Preemptive Bicriteria Scheduling for Parallel Jobs: Off-line and On-line Algorithms

Ahuva W. Mu'alem and Dror G. Feitelson

Institute of Computer Science
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
E-mail: {ahumu, feit}@cs.huji.ac.il

We introduce general algorithms for off-line and on-line scheduling of parallel and independent jobs on a multiprocessor system.

In the off-line case we consider two models by allowing and disallowing release dates. Following Hall, Schulz, Shmoys, and Wein [11], we achieve a constant approximation factor for both minimizing weighted sum of the job completion times and makespan objectives. Following Schulz and Skutella [17] [18] we achieve a smaller *randomized* constant factor for weighted sum of completion times, and a small constant factor for makespan. In both techniques we use an optimal solution to a linear programming relaxation in order to guide a task permutation as an input to a list scheduling [8] with preemption rule.

Our on-line technique gives constant factors for both the (weighted) sum of completion times and makespan objectives, and is a kind of generalization to the one-machine preemptive relaxation introduced by Chekuri, Motwani, Natarajan, and Stein [3]. To the best of our knowledge our on-line algorithms are the first known for the problems under consideration.

Key Words: approximation algorithms, competitive algorithms, randomized rounding, randomized algorithms, multi processor system

1. INTRODUCTION

Scheduling so as to approximate more than one criterion has received little attention in the literature ([23], [19]). There are many simple instances showing that an optimal preemptive schedule for the makespan is not optimal for the (weighted) sum of job completion times and vice versa. We present approximation techniques and new results for several scheduling models of parallel jobs in order to minimize both (weighted) sum of completion times and makespan objectives. In these problems each job may demand more than one processor for execution, and hence is defined as a *parallel* job. Drozdowski [5] gives a detailed overview on parallel jobs scheduling problems.

All scheduling problems under consideration ask for minimization of some objective function. The most common (and apparently relatively simple to analyze) objective function is the *makespan*. The *makespan* of a schedule is the largest completion time of a job in it. This objective reflects the interest of the machine's owner. Generally speaking, when the makespan is small, the utilization is high. Focusing on the user, the time it takes to finish individuals jobs is more important. Thus if many small jobs are postponed after some large job, it is unacceptable to the user even if the makespan is optimal. For that reason other objective functions are studied — so-called min-sum objective criteria e.g., *total completion times*, *total response times* (also known as *total flow times*), and *total waiting times*. The weighted variants of these objectives are considered as well; in this case each job has a weight and we take the weighted sums. This captures the situation when some jobs are more important than the others.

It seemed that significantly there are much more results in the area for approximating total completion times than for total flow times [5], although at optimality the two objectives are equivalent. However, the total completion times objective is concerned with maximizing the total number of finished jobs at any moment, and total flow times deals with fairness for each job [14].

The models. We are given m identical processors and n independent parallel jobs to be scheduled. The jobs are independent in the sense that no precedence constraints are imposed. Each job $j \in N = \{1, \ldots, n\}$ has the following two integral parameters: a positive processing time p_j, and a number of processors needed for execution m_j, $1 \leq m_j \leq m$. A machine can process at most one job at a time. All the m_j processors are required to start the job j at the same starting time; any set of m_j processors is proper for execution the job j. A job may be preempted from time to time and later continued to run on a possibly different set of m_j processors, without affecting its total processing time p_j. In addition, each job j may have either one or both of the following two integral parameters: a release date $r_j \geq 0$ before which it cannot be processed, and a positive weight w_j.

We sometimes refer to N as the *sequence* of the input jobs, and sometimes as the *set* of the input jobs. For simplicity we use the following notations: Let $a_j = p_j m_j$ denote the total "area" which is occupied by job j. Let $r_j^{max} = \max_{k=1,\ldots,j} r_k$ and $p_j^{max} = \max_{k=1,\ldots,j} p_k$ denote the largest release date and the largest processing time among the jobs $1, \ldots, j$, respectively.

Let C_j denote the completion time of the job j in a feasible schedule; the goal is to minimize $\sum_j C_j$ or $\sum_j w_j C_j$, together with minimizing the makespan, the largest C_j, denoted by C_{max}.

It is convenient to refer to scheduling models by the notation of Graham, Lawler, Lenstra, and Rinnooy Kan [9]. Each Problem is denoted by $\alpha|\beta|\gamma$. In our case α is either 1 or P, denoting that the scheduling environment contains either one processor, or m identical processors; β optionally contains $r_j, pmtn, m_j$[1] indicating whether there are non-trivial release date constraints, whether preemption is allowed, and whether more than one processor may be required by a job; γ indicates the objectives $\sum w_j C_j$ or $\sum C_j$ and/or C_{max}. For example $P|r_j, pmtn|\sum w_j C_j$ denotes the *identical parallel machine* model in which the goal is minimizing the weighted sum of completion times on m identical processors subject to release dates, with preemption allowed, and each job demanding a single processor for execution.

We consider three models. We call $P|m_j, r_j, pmtn|\sum w_j C_j,\ C_{max}$ the *general model*, and $P|m_j, r_j, pmtn|\sum C_j,\ C_{max}$ the *unweighted general model*. Note that any result for the general model applies to the unweighted general model, since one can refer to (a system of) unweighted jobs as having equal weights. For brevity we sometimes refer to the general model and to the unweighted general model as *the general models*. The *restricted model* has no release dates and denotes $P|m_j, pmtn|\sum w_j C_j,\ C_{max}$.

All our three models are known to be NP-hard. The general model is harder than the model $1|r_j, pmtn|\sum w_j C_j$ which was shown to be NP-hard by Labetoulle, Lawler, Lenstra, and Rinnooy Kan [13]. The unweighted general model and the restricted model are harder than the problem $P|m_j, pmtn|C_{max}$ which Drozdowski [4] proved is NP-hard.

Review of Results. Our algorithms and their analysis are heavily inspired by the work of Philips, Stein, and Wein [14]. They introduced algorithms for converting preemptive schedules into nonpreemptive ones, by scheduling the jobs in the *order of their completion times* in a preemptive schedule. This technique was generalized in many deterministic and randomized off-line and on-line results (e.g., [11], [17], [3]). We use this "ordering technique" in our simulations and relaxations to derive a permutation on the given input to our modified parallel list algorithms.

[1] m_j here differs from its original definition by Graham et al. [9] where it was used to denote the number of operations in a job shop model (they did not refer to parallel jobs). Drozdowski [5] uses instead the notation $size_j$ to denote the number of processors needed for execution of a parallel job.

Mathematically speaking, if $\bar{C}_j$ represents the job j completion time found by a relaxation or simulation, and assuming w.l.o.g the the jobs are indexed so that $\bar{C}_1 \leq \bar{C}_2 \leq \cdots \leq \bar{C}_n$, we usually derive the following bounds for $j = 1, \ldots, n$:

- $\bar{C}_j \geq p_j^{max}$.
- $\bar{C}_j \geq r_j^{max}$.
- $\bar{C}_j \geq c_1 \cdot \sum_{k=1}^{j} \frac{m_k p_k}{m}$, for some constant $0 \leq c_1 \leq 1$.
- $\sum_j \bar{C}_j \leq c_2 \cdot \sum_j C_j^*$, where $\sum_j C_j^*$ is the value of the optimal schedule with respect to the sum of completion times, and c_2 is a constant ≥ 1.

In the **off-line** case all the input data is known in advance, and the scheduler wishes to compute the optimal schedule. A (randomized) ρ-approximation algorithm is a polynomial time off-line algorithm that produces a feasible solution whose (expected) value is within a factor of ρ of the optimum value; ρ is also called the (expected) performance guarantee of the algorithm. The expectation is taken over all the random choices made by the algorithm.

As it is usually very hard to find the optimal solutions, in the off-line case we compare the output of an algorithm with a lower bound given by an optimum solution to a certain linear program (LP relaxation). Our relaxations are in fact integer programs with necessary constraints for a feasible schedule. Therefore, our integer programs compute lower bounds. Since it is hard to solve integral programs, we allow rational solutions, and hence the "integral" lower bounds are bounded from below.

Following Hall, Schulz, Shmoys, and Wein [11], we achieve a constant approximation factor for both minimizing weighted sum of the job completion times and makespan objectives[2]. Following Schulz and Skutella [17] [18] we achieve a smaller *randomized* constant factor for weighted sum of completion times, and a small constant factor for makespan. In both techniques we use an optimal solution to a linear programming relaxation in order to guide a task permutation as an input to a list scheduling [8] with preemption rule.

