Approximation Techniques for Average Completion Time Scheduling

C. Chekuri* — Stanford University
R. Motwani† — Stanford University
B. Natarajan‡ — Hewlett Packard Laboratories
C. Stein§ — Dartmouth College

Abstract

We consider the problem of nonpreemptive scheduling to minimize average (weighted) completion time, allowing for release dates, parallel machines, and precedence constraints. Recent work has led to constant-factor approximations for this problem, based on solving a preemptive or linear programming relaxation and then using the solution to get an ordering on the jobs. We introduce several new techniques which generalize this basic paradigm. We use these ideas to obtain improved approximation algorithms for one-machine scheduling to minimize average completion time with release dates. In the process, we obtain an optimal randomized on-line algorithm for the same problem that beats a lower bound for deterministic on-line algorithms. We consider extensions to the case of parallel machine scheduling, and for this we introduce two new ideas: first, we show that a preemptive one-machine relaxation is a powerful tool for designing parallel machine scheduling algorithms that simultaneously produce good approximations and have small running times; second, we show that a non-greedy "rounding" of the relaxation yields better approximations than a greedy one. We also prove a general theorem relating the value of one-machine relaxations to that of the schedules obtained for the original m-machine problems. This theorem applies even when there are precedence constraints on the jobs. We apply this result to precedence graphs such as in-trees, out-trees, and series-parallel graphs; these are of particular interest in compiler applications that partly motivated our work.

*Department of Computer Science, Stanford University, Stanford, CA 94305-9045. E-mail: chekuri@cs.stanford.edu. Supported by NSF Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

†Department of Computer Science, Stanford University, Stanford, CA 94305-9045. E-mail: rajeev@cs.stanford.edu. Supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership Award, an ARO MURI Grant DAAH04-96-1-0007, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

‡Hewlett Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304. E-mail: natarajan@hpl.hp.com.

§Department of Computer Science,Dartmouth College, Hanover, NH. E-mail: cliff@cs.dartmouth.edu. Research partly supported by NSF Award CCR-9308701. Most of this work was done while the author was visiting Stanford University

1 Introduction

We present new approximation techniques and results for *nonpreemptive* scheduling to minimize *average (weighted) completion time*. In this problem, we are given n jobs $J_1, \dots, J_n$, where job J_j has processing time p_j, release date r_j, and weight w_j. A feasible schedule S assigns jobs nonpreemptively[1] to m machines such that each job starts after its release date. Let C_j^S denote the completion time of job J_j in schedule S. The objective is to minimize the *weighted* completion time $\sum_j w_j C_j^S$; if all w_j are $1/n$, the objective becomes the *average* completion time. For the single machine case, if the release dates are 0 for all jobs, then the weighted completion time problem can be solved optimally in polynomial time [17]. We are interested in a more general setting with release dates and precedence constraints and multiple machines, any of which makes the problem $\mathcal{NP}$-hard [20]. Thus, we will consider approximation algorithms, or, in an on-line setting, competitive ratios.

An important motivation for studying weighted completion time, aside from its intrinsic theoretical interest, comes from application to compiler optimizations. Compile-time instruction scheduling is essential for effectively exploiting the fine-grained parallelism offered in pipelined, superscalar, and VLIW architectures (for example, see [13, 26]). Current research is addressing the issue of profile-based compiler optimization. In a recent paper, Chekuri, Johnson, Motwani, Natarajan, Rau, and Schlansker [3] show that weighted completion time is the measure of interest in profile-driven code optimization; some of our results are related to the heuristics described and empirically tested therein.

Recent work has led to constant-factor approximations for weighted completion time for a variety of these $\mathcal{NP}$-hard scheduling problems [21, 12, 2]. Most of these algorithms work by first constructing a relaxed solution, either a preemptive schedule or a linear-programming relaxation. These relaxations are used to obtain an ordering of the jobs, and then the jobs are list scheduled

[1] In a nonpreemptive schedule, each job runs uninterrupted on one machine from start to finish; in a preemptive schedule, a job may be interrupted or may switch machines at any time.

as per this ordering.

We introduce several new techniques which generalize this basic paradigm. We first use these to obtain improved approximation algorithms for one-machine scheduling to minimize average completion time with release dates. Our main result here is a $\frac{e}{e-1} \approx 1.58$-approximation algorithm. This algorithm can be turned into a randomized on-line algorithm with the same bound, where an algorithm is *on-line* if before time r_j it is unaware of J_j, but at time r_j it learns all the parameters of J_j. This randomized on-line algorithm is particularly interesting as it beats a lower bound for deterministic on-line algorithms [16], and matches a recent lower bound for randomized on-line algorithms [25].

We then consider extensions to parallel machine scheduling, and introduce two new ideas: first, we show that a preemptive *one-machine* relaxation is a powerful tool for designing parallel machine scheduling algorithms that simultaneously produce good approximations and have small running times; second, we show that a non-greedy "rounding" of the relaxation produces better approximations than simple list scheduling. In fact, we prove a general theorem relating the value of one-machine relaxations to that of the schedules obtained for the original m-machine problems. This theorem applies even when there are precedence constraints, yielding better approximations for precedence graphs such as in-trees, out-trees, and series-parallel graphs, which are of interest in compiler applications that partly motivated our work.

The bounds in this paper derive from proving bounds on the ratio between solutions to a nonpreemptive scheduling problem and a relaxed version of this problem. The bounds on this ratio all hold when both the original and relaxed problems have weights; however the relaxed problem with weights is typically not solvable exactly in polynomial time. Because of this, the performance ratios of our algorithms are not as good as those obtained directly with other techniques. But given future improvement in state of the art for 1-machine preemptive scheduling with release dates and/or precedence constraints, our results would imply better bounds for weighted completion time too.

We begin with a more detailed discussion of our results and their relation to earlier work.

