of chains on uniformly related machines. The idea of extending their lower bound to general precedence graphs using maximal chain decompositions is natural but does not appear to have been effectively used before.

Theorem 3 *A maximal chain decomposition can be computed in $O(n^3)$ time. If all p_j are the same, the running time can be improved to $O(n^2\sqrt{n})$.*

Proof. It is necessary to find the transitive closure of the given graph of precedence constraints. This can be done in $O(n^3)$ time using a breadth first search (BFS) from each vertex. From a theoretical point of view this can be improved to $O(n^\omega)$ where $\omega \leq 2.376$ using fast matrix multiplication [3]. A longest chain in a weighted DAG can be found in $O(n^2)$ time using standard algorithms. Using this at most n times, a maximal chain decomposition can be obtained in $O(n^3)$ time. If all p_j are the same (without loss of generality we can assume they are all 1), the length of a chain is the same as the number of vertices in the chain. We can use this additional structure to obtain an improved time bound as follows. We remove longest chains using the $O(n^2)$ algorithm as long as the longest chain length is at least $\sqrt{n}$. The total running time for this phase of the algorithm is clearly bounded by $O(n^2\sqrt{n})$. Once the length of the longest chain falls below $\sqrt{n}$ we run a different algorithm that is outlined in the proof of Lemma 1 below. That algorithm computes a maximal chain decomposition for unit weight jobs in $O(n^2 \cdot d)$ time where d is the maximum chain length. Thus if $d \leq \sqrt{n}$ we obtain an $O(n^2\sqrt{n})$ time for the second phase. Combining the two phases gives an algorithm with the desired bound. □

Lemma 1 *Let G be a DAG with all $p_j = 1$ and the longest chain length bounded by d. Given the transitive closure of G, there is an algorithm to compute a maximal chain decomposition in time $O(n^2 \cdot d)$.*

Proof. We partition the vertices into $(d+1)$ layers $L_0, L_1, \ldots, L_d$. Layer L_i is the set of all vertices v such that the longest chain ending at v is of length i. Let $\ell(v)$ denote the layer of a vertex v. For each vertex v we maintain its predecessors in $(d+1)$ classes corresponding to the layer to which they belong. Given the transitive closure it is easy to construct this partition and the predecessor classes in $O(n^2)$ time. Given a layered representation we can find a longest chain in $O(d)$ time by taking an arbitrary vertex in the highest numbered non-empty layer and walking down the layers looking for predecessors. Once we find a longest chain we remove the vertices in the chain and all the edges incident on them. We update the layered data structure and repeat the process. The update happens as follows. For each edge removed we change the predecessor structure of the vertex incident on it. Let S be the vertices that are incident on the edges removed. We examine vertices in S in increasing order of $\ell(v)$. We first update $\ell(v)$ to its new value. If $\ell(v)$ does not change by the removal of the chain we remove v from S. If $\ell(v)$ is reduced, we examine all the successors of v, update their predecessor data structure and add them to S. This takes time proportional to the out-degree of v. We continue this process as long as S is non-empty.

We analyze the running time as follows. The time to find the chain, remove the vertices and the associated edges, and form the initial set S can be amortized to total number of edges removed. Thus this time is bounded over all by $O(n^2)$. The time to update the layer information is amortized as follows. We examine the successors of a vertex v only if $\ell(v)$ is reduced. Since $\ell(v)$ can change at most $(d+1)$ times the total time is bounded by $(d+1)\sum_v \deg(v)$ which is $O(n^2 \cdot d)$. □

4 The Approximation Algorithm

The approximation algorithm we develop in this section is based on the maximal chain decompositions defined in the previous section. As mentioned in Section 2, our algorithm produces an assignment of jobs to speeds. Then we use the speed based list scheduling of [2] with the job assignment produced by our algorithm.

Algorithm Chain-Alloc is described in Figure 1. Based on Theorem 2, it first computes a lower bound B on the optimum schedule length. Then it orders the chains by non-increasing lengths and greedily allocates the chains to speeds such that no speed is loaded by more than four times the lower bound. We now prove several properties of the described allocation. Recall that D_i is the average load on a machine in speed class i.

