Approximation Techniques for Average Completion Time Scheduling*

C. Chekuri[†] R. Motwani[‡] B. Natarajan[§] C. Stein[¶]

Abstract

We consider the problem of non-preemptive 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 obtain improved approximation ratios for precedence graphs such as in-trees, out-trees, and series-parallel graphs.

---

*A preliminary version [5] of this paper appeared in SODA 1997.

[†]Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974. Email: `chekuri@research.bell-labs.com`. Work done while author was at Stanford University where he was supported by NSF Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

[‡]Dept. of Computer Science, Stanford University, Stanford, CA 94305-9045. Email: `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.

[§]Arcot Systems, 2490 Sand Hill Road, Menlo Park, CA 94025. Email: `nat@arcot.com`. Work done while the author was at Hewlett Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304.

[¶]Dept. of Computer Science, Dartmouth College, Hanover, NH 03755. Email: `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 *non-preemptive* scheduling to minimize *average (weighted) completion time* (equivalently sum of (weighted) completion times). In this problem, we are given n jobs $J_1, \ldots, J_n$, where job J_j has processing time p_j, release date r_j, and a positive weight w_j. A feasible schedule S assigns jobs non-preemptively[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 [30]. 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 [23]. Thus, we will consider approximation algorithms, or, in an on-line setting, competitive ratios.

An important motivation for studying scheduling to minimize sum of weighted completion times, 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 [17, 35]). Current research is addressing the issue of profile-based compiler optimization. In a recent paper, Chekuri et al, [4] 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 [25, 16, 3, 12, 7, 28]. 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 as per this ordering.

We introduce new techniques that generalize this basic paradigm. We 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 [21], and matches a recent lower bound for randomized on-line algorithms [33].

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

[1] In a non-preemptive 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.

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 non-preemptive 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, in some cases, are not as good as those obtained directly through 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 an average completion time problem was the following 2-approximation algorithm of Phillips, Stein and Wein [25] 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 [23] 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. Other 2-approximation algorithms have been discovered subsequently [21, 32, 12], and it is also known that no deterministic on-line algorithm has approximation ratio better than 2 [21, 32]. 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 $\frac{e}{e-1}$. This beats the deterministic on-line lower bound. Our approach is based on what we call *α-schedules* (this notion was also used by [25] and [15] 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 αp_j, an α-fraction of J_j, is completed. An *α-schedule* is a non-preemptive 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 [25] 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 observation is that a worst-case instance that induces a 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 which, given I, chooses α to minimize $\alpha(I)$, as 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. 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 $O(n^2)$ time and all our other algorithms can be implemented in either $O(n)$ or $O(n \log n)$ time. Torng and Uthaisombut [34] recently showed that our analysis of Best-α is tight by giving instances on which the approximation ratio achieved by Best-α is arbitrarily close to $e/(e-1)$.

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. The current best approximation algorithm for the preemptive case [26] has a ratio of 4/3, which yields a 2.12-approximation for the non-preemptive case. However this does not improve earlier results.

Independently, Goemans [12] has used very similar ideas to design a 2-approximation algorithm for the problem of non-preemptive scheduling on one machine so as to minimize the average *weighted* completion time. His algorithm is also a Best-α algorithm and works off a preemptive schedule that is optimal for a certain LP relaxation of the problem and the analysis is based on the LP formulation. Interestingly, Goemans proves the performance of his algorithm by choosing α uniformly at random in the interval $(0, 1]$. Further work [26, 13] based on the idea of using independent random α points for each job has resulted in an improved approximation ratio of 1.6853.

Scheduling Parallel Machines with Release Dates. We consider generalizations of the single machine problems to the case of m identical parallel machines. We first consider the problem of minimizing average completion time with release dates and no precedence constraints. Extending the techniques to the m-machine problem gives rise to two complications: the problem of computing an optimal m-machine preemptive schedule is $\mathcal{NP}$-hard [9], and the best known approximation ratio is 2 [25]; further, the conversion bounds from preemptive to non-preemptive schedules are not as good. Chakrabarti et al [3] obtain a bound of 7/3 on the conversion from preemptive to non-preemptive 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 [25] gave the first such algorithm, a 24-approximation algorithm. This has been greatly improved to $4+\epsilon$ [15], $4-\frac{1}{m}$ [16], and 3.5 [3]. Using a general on-line framework [3], 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 [20].

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 non-preemptive m-machine schedule that is a 3-approximation. Our algorithm has a running time of $O(n \log n)$ and in addition is also on-line. We then show that the approximation ratio can be improved to 2.83, using a general conversion algorithm that we develop. This improves on the approximation bounds of previous algorithms and gives a much smaller running time of $O(n \log n)$. Subsequent to our work, Schulz and Skutella [28] using some of our ideas have obtained an approximation ratio of 2 for the more general case of sum of weighted completion times.

Scheduling with Precedence Constraints. We now consider the weighted completion time problem with precedence constraints. For one-machine scheduling, minimizing the weighted completion time is $\mathcal{NP}$-hard for arbitrary precedence constraints even without release dates [11, 22]. A 2-approximation for the case of no release dates [16] and an $e \simeq 2.718$ [29] approximation with release dates are known. For arbitrary m, the problem is $\mathcal{NP}$-hard even without precedence constraints and release dates if weights are not required to be identical; 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 [10]. An expected approximation ratio of $5.33+\epsilon$ is achievable with release dates and precedence constraints [3]. This has been recently improved to 4 in [24].