One-Machine Scheduling with Release Dates. The first constant-factor approximation algorithm for a weighted completion time problem was the following 2-approximation algorithm of Phillips, Stein and Wein [21] for minimizing the average completion time on one machine with release dates. First, an optimal preemptive schedule P is found using the shortest remaining processing time (SRPT) algorithm [20] which at any time runs an available job that has the least processing time left; note that this is an on-line algorithm. Given P, the jobs are ordered by increasing completion times, C_j^P, and are scheduled according to that ordering, introducing idle time as necessary to account for release dates. A simple proof shows that each job J_j completes at time no later than $2C_j^P$, implying a 2-approximation. Two other 2-approximation algorithms have been discovered subsequently [16, 24], and it is also known that no deterministic on-line algorithm has approximation ratio better than 2 [16, 24]. This approximation technique has been generalized to many other scheduling problems, and hence finding better approximations for this basic problem is believed to be an important step towards improved approximations for more general problems.

Our main result here is a deterministic off-line algorithm for the basic problem that gives an $\frac{e}{e-1}$-approximation ($\frac{e}{e-1} \approx 1.58$). We also obtain an *optimal* randomized on-line algorithm(in the *oblivious* adversary model) with expected competitive ratio $\frac{e}{e-1}$, which beats the deterministic on-line lower bound. Our approach is based on what we call *α-schedules* (this notion was also used by [21] and [11] in a somewhat different manner). Given a preemptive schedule P and $\alpha \in [0,1]$, we define $C_j^P(\alpha)$ to be the time at which an α-fraction of J_j is completed. An *α-schedule* is a nonpreemptive schedule obtained by list scheduling jobs in order of increasing $C_j^P(\alpha)$, possibly introducing idle time to account for release dates. Clearly, an α-scheduler is an on-line algorithm; moreover, for $\alpha = 1$, the α-scheduler is exactly the algorithm of Phillips et al [21] and hence a 2-approximation. We show that for arbitrary α, an α-scheduler has a *tight* approximation ratio of $1 + 1/\alpha$. Given that $1 + 1/\alpha \geq 2$ for $\alpha \in [0,1]$, it may appear that this notion of α-schedulers is useless for obtaining ratios better than 2.

A key insight is that a worst-case instance with performance ratio $1 + 1/\alpha$ for one value of α is not a worst-case instance for many other values of α. This suggests that a randomized algorithm which picks α at random, and then behaves like an α-scheduler, may lead to an approximation better than 2. Unfortunately, we show that choosing α uniformly at random gives an expected approximation ratio of 2, and that this is tight. However, this leaves open the possibility that for any given instance I, there exists a choice $\alpha(I)$ for which the $\alpha(I)$-schedule yields an approximation better than 2. We refer to the resulting deterministic off-line algorithm as Best-α. Currently, we are unable to directly analyze the performance of Best-α.

It turns out, however, that the randomized on-line algorithm which chooses α to be 1 with probability $3/5$ and $1/2$ with probability $2/5$ has competitive ratio 1.8;

consequently, for any input I, either the 1-schedule or the $\frac{1}{2}$-schedule is no worse than an 1.8-approximation. We conjecture that even this restricted version of the Best-α algorithm is a 3/2-approximation algorithm. More significantly, the non-uniform choice in the randomized version suggests the possibility of defining randomized choices of α that may perform better than 1.8 while being easy to analyze. In fact, our main result here is that it is possible to define a distribution for α that yields a randomized on-line $\frac{e}{e-1}$-approximation algorithm implying that Best-α is an $\frac{e}{e-1}$-approximation algorithm. It should be noted that Best-α can be implemented in $O(n^2 \log n)$ time and all our other algorithms can be implemented in either $O(n)$ or $O(n \log n)$ time.

Our bounds are actually job-by-job, i.e., we produce a schedule N in which $E[C_j^N] \leq \frac{e}{e-1} C_j^P$ for all j where $E[C_j^N]$ is the expected completion time of J_j in the non-preemptive schedule. Thus, our conversion results generalize to *weighted* completion time. However, since for the weighted case even the preemptive scheduling problem is $\mathcal{NP}$-hard (given release dates), we must use an approximation for the relaxation. There is a 2-approximation for the preemptive case [12], which yields a 3.16-approximation for the nonpreemptive case. While this does not immediately improve earlier results, we believe that, as an off-line algorithm, Best-α actually has a performance ratio better than 1.58 when applied to structured preemptive schedules as opposed to arbitrary preemptive schedules. If this is true, or if there is a better preemptive approximation algorithm, our results will have bearing on the weighted case.

Independently, Goemans [9] has used some related ideas to design a 2-approximation algorithm for the problem of nonpreemptive scheduling on one machine so as to minimize the average weighted completion time. His algorithm is also a Best-α algorithm, but it works off of a linear programming relaxation rather than a preemptive schedule.

Scheduling Parallel Machines with Release Dates. The m-machine problem gives rises to two complications: the problem of computing an optimal m-machine preemptive schedule is $\mathcal{NP}$-hard [4], and the best known approximation ratio is 2 [21]; further, the conversion bounds from preemptive to nonpreemptive schedules are not as good. Chakrabarti et al [2] obtain a bound of 7/3 on the conversion from preemptive to nonpreemptive case, yielding a 14/3-approximation for scheduling on m machines with release dates. Several other algorithms don't use the preemptive schedule but use a linear programming relaxation. Phillips et al [21] gave the first such algorithm, a 24-approximation algorithm. This has been greatly improved to $4+\epsilon$ [11], $4 - \frac{1}{m}$ [12], and 3.5 [2]. Using a general on-line framework [2], one can obtain an algorithm with an approximation ratio of $2.89 + \epsilon$. Unfortunately, the algorithm with the best approximation is inefficient, as it uses the polynomial approximation scheme for makespan due to Hochbaum and Shmoys [15].

We give a new algorithm for this problem. First we introduce a different relaxation – a *preemptive one-machine relaxation*. More precisely, we maintain the original release dates and allow preemptions, but divide all the processing times by m. We then compute a one-machine schedule. The resulting completion time ordering is then used to generate a nonpreemptive m-machine schedule that is a 3-approximation. Our algorithm is not only on-line but is also efficient and runs in $O(n \log n)$ time. We then show that the approximation ratio can be improved by using the completion times from the relaxation in a *non-greedy* fashion. We call this algorithm Modified Greedy. The basic idea is that even though a job J_j appears first on the list, if there is a job J_k with $r_k < r_j$ and there is a significant amount of idle time in the schedule, we schedule J_k before J_j. We show that either Modified Greedy or list scheduling always yields an approximation bound no larger than 2.85. This improves on the approximation bounds of previous algorithms and gives a much smaller running time of $O(n \log n)$.

