An Efficient Approximation Algorithm for Minimizing Makespan on Uniformly Related Machines

CHANDRA CHEKURI[1]* and MICHAEL BENDER[2]

[1] Computer Science Department
Stanford University
chekuri@cs.stanford.edu

[2] Division of Applied Sciences
Harvard University
bender@deas.harvard.edu

Abstract. We give a new efficient approximation algorithm for scheduling precedence constrained jobs on machines with different speeds. The setting is as follows. There are n jobs $1, \ldots, n$ where job j requires p_j units of processing. The jobs are to be scheduled on a set of m machines. Machine i has a speed s_i; it takes p_j/s_i units of time for machine i to process job j. The precedence constraints on the jobs are given in the form of a partial order. If $j \prec k$, processing of k cannot start until j's execution if finished. Let C_j denote the completion time of job j. The objective is to find a schedule to minimize $C_{\max} = \max_j C_j$, conventionally called the *makespan* of the schedule. We consider non-preemptive schedules where each job is processed on a single machine with no preemptions. Recently Chudak and Shmoys [1] gave an algorithm with an approximation ratio of $O(\log m)$ significantly improving the earlier ratio of $O(\sqrt{m})$ due to Jaffe [7]. Their algorithm is based on solving a linear programming relaxation of the problem. Building on some of their ideas, we present a combinatorial algorithm that achieves a similar approximation ratio but runs in $O(n^3)$ time. In the process we also obtain a constant factor approximation algorithm for the special case of precedence constraints induced by a collection of chains. Our algorithm is based on a new lower bound which we believe is of independent interest. By a general result of Shmoys, Wein, and Williamson [10] our algorithm can be extended to obtain an $O(\log m)$ approximation ratio even if jobs have release dates.

1 Introduction

The problem of scheduling precedence constrained jobs on a set of identical parallel machines to minimize makespan is one of the oldest problems for which

* Supported primarily by an IBM Cooperative Fellowship. Remaining support was provided by an ARO MURI Grant DAAH04-96-1-0007 and NSF Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

approximation algorithms have been devised. Graham [4] showed that a simple list scheduling gives a ratio of 2 and it is the best known algorithm till date. We consider a generalization of this model in which machines have different speeds. In the scheduling literature such machines are called *uniformly related*. We formalize the problem below. There are n jobs $1, \ldots, n$, with job j requiring processing of p_j units. The jobs are to be scheduled on a set of m machines. Machine i has a speed factor s_i. Job j with a processing requirement p_j takes p_j/s_i time units to run on machine i. Let C_j denote the completion time of job j. The objective is to find a schedule to minimize $C_{\max} = \max_j C_j$ 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 [5] where problems are classified in the $\alpha|\beta|\gamma$ notation, 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 the approximation guarantee depends on the ratio of the largest to the smallest speed. This ratio could be arbitrarily large even for a small number of machines. The first algorithm to have a bound independent of the speeds was given by Jaffe [7]. He showed that list scheduling restricted to the set of machines with speeds that are within a factor of $1/\sqrt{m}$ of the fastest machine speed results in an $O(\sqrt{m})$ bound. More recently, Chudak and Shymoys [1] improved the ratio considerably and gave an algorithm which has a guarantee of $O(\log m)$. At a more basic level their algorithm has a guarantee of $O(K)$ where K is the number of *distinct* speeds. The above mentioned algorithm relies on solving a linear programming relaxation and uses the information obtained from the solution to allocate jobs to processors. We present a new algorithm which finds an allocation without solving a linear program. The ratio guaranteed by our algorithm is also $O(\log m)$ but is advantageous for the following reason. Our algorithm runs in $O(n^3)$ time and is combinatorial, hence is more efficient than the algorithm in [1]. Further, the analysis of our algorithm relies on a new lower bound which is very natural, and might be useful in other contexts. In addition we show that our algorithm achieves a constant factor approximation when the precedence constraints are induced by a collection of chains. We remark here that our work was inspired by, and builds upon the ideas in [1].