A General Conversion Algorithm. We obtain a fairly general algorithm for m-machine problems with precedence constraints, release dates and job weights. To do so, we first solve a one-machine preemptive relaxation, and apply an algorithm we call Delay List to get an m-machine non-preemptive 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. In fact we give an algorithm that gives a $(1+\beta)\rho+(1+1/\beta)$ approximation for any $\beta > 0$ which when optimized for ρ yields a $(\rho+2\sqrt{\rho}+1)$ approximation. In the absence of release dates an optimal one-machine schedule can be computed in polynomial time when the precedence graph is a forest [19], or a series-parallel graph [22, 1]. Applying our conversion algorithm for these cases results in improved approximation results. Although the algorithm fails to obtain improved results for the most general problem, we feel that it is of independent interest and likely to find applications in the future. Further, our conversion algorithm has the advantage of being simple and combinatorial. In applications such as compilers [4], 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 [3], and Stein and Wein [31] also have shown the existence of such schedules).

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 non-preemptive α-schedule derived from P, respectively. We begin by analyzing simple α-schedules. Techniques from [25, 16] are easily generalized to yield the following:

Theorem 1 *Given an instance of one-machine scheduling with release dates, for any $\alpha \in (0,1]$, an α-schedule has $\sum_j C_j^\alpha \leq (1+1/\alpha)\sum_j C_j^P$. Further, there are instances where the inequality is asymptotically tight.*

Proof: Index the jobs by the order of their α-points in the preemptive schedule P. Let $r_j^{\max} = \max_{1\leq k\leq j} r_k$ be the latest release date among jobs with α points no greater than j's. By time $r_j^{\max}$, jobs 1 through j have all been released, and hence

$$C_j^\alpha \leq r_j^{\max} + \sum_{k=1}^{j} p_k. \tag{1}$$

We know that $C_j^P \geq r_j^{\max}$, since only an α fraction j has finished by $r_j^{\max}$. We also know that $C_j^P \geq \alpha\sum_{k=1}^{j} p_k$, since the α fractions of jobs $1,\ldots,j$ must run before time C_j^P. Plugging these last two inequalities into (1) yields $C_j^\alpha \leq (1+\frac{1}{\alpha})C_j^P$. Summing over all j yields the lemma.

To see that this is tight, consider the following class of instances. Let ϵ be a small positive number. We will also allow jobs with processing time 0, although the proof can be modified even if these are not allowed. At time 0, we release a job with processing time 1. At time $\alpha-\epsilon$, we release a job with processing time ϵ and at time $\alpha+\epsilon$, we release x jobs of processing time 0. The optimal preemptive completion time is $\alpha+x(\alpha+\epsilon)+1+\epsilon$, while the completion time of the non-preemptive α-schedule is $\alpha+(\alpha+1)+x(1+\alpha)$. As x gets large and ϵ goes to 0, the ratio between the two goes to $1+\frac{1}{\alpha}$. $\square$

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 below. Thus, the idle time introduced in the non-preemptive 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 non-preemptive 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 an optimal preemptive schedule whose value is a lower bound on the value of any optimal non-preemptive 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 time and fractional processing times of jobs that ran before C_i^P. This yields the following lemma:

Lemma 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

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

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 in which

(C1) jobs $J_1,\ldots,J_i$ run non-preemptively in that order,

(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 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 $J^B = \bigcup_{\beta\geq\alpha} S_i^P(\beta)$ and $J^A = J - J^B$. We can interpret the four terms in the above equation as (1) the idle time in the preemptive schedule before C_i^P, (2) the pieces of jobs in J^A that ran before C_i^P, (3) for each job $J_j \in J^B$, the pieces of J_j that ran before $C_j^P(\alpha)$, and (4) for each job $J_j \in J^B$, the pieces of J_j that run between $C_j^P(\alpha)$ and C_i^P. Let x_j be the β for which $J_j \in S_i^P(\beta)$, that is the fraction of J_j that was completed before C_i^P. Then $\sum_{\beta\geq\alpha}(\beta-\alpha)S_i^P(\beta)$ can be rewritten as $\sum_{J_j\in J^B}(x_j-\alpha)p_j$. Observe that $(x_j-\alpha)p_j$ is the fraction of job J_j that ran between $C_j^P(\alpha)$ and C_i^P.