Extension to Precedence Constraints. We now consider the weighted completion time problem with precedence constraints; for the time being, assume that all release dates are 0, but note that our results apply unchanged for arbitrary release dates. For one-processor scheduling, minimizing the weighted completion time is $\mathcal{NP}$-hard for arbitrary precedence constraints [7, 19], although a 2-approximation is possible [12]. A polynomial-time solution is known when the precedence graph is a forest [14], or a series-parallel graph [19, 1]. For arbitrary m, the problem is $\mathcal{NP}$-hard even without precedence constraints, unless the weights are all equal (without release dates); on the other hand, the problem is strongly $\mathcal{NP}$-hard even when all weights are identical and the precedence graph is a union of chains [5]. An expected approximation ratio of $5.33 + \epsilon$ is achievable with release dates and precedence constraints [2].

We extend our techniques to obtain a fairly general algorithm for m-machine problems with precedence constraints, in addition to release dates and job weights. The idea is similar to that in our other algorithms: solve a one-machine preemptive relaxation, and apply an algorithm we call Delay List (similar to Modified Greedy), to get an m-machine nonpreemptive schedule. Since, in general, the one-machine preemptive relaxation is also $\mathcal{NP}$-hard, we would have to settle for a ρ-approximation for it; then, our algorithm would give a

$(2\rho+2)$-approximation for the m-machine case. While it does not, at present, improve the best known bounds for any scheduling problem, we feel that our general theorem is of independent interest and likely to find applications in the future. Further, our conversion algorithm has the advantage of being extremely simple and combinatorial. Note that in applications such as compilers [3], speed and simplicity are sometimes more important than getting the best possible ratio. Finally, our algorithm has a surprising property: it gives schedules that are good for both makespan and average completion time (Chakrabarti et al [2] also have shown the existence of such schedules).

We also show that a similar but simpler technique gives a 2-approximation if the precedence graph is an in-tree; note that the optimal one-processor schedule for in-trees is polynomial-time computable. The idea is to use list scheduling with a one-processor schedule as the "list," which is a natural heuristic for *makespan* [10].

2 One-Machine Scheduling with Release Dates

In this section, we present our results for one-machine scheduling with release dates to minimize average completion time. Let P be a preemptive schedule, and let C_i^P and C_i^α denote the completion time of J_i in P and in the nonpreemptive α-schedule derived from P, respectively. We begin by analyzing simple α-schedules. Techniques from [21, 12] are easily generalized to yield the following:

THEOREM 2.1. *For $\alpha \in [0,1]$, an α-schedule has $\sum_j C_j^\alpha \leq (1+1/\alpha)\sum_j C_j^P$, and this is tight.*

This theorem, in and of itself, always yields approximation bounds that are worse than 2.

We thus introduce a new fact that ultimately yields better algorithms. We will show that the *makespan* of an α-schedule is within a $(1+\alpha)$-factor of the makespan of the corresponding preemptive schedule; in fact, we will prove a stronger result in Lemma 2.2 below. Thus, the idle time introduced in the nonpreemptive schedules decreases as α is reduced from 1 to 0. On the other hand, the (worst-case) bound on the completion time of any specific job increases as α goes from 1 to 0. It is the balancing of these two effects that leads to better approximations. In the following discussion we do not assume that the preemptive schedule is the optimal preemptive schedule found using SRPT. In fact our results on converting preemptive schedules to nonpreemptive schedules apply in general but when we want to prove upper bounds on the performance ratio for total completion time, we assume that the preemptive schedule is the optimal preemptive schedule which is a lower bound on the optimal nonpreemptive schedule.

Let $S_i^P(\beta)$ denote the set of jobs which complete exactly β fraction of their processing time before C_i^P in the schedule P (note that J_i is included in $S_i^P(1)$). We overload notation by using $S_i^P(\beta)$ to also denote the sum of processing times of all jobs in the set $S_i^P(\beta)$; the meaning should be clear from the context. Let T_i be the total idle time in P before J_i completes.

The preemptive completion time of J_i can be written as the sum of the idle times and processing times of jobs that ran between C_i^P. This yields the following lemma:

LEMMA 2.1. *$C_i^P = T_i + \sum_{0<\beta\leq 1}\beta S_i^P(\beta)$.*

We next upper bound the completion time of a job J_i in the α-schedule.

LEMMA 2.2.

$$C_i^\alpha \leq T_i + (1+\alpha)\sum_{\beta\geq\alpha} S_i^P(\beta) + \sum_{\beta<\alpha}\beta S_i^P(\beta).$$

Proof. Let $J_1,\ldots,J_{i-1}$ be the jobs that run before J_i in the α schedule. We will give a procedure which converts the preemptive schedule into a schedule which (C1) jobs $J_1,\ldots,J_i$ run non-preemptively, (C2) the remaining jobs run preemptively, and (C3) the completion time of J_i obeys the bound given in the lemma. Since the actual C_i^α is no greater than the completion time of J_i in this schedule, the lemma will be proven.

Splitting up the second term in the bound from Lemma 2.1, we get the following equation:

$$C_i^P = T_i + \sum_{\beta<\alpha}\beta S_i^P(\beta) + \sum_{\beta\geq\alpha}\alpha S_i^P(\beta) + \sum_{\beta\geq\alpha}(\beta-\alpha)S_i^P(\beta).$$

Let $J^B = \cup_{j\leq i} J_j$ and $J^A = J - J^B$ We can interpret the four terms in the equation as (1) the idle time in the preemptive schedule before C_i^P, (2) the pieces of jobs in J^A that run before C_i^P, (3) the pieces of jobs in $J_j \in J^B$ that run before $C_j^P(\alpha)$, and (4) the pieces of jobs in J^B that run between $C_j^P(\alpha)$ and C_i^P. Let x_j be the β for which $J_j \in S_i^P(\beta)$. Then $\sum_{\beta\geq\alpha} S_i^P(\alpha)$ can be rewritten as $\sum_{J_j\in J^B}(x_j-\alpha)$ and observe that $x_j-\alpha$ is the amount of processing of job J_j that occurs between $C_j^P(\alpha)$ and C_i^P.