The rest of the paper is organized as follows. Section 2 contains some of the ideas from the paper of Chudak and Shmoys [1] that are useful to us. We present our lower bound in Section 3, and give the approximation algorithm and the analysis in Section 4.

2 Preliminaries

We summarize below the basic ideas in the work of Chudak and Shmoys [1]. Their main result is an algorithm which gives a ratio of $O(K)$ for the problem of $Q|prec|C_{\max}$ where K is the number of distinct speeds. They also show how to reduce the general case 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 while losing only a constant factor in the approximation ratio. Using this observation, we will restrict ourselves to the case where we have K distinct speeds.

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 $j_1 \prec j_2 \ldots \prec j_r$ such that a machine is idle only when one of the jobs in the above chain is being processed. The time spent processing the above chain is a lower bound on the optimal makespan. In addition, the measure of the time instants during which all machines are busy is also a lower bound by 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 [1], the difficulty is that the time spent in processing the chain identified from the list scheduling analysis, is not a lower bound. The only claim that can be made is that the processing time of any chain on the fastest machine is a lower bound. However the jobs in the chain guaranteed by the list scheduling analysis do not necessarily run on the fastest machine. Based on this observation, the algorithm in [1] tries to find 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 [1] for sake of continuity and convenience. Recall that we have K distinct speeds. Let m_k be the number of machines with speed $\overline{s}_k$, $k = 1, \ldots, K$, where $\overline{s}_1 > \ldots > \overline{s}_K$. Let M_u^v denote the sum $\sum_{k=u}^{v} m_k$. 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 which are totally ordered by the precedence constraints. Let $\mathcal{P}$ be the set of all chains induced by the precedence constraints. We compute a quantity C defined by the following equation.

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

A natural variant of list scheduling called speed based list scheduling is developed in [1] which is constrained to schedule according to the speed assignments of the jobs. In classical list scheduling, the first available job from the list is scheduled as soon as a machine is free. In speed based list scheduling, an available job is scheduled on a free machine provided the speed of free machine matches the speed assignment of the job. The following theorem is from [1]. The analysis is a simple generalization of Graham's analysis of list scheduling.

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

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

The authors of [1] use a linear programming relaxation of the problem to obtain a job assignment that satisfies the two conditions: $\sum_{k=1}^{K} D_k \leq (K + \sqrt{K})C^*_{\max}$, and $C \leq (\sqrt{K}+1)C^*_{\max}$, where $C^*_{\max}$ is the optimal makespan. Combining these with Theorem 1 gives them an $O(K)$ approximation. We will show how to 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 implies that the maximum chain length (a chain's length is sum of processing times of jobs in it) divided by the fastest speed is a lower bound. However it is easy to generate examples where the maximum of this bound and the average load is $O(1/m)$ times the optimal. We describe the general nature of such examples to motivate our new bound. Suppose we have two speeds with $\overline{s}_1 = D$ and $\overline{s}_2 = 1$. The precedence constraints between the jobs are induced by a collection of $\ell > 1$ chains, each of the same length D. Suppose $m_1 = 1$, and $m_2 = \ell \cdot D$. The average load can be seen to be upper bounded by 1. In addition the time to process any chain on the fastest processor is 1. However if $D \gg \ell$ it is easy to observe that the optimal is $\Omega(\ell)$ since *only* ℓ machines can be busy at any time instant. 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 need a few definitions to formalize the intuition. We view the precedence relations between the jobs as a weighted poset where each element of the poset has a weight associated with it that is the same as the processing time of the associated job. We will further assume that we have the transitive closure of the poset.

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.*

Definition 2. *A* chain decomposition $\mathcal{P}$ *of a set of precedence constrained jobs is an partition of the poset in to a 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.*

Though we define a maximal chain decomposition as a set of chains, we will implicitly assume that it is an *ordered* set, that is $|P_1| \geq |P_2| \geq \ldots |P_r|$.