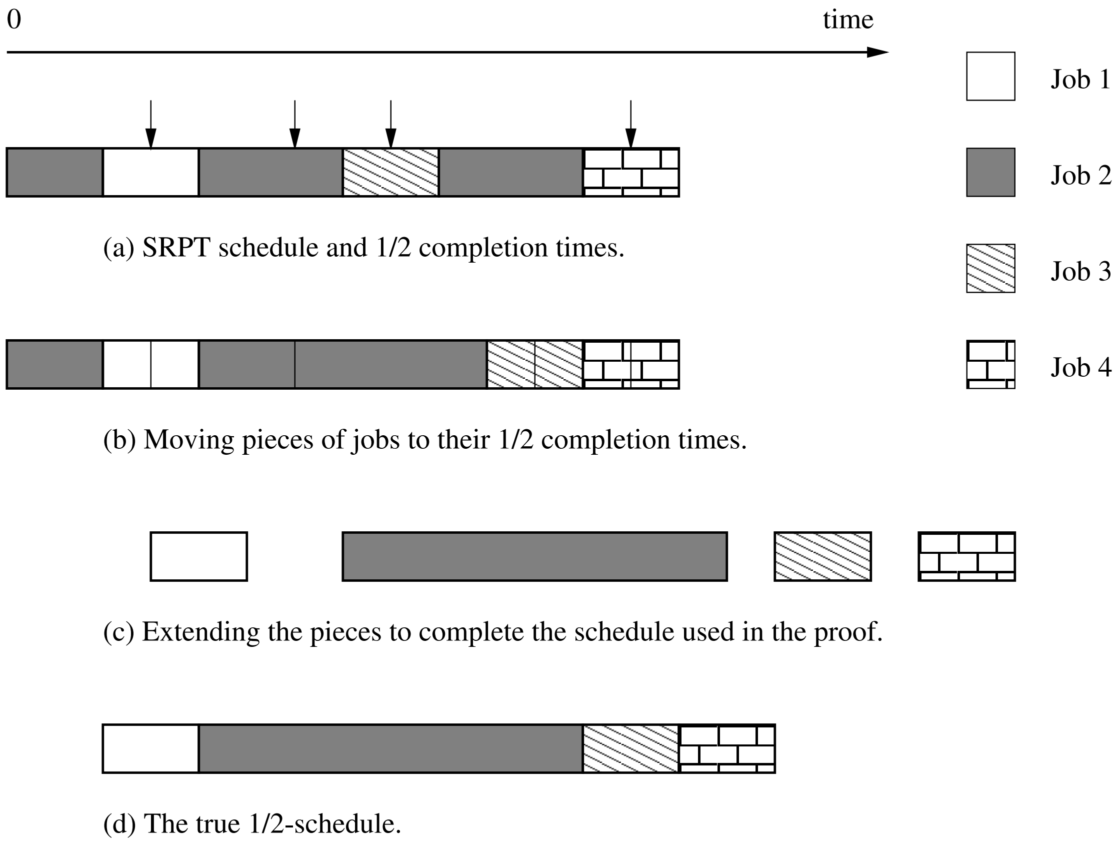

Figure 1: Illustration of proof of Lemma 2 with $\alpha = 1/2$ and $i = 4$.

Let $J^C = \{J_1, \ldots, J_i\}$. Clearly J^C is a subset of J^B. Now think of schedule P as an ordered list of pieces of jobs (with sizes). For each $J_j \in J^C$ 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 $(x_j-\alpha)p_j$ at the point corresponding to $C_j^P(\alpha)$. In this list, we have pieces of size $(x_j-\alpha)p_j$ 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 $(x_j - \alpha)p_j$, we only moved pieces later in time, so no additional idle time need be introduced.

Now, for each job $J_j \in J^C$, extend the piece of size $(x_j - \alpha)p_j$ to one of size p_j by adding $p_j - (x_j - \alpha)p_j$ units of processing and replace the pieces of J_j that occur earlier, of total size αp_j, by idle time. Figure 1 illustrates this transformation. 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

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

where the second equality just comes from re-indexing terms by β instead of j, and the third comes from plugging in the value of C_i^P from Lemma 1. To complete the proof, we observe that the remaining pieces in the schedule are all from jobs in $J - J^C$, and we have thus met the conditions (C1), (C2) and (C3) above. □

Although we will not use it directly, applying Lemma 2 to the last job to complete in the α-schedule yields:

Corollary 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, 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 general bound on this algorithm, which we call Random-α.

Lemma 3 *Suppose α is chosen from a probability distribution over $(0, 1]$ with a density function f. Then for each job J_i, $E[C_i^\alpha] \leq (1 + \delta) C_i^P$, where*

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

It follows that $E[\sum_i C_i^\alpha] \leq (1 + \delta) \sum_i C_i^P$.

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 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, $E[C_i^\alpha] = \int_0^1 f(\alpha) C_i^\alpha \mathrm{d}\alpha$, 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 *For the problem of scheduling to minimize weighted completion time with release dates,* RANDOM-α *performs as follows:*

1. *If α is chosen uniformly in $(0,1]$, the expected approximation ratio is at most 2.*
2. *If α is 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. *If α is chosen from $(0,1]$ according to the density function $f(\alpha) = \frac{e^\alpha}{e-1}$, the expected approximation ratio is at most $\frac{e}{e-1} \approx 1.58$.*

Proof:

1. Choosing α uniformly corresponds to the pdf $f(\alpha) = 1$. Plugging into the bound from Lemma 3, we get an approximation ratio of

$$\begin{aligned}
1 + \max_{0<\beta\le 1} \int_0^\beta \frac{1+\alpha-\beta}{\beta} \mathrm{d}\alpha &= 1 + \max_{0<\beta\le 1} \frac{1}{\beta}\left((1-\beta)\beta + \frac{\beta^2}{2}\right) \\
&= 1 + \max_{0<\beta\le 1} \left(1 - \frac{\beta}{2}\right) \\
&\le 2.
\end{aligned}$$

2. Omitted.

3. If $f(\alpha) = \frac{e^\alpha}{e-1}$ then

$$\begin{aligned}
\max_{0<\beta\le 1} \int_0^\beta \left(\frac{1+\alpha-\beta}{\beta}\right)\left(\frac{e^\alpha}{e-1}\right) \mathrm{d}\alpha &= \max_{0<\beta\le 1} \frac{1}{\beta(e-1)}\Big(((1-\beta)+(\beta-1))e^\beta - ((1-\beta)-1)\Big) \\
&= \max_{0<\beta\le 1} \frac{1}{e-1} \\
&= \frac{1}{e-1}.
\end{aligned}$$

Therefore

$$1 + \max_{0<\beta\le 1} \int_0^\beta \left(\frac{1+\alpha-\beta}{\beta}\right)\left(\frac{e^\alpha}{e-1}\right) \mathrm{d}\alpha \;\le\; \frac{e}{e-1}.$$

□

It can be shown that the density function $\frac{e^\alpha}{e-1}$ minimizes the expression $\max_{0<\beta\le 1} \int_0^\beta \frac{1+\alpha-\beta}{\beta} f(\alpha)\mathrm{d}\alpha$ over all choices of $f(\alpha)$. In the off-line setting, rather than choosing α randomly, we can try different values of α and choose the one that yields the best schedule. We call the algorithm which computes the schedule of value $\min_\alpha \sum_j C_j^\alpha$, Best-$\alpha$.

Corollary 2 *Algorithm* Best-α *is an* $e/(e-1)$*-approximation algorithm for non-preemptive scheduling to minimize average completion time on one machine with release dates. It runs in* $O(n^2)$ *time.*

Proof: The approximation bound follows from Theorem 2. For the running time, we observe that given a preemptive SRPT schedule we can efficiently determine the best possible choice of α. The SRPT schedule only preempts at release dates. Thus it has at most $n-1$ preemptions and there are are at most n "combinatorially distinct" values of α for a given preemptive schedule. The SRPT schedule can be computed in $O(n \log n)$ time using a simple priority queue and given that schedule and an α, the corresponding α-schedule can be computed in linear time by a simple scan.