Now think of schedule P as an ordered list of pieces of jobs (with sizes). For each $J_j \in J^B$ modify the list by (1) removing all pieces of jobs that run between $C_j^P(\alpha)$ and C_i^P and (2) inserting a piece of size $x_j - \alpha$ at the point corresponding to $C_j^P(\alpha)$. In this schedule, we have pieces of size $x_j-\alpha$ of jobs $J_1,\ldots,J_i$ in the correct order (plus other pieces of jobs). Now convert this ordered list back into a schedule by scheduling the pieces in the order of the list, respecting release dates. We claim that job i still completes at time C_i^P. To see this observe that the total processing time before C_i^P remains unchanged and that other than the pieces of size $x_j - \alpha$, we only moved pieces later in time, so no additional idle time need be introduced.

Now, for each job $J_j \in J^B$, extend the piece of size $x_j - \alpha$ to one of size p_j by adding $p_j - (x_j - \alpha)$ units of processing and replace the pieces of J_j that occur earlier, of total size αp_j, by idle time. We now have a schedule in which $J_1, \ldots J_i$ are each scheduled non-preemptively for p_j units of time and in which the completion time of J_i is

$$\begin{aligned} & C_i^P + \sum_{J_j \in B^j} (p_j - (x_j - \alpha)) \\ = \;& C_i^P + \sum_{\beta \geq \alpha} (1 - \beta + \alpha) S_i^P \\ = \;& T_i + (1+\alpha) \sum_{\beta \geq \alpha} S_i^P(\beta) + \sum_{\beta < \alpha} \beta S_i^P(\beta), \end{aligned}$$

where the first equality just comes from reindexing terms by β instead of j, and the second comes from plugging in the value of C_i^P. To complete the proof, we observe that the remaining pieces in the schedule are all from jobs in J^A, and we have thus met the conditions (C1), (C2) and (C3) above. ∎

Although we will not use it directly, applying Lemma 2.2 to the last job yields:

Corollary 2.1. *The makespan of the α-schedule is at most $(1+\alpha)$ times the makespan of the corresponding preemptive schedule, and there are instances for which this bound is tight.*

Having analyzed completion times as in Lemma 2.2, we see that the approximation ratio is going to depend on the distribution of the different sets $S_i^P(\beta)$. To avoid the worst case α, we choose α randomly, according to some probability distribution. We now give a very general bound on this algorithm.

Lemma 2.3. *Suppose α is chosen from a probability distribution over $[0,1]$ with a density function f. Then, the $E[\sum_i C_i^\alpha] \leq (1+\delta) \sum_i C_i^P$, where*

$$\delta = \max_{0<\beta\leq 1} \int_0^\beta \frac{1+\alpha-\beta}{\beta} f(\alpha) \mathrm{d}\alpha.$$

Proof. We will show that the expected completion time of any job J_i is within $(1+\delta)$ of its preemptive completion time. From Lemma 2.2 it follows that for any given α,

$$C_i^\alpha \;\leq\; T_i + (1+\alpha) \sum_{\beta \geq \alpha} S_i^P(\beta) + \sum_{\beta < \alpha} \beta S_i^P(\beta).$$

Therefore, when α is chosen according to f, the expected completion time of J_i is $E[C_i^\alpha] = \int_0^1 f(\alpha) C_i^\alpha \mathrm{d}\alpha$ is bounded by

$$T_i + \int_0^1 f(\alpha) \left((1+\alpha) \sum_{\beta \geq \alpha} S_i^P(\beta) + \sum_{\beta < \alpha} \beta S_i^P(\beta) \right) \mathrm{d}\alpha$$

since T_i is independent of α. We now bound the second term in the above expression

$$\begin{aligned} & \int_0^1 f(\alpha) \left((1+\alpha) \sum_{\beta \geq \alpha} S_i^P(\beta) + \sum_{\beta < \alpha} \beta S_i^P(\beta) \right) \mathrm{d}\alpha \\ = \;& \sum_{0<\beta\leq 1} S_i^P(\beta) \left(\int_0^\beta (1+\alpha) f(\alpha) \mathrm{d}\alpha + \int_\beta^1 \beta f(\alpha) \mathrm{d}\alpha \right) \\ = \;& \sum_{0<\beta\leq 1} \beta S_i^P(\beta) \left(1 + \int_0^\beta \frac{1+\alpha-\beta}{\beta} f(\alpha) \mathrm{d}\alpha \right) \\ \leq \;& \left(1 + \max_{0<\beta\leq 1} \int_0^\beta \frac{1+\alpha-\beta}{\beta} f(\alpha) d\alpha \right) \sum_{0<\beta\leq 1} \beta S_i^P(\beta) \\ \leq \;& (1+\delta) \sum_{0<\beta\leq 1} \beta S_i^P(\beta). \end{aligned}$$

It follows that

$$E[C_i^\alpha] \leq T_i + (1+\delta) \sum_{0<\beta\leq 1} \beta S_i^P(\beta) \leq (1+\delta) C_i^P.$$

Using linearity of expectations, it is easy to show that the expected total completion time of the schedule is within $(1+\delta)$ of the preemptive schedule's total completion time. ∎

With the above lemma in place, we can simply choose different pdf's to establish different bounds.

Theorem 2.2.

1. *For α chosen uniformly, the expected approximation ratio is at most 2.*
2. *For α chosen to be 1 with probability $3/5$ and $1/2$ with probability $2/5$, the expected approximation ratio is at most 1.8.*
3. *For α chosen with the density function $f(\alpha) = \frac{e^\alpha}{e-1}$, the approximation ratio is at most $\frac{e}{e-1} \approx 1.58$.*

Theorem 2.2.3 immediately yields an on-line randomized algorithm. To get a deterministic algorithm with the same approximation ratio, we observe that given a preemptive SRPT schedule we can efficiently determine the best possible choice of α. The SRPT schedule has at most n preemptions and hence at most $2n$ job pieces. This implies that there are at most $2n$ "combinatorially distinct" values of α for a given preemptive schedule.