Definition 3. *Let* $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ *be any maximal chain decomposition of the precedence graph of the jobs. We define a quantity called* $L_{\mathcal{P}}$ *associated with* $\mathcal{P}$ *as follows.*

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

With the above definitions in place we are ready to state and prove the new lower bound.

Theorem 2. *Let* $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ *be any maximal chain decomposition of the precedence graph of the jobs. Let* $AL = (\sum_{j=1}^{n} p_i)/(\sum_{i=1}^{m} s_i)$ *which represents the average load. Then*

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

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

Proof. It is easy to observe that $C^*_{\max} \geq AL$. We will show the following for $1 \leq j \leq m$

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

which 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 the preemptive case, the claimed lower bound applies even if preemptions are allowed.

Horvath, Lam, and Sethi [6] 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 precedences 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 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 [2]. 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. It is possible to use this additional structure to obtain a maximal chain decomposition in $O(n^2\sqrt{n})$ time. We omit the details.

4 The approximation algorithm

The approximation algorithm we develop in this section will be based on the maximal chain decompositions defined in the previous section. As mentioned in Section 2, we will describe an algorithm to produces an assignment of jobs to speeds at which they will be processed. Then we use the speed based list scheduling of [1] with the job assignment produced by our algorithm.

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

Fig. 1. Algorithm Chain-Alloc

The algorithm in Figure 1 computes a lower bound B on the optimal using Theorem 2. It then orders the chains in non-increasing lengths and 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 above described allocation which leads to the approximation guarantee of the algorithm.

Lemma 1. *Let $P_{\ell(u)}, \ldots, P_r$ be the chains remaining when Chain-Alloc considers speed u in step 3 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 follows that $|P_j|/\overline{s}_1 \leq B$ for $1 \leq j \leq r$. Let t be the last chain allocated to $\overline{s}_1$. If $t = r$ we are done. If $t < r$, it must be the case that adding $P_{(t+1)}$ increases the average load on $\overline{s}_1$ to more than $4B$. Since $P_{(t+1)}/\overline{s}_1 \leq B$, we conclude that $D_1 = \sum_{j=1}^{t} |P_j|/m_1\overline{s}_1 > 3B > 2B$.

Assume that the conditions of the lemma are satisfied for speeds s_1 to s_{u-1} and consider speed s_u. We will assume that $\ell(u) < r$ for otherwise there is nothing to prove. 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. Suppose $|P_{\ell(u)}|/\overline{s}_u > 2B$. We will derive a contradiction later. Let $j = \ell(u)$ and let v be the index such that $M_1^{v-1} < j \leq M_1^v$ (recall that $M_1^v = \sum_{k=1}^{v} m_k$). If $j > m$, no such index exists and we set v to K, the slowest speed. If $j \leq m$, for convenience of notation we assume that $j = M_1^v$ simply by ignoring other machines of speed $\overline{s}_v$. It is easy to see that $v \geq u$ and $j > M_1^{u-1}$. From the definition of $L_{\mathcal{P}}$, AL, and B, we get the following. 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 $AL \geq (\sum_{i=1}^{j} |P_i|)/(\sum_{k=1}^{K} m_k\overline{s}_k)$. In either case we obtain the fact that

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

Since $|P_j|/\overline{s}_u > 2B$, it must be the case that $|P_i|/\overline{s}_u > 2B$ for all $M_1^{u-1} < i \leq j$. This implies that

$$\begin{aligned} \sum_{M_1^{u-1}<i}^{j} |P_i| &> 2B(j - M_1^{u-1})\overline{s}_u \\ &\geq 2B\sum_{k=u}^{v} m_k\overline{s}_k \\ \Rightarrow \sum_{i=1}^{j} |P_i| &> 2B\sum_{k=u}^{v} m_k\overline{s}_k \end{aligned} \tag{2}$$

The last inequality follows since we are summing up more terms on the left hand side. 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

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

$$\Rightarrow \sum_{i=1}^{j} |P_i| > 2B\sum_{k=1}^{u-1} m_k\overline{s}_k \tag{4}$$

Combining Equations 2 and 4 we obtain the following.