In the **on-line** case we schedule jobs that continually arrive. For each time t, we construct a schedule *until time* t, without any knowledge regarding the jobs that will arrive afterwards. Naturally, in the on-line case we refer solely to the general models (with the non-trivial release dates).

[2]The observation that the algorithms of Hall et al. [11] are approximations for makespan objective as well was made afterwards and appeared in the literature (e.g. [21]), and was made independently by us.

An on-line scheduler is (randomized) f-competitive if for any finite input sequence N of jobs the (expected) value obtained by the on-line scheduler on N divided by the best off-line scheduler on N is at most f. If a scheduler is f-competitive we sometimes say that the scheduler achieves a competitive ratio of f or that it has performance guarantee of f.

The on-line technique that we introduce gives a constant competitive ratio factor for the (weighted) sum of completion times, and constant competitive ratio for makespan, and is a kind of generalization to the one-machine preemptive relaxation introduced by Chekuri, Motwani, Natarajan, and Stein [3]. Turek et al. used a similar technique for deriving a lower bound for the sum of completion times in the restricted model (with no release dates), called the squashed area bound [25]. To the best of our knowledge our on-line algorithms are the first known for the general models [21].

Related work. Most of the constant factor approximations and competitive algorithms for scheduling problems with release dates so as to minimize (weighted) sum of completion time were developed in the last five years. In the following we first relate to the most "classical" ones amongst them (not necessarily the best results) considering single machine and identical parallel machine models. However, for parallel jobs almost nothing is known about the approximability of the general models (with the non-trivial release dates).

It is well known that algorithm SRPT [2] which always schedules the shortest remaining processing time job is a polynomial optimal on-line algorithm for the problem $1|r_j, pmtn|\sum C_j$. This problem is the hardest scheduling problem which is polynomially solvable in the sense that imposing harder constraints makes it a NP problem.

Goemans, Wein, and Williamson [10] proved that the on-line preemptive algorithm which schedules the jobs in order of nonincreasing ratios w_j/p_j is 2-competitive for the model $1|r_j, pmtn|\sum w_j C_j$. Its simple and short proof is quoted in [16]. Schulz and Skutella [16] gave a randomized preemptive list based algorithm which is 4/3-competitive for $1|r_j, pmtn|\sum w_j C_j$.

Phillips, Stein, and Wein [14] showed that the on-line algorithm SPT that always runs the m jobs with the shortest remaining processing time achieves a competitive ratio of 2 for total sum of completion times for the model $P|r_j, pmtn|\sum C_j$.

Hall, Schulz, Shmoys, and Wein [11] achieved an off-line $(4-\frac{1}{m})$-approximation and an on-line $(4+\epsilon)$-competitive algorithm for $P|r_j|\sum w_j C_j$.

Chekuri, Motwani, Natarajan, and Stein achieved an off-line 2.85-approximation and an on-line 3-competitive algorithm for the model $P|r_j|\sum C_j$ (that is, with no weights and no preemptions allowed) [3].

Schulz and Skutella [17] achieved an off-line $(2+\epsilon)$-approximation algorithm and an on-line 2-competitive randomized algorithm for $P|r_j|\sum w_j C_j$.

Very recently the first polynomial time approximation schemes (PTASs) for minimizing the (weighted) sum of completion times for several scheduling models have been presented by Afrati et al. [1]. PTAS is an algorithm that for any fixed $\epsilon \geq 0$ finds an off-line solution within a $(1+\epsilon)$ factor of the optimum in polynomial time. The PTASs in [1] refer among other to: $1|r_j|\sum w_j C_j$, $1|r_j, pmtn|\sum w_j C_j$, $P|r_j|\sum w_j C_j$, and $P|r_j, pmtn|\sum w_j C_j$.

For parallel jobs with disallowing preemptions and trivial release dates Schwiegelshohn, Ludwig, Wolf, Turek, and Yu [20] introduced the SMART off-line algorithms. Schwiegelshohn et al. acheived an 8-approximation for the model $P|m_j|\sum C_j$ and a 10.45-approximation for the weighted model $P|m_j|\sum w_j C_j$. Schwiegelshohn [19] presented several versions of the off-line preemptive deterministic algorithm PSRS, which can be fine tuned to minimize either sum of completion times or makespan. He considered penalties for preemptions and transforming a preemptive schedule into a nonpreemptive one. Part of his related results (with no imposed penalties) are summarized in Table 1.

TABLE 1.

Summary of Schwiegelshohn's off-line results [19]

model	approx. factor for $\sum w_j C_j$	approx. factor for C_{max}
$P|m_j, pmtn|\sum w_j C_j,\ C_{max}$	2.37	3.20
$P|m_j, pmtn|\sum w_j C_j,\ C_{max}$	2.42	3.00
$P|m_j|\sum w_j C_j,\ C_{max}$	7.11	9.60
$P|m_j|\sum w_j C_j,\ C_{max}$	7.26	9.00

Organization of this paper. In section 2 we review a nonpreemptive list scheduling algorithm introduced by Garey and Graham [8] and then modify it to allow preemptions. In section 3 we continue to modify it to handle release dates as well. In the next sections we consider our three models: the general, the unweighted general, and the restricted. The results are summarized in tables 2 through 4. Section dependencies are listed in Table 5.

TABLE 2.

Summary of the off-line results for the restricted model: $P|m_j, pmtn|\sum w_jC_j,\ C_{max}$.

type	approx. factor		sect	comment
	for $\sum w_jC_j$	for C_{max}		
deterministic	4	2	4	
randomized	3	2	5	pseudo-polynomial-time algorithm
randomized	$3+\epsilon$	2	6	polynomization of the above randomized algorithm

TABLE 3.

Summary of the off-line results for the general model: $P|m_j, r_j, pmtn|\sum w_jC_j,\ C_{max}$.

type	approx. factor		sect	comment
	for $\sum w_jC_j$	for C_{max}		
deterministic	5	3	4	
randomized	4	3	5	pseudo-polynomial-time algorithm
randomized	$4+\epsilon$	3	6	polynomization of the above randomized algorithm
deterministic	$4+\epsilon$	3	6	polynomization and derandomization of the above randomized algorithm

TABLE 4.

Summary of the on-line results.

model	type	competitive ratio		sect
		for $\sum(w_j)C_j$	for C_{max}	
$P\|m_j, r_j, pmtn\|\sum C_j,\ C_{max}$	deterministic	9	4	7
$P\|m_j, r_j, pmtn\|\sum w_jC_j,\ C_{max}$	deterministic	18	4	7
$P\|m_j, r_j, pmtn\|\sum w_jC_j,\ C_{max}$	randomized	12	4	7

2. PREEMPTIONS CAN HELP

First we review a nonpreemptive parallel list scheduling algorithm introduced by Garey and Graham [8] for the model $P|m_j|C_{max}$. Then we modify the algorithm to handle preemptions. In the next section we modify the preemptive version of the parallel list algorithm to handle the case of non-trivial release dates. The two modifications produces the algorithm List-GG-PR. Its main idea is that jobs are packed onto selves in the order

TABLE 5.

Sections Dependencies.

section	section dependencies
3	2
4	2, 3
5	2, 3
6	2, 3, 5
7	3

defined by the input list. If the next job in the list requires too many processors, it is skipped, and jobs further down in the list are considered. Thus the shelf is considered full only whan no additional job from the list can fit in. However, the height of each shelf is only until the first job in the shelf terminates, or a new job that precedes some scheduled job is released, whichever comes first. When this happens, all jobs are preempted and a new shelf is packed in the same way.

The virtue of adding preemptions is that each completion time of a job in the modified algorithm is solely dictated by its predecessors in the input permutation and not by successive jobs, consistently with the classical list algorithm developed by Graham in 1966 for the identical parallel machine model, and in contrast with the original parallel list algorithm [8]. This enables us to generalize techniques which were developed for other machine models.

Here we use and quote without proofs some results that appear in Turek, Wolf, and Yu [24]. Note that the optimum makespan is bounded from below by (for example see [24]):

$$\text{Makespan}_{(\mathsf{Opt})} \geq \max\left\{\sum_{k=1}^{n} \frac{a_k}{m},\ r_n^{max},\ p_n^{max}\right\}.$$

This trivial result holds for both the preemptive and the nonpreemptive cases.

Nonpreemptive Algorithm List-GG [8]:

A job list $\bar{L}$ is given, ordered by some permutation π, each job in the list has $r_j = 0$.