COROLLARY 2.2. *There is a randomized on-line algorithm with an expected competitive ratio* 1.58 *for the problem of minimizing total completion time in the presence of release dates. Further, there is a deterministic off-line algorithm,* BEST-α*, which is a* 1.58*-approximation algorithm for the same problem and runs in* $O(n^2 \log n)$ *time.*

We omit the proof of the following negative results for the various algorithms. Note that, after learning of our results, Stougie and Vestjens [25] improved the lower bound for randomized on-line algorithms to $e/(e-1)$. This implies that our randomized on-line algorithm is *optimal.*

THEOREM 2.3.

1. *For* α *chosen uniformly, the expected approximation ratio is at least* 2.
2. *For* α *chosen to be* 1 *with probability* 3/5 *and* 1/2 *with probability* 2/5*, the expected approximation ratio is at least* 1.5.
3. *For the* BEST-α *algorithm, the approximation ratio is at least* 4/3.
4. *Any randomized on-line algorithm has expected competitive ratio at least* 4/3.

3 Parallel Scheduling with Release Dates

We now give an improved algorithm for scheduling on m machines with release dates to minimize total (or, average) completion time. Our algorithm does not use linear programming, or slow dynamic programming, and is a 2.85-approximation algorithm. It introduces the notion of a one-machine preemptive relaxation, and of using an algorithm other than list scheduling for scheduling the jobs nonpreemptively.

3.1 Obtaining a 3-approximation. We introduce the one-machine preemptive relaxation by using it to give a weaker 3 approximation. The algorithm is not only simple but is also on-line.

Given an instance I for nonpreemptive scheduling on m machines, we define a relaxation I1 in which we have processing times $p'_j = p_j/m$ and release dates $r'_j = r_j$. We call a relaxation *valid* if it maps feasible m-machine schedules for I to feasible one-machine schedules for I1 such that the average completion time of a schedule for I1 is no more than that of the corresponding schedule for I. We omit the proof of Lemma 3.1.

LEMMA 3.1. *An instance I1 is a valid one-machine preemptive relaxation of the m-machine problem I.*

Given a solution to $P1$, we form N by ordering jobs by C_j^{P1} and then scheduling them nonpreemptively in that order, respecting release dates. Let C_j^* be the completion time of J_j in an optimal schedule for I.

LEMMA 3.2. $\sum_j C_j^N \leq (3 - \frac{1}{m}) \sum_j C_j^*$.

Proof. We first focus on a particular job J_j. For convenience, we assume that the jobs are ordered according to their completion times in $P1$. Thus J_j is the jth job to complete in $P1$. We now derive three lower bounds on C_j^{P1}. First, we have the trivial bound $C_j^{P1} \geq r'_j + p'_j$. Further, C_j^{P1} is at least as big as the processing times of the jobs that precede it. Therefore,

$$C_j^{P1} \geq \sum_{k=1}^{j} p'_k = \sum_{k=1}^{j} \frac{p_k}{m}. \tag{3.1}$$

Let $r_j^{\max} = \max_{1 \leq k \leq j} r'_k$ be the latest release date among jobs that complete before j; then, $C_j^{P1} \geq r_j^{\max}$.

Now consider the list schedule N. Clearly by time $r_j^{\max}$ all jobs $J_1, \ldots J_j$ have been released. Even if no job starts before time $r_j^{\max}$, by standard makespan arguments J_j will complete by

$$\begin{aligned} C_j^N &\leq r_j^{\max} + \sum_{k=1}^{j-1} \frac{p_k}{m} + p_j \\ &\leq C_j^{P1} + C_j^{P1} + p_j \left(1 - \frac{1}{m}\right), \qquad (3.2) \end{aligned}$$

where the second inequality follows from (3.1) and $C_j^{P1} \geq r_j^{\max}$ above. Summing (3.2) over all jobs, we get a total completion time of

$$\sum_j C_j^N \leq 2 \sum_j C_j^{P1} + \left(1 - \frac{1}{m}\right) \sum_j p_j. \tag{3.3}$$

By Lemma 3.1, $\sum_j C_j^{P1} \leq \sum_j C_j^*$, and trivially the optimal solution to I must have total completion time $\sum_j C_j^* \geq \sum p_j$, so this algorithm is a 3-approximation. ∎

The running time is just the time to run SRPT on the one machine relaxation and the time to list schedule, for a total of $O(n \log n)$. This algorithm can be made on-line by simulating the preemptive schedule and adding a job to the list when it completes in the preemptive schedule.

3.2 Improvement to a 2.85-approximation. We now improve the approximation bound to 2.85, which improves the the earlier ratio of $2.89 + \epsilon$ [2]. We observe that it is possible that, when we order the jobs by C_j^{P1}, there are pairs of jobs j and k with $C_j^{P1} > C_k^{P1}$ and

$r_j < r_k$. Thus J_k precedes J_j on the list and J_j has to wait for J_k to be released before there is any chance it will be scheduled. Often this is the correct thing to do, and in a sense the whole point of using these relaxations is to identify this situation. However, we can sometimes do better by, in effect, moving J_j ahead of J_k in the list. For example, if, on some machine, there are more than p_j consecutive units of idle time between time r_j and r_k, then clearly we can run J_j earlier and the average completion time can only decrease.

We generalize this idea in an algorithm we call Modified Greedy. We again start with the schedule $P1$ and order the jobs on a list by C_j^{P1}. We loop through time one unit at a time, choosing jobs to start. Let β be a parameter to be chosen later. Let t be the current time, there are three cases to consider: **(1)** [*there is an idle machine and the first job J_j on the list has $r_j \le t$*] — in this case schedule J_j on an idle machine; **(2)** [*there is an idle machine M, the first job J_j on the list has $r_j > t$, another job J_k on the list has $r_k \le t$*] — focusing on the job J_k which is first on the list with $r_k \le t$, if there is at least βp_k uncharged idle time, then schedule J_k on M and charge the βp_k units of idle time to J_k; **(3)** [*no idle machines, or no jobs J_j satisfying either of the previous two cases*] — in this case do not schedule anything, merely increment t.