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

$$> 2B \sum_{k=1}^{v} m_k \overline{s}_k$$

$$\Rightarrow \sum_{i=1}^{j} |P_i| > B \sum_{k=1}^{v} m_k \overline{s}_k \tag{5}$$

Equation 5 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.*

Lemma 2. *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.

Lemma 3. *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 i. Let P_ℓ be the longest chain assigned to speed i by the algorithm. We claim that $|P_\ell| \geq \sum_{j \in A_i} p_i$. This is because the jobs in A_i form a chain when we picked P_ℓ to be the longest chain in the max chain decomposition. From Corollary 1 we know that $|P_\ell|/\overline{s}_i \leq 2B \leq 2C^*_{\max}$. Therefore it follows that

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

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 2 we have $D_k \leq 4C^*_{\max}$ for $1 \leq k \leq K$ and from Lemma 3 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.

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

It is easy to see that the speed based list scheduling can be implemented in $O(n^2)$ time. The running time is dominated by the time to do the maximum chain decomposition. Theorem 3 gives the desired bounds.

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

We remark here that the leading constant in the LP based algorithm in [1] 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 [1] 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 [1] 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 a factor of $\Omega(\log m / \log\log m)$ away from the optimal.*

Proof. The proof of Theorem 3.3 in [1] provides the instance and it is easily verified that *any* maximum 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 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 the problem $Q|prec, r_j|C_{\max}$ 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 give an informal proof in this version of the paper. 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 $C_{\max} \le C + \max_{1 \le k \le K} D_k$. Further we can bound C as $\max_{1 \le i \le r} |P_i| / \overline{s}_{k(i)}$ where chain P_i is allocated to speed $k(i)$. From Corollary 1 it follows that $C \le 2B$. Lemma 2 implies that $\max_{1 \le k \le K} D_k \le 4B$. 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 Conclusions

The main contribution of this paper is a simple and efficient $O(\log m)$ approximation to the scheduling problem $Q|prec|C_{\max}$. Chudak and Shmoys [1] provide similar approximations for the more general case when the objective function is the average weighted completion time ($Q|prec|\sum w_j C_j$) using linear programming relaxations. We believe that the techniques of this paper can be extended to obtain a simpler and combinatorial algorithm for that case as well. 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]. However, for the single speed case a 2 approximation is known, while the current ratio for the multiple speed case is only $O(\log m)$. Obtaining a constant factor approximation, or improving the hardness are interesting open problems.

Acknowledgments

We thank Monika Henzinger for simplifying the proof of Lemma 1.

References

1. F. Chudak and D. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1997.
2. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progression. *Proceedings of the 19th ACM Symposium on Theory of Computing*, 1–6, 1987.
3. M.R. Garey and D.S. Johnson. **Computers and Intractability: A Guide to the Theory of NP-completeness,** Freeman, San Francisco (1979).
4. R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Tech. J.*, 45:1563–81, 1966.
5. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* 5:287–326, 1979.
6. E. Horvath, S. Lam, and R. Sethi. A Level Algorithm for Preemptive Scheduling. *Journal of the ACM*, vol 24(1), 32–43, 1977.
7. J. Jaffe. Efficient scheduling of tasks without full use of processor resources. *Theoretical Computer Science*, 26:1–17, 1980.
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. Lui and C.L. Lui. Bounds on scheduling algorithms for heterogeneous computing systems. In J.L Rosenfeld (ed.), *Information Processing 74*, North-Holland, 349–353, 1974.
10. D. Shmoys, J. Wein, and D. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, vol 24, 1313–31, 1995.