Base step: The first job in the list is scheduled at time $\tau_1 := 0$ and removed from $\bar{L}$.

Iterative step: $\hat{j}_i$ is selected from the remaining jobs in $\bar{L}$ by scanning through $\bar{L}$ in order and selecting the job in the list that can be scheduled at the minimum time $\tau_i \geq \tau_{i-1}$ (ties are broken by selecting the first such job in the list). Job $\hat{j}_i$ is then removed from $\bar{L}$, and the step is repeated until the list $\bar{L}$ is empty.

LEMMA 2.1. *The makespan produced by* List-GG *is bounded from above by:*

$$\text{Makespan}_{(\mathsf{List-GG})} \leq 2 \cdot \max\left\{\sum_{k=1}^{n} \frac{a_k}{m},\ p_n^{max}\right\}.$$

Therefore, List-GG is a 2-approximation to the model $P|m_j|C_{max}$. For proof see Turek et al. [24].

Note that this upper bound is asymptomatically tight. Consider the input: $n = m \cdot (m-1) + 1,\ m_j = 1$ and $r_j = 0$ for $j = 1, \ldots, n$. As the optimal permutation for makespan consider $p_1 = m,\ p_k = 1\ , k = 2, \ldots, n$. As a nonoptimal schedule for makespan consider $p_k = 1,\ k = 1, \ldots, n-1,\ p_n = m$.

Preemptive Shelf-Based Algorithm List-GG-P:
A job list $\bar{L}$ is given, ordered by some permutation π. Each job in the list has $r_j = 0$. In each iteration we update the remaining processing times $p_k,\ k = 1, \ldots, n$.
Base step: set $\tau_1 := 0,\ i := 1$.
Iterative step: Construct shelf i: schedule the first job in the list $\bar{L}$ at time τ_i. Greedily select the next jobs to be scheduled from the remaining jobs in $\bar{L}$ by scanning through $\bar{L}$ in order and selecting the next job on the list such that its processors demand is not greater than the number of the currently free processors at time τ_i. Now, run the jobs in shelf i. When the shortest job, say $\hat{j}$, in shelf i terminates, remove it from $\bar{L}$, set $\tau_{i+1} := \tau_i + p_{\hat{j}},\ i := i+1$, preempt all remaining jobs in shelf i, and update the remaining processing times $p_k := p_k - p_{\hat{j}},\ k \neq \hat{j},\ k \in$ shelf i. The step is repeated until the list $\bar{L}$ is empty.

Note that in each iteration of List-GG-P at least one job completes its execution and is removed from $\bar{L}$. The greedy style of List-GG-P causes a preemption to affect only the successive jobs following the removed job(s) in the ordered list $\bar{L}$; the head of the successive shelf remains unchanged, however the tail of the successive shelf may be different, since some other jobs may be selected. Equivalently, the completion times produced by List-GG-P on an induced ordered list consisting of any prefix of $\bar{L}$ are identical to the respective completion times produced by List-GG-P on the entire list

$\bar{L}$. In addition, note that the number of (shelf) preemptions is bounded from above by n: in the worst case only a single job terminates in each shelf.

Consider the following ordered job system J: $m_1 \leq m_2 \leq \cdots \leq m_n,\ p_1 \leq p_2 \leq \cdots \leq p_n$. The completion times of List-GG-P on J are identical to the completion times of List-GG on J. This implies that the way in which we add the preemption to List-GG does not necessarily improve the bounds on the achieved makespan. In the next lemma we shall see that the upper bound stated in Lemma 2.1 still holds.

For a given job list $\bar{L}$ ordered by π consider the fragment jobs list $\hat{L}$ created by running List-GG-P on $\bar{L}$, and splitting each job j into sub jobs $\hat{j}_1, \hat{j}_2, \ldots$ with respect to the natural way dictated by the shelves and the preemptions that occurred. Note that $\sum_{k\in\hat{L}} \frac{a_k}{m} \equiv \sum_{k\in\bar{L}} \frac{a_k}{m}$ since the total areas sum of the jobs in both systems $\bar{L}$ and $\hat{L}$ are equivalent. In the fragment system the maximum processing time of the sub jobs may be reduced, and hence $\max_{k\in\hat{L}} p_k \leq \max_{k\in\bar{L}} p_k$.

LEMMA 2.2. *The makespan produced by* List-GG-P *is bounded from above by:*

$$\text{Makespan}_{(\text{List-GG-P})} \leq 2 \cdot \max\left\{\sum_{k=1}^{n} \frac{a_k}{m},\ p_n^{max}\right\}.$$

Therefore, List-GG-P is a 2-approximation for $P|m_j, pmtn|C_{max}$.

Proof. Consider the following natural permutation $\hat{\pi}$ on $\hat{L}$ constructed by running List-GG-P on the system $(\bar{L},\pi)$: let the first jobs in $\hat{\pi}$ be the sub jobs in the first shelf in the same order they were scheduled into that shelf. Let the next jobs in $\hat{\pi}$ be the sub jobs in the second shelf in the same scheduling order, and so on. Run List-GG on the fragment system $\hat{L}$ permuted by $\hat{\pi}$. Since List-GG is a greedy algorithm, the resulting feasible schedule of List-GG on $(\hat{L}, \hat{\pi})$ is identical to the schedule constructed by List-GG-P on $(\bar{L}, \pi)$. Hence from lemma 2.1 we get:

$$Makespan_{(\text{List-GG-P})}(\bar{L}, \pi) \equiv Makespan_{(\text{List-GG})}(\hat{L}, \hat{\pi}) \leq$$

$$\leq\ 2 \cdot \max\left\{\sum_{k\in\hat{L}} \frac{a_k}{m},\ \max_{k\in\hat{L}} p_k\right\}\ \leq\ 2 \cdot \max\left\{\sum_{k\in\bar{L}} \frac{a_k}{m},\ \max_{k\in\bar{L}} p_k\right\}.$$

∎

COROLLARY 2.1. *Given any permutation π on the input, the completion time of job $\pi(j)$ in* List-GG-P *in the model $P|m_j, pmtn|C_{max}$ is bounded from above by:*

$$2 \cdot \max\left\{ \sum_{\pi(k)\le\pi(j)} \frac{a_{\pi(k)}}{m},\ p^{max}_{\pi(j)} \right\},$$

where

$$p^{max}_{\pi(j)} = \max_{\pi(k)\le\pi(j)} p_{\pi(k)} \ \text{ and } \ a_{\pi(k)} = m_{\pi(k)} p_{\pi(k)}.$$

3. ADDING RELEASE DATES TO LIST-GG-P

Recall that the classical list algorithm of Graham for identical parallel machine, that schedules the next job in the input list on the least loaded machine, is on-line with the same performance guarantee as in the off-line case, and that the same analysis applies to both the off-line and the on-line cases. The same holds for the parallel list algorithm List-GG. In this section we analyze the notion of adding release dates to List-GG-P. This yields an on-line algorithm with competitive ratio of 3 for makespan for the model $P|m_j, r_j, pmtn|C_{max}$.

Preemptive Shelf-Based Algorithm with Release Dates List-GG-PR:
A job list $\bar{L}$ is given, ordered by some permutation π. Each job in the list has $r_j \ge 0$. In each iteration we update the remaining processing times $p_k,\ k = 1, \ldots, n$.
Base step: Set $\tau_1 := \min_{j\in\bar{L}} r_j,\ i := 1$.
Iterative step: Construct shelf i: schedule the first job j in the list $\bar{L}$ such that $r_j \le \tau_i$. Greedily select the next jobs to be scheduled from the remaining jobs in $\bar{L}$ by scanning through $\bar{L}$ in order and selecting the next job on the list that its processors demand is not greater than the number of the current free processors at time τ_i, and that its release date is not greater than τ_i. Now, run the jobs in the shelf i. Preempt shelf i from running at the first point in time at which the shortest job in shelf i terminates, or some job which is placed in the permutation before a certain job in the current shelf has been released, whichever is earlier. Now, update the remaining processing times of the jobs in the current shelf. If some job has completed its execution remove it from $\bar{L}$. Set $i := i + 1$. Then set τ_{i+1} to

be the current time, or $\tau_{i+1} = \min_{j \in \bar{L}} r_j$ whichever is larger. The step is repeated until the list $\bar{L}$ is empty.

Note that the completion times produced by List-GG-PR on an induced ordered list consisting of any prefix of $\bar{L}$, are identical to the respective completion times produced by List-GG-PR on the entire list $\bar{L}$. In addition, note that the number of (shelf) preemptions is bounded from above by $2n$: in the worst case each job release and each job completion cause a preemption to a shelf.