Lemma 3.3. *If we apply* Modified Greedy *to P1, the resulting schedule D has total completion time*

$$\sum C_j^D \le (2+\beta)C_j^* + \frac{1}{\beta}\sum_j r_j.$$

Proof. We focus on a particular job J_j. Let $B_j = \{J_1, \ldots, J_{j-1}\}$ be the set of jobs that come before J_j on the list and let A_j be the set of jobs J_k that come after J_j on the list, but have $r_k < r_j$. Let S_j be the starting time of job J_j. First observe that no job in A_j will start after time r_j but before time S_j, and that there is no idle time between r_j and S_j. Thus every job J_k from A_j that runs before J_j runs because of case 2, and gets charged against βp_k idle time. Since there is no idle time between time r_j and time S_j the total amount of idle time before time S_j is at most mr_j and the charges sum to at most mr_j. Letting Z_j be the set of jobs from A_j that actually run before J_j, we get that $\sum_{J_k \in Z_j} \beta p_k \le mr_j$ which implies $\sum_{J_k \in Z_j} p_k \le \frac{mr_j}{\beta}$. Thus the jobs in Z_j can contribute at most $\frac{mr_j}{\beta}$ processing time to the interval between r_j and S_j.

We now focus on jobs in B_j. For each job $J_k \in B_j$, in may have had to wait for βp_k idle time to accumulate before it ran. Thus, summing over all the jobs that could run before J_j gives $(1+\beta)\sum_{i=1}^{j-1} p_i$ of processing plus forced idle time. Thus the jobs in B_j can contribute at most $(1+\beta)\sum_{i=1}^{j-1} p_i$ processing plus forced idle time to the interval between 0 and S_j.

Assuming pessimistically that all the processing and forced idle time associated with B_j occurs after r_j, we conclude that the total amount of processing plus forced idle time between r_j and S_j is bound by $mr_j/\beta + (1+\beta)\sum_{i=1}^{j-1} p_i$. Since after time r_j, all the jobs are available, all m machines will always be busy, and we conclude that $S_j \le r_j + (1+\beta)/m\sum_{i=1}^{j-1} p_i + (1/\beta)r_j$, and that J_j completes by time $C_j^D \le r_j + p_j + (1+\beta)/m\sum_{i=1}^{j-1} p_i + (1/\beta)r_j$. Now use the trivial bound of $r_j + p_j \le C_j^*$, and recall that $\sum_{i=1}^{j-1} p_i/m \le C_j^{P1}$ by (3.1). This gives us that $C_j^D \le (2+\beta)C_j^* + \frac{r_j}{\beta}$. We sum over all jobs to obtain that $\sum_j C_j^D \le (2+\beta)\sum_j C_j^* + \sum_j \frac{r_j}{\beta}$. ■

We now show that we can balance the two algorithms, list scheduling and Modified Greedy to achieve an approximation ratio better than 2.85.

Lemma 3.4. *For any input I, either list scheduling from P1 or using* Modified Greedy *produces a schedule with $\sum_j C_j \le 2.85\sum_j C_j^*$ and runs in $O(n \log n)$ time.*

Proof. By (3.3), we know that

$$\sum_j C_j^N \le 2\sum_j C_j^{P1} + \sum_j p_j. \tag{3.4}$$

If, for some α, we know that $\sum_j p_j \le \alpha \sum_j C_j^*$, then the list scheduling algorithm is a $2+\alpha$ approximation algorithm.

Now consider the case when $\sum_j p_j > \alpha \sum_j C_j^*$. If we combine this equation with the simple bound that $\sum_j C_j^* \ge \sum_j (p_j + r_j)$, we get that $\sum_j p_j \ge \alpha \sum_j (p_j + r_j)$, which implies that

$$\sum_j r_j \le \frac{1-\alpha}{\alpha}\sum_j p_j \le \frac{1-\alpha}{\alpha}\sum C_j^*. \tag{3.5}$$

We can now plug (3.5) into the upper bound on C_j^D from Lemma 3.3 to get

$$\sum_j C_j^D \le \left(2+\beta+\frac{1-\alpha}{\alpha\beta}\right)\sum_j C_j^*.$$

We do not know the value of α, but for each possible α we can choose the β that minimizes the two terms. Simple algebra and calculus shows that given α, we can choose β to be $\sqrt{\frac{1-\alpha}{\alpha}}$. Thus, trying all possible α yields the result.

We can obtain running time $O(n \log n)$ by observing that the analysis of the 3-approximation algorithm allows us to estimate C_j^* to within a factor 3. We can then perform a binary search on α. ■

4 Precedences and Generic Conversion

We now extend our results to deal with precedence constraints, which are represented in the usual way by a directed acyclic graph (DAG). For simplicity, we will assume that all release times are 0; our bounds extend to the case of arbitrary release dates. We say that i immediately precedes j, denoted $i \prec j$, if and only if there is an edge from i to j in the graph; a vertex i precedes a vertex j, denoted $i \stackrel{*}{\prec} j$, if and only if there is a path from i to j.

We use the superscript m to denote the number of processors; thus, S^m denotes a schedule for m processors, C^m denotes the completion time of S^m, and C_j^m denotes the completion time of job J_j under schedule S^m.The subscript OPT refers to an optimal schedule; thus, an optimal schedule is denoted by S^m_{OPT}, and its weighted completion time is denoted by C^m_{OPT}. For a set of jobs A, $p(A)$ denotes the sum of processing times of jobs in A. Observe that our one machine schedule may be preemptive and all the bounds hold.

Definition 4.1. *For any vertex j, define the quantity $\kappa_j = \max_{i \stackrel{*}{\prec} j} p(P_{ij})$ where P_{ij} is a path from i to j. Any path P_{ij} where $p(P_{ij}) = \kappa_j$ is referred to as a* critical *path to j.*

The following two lemmas provide lower bounds on the optimal parallel schedule for the general case; the proofs are omitted.

Lemma 4.1. *$C^m_{\text{OPT}} \geq C^1_{\text{OPT}}/m$.*

Lemma 4.2. *$C^m_{\text{OPT}} \geq C^\infty_{\text{OPT}} = \sum_i w_i \kappa_i$.*

We now describe the Delay List algorithm. Given a one-machine schedule which is a ρ-approximation, Delay List produces a schedule for $m \geq 2$ processors whose value is within a factor $(k_1\rho + k_2)$ of the optimal m-machine schedule, where k_1 and k_2 are small constants. We will describe a variant of this scheduling algorithm which yields $k_1 = (1+\beta)$ and $k_2 = (1 + 1/\beta)$ for any $\beta > 0$. Therefore, for cases where we can find optimal sequential schedules (trees, series-parallel), we obtain a 4-approximation for m-machines by setting $\beta = 1$. To our knowledge, these are the best results for these special cases.