□

In the on-line setting, we cannot implement Best-α. However, if we choose α randomly we get the following:

Theorem 3 *There is a polynomial-time randomized on-line algorithm with an expected competitive ratio $e/(e-1)$ for the problem of minimizing total completion time in the presence of release dates.*

Proof: The randomized on-line algorithm is the following. The algorithm picks an $\alpha \in (0,1]$ at random according to the density function $f(x) = \frac{e^x}{e-1}$ before receiving any input (this is the only randomness used in the algorithm). The algorithm simulates the on-line preemptive SRPT schedule. At the exact time when a job finishes α fraction of its processing time in the simulated SRPT schedule it is added to the queue of jobs to be executed non-preemptively. The non-preemptive schedule is obtained by executing jobs in the strict order of their insertion into the queue while respecting the insertion times into the queue. Observe that this rule leads to a valid on-line non-preemptive schedule and that in fact the order of the jobs scheduled is exactly the same as in the α-schedule. The schedule respects the insertion times, therefore no job is executed in the non-preemptive schedule before its α point in the SRPT schedule. To show the bound on the expected competitive ratio, we claim that the bounds in Lemma 2 (and hence Theorem 2 also) hold for the non-preemptive schedule created by the on-line algorithm. The main observation is that the proof of Lemma 2 does not use the true α-schedule but a weaker one in which for every job J_i the first α fraction of its processing time in the SRPT schedule is left as idle time. A careful examination of the proof of Lemma 2 with Figure 1 as an illustration makes this clear. □

We also give some negative results for the various algorithms.

Theorem 4 *For the problem of scheduling to minimize weighted completion time with release dates,* Random-α *performs as follows:*

1. *If α is chosen uniformly, the expected approximation ratio is at least* 2.
2. *For the* Best-α *algorithm, the approximation ratio is at least* $4/3$.

Proof: We will use a set of parameterized instances to show all the above bounds. We define an instance $I(\delta, n)$ as follows. At time 0 a job of size 1 is released and at time $\delta < 1$, n jobs of size 0 are released (we use zero length jobs for ease of exposition). The optimal preemptive schedule for this instance has a total completion time of $1 + n\delta$. The optimal non-preemptive schedule for this instance can be obtained by first completing all the small jobs and running the large job after them for a total completion time of $1 + \delta + n\delta$. It is easy to see that there are only two combinatorially distinct schedules corresponding to the values of $\alpha \leq \delta$ and $\alpha > \delta$ and we can restrict our attention to those two schedules and the probability with which they are chosen. Let $S1$ and $S2$ be the two schedules and $C1$ and $C2$ be their total completion times respectively. It is easy to see that $C1 = 1 + n$ and $C2 = 1 + \delta + n\delta$.

1. If α is chosen uniformly at random, $S1$ is chosen with probability δ and $S2$ is chosen with probability $(1 - \delta)$ and a simple calculation shows that if we choose $n \gg 1$ and $1 \gg \delta$, the expected approximation ratio approaches 2.

2. Consider an instance I in which in addition to the jobs of $I(1/2, n)$ we release n more jobs of size 0 at time 1. The optimal preemptive schedule for I consists of the preemptive schedule for $I(1/2, \epsilon, n)$ followed by the additional n small jobs. The completion time of the optimal preemptive schedule is term $1 + 3n/2$. An optimal non-preemptive schedule schedules the large job after all the small jobs and has a total completion time $2 + 3n/2$. It is easy to see that there are only two combinatorially distinct α-schedules one corresponding to $\alpha \leq 1/2$ and the other corresponding to $\alpha > 1/2$. In both cases it is easy to verify that the completion time of the schedule is $1 + 2n$. Thus the approximation ratio of the BEST-α cannot be better than $4/3$.

□

After learning of our results, Stougie and Vestjens [33] improved the lower bound for randomized on-line algorithms to $e/(e-1)$. This implies that our randomized on-line algorithm is *optimal*. Torng and Uthaisombut [34] have shown that there are instances on which the approximation ratio of BEST-α can be made arbitrarily close to $\frac{e}{e-1}$. This improves our lower bound of $4/3$ on BEST-α's performance and also implies that our upper bound analysis is tight.

3 Parallel Machine Scheduling with Release Dates

We now turn to the problem of minimizing average completion time on parallel machines, in the presence of release dates. In this section, we give a simple 3-approximation algorithm for the problem that is also an on-line algorithm. Our algorithm does not use linear programming, or slow dynamic programming. It introduces the notion of a one-machine preemptive relaxation. In the next section, we will show how to improve this to a 2.83-approximation algorithm, using more involved techniques.

Given an instance I for non-preemptive scheduling on m machines, we define a one machine preemptive relaxation $I1$ as follows. $I1$ has the same set of jobs as those of I and has one machine. The processing time of J_j in $I1$ is $p'_j = p_j/m$ and release date is $r'_j = r_j$.

Lemma 4 *The value of an optimal solution to $I1$ is a lower bound on the value of an optimal solution to I.*

Proof: We show how to convert a feasible schedule N, for input I, to a feasible schedule $P1$, for input $I1$, without increasing the average completion time. Take any schedule N and consider a particular time unit t that is sufficiently small. Let the $k \leq m$ jobs that are running during that time be $J_1, \ldots, J_k$. In $P1$, at time t, run $1/m$ units of each of jobs $J_1, \ldots, J_k$, in arbitrary order. The completion time of job J_j in $P1$, C_j^{P1} is clearly no greater than C_j^N, the completion time of J_j in N. □

Given an optimal preemptive schedule $P1$ for $I1$, we form a list schedule N by ordering jobs by C_j^{P1} and then scheduling them non-preemptively in that order, respecting release dates. Let C_j^* be the completion time of J_j in an optimal schedule for I. $I1$ may be a bad relaxation, in the sense that $\sum_j C_j^{P1}$ may be much less than $\sum_j C_j^*$. However, we can still use this relaxation to obtain a good non-preemptive schedule.