The most important effect of adding the notion of non-trivial release dates is that we have now a "realistic" on-line algorithm. Jobs may be added dynamically to any place in the permutation right after being released. Note that the analysis of both the off-line and on-line cases is the same, since the algorithm does not schedule a job before its release.

LEMMA 3.1. *Allowing release dates, the makespan produced by* List-GG-PR *is bounded from above by:*

$$\begin{aligned}\text{Makespan}_{(\mathsf{List-GG-PR})} &\leq r_n^{max} + 2 \cdot \max\left\{\sum_{k=1}^{n} \frac{a_k}{m},\ p_n^{max}\right\}\\ &\leq 3 \cdot \text{Makespan}_{(\mathsf{OPT})}.\end{aligned}$$

Therefore, List-GG-PR is a 3-approximation to the model $P|m_j, r_j, pmtn|C_{max}$.

COROLLARY 3.1. *Given any permutation on the input, the completion time of job the $\pi(j)$ in* List-GG-PR *in a model $P|m_j, r_j, pmtn|C_{max}$ is bounded from above by:*

$$r_{\pi(j)}^{max} + 2 \cdot \max\left\{\sum_{\pi(k) \leq \pi(j)} \frac{a_{\pi(k)}}{m},\ p_{\pi(k)}^{max}\right\},$$

where

$$r_{\pi(j)}^{max} = \max_{\pi(k) \leq \pi(j)} r_{\pi(k)}\ ,\ \ p_{\pi(j)}^{max} = \max_{\pi(k) \leq \pi(j)} p_{\pi(k)}\ ,\text{and}\ \ a_{\pi(k)} = m_{\pi(k)} p_{\pi(k)}.$$

4. DETERMINISTIC LP RELAXATION

Following Hall, Schulz, Shmoys, and Wein [11], we achieve a constant approximation factor for both the weighted sum of completion times objective and the makespan objective. Our algorithms in this section are "natural" preemptive generalization of the nonpreemptive techniques of [11] for the identical parallel machine model. We first introduce an algorithm for the general model $P|m_j, r_j, pmtn|\sum w_j C_j,\ C_{max}$, then we conclude an algorithm for the restricted model.

From now on we use the following notations. Consider an optimal schedule that minimize $\sum_j (w_j) C_j$, $j \in N$, and let C_j^* denote the completion time of job j in it; thus $\sum_j (w_j) C_j^*$ denotes the optimal (weighted) sum of completion times value. We shall use the notation $\bar{C}_j$ to denote the value assigned to the job j by a relaxation. Therefore $\sum_j (w_j)\bar{C}_j$ is a lower bound on $\sum_j (w_j) C_j^*$. We use the notation $\tilde{C}_j$ to denote the completion time of job j in the schedule that our algorithm computes.

LEMMA 4.1. *Let $C_1, \ldots, C_n$ denote the job completion times in a feasible schedule. Then the following (exponentially many) inequalities are hold:*

$$\sum_{j\in S} a_j C_j \geq \frac{1}{2m}\left(\left(\sum_{j\in S} a_j\right)^2 + \sum_{j\in S} a_j^2\right) \quad \textit{for each } S \subseteq N. \tag{1}$$

Proof. Without loss of generality, assume there is no forced idle time in the schedule, and that the jobs are indexed so that $C_1 \leq C_2 \leq \cdots \leq C_n$. Consider the schedule induced on the subset of jobs $J = \{1, 2, \ldots, j\}$. Among the jobs of J the last job to finish is j. Recall that C_j is bounded from below by $\sum_{k=1}^{j} \frac{m_k p_k}{m} = \sum_{k=1}^{j} \frac{a_k}{m}$, the lower bound on the makespan of J in any preemptive or nonpreemptive schedule. But then

$$\sum_{j=1}^{n} a_j C_j \;\geq\; \frac{1}{m}\sum_{j=1}^{n}\left(a_j \sum_{k=1}^{j} a_k\right) \;=\; \frac{1}{m}\sum_{1\leq i\leq j\leq n} a_i a_j \;=\; \frac{1}{2m}\left(\left(\sum_{j=1}^{n} a_j\right)^2 + \sum_{j=1}^{n} a_j^2\right),$$

so inequality (1) is true for $S = N$. The general case follows from the fact that the restriction of a schedule for the entire set of jobs to a subset can be interpreted as a schedule for this subset. ∎

LEMMA 4.2. *Let $C_1, \ldots, C_n$ be arbitrary numbers (not necessarily representing a feasible schedule) that satisfy (1) and assume that $C_1 \leq \cdots \leq C_n$. Let $S = \{1, \ldots, j\}$, $1 \leq j \leq n$. Then:*

$$C_j \;\geq\; \frac{1}{2m}\sum_{k=1}^{j} a_k.$$

Proof. Since $C_k \leq C_j$ for each $k = 1, 2, \ldots, j$ and using (1), we have

$$C_j \sum_{k=1}^{j} a_k \;\geq\; \sum_{k=1}^{j} a_k C_k \;\geq\; \frac{1}{2m}\left(\left(\sum_{k=1}^{j} a_k\right)^2 + \sum_{k=1}^{j} a_k^2\right) \;\geq\; \frac{1}{2m}\left(\sum_{k=1}^{j} a_k\right)^2$$

from which we obtain the result. ▮

We now introduce the following LP relaxation, denoted by **LP1**. Both constraints in the following linear program **LP1** are necessary conditions for any feasible schedule. The first constraint ensures that the completion time of a job will not be less than its length. The second constraint is determined by the inequalities (1).

$$\text{minimize} \;\; \sum_{j=1}^{n} w_j C_j$$

(LP1) subject to:

$$C_j \geq p_j + r_j \qquad\qquad \forall j \in N$$

$$\sum_{j \in S} a_j C_j \geq \frac{1}{2m}\left(\left(\sum_{j \in S} a_j\right)^2 + \sum_{j \in S} a_j^2\right) \qquad \forall S \subseteq N$$

Algorithm Start-Parallel-Jobs-by-$\bar{C}_j$ works as follows. We first compute an optimal solution $\bar{C}_1, \ldots, \bar{C}_n$ to **LP1**; we assume that $\bar{C}_1 \leq \cdots \leq \bar{C}_n$ (by re-indexing). Then we use this permutation, which was dictated by the linear program, as an input to the algorithm List-GG-PR.

Note that we use **LP1** to set the input permutation to List-GG-PR. Hence Start-Parallel-Jobs-by-$\bar{C}_j$ is a 3-approximation for the makespan.

LEMMA 4.3. *Let $\bar{C}_1 \leq \cdots \leq \bar{C}_n$ be an optimal solution to* **LP1**, *and let $\tilde{C}_1, \ldots, \tilde{C}_n$ denote the completion times in the schedule found by* Start-Parallel-Jobs-by-$\bar{C}_j$. *For each $j = 1, \ldots, n$ in the model $P|m_j, r_j, pmtn|\sum w_j C_j, C_{max}$,*

$$\tilde{C}_j \leq 5\,\bar{C}_j.$$

Proof. Since we use List-GG-PR, any job j solely depends on its predecessors in the permutation, $k = 1, \ldots, j$. Hence, from corollary 3.1 we get:

$$\tilde{C}_j \leq r_j^{max} + 2 \cdot \max\left\{\sum_{k=1}^{j} \frac{a_k}{m}\;,\; p_j^{max}\right\}.$$

Recall that $\bar{C}_j \geq \bar{C}_k$ for $k = 1, \ldots, j$. From the first constraint in **LP1** we have $\bar{C}_j \geq \bar{C}_k \geq \max\{p_k, r_k\}$ for $k = 1, \ldots, j$. Namely, $\bar{C}_j \geq \max\{p_j^{max}, r_j^{max}\}$. Hence, $\tilde{C}_j \leq \bar{C}_j + 2 \cdot \max\left\{\sum_{k=1}^{j} \frac{a_k}{m}, \bar{C}_j\right\}$. By Lemma 4.2 $2\bar{C}_j \geq \frac{1}{m}\sum_{k=1}^{j} a_k$. Therefore, $\tilde{C}_j \leq \bar{C}_j + 2 \cdot \max\left\{2\bar{C}_j, \bar{C}_j\right\} = 5\bar{C}_j$. ▮

Hence we conclude that $\sum w_j \tilde{C}_j \leq 5 \cdot \sum w_j \bar{C}_j \leq 5 \cdot \sum w_j C_j^*$ where $\sum w_j C_j^*$ is the value of the optimal schedule for minimizing the weighted sum of completions.