The main idea is as follows. The one-processor schedule taken as a list provides some priority information on which jobs to schedule earlier. Unlike with makespan, the completion time of every job is important for weighted completion time. When trying to convert the one-processor schedule into a parallel one, precedence constraints prevent complete parallelization. Thus, we may have to execute jobs out-of-order from the list to benefit from parallelism. If all p_i are identical (say 1), we can afford to use naive list scheduling [2]. If there is an idle processor and we schedule some available job on it, it is not going to delay jobs which become available soon, since it completes in one time unit. On the other hand if not all p_i's are the same, a job could keep a processor busy, delaying more profitable jobs that become available soon. Also, we cannot afford to keep processors idle. We strike a balance between the two extremes: schedule a job out-of-order only if there has been enough idleness already to justify scheduling it.

Assume, for ease of exposition, that all processing times are integers and time is discrete. A job is *ready* as soon as all its predecessors are done. Let $\beta > 0$ be some constant. At each time step t, the algorithm applies one of the following three cases: **(1)** [*there is an idle machine M and the first job J_j on the list is ready at time t*] — schedule J_j on M; **(2)** [*there is an idle machine and the first job J_j in the list is not ready at t but there is another ready job J_k on the list*] — focusing on the job J_k which is the first in the list among the ready jobs, schedule it if there is at least βp_k *uncharged* idle time among all processors, and charge βp_k idle time to J_k; **(3)** [*there is no idle time or the above two cases do not apply*] — do not schedule any job, merely increment t. In contrast with Modified Greedy, in this case we charge against idle time on all machines rather than on one machine.

Definition 4.2. *A job is said to be scheduled* in order *if it is scheduled when it is at the head of the list. Otherwise it is said to be scheduled* out of order. *The set of jobs which are scheduled before a job J_i but which come later in the list than J_i is denoted by O_i. The set of jobs which come after J_i in the list is denoted by A_i and those which come before by B_i (includes J_i).*

Definition 4.3. *For each job J_i, define a path $P'_i = J_{j_1}, J_{j_2}, \ldots, J_{j_k} = J_i$ with respect to the schedule S^m as follows. The job J_{j_k} is the immediate predecessor of $J_{j_{k+1}}$ with the largest completion time (in S^m) among all the immediate predecessors of $J_{j_{k+1}}$; ties are broken arbitrarily. The jobs in P'_i define a disjoint set of time intervals in the schedule. Let κ'_i denote the sum of the processing times of $J_{j_1}, \ldots, J_{j_k}$.*

Fact 4.1. *$\kappa'_i \leq \kappa_i$.*

Fact 4.2. *The idle time charged to each job J_i is less than or equal to βp_i.*

[2] In this section, by *list scheduling* we mean the algorithm which schedules the first available job in the list if a processor is free. This is in contrast to another variant considered in earlier sections in which jobs are scheduled strictly in the order of the list.

A crucial feature of the algorithm is that when it schedules jobs, it considers only the first job in the ready list, even if there is enough idle time for other ready jobs that are later in the list.

LEMMA 4.3. *In the time interval* (r_i^m, s_i^m) *where* r_i^m *is the time when the job* J_i *is ready and* s_i^m *is its start time, there is no uncharged idle time and furthermore all the idle time is charged only to jobs in* B_i.

Proof. By the preceding remarks, it is clear that no job in A_i is scheduled in the time interval (r_i^m, s_i^m) since J_i was ready at r_i^m. From this we can conclude that there is is no idle time charged to jobs in A_i in that time interval. Since J_i is ready in that interval and was not scheduled, there cannot be any uncharged idle time. ■

The following lemma shows that for any job J_i, the algorithm does not schedule too many jobs from A_i before scheduling J_i itself.

LEMMA 4.4. *For every job* J_i, *the total idle time charged to jobs in* A_i, *in the interval* $(0, s_i^m)$, *is bounded by* $(m-1)\kappa_i'$. *It follows that* $p(O_i) \leq (m-1)\kappa_i'/\beta \leq (m-1)\kappa_i/\beta$.

Proof. Consider a job J_{j_k} in P_i'. The job $J_{j_{k+1}}$ is ready to be scheduled at the completion of J_{j_k}, that is $r_{i_{k+1}}^m = C_{i_k}^m$. From Lemma 4.3, it follows that in the time interval between $(C_{j_k}^m, s_{j_{k+1}}^m)$ there is no idle time charged to jobs in $A_{j_{k+1}}$. Since $A_{j_{k+1}} \supset A_i$ it follows that all the idle time for jobs in A_i has to be accumulated in the time intervals defined by P_i'. This quantity is clearly bounded by $(m-1)\kappa_i'$. The second part follows since the total processing time of the jobs in O_i is bounded by $1/\beta$ times the total idle time that can be charged to jobs in A_i (recall that $O_i \subseteq A_i$). ■

THEOREM 4.1. *Let* S^m *be the schedule produced by the algorithm* DELAY LIST *using a sequential schedule* S^1 *as the list. Then for each job* J_i, $C_i^m \leq (1+\beta)C_i^1/m + (1+1/\beta)\kappa_i$.

Proof. Consider a job J_i. We can split the time interval $(0, C_i^m)$ in to two disjoint sets of time intervals T_1 and T_2 as follows. The set T_1 consists of all the intervals in which some job in P_i' runs. The set T_2 consists of the time intervals obtained by removing the intervals in T_1 from $(0, C_i^m)$. Let t_1 and t_2 be the sum of the times of the intervals in T_1 and T_2 respectively. From the definition of T_1, it follows that $t_1 = \kappa_i' \leq \kappa_i$. From Lemma 4.3, in the time intervals of T_2, all the idle time is either charged to jobs in B_i and, the only jobs which run are from $B_i \cup O_i$. From Fact 4.2, the idle time charged to jobs in B_i bounded by $\beta p(B_i)$. Therefore the time t_2 is bounded by $(\beta p(B_i) + p(B_i) + p(O_i))/m$. By Lemma 4.4 and the fact that all the jobs in B_i come before J_i in the sequential schedule, it follows that $t_2 \leq (1+\beta)C_i^1/m + (m-1)\kappa_i/(\beta m)$. Therefore we conclude that $C_i^m \leq (1+\beta)C_i^1/m + (1+1/\beta)\kappa_i$. ■