Lemma 5 *The non-preemptive list schedule N satisfies $\sum_j C_j^N \leq (3 - \frac{1}{m}) \sum_j C_j^*$.*

Proof: We 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,

$$C_j^{P1} \geq \sum_{k=1}^{j} p'_k = \sum_{k=1}^{j} \frac{p_k}{m}. \tag{2}$$

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

$$\begin{aligned} C_j^N &\leq r_j^{\max} + \sum_{k=1}^{j-1} \frac{p_k}{m} + p_j \\ &\leq C_j^{P1} + C_j^{P1} + p_j \left(1 - \frac{1}{m}\right), \end{aligned} \tag{3}$$

where the second inequality follows from (2) and $C_j^{P1} \geq r_j^{\max}$ above. Summing (3) over all jobs, we get a total completion time of

$$\sum_j C_j^N \leq 2 \sum_j C_j^{P1} + \left(1 - \frac{1}{m}\right) \sum_j p_j. \tag{4}$$

By Lemma 4, $\sum_j C_j^{P1} \leq \sum_j C_j^*$, 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 $(3 - \frac{1}{m})$-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.

4 A General Conversion Algorithm

In this section we develop a technique to obtain parallel machine schedules from one-machine schedules that works even when jobs have precedence constraints and release dates. Given an average weighted completion time scheduling problem, we show that if we can approximate the one-machine preemptive variant, then we can also approximate the m-machine non-preemptive variant, with a slight degradation in the quality of approximation.

Precedence constraints will be represented in the usual way by a directed acyclic graph (DAG) whose vertices correspond to jobs, and whose edges represent precedence constraints.

In this section, we use a slightly different one-machine relaxation from the previous section, namely, we do not divide the processing times by m. We use the superscript m to denote the number of machines; thus, S^m denotes a schedule for m machines, C^m denotes the sum of weighted 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_{OPT}^m, and its weighted completion time is denoted by C_{OPT}^m. For a set of jobs A, $p(A)$ denotes the sum of processing times of jobs in A.

Definition 1 *For any vertex j, recursively define the quantity κ_j as follows. For a vertex j with no predecessors $\kappa_j = p_j + r_j$. Otherwise define $\kappa_j = p_j + \max\{\max_{i \prec j} \kappa_i, r_j\}$. Any path P_{ij} from i to j where $p(P_{ij}) = \kappa_j$ is referred to as a* critical path *to j.*

4.1 Conversion algorithm Delay List

We now describe the Delay List algorithm. Given a one-machine schedule which is a ρ-approximation, Delay List produces a schedule for $m \geq 2$ machines 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 one-machine schedules (trees and series-parallel without release dates), 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-machine schedule taken as a list (jobs in order of their completion times in the schedule) provides some priority information on which jobs to schedule earlier [2]. Unlike with makespan, the completion time of every job is important for weighted completion time. When trying to convert the one-machine schedule into an m-machine 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 [3]. If there is an idle machine 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

[2] In the rest of the paper we assume without loss of generality that a *list* obeys the precedence constraints, that is, if $i \prec j$ then i comes earlier in the list than j.

[3] In this section, by *list scheduling* we mean the algorithm which schedules the first available job in the list if a machine 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.

if not all p_i's are the same, a job could keep a machine busy, delaying more profitable jobs that become available soon. At the same time, we cannot afford to keep machines idle. We strike a balance between the two extremes: schedule a job out-of-order only if there has been enough idle time already to justify scheduling it. To measure whether there has been enough idle time, we introduce a charging scheme.

Assume, for ease of exposition, that all processing times are integers and that time is discrete. This restriction can be removed without much difficulty and we use it only in the interests of clarity and intuition. A job is *ready* if it has been released and all its predecessors are done.

Definition 2 *The time at which job J_i is ready in a schedule S is denoted by q_i^S and time at which it starts is denoted by s_i^S.*

We use S^m to denote the m-machine schedule that our algorithm constructs and for ease of notation the superscript m will be used in place of S^m to refer to quantities of interest in this schedule. Let $\beta > 0$ be some constant. At each discrete 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 and charge all uncharged idle time in the interval (q_j^m, s_j^m) to J_j.

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 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 machines, 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.

Definition 3 *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 J_i by B_i (includes J_i).*

Definition 4 *For each job J_i, define a path $P_i' = J_{j_1}, J_{j_2}, \ldots, J_{j_\ell}$, with $J_{j_\ell} = J_i$ with respect to the schedule S^m as follows. The job J_{j_k} is the predecessor of $J_{j_{k+1}}$ with the largest completion time (in S^m) among all the predecessors of $J_{j_{k+1}}$ such that $C_{j_k}^m \geq r_{j_{k+1}}$; ties are broken arbitrarily. J_{j_1} is the job where this process terminates when there are no predecessors which satisfy the above condition. The jobs in P_i' define a disjoint set of time intervals $(0, r_{j_1}], (s_{j_1}^m, C_{j_1}^m], \ldots, (s_{j_\ell}^m, C_{j_\ell}^m]$ in the schedule. Let κ_i' denote the sum of the lengths of the intervals.*

Fact 1 $\kappa_i' \leq \kappa_i$.