To solve the linear program **LP1** in polynomial time we use the ellipsoid algorithm; the separability of the constraints follows from the fact that (1) is merely a rescaled version of the inequalities for a single machine scheduling model which Queyranne [15] proved are separable.

Corollary 4.1. *For the general model $P|m_j, r_j, pmtn|\sum w_j C_j$, C_{max} algorithm* Start-Parallel-Jobs-by-$\bar{C}_j$ *is a 5-approximation for weighted sum of completion times, and a 3-approximation for makespan.*

Corollary 4.2. *By setting the release dates in* **LP1** *to zero and by using* List-GG-P *instead of* List-GG-PR *in algorithm* Start-Parallel-Jobs-by-$\bar{C}_j$, *we get for the restricted model $P|m_j, pmtn|\sum w_j C_j, C_{max}$ a 4-approximation for the weighted sum of completion times, and a 2-approximation for the makespan.*

5. RANDOMIZED LP RELAXATION

Following Schulz and Skutella ([17], [18]), we introduce a randomized technique. Schulz and Skutella achieve with this technique a $(2 + \epsilon)$-approximation algorithm for $P|r_j|\sum w_j C_j$.

We first introduce an algorithm for the general model, then we conclude performance guarantee for the restricted model.

Let $T = r_j^{max} + \sum_{j \in N} p_j - 1$ be the time horizon. Note that the makespan is bounded from above by $T + 1$. For each $j \in N$ and each point $t = 0, \ldots, T$ in time, the variable x_{jt} represents the duration that job j is being processed within the time interval $(t, t + 1]$. The variable x_{jt} should be zero before the release of job j.

The following LP relaxation, denoted by **LP2**, is a generalization of Schulz and Skutella's linear program for identical parallel machine model, which in turn extends the single machine LP relaxation of Dyer and Wolsey[6]. Unfortunately the following formulation has exponentially many variables. However, in the next section, following Schulz and Skutella ([17], [18]), we overcome this drawback with other variables which are not associated with time intervals of length 1, but rather with intervals of geometrically increasing size.

$$\text{minimize} \sum_{j=1}^{n} w_j C_j$$

(LP2) subject to:

$$\begin{array}{lll}
(1) & \displaystyle\sum_{t=0}^{T} \frac{x_{jt}}{p_j} = 1 & \forall j \in N \\
(2) & \displaystyle\sum_{j \in N} x_{jt}\, m_j \leq m & t = 0, \ldots, T \\
(3) & C_j \geq p_j & \forall j \in N \\
(4) & \displaystyle C_j \geq \frac{p_j}{2} + \sum_{t=0}^{T} \frac{x_{jt}}{p_j}\,(t + \frac{1}{2}) & \forall j \in N \\
(5) & x_{jt} = 0 & \forall j \in N \text{ and } t = 0, \ldots, r_j - 1 \\
(6) & 0 \leq x_{jt} \leq 1 & \forall j \in N \text{ and } t = r_j, \ldots, T
\end{array}$$

Equation (1) says that all the fractions of a job (which later on we shall interpret as probabilities) must sum up to the whole job. Inequality (2) is the machine's capacity constraint. Constraint (3) assures that the completion time of a job will not be less than its length p_j.

Consider an arbitrary feasible schedule, where job j is being continuously processed between $C_j - p_j$ and C_j on m_j machines. Then the expression for C_j in (4) corresponds to the real completion time, if we assign the values to the LP variables x_{jt} as defined above, i.e., $x_{jt} = 1$ if j is being processed in the time interval $(t, t+1]$, otherwise $x_{jt} = 0$. If j is not being continuously processed but preempted once or several times, the expression for C_j in (4) is a lower bound for the real completion time.

The following algorithm takes an optimum LP solution, and then constructs a feasible schedule by using a kind of randomized rounding.

Algorithm Multi-Random-Assignment works as follows: Compute an optimum solution x to **LP2**. Assign job j to t at random with probability $\frac{x_{jt}}{p_j}$. Define the random assignment t for job j as t_j. Sort the jobs according to nondecreasing order of t_j (ties are broken randomly), and use this permutation as an input to algorithm List-GG-PR.

Note that all the random assignments are to be performed independently from each other, and that the number of random assignments is exactly n, the number of jobs.

Let $\bar{C}_j$ denote the completion time of the job j by the solution x of **LP2**, and $\tilde{C}_j$ denote the completion time in the schedule computed by Multi-Random-Assignment.

LEMMA 5.1. *The expected completion time of the job j in the schedule computed by* Multi-Random-Assignment *for the general model is bounded from above by 4 times the optimum LP relaxation completion time.*

Proof. We first consider the conditional expectation of the j's job completion time under the assumption that j was assigned to t. We denote this conditional expectation by $E_t\,[\tilde{C}_j]$.

It follows from constraint (5) and the construction of Multi-Random-Assignment that the maximum release date of jobs which were placed in the input permutation to List-GG-PR before j is bounded from above by t.

By Corollary 3.1:

$$E_t[\tilde{C}_j] \;\leq\; t + 2\cdot\max\left\{ \left(\sum_{k\neq j} \left(\frac{a_k}{m} \;\cdot\; \Pr[\;k \text{ before } j\;] \right) \right) + \;\frac{a_j}{m}\;,\; p_k\cdot \Pr[\;k \text{ before } j\;]\;,\; p_j \right\}$$

By replacing the probability expression we get, and using the fact that in algorithm Multi-Random-Assignment ties are broken randomly:

$$\begin{aligned}\leq\; t \;+\; 2\cdot\max\; &\left\{ \left(\sum_{k\neq j} \frac{m_k\,p_k}{m} \cdot \left(\sum_{l=0}^{t-1} \frac{x_{kl}}{p_k} + \frac{1}{2}\frac{x_{kt}}{p_k} \right) \right) \;+\; \frac{m_j\,p_j}{m}\;, \right.\\ &\left. \forall k\neq j:\; p_k\cdot \left(\sum_{l=0}^{t-1} \frac{x_{kl}}{p_k} + \frac{1}{2}\frac{x_{kt}}{p_k} \right)\;,\; p_j \right\}\end{aligned}$$

$$\begin{aligned}\leq\; t \;+\; 2\cdot\max\; &\left\{ \left(\sum_{l=0}^{t-1}\sum_{k\neq j} \frac{x_{kl}\,m_k}{m} \right) \;+\; \left(\frac{1}{2}\sum_{k\neq j} \frac{x_{kt}\,m_k}{m} \right) \;+\; \frac{m_j\,p_j}{m}\;, \right.\\ &\left. \forall k\neq j:\; \sum_{l=0}^{t-1} x_{kl} + \frac{1}{2}x_{kt}\;,\; p_j \right\}\end{aligned}$$

The capacity constraint (2) together with (6) yields:

$$\leq\; t + 2\cdot\max\left\{ t + \frac{1}{2} + \frac{m_j p_j}{m}\;,\; t + \frac{1}{2}\;,\; p_j \right\} \;\leq\; t + 2\cdot\left(t + \frac{1}{2} + p_j \right).$$

Summing over all possible assignments of j to t we finally bound the expectation of j's completion time as follows:

$$E[\tilde{C}_j] \;=\; \sum_{t=0}^{T} \frac{x_{jt}}{p_j}\, E_t[\tilde{C}_j] \;\leq\; t \;+\; 2\cdot \sum_{t=0}^{T} \frac{x_{jt}}{p_j}\;(t+\frac{1}{2}+p_j)$$

$$\leq\; t \;+\; 2\cdot \left(\sum_{t=0}^{T} \frac{x_{jt}}{p_j} \left(t+\frac{1}{2}+\frac{p_j}{2}\right) \;+\; \sum_{t=0}^{T} \frac{x_{jt}}{p_j}\;\frac{p_j}{2} \right).$$

From (1), (3) and (4) we get:

$$=\; t \;+\; 2\cdot \left(\frac{p_j}{2} + \sum_{t=0}^{T} \frac{x_{jt}}{p_j}(t+\frac{1}{2}) \right) \;+\; 2\cdot\frac{p_j}{2} \;\leq\; \bar{C}_j + 2\bar{C}_j + \bar{C}_j \;=\; 4\bar{C}_j.$$

∎

COROLLARY 5.1. *for the general model $P|m_j, r_j, pmtn|\sum w_j C_j$, C_{max} Lemma 5.1 implies a randomized performance guarantee of 4 for weighted sum of completion times achieved by algorithm* Multi-Random-Assignment. *Using* List-GG-PR *achieves a performance guarantee of 3 for the makespan.*

COROLLARY 5.2. *By dropping constraint (5) of* **LP2** *and using* List-GG-P *instead of* List-GG-PR *in algorithm* Multi-Random-Assignment, *we get for the restricted model $P|m_j, pmtn|\sum w_j C_j$, C_{max} a randomized performance guarantee of 3 for the weighted sum of completion times, and a 2-approximation for makespan.*

Unfortunately the above formulation refer to exponentially many time intervals. Consequently, the algorithm Multi-Random-Assignment is a pseudo-polynomial-time algorithm.