1. **compute** a maximal chain decomposition of the jobs $\mathcal{P} = (P_1, \ldots, P_r)$.
2. **set** $B = \max\{AL, L_{\mathcal{P}}\}$.
3. **set** $\ell = 1$.
4. **for** $i = 1$ to K **do**
   (a) **let** $t \leq r$ be the maximum index such that $\sum_{\ell \leq j \leq t} |P_j|/(m_i \overline{s}_i) \leq 4B$.
   (b) **assign** jobs in chains $P_\ell, \ldots, P_t$ to speed i.
   (c) **set** $\ell = t + 1$. If $\ell > r$ **return**.
5. **return**.

Figure 1: Algorithm Chain-Alloc

Lemma 2 *Let $P_{\ell(u)}, \ldots, P_r$ be the chains remaining when Chain-Alloc considers speed u in step 4 of the algorithm. Then*

1. $|P_{\ell(u)}|/\overline{s}_u \leq 2B$ and

2. Either $P_{\ell(u)}, \ldots, P_r$ are allocated to speed u or *$D_u > 2B$.*

Proof. We prove the above assertions by induction on u. Consider the base case when $u = 1$ and $\ell(1) = 1$. From the definition of $L_{\mathcal{P}}$ it follows that $|P_1|/\overline{s}_1 \leq B$. Since P_1 is the longest chain, it also follows that $|P_j|/\overline{s}_1 \leq B$ for $1 \leq j \leq r$. Let t be the largest index such that P_t is assigned to $\overline{s}_1$. If $t = r$ we are done. If $t < r$, it must be the case that assigning $P_{(t+1)}$ to $\overline{s}_1$ machines increases their average load to more than $4B$. Since $|P_{(t+1)}|/\overline{s}_1 \leq B$, we conclude that $D_1 = \sum_{j=1}^{t} |P_j|/m_1\overline{s}_1 = \sum_{j=1}^{t+1} |P_j|/m_1\overline{s}_1 - |P_{t+1}|/m_1\overline{s}_1 > (4 - 1/m_1)B > 2B$.

Assume that the conditions of the lemma are satisfied for speeds $\overline{s}_1$ to $\overline{s}_{u-1}$ and consider speed $\overline{s}_u$. We will assume that $\ell(u) < r$ for otherwise there are no chains left to be assigned to $\overline{s}_u$ and the lemma is true by induction. We observe that the second condition follows from the first using an argument similar to the one used above for the base case. Therefore it is sufficient to prove the first condition. We also note that the two conditions imply that if $D_k > 2B$ then the number of chains assigned to speed $\overline{s}_k$ is at least m_k.

Suppose $|P_{\ell(u)}|/\overline{s}_u > 2B$. We will derive a contradiction. Let M^v denote the sum $\sum_{k=1}^{v} m_k$. Let $j = \ell(u)$ and let v be the index such that $M^{v-1} < j \leq M^v$. If $j > m$, no such index v exists and we set v to K, the slowest speed. If $j \leq m$, for convenience of notation we assume that $j = M^v$ simply by ignoring other machines of speed $\overline{s}_v$. We note that $P_{\ell(u)-1}$ is the last chain assigned to speed $\overline{s}_{u-1}$. We claim that $\ell(u) - 1 \geq M^{u-1}$ since for each speed $k < u$, $D_k > 2B$, and therefore the number of chains assigned to speed $\overline{s}_k$ is at least m_k. Therefore $j = \ell(u) > M^{u-1}$ and hence from the definition of v it follows that $v \geq u$.

From the definition of $L_{\mathcal{P}}$, AL, and B, we get the following two facts. If $j \leq m$ then $L_{\mathcal{P}} \geq (\sum_{i=1}^{j} |P_i|)/(\sum_{k=1}^{v} m_k \overline{s}_k)$. If $j > m$ then $AL \geq (\sum_{i=1}^{j} |P_i|)/(\sum_{k=1}^{K} m_k \overline{s}_k)$. Combining them we obtain the following,

$$\frac{\sum_{i=1}^{j} |P_i|}{\sum_{k=1}^{v} m_k \overline{s}_k} \leq \max\{L_{\mathcal{P}}, AL\} = B. \tag{1}$$

Since $|P_j|/\overline{s}_u > 2B$, it is the case that $|P_i|/\overline{s}_u > 2B$ for all $M^{u-1} < i \leq j$. This implies that

$$\sum_{M^{u-1} < i}^{j} |P_i| \; > \; 2B(j - M^{u-1})\overline{s}_u \; \geq \; 2B \sum_{k=u}^{v} m_k \overline{s}_k$$

Adding more terms to the left hand side of the above equation we get the following.