Fact 2 *The idle time charged to each job J_i is less than or equal to βp_i.*

Proof: The fact is clear if idle time is charged to J_i according to case 2 in the description of our algorithm. Suppose case 1 applies to J_i. Since J_i was ready at q_i^m and was not scheduled according to case 2 earlier, the idle time in the interval (q_i^m, s_i^m) that is charged to J_i is less than βp_i. We remark that the algorithm with discrete time units might charge more idle time due to integrality of the time unit. However that is easily fixed in the continuous case where we schedule J_i at the first time instant when at least βp_i units of uncharged idle time have accumulated. □

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

Lemma 6 *For every job J_i, there is no uncharged idle time in the time interval (q_i^m, s_i^m), 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 started in the time interval (q_i^m, s_i^m) since J_i was ready at q_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 at q_i^m and was not scheduled before s_i^m, from cases 1 and 2 in the description of our algorithm 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 7 *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 $m(\kappa_i' - p_i)$. It follows that $p(O_i) \leq m(\kappa_i' - p_i)/\beta \leq m(\kappa_i - p_i)/\beta$.*

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 $q_{j_{k+1}}^m = C_{j_k}^m$. From Lemma 6, 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 intersection between $(0, s_i^m)$ and the time intervals defined by P_i'. This quantity is clearly bounded by $m(\kappa_i' - p_i)$. 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 5 *Let S^m be the schedule produced by the algorithm* Delay List *using a list S^1. Then for each job J_i, $C_i^m \leq (1+\beta)p(B_i)/m + (1 + 1/\beta)\kappa_i' - p_i/\beta$.*

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 disjoint time intervals defined by P_i'. 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 6, 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 2, the idle time charged to jobs in B_i bounded by $\beta p(B_i)$. Therefore the time t_2 is bounded by $(\beta p(B_i) + p(B_i) + p(O_i))/m$. Using Lemma 7 we see that $t_1 + t_2$ is bounded by $(1+\beta)p(B_i)/m + (1 + 1/\beta)\kappa_i' - p_i/\beta$. □

4.2 One Machine Relaxation

In order to use Delay List, we will need to start with a one machine schedule. The following two lemmas provide lower bounds on the optimal m-machine schedule in terms of the optimal one-machine schedule. This one-machine schedule can be either preemptive or non-preemptive, the bounds hold in either case.

Lemma 8 $C^m_{\mathrm{OPT}} \geq C^1_{\mathrm{OPT}}/m$.

Proof: Given a schedule S^m on m machines with total weighted completion time C^m, we will construct a one-machine schedule S^1 with total weighted completion time at most mC^m as follows. Order the jobs according to their completion times in S^m with the jobs completing early coming earlier in the ordering. This ordering is our schedule S^1. Note that there could be idle time in the schedule due to release dates. If $i \overset{\star}{\prec} j$ then $C_i^m \leq s_j^m \leq C_j^m$ which implies that there will be no precedence violations in S^1. We claim that $C_i^1 \leq mC_i^m$ for every job J_i. Let P be the sum of the processing times of all the jobs which finish before J_i (including J_i) in S^m. Let I be the total idle time in the schedule S^m before C_i^m. It is easy to see that $mC_i^m \geq P + I$. We claim that $C_i^1 \leq P + I$. The idle time in the schedule S^1 can be charged to idle time in the schedule S^m and P is the sum of all jobs which come before J_i in S^1. This implies the desired result. □

Lemma 9 $C^m_{\mathrm{OPT}} \geq \sum_i w_i \kappa_i = C^\infty_{\mathrm{OPT}}$.

Proof: The length of the critical path κ_i, is an obvious lower bound on the completion time C_i^m of job J_i. Summing up over all jobs gives the first inequality. It is also easy to see that if the number of machines is unbounded that every job J_i can be scheduled at the earliest time it is available and will finish by κ_i yielding the equality. □

4.3 Obtaining generic m-machine schedules

In this section we derive our main theorem relating m-machine schedules to one-machine schedules.

We begin with a corollary to Theorem 5.

Corollary 3 *Let S^m be the schedule produced by the algorithm* Delay List *using a one-machine 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: Since all jobs in B_i come before J_i in the one-machine schedule, it follows that $p(B_i) \leq C_i^1$. Plugging this and Fact 1 into the bound in Theorem 5, we conclude that $C_i^m \leq (1+\beta)C_i^1/m + (1+1/\beta)\kappa_i$. □

Theorem 6 *Given an instance I of scheduling to minimize sum of weighted completion times and a one-machine schedule for I that is within a factor ρ of an optimal one-machine schedule,* Delay List *gives a m-machine schedule for I that is within a factor $(1+\beta)\rho + (1+1/\beta)$ of an optimal m-machine schedule.*

Proof: Let S^1 be a schedule which is within a factor ρ of the optimal one-machine schedule. Then $C^1 = \sum_i w_i C_i^1 \leq \rho C_{\text{OPT}}^1$. By Corollary 3, the schedule created by the algorithm Delay List satisfies,

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

From Lemmas 8 and 9 it follows that

$$
\begin{aligned}
C^m &\leq \frac{(1+\beta)\rho C_{\text{OPT}}^1}{m} + \left(1+\frac{1}{\beta}\right) C_{\text{OPT}}^\infty \\
&\leq \left((1+\beta)\rho + \left(1+\frac{1}{\beta}\right)\right) C_{\text{OPT}}^m .
\end{aligned}
$$

□

Corollary 4 *There is an $O(n\log n)$ time 4-approximation algorithm for weighted completion time on parallel machines when the precedence graphs are restricted to be series-parallel graphs.*

Proof: The optimal single machine schedule with release dates ignored can be computed in $O(n\log n)$ time for series-parallel graphs [1]. Applying the Delay List algorithm with $\beta = 1$ to this schedule gives the desired result. □

Remark 1 *Since the bounds in our conversion algorithm are job-by-job, the algorithm is applicable to a more general class of metrics as well.*