6. INTERVAL-INDEXED FORMULATION

Following Schulz and Skutella ([17], [18]) we introduce "interval-indexed variables" which refer to polynomial number of intervals of geometrically increasing size. The idea of using interval-indexed variables to get polynomial-time solvable LP relaxations was introduced by Hall, Shmoys, and Wein [12]. In order to get polynomial-time approximation in this way, we have to pay with a slightly worse performance guarantee. For instance, for any constant $\epsilon > 0$, we get a randomized approximation algorithm with performance guarantee $4+\epsilon$ for the sum of completion times for the general model.

For a given $0 < \epsilon \leq 1$, L is defined as to be the smallest integer satisfying $(1+\epsilon)^L \geq T+1$. Consequently, L is polynomially bounded in the input size of the considered scheduling problem. Let $I_0 = [0,1]$ and for $1 \leq l \leq L$ let $I_l = \left((1+\epsilon)^{l-1}, (1+\epsilon)^l\right]$. We denote by $|I_l|$ the length of the l-th interval, i.e., $|I_l| = \epsilon(1+\epsilon)^{l-1}$ for $1 \leq l \leq L$, and $|I_0| = 1$. To simplify notation we define $(1+\epsilon)^{-1}$ to be $\frac{1}{2}$ in the interval I_0. For each $j \in N$ and $l = 0, \ldots, L$, define the variable y_{jl} as the portion of I_l that is being used by the j-th job. Consider the following linear program **LP3**, in these time-indexed variables:

$$\text{minimize} \quad \sum_{j=1}^{n} w_j C_j$$

(LP3) subject to:

$$\begin{array}{lll}
(1) & \displaystyle\sum_{l=0}^{L} \frac{y_{jl}|I_l|}{p_j} = 1 & \forall j \in N \\[2ex]
(2) & \displaystyle\sum_{j \in N} y_{jl}\, m_j \leq m & l = 0, \ldots, L \\[2ex]
(3) & C_j \geq p_j & \forall j \in N \\[2ex]
(4) & C_j \geq \dfrac{p_j}{2} + \displaystyle\sum_{l=0}^{L} \frac{y_{jl}\,|I_l|}{p_j}\,(1+\epsilon)^{l-1} & \forall j \in N \text{ and } l = 0, \ldots, L \\[2ex]
(5) & y_{jl} = 0 & \forall j \in N \text{ and } (1+\epsilon)^l \leq r_j \\[2ex]
(6) & 0 \leq y_{jl} \leq 1 & \forall j \in N \text{ and } l = 0, \ldots, L
\end{array}$$

Note that in constraint (4) instead of taking the middle point at the appropriate interval, we set here $(1+\epsilon)^{l-1}$ as a lower bound on the interval. Note that since $(1+\epsilon)^{-1} = \frac{1}{2}$ for the interval I_0, so the bounds holds for any $l = 0, \ldots, L$. Thus the above LP is a polynomial relaxation for our scheduling problem.

The following algorithm uses optimum LP solution to construct a feasible schedule by using a kind of a randomized rounding.

Algorithm Multi-Random-ϵ-Assignment works as follows: Compute an optimum solution y to **LP3**. Assign job j to l at random with probability $\frac{y_{jl}|I_l|}{p_j}$. Define the random assignment l of job j as l_j. Sort the jobs according to nondecreasing order of l_j (ties are broken randomly), and use the resulting permutation as an input to algorithm List-GG-PR.

Note that all the random assignments need to be performed independently from each other.

Let $\bar{C}_j$ denote the completion time of the job j by the solution y of the LP relaxation, and $\tilde{C}_j$ denote the completion time in the schedule computed by Multi-Random-ϵ-Assignment.

LEMMA 6.1. *$E\,[\tilde{C}_j] \leq (4+\epsilon)\bar{C}_j$ for the general model.*

Proof. First consider the conditional expectation of j's completion time under the assumption that j was assigned to l. we denote this conditional expectation by $E_l\,[\tilde{C}_j]$.

By Corollary 3.1:

$$E_l[\tilde{C}_j] \;\leq\; l{+}2{\cdot}\max\left\{\left(\sum_{k\neq j}\left(\frac{a_k}{m}\;\cdot\;\Pr\,[\;k \text{ before } j\;]\right)\right) + \frac{a_j}{m}\;,\;\; p_k\cdot\Pr\,[\;k \text{ before } j\;]\;,\;\; p_j\right\}.$$

By replacing the probability expression we get:

$$\begin{aligned}
\leq\;\; l \,+\, 2\cdot\max\;&\left\{\sum_{k\neq j}\frac{m_k\,p_k}{m}\cdot\left(\sum_{i=0}^{l-1}\frac{y_{ki}\,|I_i|}{p_k}+\frac{1}{2}\frac{y_{kl}\,|I_l|}{p_k}\right)\;+\;\frac{m_j\,p_j}{m}\;,\right.\\
&\left.\qquad \forall k\neq j:\; p_k\cdot\left(\sum_{i=0}^{l-1}\frac{y_{ki}\,|I_i|}{p_k}+\frac{1}{2}\frac{y_{kl}\,|I_l|}{p_k}\right)\;,\; p_j\right\}
\end{aligned}$$

$$\begin{aligned}
\leq\; l \,+\, 2\cdot\max\;&\left\{\left(\sum_{i=0}^{l-1}|I_i|\sum_{k\neq j}\frac{y_{ki}\,m_k}{m}\right)\;+\;\frac{|I_l|}{2}\sum_{k\neq j}\frac{y_{kl}\,m_k}{m}\;+\;\frac{m_j\,p_j}{m}\;,\right.\\
&\left.\qquad \forall k\neq j:\;\left(\sum_{i=0}^{l-1}y_{ki}\,|I_i|+\frac{1}{2}y_{kl}\,|I_l|\right)\;,\; p_j\right\}
\end{aligned}$$

$$\leq\; l{+}2{\cdot}\max\left\{(1+\epsilon)^{l-1}+\frac{1}{2}\epsilon(1+\epsilon)^{l-1}+\frac{m_j p_j}{m}\;,\;\; (1+\epsilon)^{l-1}+\frac{1}{2}\epsilon(1+\epsilon)^{l-1}\;,\;\; p_j\right\}$$

$$\leq\;\; l{+}2{\cdot}\left((1+\epsilon)^{l-1}+\frac{1}{2}\epsilon(1+\epsilon)^{l-1}+p_j\right) \;\;=\;\; l{+}2{\cdot}\left((1+\epsilon)^{l-1}(1+\frac{\epsilon}{2})+p_j\right).$$

Summing over all possible assignments of j to l we finally bound the expectation of j's completion time as follows:

$$E[\tilde{C}_j] \;\;=\;\; \sum_{l=0}^{L}\frac{y_{jl}\,|I_l|}{p_j}\,E_l[\tilde{C}_j] \;\;\leq\;\; l{+}2{\cdot}\sum_{l=0}^{L}\frac{y_{jl}\,|I_l|}{p_j}{\cdot}\left((1+\epsilon)^{l-1}(1+\frac{\epsilon}{2})+p_j\right)$$

$$\leq\ l\ +\ 2\cdot\left(\frac{p_j}{2}+\sum_{l=0}^{L}\frac{y_{jl}\,|I_l|}{p_j}(1+\epsilon)^{l-1}\right)\ (1+\frac{\epsilon}{2})\ \ +\ \ 2\cdot\frac{p_j}{2}$$

$$\leq\ \bar{C_j}+2(1+\frac{\epsilon}{2})\bar{C_j}+\bar{C_j}\ \ =\ \ (4+\epsilon)\bar{C_j}\,.$$

∎

COROLLARY 6.1. *Lemma 6.1 implies a randomized performance guarantee of* $4+\epsilon$ *for the weighted sum of completion times achieved by the algorithm* Multi-Random-ϵ-Assignment *for the general model with release dates. Using* List-GG-PR *achieves performance guarantee of 3 for makespan.*

Following Schulz and Skutella [17] we can efficiently derandomized the above technique for the general model with slight modifications to algorithm Multi-Random-ϵ-Assignment. Schulz and Skutella use the method of conditional probabilities developed implicitly by Erdős and Selfridge [7], and later developed in a more general context by Spencer [22].