$$\sum_{i=1}^{j} |P_i| > 2B \sum_{k=u}^{v} m_k \overline{s}_k. \tag{2}$$

From the induction hypothesis it follows that speeds $\overline{s}_1$ to $\overline{s}_{u-1}$ have an average load greater than $2B$. From this we obtain

$$\sum_{i=1}^{j-1} |P_i| > 2B \sum_{k=1}^{u-1} m_k \overline{s}_k$$

and adding one more term to the left hand side of the above equation we get the following.

$$\sum_{i=1}^{j} |P_i| > 2B \sum_{k=1}^{u-1} m_k \overline{s}_k. \tag{3}$$

Combining Equations 2 and 3 we obtain the following.

$$\begin{aligned} 2\sum_{i=1}^{j} |P_i| \; &> \; 2B \sum_{k=1}^{u-1} m_k \overline{s}_k + 2B \sum_{k=u}^{v} m_k \overline{s}_k \\ &> \; 2B \sum_{k=1}^{v} m_k \overline{s}_k. \end{aligned} \tag{4}$$

Equation 4 contradicts Equation 1. □

Corollary 1 *If chain P_j is assigned to speed i, then $\frac{|P_j|}{\overline{s}_i} \leq 2B$.*

Corollary 2 *Algorithm Chain-Alloc allocates all chains.*

Proof. Suppose some chain is left unassigned after speed $\overline{s}_K$ is considered by the algorithm. Then by Lemma 2, $D_i > 2B$ for $1 \leq i \leq K$. This would imply that $\sum_{j=1}^{n} p_j \geq 2B \sum_{i=1}^{K} m_k \overline{s}_k$ violating the condition that $B \geq AL = \sum_{j=1}^{n} p_j / (\sum_{i=1}^{K} m_k \overline{s}_k)$. □

Lemma 3 *For $1 \leq k \leq K$, $D_k \leq 4C^*_{\max}$.*

Proof. Since $B \leq C^*_{\max}$ and the algorithm never loads a speed by more than an average load of $4B$, the bound follows. $\square$

Lemma 4 *For the job assignment produced by Chain-Alloc $C \leq 2KC^*_{\max}$.*

Proof. Let P be any chain. We will show that $\sum_{j \in P} p_j/\overline{s}_{k(j)} \leq 2KC^*_{\max}$ where $k(j)$ is the speed to which job j is assigned. Let A_i be the set of jobs in P which are assigned to speed $\overline{s}_i$. Let P_{ℓ_i} be the longest chain assigned to speed $\overline{s}_i$ by the algorithm. We claim that $|P_{\ell_i}| \geq \sum_{j \in A_i} p_i$. This is because the jobs in A_i form a chain when P_{ℓ_i} is picked as the longest chain in the maximal chain decomposition. From Corollary 1 we know that $|P_{\ell_i}|/\overline{s}_i \leq 2B \leq 2C^*_{\max}$. Therefore it follows that

$$\sum_{j \in P} \frac{p_j}{\overline{s}_{k(j)}} = \sum_{i=1}^{K} \frac{\sum_{j \in A_i} p_j}{\overline{s}_i} \leq \sum_{i=1}^{K} |P_{\ell_i}|/\overline{s}_i \leq 2KC^*_{\max}.$$

$\square$

Theorem 4 *Using speed based list scheduling on the job assignment produced by Algorithm Chain-Alloc gives a $6K$ approximation where K is the number of distinct speeds. Furthermore the algorithm runs in $O(n^3)$ time. The running time can be improved to $O(n^2\sqrt{n})$ if all p_j are the same.*

Proof. From Lemma 3 we have $D_k \leq 4C^*_{\max}$ for $1 \leq k \leq K$ and from Lemma 4 we have $C \leq 2KC^*_{\max}$. Putting these two facts together, for the job assignment produced by the algorithm Chain-Alloc, speed based list scheduling gives the following upper bound by Theorem 1.

$$C_{\max} \leq C + \sum_{k=1}^{K} D_k \leq 2KC^*_{\max} + 4KC^*_{\max} \leq 6KC^*_{\max}.$$

Speed based list scheduling can be implemented in $O(n \log n)$ time. The running time is dominated by the time to compute a maximal chain decomposition. Theorem 3 gives the desired bounds. $\square$

Corollary 3 *There is an algorithm which runs in $O(n^3)$ time and gives an $O(\log m)$ approximation ratio for the problem of scheduling precedence constrained jobs on uniformly related machines to minimize makespan.*