THEOREM 4.2. *Given a sequential schedule within a factor* ρ *of an optimal sequential schedule,* DELAY LIST *gives a m-processor schedule within a factor* $(1+\beta)\rho + (1+1/\beta)$ *of an optimal parallel schedule.*

Proof. Let S^1 be a schedule which is within a factor ρ of the optimal sequential schedule. Then $C^1 = \sum_i w_i C_i^1 \leq \rho C_{\mathrm{OPT}}^1$. By Theorem 4.1, the schedule created by the algorithm DELAY LIST satisfies,

$$\begin{aligned}
C^m &= \sum_i w_i C_i^m \\
&\leq \sum_i w_i \left((1+\beta)\frac{C_i^1}{m} + \left(1+\frac{1}{\beta}\right)\kappa_i \right) \\
&\leq \frac{1+\beta}{m}\sum_i w_i C_i^1 + \left(1+\frac{1}{\beta}\right)\sum_i w_i \kappa_i \\
&\leq \frac{(1+\beta)\rho C_{\mathrm{OPT}}^1}{m} + \left(1+\frac{1}{\beta}\right) C_{\mathrm{OPT}}^\infty \\
&\leq \left((1+\beta)\rho + \left(1+\frac{1}{\beta}\right)\right) C_{\mathrm{OPT}}^m .
\end{aligned}$$

■

COROLLARY 4.1. *There is a simple* $O(n \log n)$ *time* 4*-approximation algorithm for weighted completion time on parallel machines when the precedence graphs are restricted to be trees or series-parallel graphs.*

REMARK 4.1. *The algorithm* DELAY LIST *can be generalized to handle release dates on the jobs. In fact we can use a preemptive schedule instead of a nonpreemptive schedule and the algorithms guarantees a ratio of* $(1+\beta)\rho + (1+1/\beta)$ *where* ρ *is the approximation ratio guaranteed by the preemptive schedule.*

REMARK 4.2. *Since the bounds in our conversion algorithm are job-by-job, the algorithm is applicable to a more general class of metrics as well.*

We obtain stronger results for in-tree precedence without release dates. The details are omitted.

THEOREM 4.3. *There is an* $O(n \log n)$*-time algorithm with approximation ratio* 2 *for minimizing weighted completion time on* m *machines for in-tree precedence without release dates.*

Acknowledgments. We thank Javed Aslam, Moses Charikar, Cindy Phillips, David Shmoys, Joel Wein, and Santosh Vempala for valuable discussions.

References

[1] D. Adolphson. Single machine job sequencing with precedence constraints. *SIAM Journal on Computing*, 6:40–54 (1977).

[2] S. Chakrabarti, C. Phillips, A. Schulz, D.B. Shmoys, C. Stein, and J. Wein. Improved Scheduling Algorithms for Minsum Criteria. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, Springer, 1996.

[3] C. Chekuri, R. Johnson, R. Motwani, B.K. Natarajan, B.R. Rau, and M. Schlansker. Profile-Driven Instruction Level Parallel Scheduling with Applications to Super Blocks. To appear in *Proceeding of the 29th Annual International Symposium on Microarchitechture (MICRO-29)*, Paris, France, December 2–4, 1996.

[4] J. Du, J.Y.T. Leung, and G.H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75 (1990).

[5] J. Du, J.Y.T. Leung, and G.H. Young. Scheduling chain structured tasks to minimize makespan and mean flow time. *Information and Computation*, 92:219–236 (1991).

[6] M.R. Garey. Optimal task sequencing with precedence constraints. *Discrete Mathematics*, 4:37–56 (1973).

[7] M.R. Garey and D.S. Johnson. **Computers and Intractability: A Guide to the Theory of NP-completeness,** Freeman, San Francisco (1979).

[8] M.X. Goemans. A Supermodular Relaxation for Scheduling with Release Dates. In *Proceedings of the Fifth Conference On Integer Programming and Combinatorial Optimization*, 288–301, 1996.

[9] M.X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[10] R.L. Graham Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429 (1969).

[11] L.A. Hall, D.B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Offline and online algorithms. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, 1996, pp. 142–151.

[12] L.A. Hall, A.S. Schulz, D.B. Shmoys, J. Wein. Scheduling to Minimize Average Completion time: Off–line and On–line Approximation Algorithms. Preprint, 1996.

[13] J.L. Hennessy and T. Gross. Postpass code optimization of pipeline constraints. *ACM Transactions on Programming Languages and Systems*, 5:422-448 (1983).

[14] W.A. Horn. Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM Journal of Applied Mathematics*, 23:189–202 (1972).

[15] D.S. Hochbaum, and D.B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17:539–551 (1988).

[16] J.A. Hoogeveen, and A.P.A. Vestjens. Optimal On-Line Algorithms for Single-Machine Scheduling. In *Proceedings of the Fifth Conference On Integer Programming and Combinatorial Optimization*, 1996, pp. 404–414.

[17] W.A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185 (1974).

[18] T.C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9:841–848 (1961).

[19] E.L. Lawler. Sequencing jobs to minimize total weighted completion time. *Annals of Discrete Mathematics*, 2:75–90 (1978).

[20] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, Sequencing and Scheduling: Algorithms and Complexity, In *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory* (1990).

[21] C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. In *Proceedings of the Fourth International Workshop on Algorithms and Data Structures*, 1995, pp. 86–97.

[22] R. Ravi, A. Agrawal, and P. Klein. Ordering problems approximated: single-processor scheduling and interval graph completion. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, 1991, pp. 751–762.

[23] A.S. Schulz. Scheduling to minimize total weighted completion time: performance guarantees of lp based heuristics and lower bounds. In *Proceedings of the Fifth Conference On Integer Programming and Combinatorial Optimization*, 1996, pp. 301–315.

[24] L. Stougie, 1995. Personal communication cited in [16].

[25] L. Stougie and A. Vestjens, 1996. Personal communication.

[26] S. Weiss and J.E. Smith. A study of scalar compilation techniques for pipelined supercomputers. In *Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1987.

[27] A.C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, 1977, pp. 222–227.