COROLLARY 6.2. *By dropping constraint (5) of* **LP3** *and using* List-GG-P *instead of* List-GG-PR *in the algorithm* Multi-Random-ϵ-Assignment*, we get for the restricted model* $P|m_j,pmtn|\sum w_jC_j$*,* C_{max} *a randomized performance guarantee of* $3+\epsilon$ *for weighted sum of completion times, and a 2-approximation for makespan.*

7. ON-LINE ALGORITHMS WITH CONSTANT RATIOS

First we introduce a simple on-line algorithm for the unweighted general model, which we shall adjust later on to the genral model. Our algorithms produce constant competitive ratios for both makespan and sum of completion times objectives.

Our technique is a kind of a generalization to the deterministic on-line algorithm of Chekuri, Motwani, Natarajan, and Stein [3], which we quote and analyze for the sake of completeness. We denote this algorithm as CMNS.

7.1. Algorithm CMNS

Algorithm CMNS of Chekuri et al. addresses the problem of nonpreemptively scheduling on identical parallel machine with release dates to minimize the sum of completion times. In this model each job demand a single processor, and m is the total number of processors. Chekuri et al. rely on the well known fact that the preemptive algorithm SRPT [2] which always schedules the shortest remaining processing time job is a polynomial optimal on-line algorithm for the problem $1|r_j, pmtn|\sum C_j$. By using a relaxation to the one-machine preemptive problem, they achieve a competitive ratio of 3.

Given an instance J for nonpreemptive scheduling on m machines Chekurei et al. define a relaxation J'' in which the processing times $p_j'' = p_j/m$ and the release dates remain $r_j'' = r_j$. They find the optimal solution for the instance J'' of the problem $1|r_j, pmtn|\sum C_j$ by running SRPT. They argue that the sum of completion times achieved by SRPT on J'' is a lower bound for any feasible schedule that minimizes the sum of completion times of J on m machines (Lemma 7.3 generalizes this idea). Turek et al. used a similar technique to derive a lower bound w.r.t. sum of completion times for parallel jobs in the restricted model (with no release dates), called the squashed area bound [25].

Algorithm CMNS works as follows.
For the instance J construct the instance J''. Set L to be an empty list. Then simulate SRPT on J''. Let $\bar{C}_j$ denote the completion time of the job j in the simulation. As soon as a job is finished in the simulation add it to the end of the list L. Schedule the jobs nonpreemptively in the identical parallel machine in the order of the list L. Let $\tilde{C}_j$ denote the completion time in the schedule found by CMNS for the job j.

Without loss of generality we may assume that the jobs are indexed so that $\bar{C}_1 \leq \bar{C}_2 \leq \cdots \leq \bar{C}_n$. Thus job j is the jth job to complete in the simulation.

LEMMA 7.1. $\sum \tilde{C}_j \leq 3 \sum C_j^*$.

Proof. First we derive a lower bound on $\bar{C}_j$. $\bar{C}_j$ is at least as big as the processing times of the jobs that precede it in the simulation. Therefore, $\bar{C}_j \geq \sum_{k=1}^{j} p_k'' = \sum_{k=1}^{j} p_k/m$. From the point of view of the list L, the job j is released at the time $\bar{C}_j$. Hence, by a standard makespan argument the job j will complete by:

$$\tilde{C}_j \leq \bar{C}_j + \sum_{k=1}^{j-1} \frac{p_k}{m} + p_j \leq 2 \cdot \bar{C}_j + p_j$$

Summing over all jobs we get the result. ∎

By a simple argument $\bar{C}_n$, the completion time of the last job on the simulation SRPT, is a lower bound for the makespan, and hence we conclude that CMNS gives a 3 competitive ratio for the makespan as well.

TABLE 6.

Related instances of the job system.

instance	release date	processing time	processors demand	total processors	weight
I	r_j	p_j	m_j	m	w_j
II	$r'_j = \max\{r_j, p_j\}$	$p'_j = p_j$	$m'_j = m_j$	m	$w'_j = w_j$
III	$r''_j = \max\{r_j, p_j\}$	$p''_j = \frac{p_j m_j}{m}$	$m''_j = 1$	1	$w''_j = w_j$

7.2. Back to our unweighted general model

In this subsection we shall introduce and analyze the algorithm Simulate-SRA - simulate smallest remaining area. Given an instance I for the unweighted general model $P|m_j, r_j, pmtn|\sum C_j, C_{max}$, allowing preemptions, we refer to the following related instances:

If the job j is released at time $r_j \geq p_j$ in the original instance I, it is simultaneously released at instances II and III, otherwise j will be release at time p_j. In the unweighted model we ignore the right most column of the weights. Let $\sum C^*_j$ and $\sum C^{**}_j$ denote the (preemptive) optimal sums of completion times for the instances I and II, respectively. In this subsection any optimal schedule for an instance means the optimal schedule for sum of completion times.

Let SI^* be any optimal schedule for I, and let C^*_j be the completion time of the job j in it. Assume that the jobs are indexed so that $C^*_1 \leq \cdots \leq C^*_n$. Since SI^* is a feasible schedule we immediately derive that: $C^*_j \geq p_j + r_j$, and $C^*_j \geq \sum_{k=1}^{j} \frac{m_k p_k}{m}$. In addition, from the re-indexing $C^*_1 \leq \cdots \leq C^*_n$, we get that $C^*_j \geq p^{max}_j$.

LEMMA 7.2. $\sum C^{**}_j \leq 3 \cdot \sum C^*_j$.

Proof. Since $\sum C^{**}_j$ is a lower bound for II, given $C^*_1 \leq \cdots \leq C^*_n$, the completion times of the optimal schedule SI^*, it is sufficient to prove that there is a schedule SII for II that its sum of completion times is bounded from above by $3 \cdot \sum C^*_j$. Following is an algorithm to find such an SII.

Algorithm A: Set L to be an empty list. Add job j to the end of the list L at time C^*_j. Run List-GG-PR on the list L. Let C'_j denote the completion time in the schedule found by algorithm A for the job j.

Since $C^*_j \geq r_j + p_j \geq max\{r_j, p_j\} = r'_j$, we conclude that the job j has entered to L, after being released in II. Hence, algorithm A produces a feasible schedule for II. By using List-GG-PR in A we have:

$$C'_j \;\leq\; C^*_j + 2\cdot\max\left\{\sum_{k=1}^{j}\frac{m_k p_k}{m},\; p_j^{max}\right\} \;\leq\; C^*_j + 2\cdot\max\left\{C^*_j,\; C^*_j\right\} = 3C^*_j.$$

Hence, we get immediately: $\sum C_j^{**} \leq \sum C'_j \leq 3\cdot\sum C^*_j$. ▌

Recall that algorithm SRPT on the instance III gives us an optimum solution for III. Let $C_j^{*^*{*}}$ denote the completion time of job j in it, and assume that the jobs are indexed so that $C_1^{*^*{*}} \leq \cdots \leq C_n^{*^*{*}}$. SRPT gives a feasible schedule for III on a single processor, and hence: $C_j^{*^*{*}} \geq \sum_{k=1}^{j} p''_k = \sum_{k=1}^{j}\frac{m_k p_k}{m}$, and $C_j^{*^*{*}} \geq r''_k = \max\{r_j, p_j\}$. Once again, by the re-indexing $C_1^{*^*{*}} \leq \cdots \leq C_n^{*^*{*}}$ we get that $C_j^{*^*{*}} \geq p_j^{max}$ and $C_j^{*^*{*}} \geq r_j^{max}$. The benefit of considering instances II and III, is the upper bounding of p_j^{max} by $C_j^{*^*{*}}$.

Lemma 7.3. $\sum C_j^{*^*{*}} \leq \sum C_j^{**}$.