Remark 1 *The maximal chain decomposition depends only on the jobs of the given instance and is independent of the machine environment. If a maximal chain decomposition is given, the algorithm Chain-Alloc and speed based list scheduling can be implemented in $O(n \log n)$ time.*

We note here that the leading constant in the LP based algorithm in [2] is better. We also observe that the above bound is based on our lower bound which is valid for preemptive schedules as well. Hence our approximation ratio is also valid for preemptive schedules. In [2] it is shown that the lower bound provided by the LP relaxation is a factor of $\Omega(\log m/\log\log m)$ away from the optimal. Surprisingly it is easy to show using the same example as in [2] that our lower bound from Section 3 is also a factor of $\Omega(\log m/\log\log m)$ away from the optimal.

Theorem 5 *There are instances where the lower bound given in Theorem 2 is* $\Omega(\log m/\log\log m)$ *factor away from the optimal schedule value.*

Proof. The proof of Theorem 3.3 in [2] provides the instance and it is easily verified that *any* maximal chain decomposition of that instance is a factor of $\Omega(\log m/\log\log m)$ away from the optimal. □

4.1 Release Dates

Now consider the scenario where each job j has a release date r_j before which it cannot be processed. By a general result of Shmoys, Wein, and Williamson [10] an approximation algorithm for the problem of minimizing makespan without release dates can be transformed to one with release dates losing only a factor of 2 in the process. Therefore we obtain the following.

Theorem 6 *There is an* $O(\log m)$ *approximation algorithm for scheduling to minimize makespan of jobs with precedence constraints and release dates on uniformly related machines* ($Q|prec, r_j|C_{\max}$) *that runs in time* $O(n^3)$.

4.2 Scheduling Chains

In this subsection we show that Chain-Alloc followed by speed based list scheduling gives a constant factor approximation if the precedence constraints are induced by a collection of chains. We first observe that any maximal chain decomposition of a collection of chains is simply the collection itself. The crucial observation is that the algorithm Chain-Alloc allocates all jobs of any chain to the same speed class. The two observation together imply that there are no precedence relations between jobs allocated to different speeds. This allows us to obtain a stronger version of Theorem 1 where we can upper bound the makespan obtained by speed based list scheduling as $C_{\max} \leq C + \max_{1\leq k\leq K} D_k$. Further we can bound C as $\max_{1\leq i\leq r} |P_i|/\overline{s}_{k(i)}$ where chain P_i is allocated to speed $k(i)$. From Corollary 1 it follows that $C \leq 2B$. Lemma 3 implies that $\max_{1\leq k\leq K} D_k \leq 4B$. Combining the above observations yields a 6 approximation.

Theorem 7 *There is a* 6 *approximation for the problem* $Q|chains|C_{\max}$ *and a* 12 *approximation for the problem* $Q|chains, r_j|C_{\max}$.

Computing the maximal chain decomposition of a collection of chains is trivial and the above algorithm can be implemented in $O(n \log n)$ time.

5 Concluding Remarks

It is known that the problem of minimizing makespan is hard to approximate to within a factor of 4/3 even if all machines have the same speed [8]. This is the best known hardness of approximation for the multiple speed case too while the best upper bound we have is $O(\log m)$. Improving the gap is an obvious and very interesting research problem. We conjecture that there is a $O(1)$ approximation algorithm for the problem $Q|prec|C_{\max}$ and even for the more general problem of minimizing sum of weighted completion times $(Q|prec|\sum w_j C_j)$.

Acknowledgments

We thank Monika Henzinger for simplifying the proof of Lemma 2.

References

[1] C. Chekuri and M. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 383–393. Springer, 1998.

[2] F. Chudak and D. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 581–590, 1997.

[3] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–80, 1990.

[4] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Tech. J.*, 45:1563–81, 1966.

[5] E. Horvath, S. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24(1):32–43, 1977.

[6] J. Jaffe. Efficient scheduling of tasks without full use of processor resources. *Theoretical Computer Science*, 26:1–17, 1980.

[7] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S. C. Graves etal, editor, *Handbooks in OR & MS*, volume 4, pages 445–522. Elsevier Science Publishers, 1993.

[8] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[9] J. W. S. Liu and C. L. Liu. Bounds on scheduling algorithms for heterogeneous computing systems. In J. L Rosenfeld, editor, *Information Processing 74*, pages 349–353. North-Holland, 1974.

[10] D. Shmoys, J. Wein, and D. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24:1313–31, 1995.