There is an interesting property of the conversion algorithm that is useful in its applications and worth pointing out explicitly. We explain it via an example. Suppose we want to compute an m-machine schedule with release dates and precedence constraints. From Theorem 6 it would appear that we need to compute a one-machine schedule for the problem that has both precedence constraints and release dates. However, we can completely ignore the release dates in computing the one machine schedule S^1! This follows from a careful examination of the upper bound proved in Theorem 5 and the proof of Theorem 6. This is useful since the approximation ratio for the problem $1|prec|\sum_j w_j C_j$ is 2 [16] while it is 3 for $1|prec, r_j|\sum_j w_j C_j$ [16]. In another example the problem $1||\sum_j w_j C_j$ has a very simple polynomial time algorithm using Smith's ratio rule while $1|r_j|\sum_j w_j C_j$ is NP-hard. Thus release dates play a role only in the conversion algorithm and not in the single machine schedule. A similar claim can be made when there are delays between jobs. In this setting a positive delay d_{ij} between jobs i and j indicates that i is a predecessor of j and that j cannot start until d_{ij} time units after i completes. We can generalize our conversion algorithm and its analysis to handle delays, and obtain the same results as those in Theorems 5 and 6. The only change required is in the definition of ready time of a job which now depends also on the delay after a predecessor finishes. As with release dates we can ignore the delay values (not the precedence constraints implied by them though) in computing the single machine schedule. Munier etal. [24] use linear programming ideas to generalize results for problems with precedence constraints to those with delay constraints.

4.4 Applying conversion to in-tree precedence

We obtain stronger results for in-tree precedence *without* release dates. The problem is strongly NP-hard even for this case. We analyze the standard *list scheduling* algorithm which starts with an ordering on the jobs (the list), and greedily schedules each successive job in the list at the earliest possible time. We use the optimal one-machine schedule for trees as the list. We show that this algorithm gives a 2-approximation for in-trees. Recall that s_i^m is the start time of J_i in the schedule S^m.

Lemma 10 *If S^m is the list schedule using a one-machine schedule S^1 as the list, then for any job J_i, $C_i^m \leq \kappa_i + C_i^1/m$.*

Proof: Since there are no release dates, we can assume that the schedule S^1 has no idle time. Without loss of generality assume that $J_1, \ldots, J_n$ is the ordering of the jobs ordered according to their *start* times s_i^m in S^m (we break ties arbitrarily). We will prove the lemma by induction on i. We strengthen the hypothesis by adding the following invariant. If $C_i^m > C_j^m + p_i$ where J_j is the last predecessor of J_i to finish in S^m, then all the jobs scheduled before s_i^m in S^m are ahead of J_i in the list S^1 and there is no idle time in the schedule, before time s_i^m. In this case it follows that $C_i^m \leq p_i + C_i^1/m$. The base case is trivial since $\kappa_1 = p_1$ and the first job finishes at time p_1. Suppose that the hypothesis holds for all jobs J_k, $k < i$, we will prove it holds for J_i. If J_i has no predecessor it is easily seen that there is no idle time before J_i is scheduled and that $C_i^m \leq p_i + C_i^1/m$. Among the predecessors of i, let J_j, $j < i$ be the last to finish in the schedule S^m (ties are broken arbitrarily). We consider two cases.

1. $C_i^m = C_j^m + p_i$. By the hypothesis, $C_j^m \le \kappa_j + C_j^1/m$. It follows that $C_i^m \le \kappa_i + C_i^1/m$ since $\kappa_i \ge \kappa_j + p_i$ and $C_j^1 < C_i^1$.

2. $C_i^m > C_j^m + p_i$. Let $t = C_j^m$. Let P be the set of jobs which finish exactly at time t and P' be the set of jobs which had their last predecessor running until time t. Note that $J_i \in P'$, $J_j \in P$, and all the jobs in P' are ready to be run at time t. In an in-tree a node has at most one immediate successor therefore $|P'| \le |P|$. Therefore the number of jobs that are ready at t but were not ready at t^- is at most $|P|$. If J_i was not scheduled at t there must exist a job $J_l \notin P'$ which is scheduled at t. This implies that J_l occurs before J_i in the list S^1. Since no immediate predecessor of J_l finished at t, by the induction hypothesis we conclude that there was no idle time and no job which comes later than J_l in S^1 is scheduled before time t. Since J_i was ready at time t, it follows that there is no idle time and no job later than J_i in S^1 is scheduled between time t and the s_i^m. From these observations it follows that $C_i^m \le p_i + C_i^1/m \le \kappa_i + C_i^1/m$.

In both cases we see that the induction hypothesis is established for J_i and this finishes the proof. □

Theorem 7 *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.*

Proof: The proof is similar to that of Theorem 6 except that we use the stronger bounds from Lemma 10. The running time is dominated by the time to compute the optimal one machine schedule which can be done in $O(n \log n)$ time [1]. □

4.5 A 2.83-approximation for scheduling without precedence constraints

We now improve the approximation bound for parallel machine scheduling with release dates to 2.83 which improves the earlier ratio of $2.89 + \epsilon$ [3]. We combine ideas of the one machine relaxation developed in Section 3 and the idea of using delay based list scheduling to derive an alternate algorithm which has worse ratio than the algorithm in Section 3. But we observe that the bounds we get from the analysis of these two algorithms can be combined to get an improved lower bound on the optimal which leads to the improvement.

Recall from Section 3 that $I1$ is the one-machine relaxation for a given instance I and $P1$ is an optimal preemptive schedule for $I1$.

Lemma 11 *If we apply* Delay List *to $P1$ with parameter β, 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. From Theorem 5 and Fact 1 we conclude that $C_j^D \leq (1+\beta)p(B_j)/m + (1+1/\beta)\kappa_j - p_j/\beta$. Since we do not have precedence constraints on the jobs, $\kappa_j = r_j + p_j$. From the definition of B_j and the fact that the list is the order in which jobs finish in $P1$, it follows that $p(B_j)/m \leq C_j^{P1}$. We therefore conclude that $C_j^D \leq (1+\beta)C_j^{P1} + p_j + r_j/\beta$. Summing this over all jobs we obtain

$$\sum_j C_j^D \leq (1+\beta)\sum_j C_j^{P1} + \sum_j p_j + \frac{1}{\beta}\sum_j r_j$$

Since both $\sum_j C_j^{P1}$ and $\sum_j p_j$ are lower bounds on the optimal schedule value, it follows that

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

□

We now balance the two algorithms, list scheduling and Delay List to achieve an approximation ratio of 2.83.

Lemma 12 *For any input I, either list scheduling from $P1$ or using* Delay List *on $P1$ for an appropriate choice of β produces a schedule with $\sum_j C_j \leq 2.83\sum_j C_j^*$ and runs in $O(n\log n)$ time.*

Proof: By (4), we know that

$$\sum_j C_j^N \leq 2\sum_j C_j^{P1} + \sum_j p_j. \tag{5}$$

If, for some α, we know that $\sum_j p_j \leq \alpha\sum_j C_j^*$, 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 $\sum_j C_j^* \geq \sum_j (p_j + r_j)$, we get

$$\sum_j r_j \leq (1-\alpha)\sum_j C_j^*. \tag{6}$$

We can now plug (6) into the upper bound on C_j^D from Lemma 11 to get

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

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 $\sqrt{(1-\alpha)}$. The expression $\min\{2+\alpha, 2+2\sqrt{1-\alpha}\}$ is minimized when $\alpha = 2\sqrt{2}-2$ thus yielding a ratio of $2\sqrt{2} \simeq 2.83$.

To obtain the guaranteed approximation the algorithm runs both the list scheduling algorithm and the Delay List algorithm with $\beta = \sqrt{3-2\sqrt{2}}$, and chooses the better of the two schedules. The schedules can be computed in $O(n\log n)$ time each. □

5 Conclusions

As mentioned earlier, many variants of the problem of minimizing average weighted completion time were shown to have constant factor approximations. Hoogeveen et al [18] investigated the hardness of approximation of average completion time scheduling and in particular showed that the problems $P|prec, p_j = 1|\sum_j C_j$ and $R|r_j|\sum_j C_j$ are APX-hard, in other words they do not admit a PTAS unless $P = NP$. Recently, much progress was made in obtained improved upper bounds as well. Several groups of authors obtained efficient polynomial time approximation schemes for problems involving only release dates. A preliminary version describing these results is [2]. The maximal cases that were shown to have a PTAS are $P|r_j|\sum_j w_j C_j$, $Rm|r_j|\sum_j w_j C_j$, and their preemptive versions. An interesting open problem is the complexity of $1|prec|\sum_j w_j C_j$. A 2-approximation for this problem is known but no APX-hardness has been established. Subsequent to the linear programming based methods [16], simple combinatorial algorithms [6, 8] were developed for this problem matching the approximation ratio of 2. By coupling these algorithms with the DELAY LIST algorithm in this paper we obtain the first efficient and combinatorial approximation algorithms even with precedence constraints and delays. The ratio achieved for $P|r_j, prec|\sum_j w_j C_j$ is 5.83 and is worse than the currently best known ratio of 4 [24]. However the algorithm in [24] is based on solving a linear program via the ellipsoid method.

Acknowledgments

We thank Javed Aslam, Moses Charikar, Cindy Phillips, David Shmoys, Santosh Vempala, and Joel Wein for valuable discussions. We are also grateful to the two anonymous referees for their comments, corrections, and suggestions for improvements.

References

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

[2] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates. In *Proceedings of the 40th Symposium on the Foundations of Computer Science*, 32–43, 1999.

[3] 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*, 646–57, Springer, 1996.

[4] 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. In *Proceeding of the 29th Annual International Symposium on Microarchitechture (MICRO-29)*, 58–67, 1996.

[5] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation Techniques for Average Completion Time Scheduling. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, 609–618, 1997.

[6] C. Chekuri and R. Motwani. Precedence constrained scheduling to minimize weighted completion time on a single machine. In *Discrete Applied Mathematics*, 98:29–38, 1999.

[7] F. Chudak, and D. Shmoys. Approximation Algorithms for Precedence-Constrained Scheduling Problems on Parallel Machines that Run at Different Speeds. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, 581–590, 1997.

[8] F. Chudak and D. Hochbaum. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. In *Operations Research Letters*, 25(5):199–204, 1999.

[9] J. Du, J.Y.T. Leung, and G.H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75:347–355, 1990.

[10] 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.

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

[12] M.X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, 591–598, 1997.

[13] M. Goemans, M. Queyranne, A. Schulz, M. Skutella, and Y. Wang. Personal communication, 1997.

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

[15] 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*, 142–151, 1996.

[16] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Offline and online algorithms. *Math. of Operations Research*, 22:513–544, 1997.

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

[18] J. A. Hoogeveen, P. Schuurman, and G. J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. 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 353–366. Springer, 1998.

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

[20] 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.

[21] 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*, 404–414, 1996.

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

[23] 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).

[24] A. Munier, M. Queyranne, and A. S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. 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 367–382. Springer, 1998.

[25] 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*, 86–97, 1995.

[26] A.S. Schulz and M. Skutella. Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria. Technical Report 533-1996, Fachbereich Mathematik, Technische Universitt Berlin, Berlin, Germany, 1996.

[27] 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*, 301–315, 1996.

[28] A.S. Schulz and M. Skutella. Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria. In *Proceedings of the Fifth Annual European Symposium on Algorithms*, 416–29, 1997.

[29] A. S. Schulz and M. Skutella. Random-based scheduling: New approximations and lp lower bounds. In J. Rolim, editor, *Randomization and Approximation Techniques in Computer Science (RANDOM)*, volume 1269 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 1997.

[30] W.E. Smith. Various optimizers for single-stage production. *Naval Res. Logist. Quart.*, 3:59–66, 1956.

[31] C. Stein and J. Wein. On the Existence of Schedules that are Near-Optimal for both Makespan and Total Weighted Completion Time. In *Operations Research Letters*, 21(3):115–22, 1997.

[32] L. Stougie, 1995. Personal communication cited in [21].

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

[34] E. Torng and P. Uthaisombut. Lower bounds for srpt-subsequence algorithms for nonpreemptive scheduling. Manuscript, July 1998.

[35] 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)*, 105–9, 1987.