Proof. Let SII denote any feasible schedule for II. We shall produce from SII a feasible schedule for III, denoted by SIII. Let T be the makespan of SII. For each $t = 0, \ldots, T-1$, we consider the following one unit time interval $I_t = (t, t+1]$. Assume w.l.o.g that in the interval I_t the jobs $1, \ldots, k$ are those which are running in SII. From the capacity constraint we have $\sum_{i=1}^{k} m_i \leq m$. Equivalently, $\sum_{i=1}^{k}\frac{m_i}{m} \leq 1$. Thus, in SIII, we run the job i, $i = 1, \ldots, k$, during $\frac{m_i}{m}$ unit of of time in the interval I_t. We argue that if the job j runs p_j units of time and completes at time t' in SII, it runs $\frac{p_j m_j}{m}$ units of time and completes at time $\leq t'$ in SIII. Especially, when SII is an optimal schedule for II, the sum of completion times of SIII is bounded from above by $\sum C_j^{**}$ and bounded from below by $\sum C_j^{*^*{*}}$. ▌

From the above two lemmas, we conclude that $\sum C_j^{*^*{*}} \leq 3\cdot\sum C^*_j$. Now we are ready to introduce our on-line algorithm for the unweighted general model.

Algorithm Simulate-SRA works as follows.
Given the instance I construct the instance III. Set L to be an empty list. Simulate SRPT on III. As soon as a job is finished to run on the simulation add it to the end of the list L. Run List-GG-PR on the list L.

Let $C_j^{*^*{*}}$ denote the completion time of the job j in the simulation. Assume w.l.o.g that the jobs are indexed so that $C_1^{*^*{*}} \leq \cdots \leq C_n^{*^*{*}}$. Let $\tilde{C}_j$ denote the completion time in the schedule found by Simulate-SRA for the job j. Clearly the job j is being released into the list L at time $C_j^{*^*{*}}$.

Lemma 7.4. Simulate-SRA *achieves competitive ratio of 9 for the sum of completion times.*

Proof. By using List-GG-PR in Simulate-SRA we have by corollary 3.1:

$$\tilde{C}_j \;\le\; C_j^{*^{*}*} + 2{\cdot}\max\left\{\sum_{k=1}^{j}\frac{m_k p_k}{m},\; p_j^{max}\right\} \;\le\; C_j^{*^{*}*} + 2{\cdot}\max\left\{C_j^{*^{*}*},\; C_j^{*^{*}*}\right\} = 3C_j^{*^{*}*}.$$

Combining the previous results we have:

$$\sum \tilde{C}_j \;\le\; 3\cdot\sum C_j^{*^{*}*} \;\le\; 9\cdot\sum C_j^{*}.$$

∎

LEMMA 7.5. Simulate-SRA *achieves competitive ratio of 4 for makespan.*

Proof. This comes from the following arguments:

- $\max_{j=1,\ldots,n}\tilde{C}_j \;\le\; C_n^{*^{*}*} \;+\; 2\cdot\max\{\sum_{k=1}^{n}\frac{m_k p_k}{m},\; p_n^{max}\}$.
- $C_n^{*^{*}*} \;\le\; r_n^{\prime\prime max} + \sum_{k=1}^{n} p_k^{\prime\prime} \;\le\; \max\{r_n^{max},\; p_n^{max}\} + \sum_{k=1}^{n}\frac{m_k p_k}{m}$, since SRPT is a list based algorithm.
- Let C_{max}^{*} denotes the optimal makespan. The following trivial lower bounds for the optimal makespan hold: $\max\{\sum_{k=1}^{n}\frac{m_k p_k}{m},\; p_n^{max},\; r_n^{max}\} \le C_{max}^{*}$
- Combining the above we have that: $\max_{j=1,\ldots,n}\tilde{C}_j \le 4\cdot C_{max}^{*}$.

∎

7.3. Concluding remarks regarding our general model

Considering weighted jobs, a similar technique may be applied to the general model.

Let $\sum w_j C_j^{*}$, $\sum w_j C_j^{**}$, and $\sum w_j C_j^{*^{*}*}$ denote the (preemptive) optimal weighted sums of completion times for the instances I, II, and III.

Note that the analysis in the proofs of Lemmas 7.2 and 7.3 is per job (job-by-job). Thus adding weights does not affect the respective results. Hence we conclude that $\sum w_j C_j^{*^{*}*} \le 3\cdot\sum w_j C_j^{*}$.

Following is a deterministic algorithm with constant competitive ratios for the general model. Our algorithm Simulate-wSRA, simulates the on-line algorithm "preemptive list scheduling in order of nonincreasing ratios w_j/p_j", which Goemans, Wein, and Williamson [10] proved to be 2-competitive for $1|r_j, pmtn|\sum w_j C_j$. Its simple and short proof is quoted in [16].

Algorithm Simulate-wSRA works as follows.

Given the instance I construct the instance III. Set L to be an empty list. Simulate preemptive list scheduling in order of nonincreasing ratios w_j/p_j

on III. As soon as a job is finished to run in the simulation add it to the end of the list L. Run List-GG-PR on the list L.

Let $\hat{C}_j$ denote the completion time of the job j in the simulation. Assume w.l.o.g that the jobs are indexed so that $\hat{C}_1 \leq \cdots \leq \hat{C}_n$. Let $\tilde{C}_j$ denote the completion time in the schedule found by Simulate-wSRA for the job j.

LEMMA 7.6. Simulate-wSRA *achieves competitive ratio of 18 for the weighted sum of completion times.*

Proof. By using List-GG-PR in Simulate-wSRA we have:

$$\tilde{C}_j \leq \hat{C}_j + 2 \cdot \max\left\{\sum_{k=1}^{j} \frac{m_k p_k}{m},\ p_j^{max}\right\} \leq \hat{C}_j + 2 \cdot \max\left\{\hat{C}_j,\ \hat{C}_j\right\} = 3\hat{C}_j.$$

Combining the previous results we have:

$$\sum w_j \tilde{C}_j \leq 3 \cdot \sum w_j \hat{C}_j \leq 3 \cdot 2 \cdot \sum w_j {C_j^{**}}^{*} \leq 18 \cdot \sum C_j^{*}.$$

∎

Following is a randomized algorithm for the general model. Our algorithm Simulate-rwSRA, simulates the on-line randomized algorithm of Schulz and Skutella [16], which we shall denote as $\alpha - SS$. Algorithm $\alpha - SS$ is randomized preemptive list based algorithm, which Schulz and Skutella proved to be 4/3-competitive for $1|r_j, pmtn|\sum w_j C_j$.

Algorithm Simulate-rwSRA works as follows.

Given the instance I construct the instance III. Set L to be an empty list. Simulate $\alpha - SS$ on III. As soon as a job is finished to run on the simulation add it to the end of the list L. Run List-GG-PR on the list L.

Let $\hat{C}_j$ denote the completion time of the job j in the simulation. Aassume w.l.o.g that the jobs are indexed so that $\hat{C}_1 \leq \cdots \leq \hat{C}_n$. Let $\tilde{C}_j$ denote the completion time in the schedule found by Simulate-rwSRA for the job j.

LEMMA 7.7. Simulate-rwSRA *achieves randomized competitive ratio of 12 for the weighted sum of completion times.*

Proof. By using List-GG-PR in Simulate-rwSRA we have:

$$\tilde{C}_j \leq \hat{C}_j + 2 \cdot \max\left\{\sum_{k=1}^{j} \frac{m_k p_k}{m},\ p_j^{max}\right\} \leq \hat{C}_j + 2 \cdot \max\left\{\hat{C}_j,\ \hat{C}_j\right\} = 3\hat{C}_j.$$

Combining the previous results we have:

$$E[\sum w_j \tilde{C}_j] \leq 3 \cdot E[\sum w_j \hat{C}_j] \leq 3 \cdot \frac{4}{3} \cdot \sum w_j C_j^{**^*} \leq 12 \cdot \sum C_j^*.$$

▮

LEMMA 7.8. (Simulate-wSRA) Simulate-rwSRA *achieves competitive ratio of 4 for makespan.*

Proof. This comes from the following arguments:

- $\max_{j=1,\ldots,n} \tilde{C}_j \leq \hat{C}_n + 2 \cdot \max\{\sum_{k=1}^{n} \frac{m_k p_k}{m}, p_n^{max}\}$.
- $\hat{C}_n \leq r_n''^{max} + \sum_{k=1}^{n} p_k'' \leq \max\{r_n^{max}, p_n^{max}\} + \sum_{k=1}^{n} \frac{m_k p_k}{m}$.
- Combining the above we have that: $\max_{j=1,\ldots,n} \tilde{C}_j \leq 4 \cdot C_{max}^*$.

▮

8. OPEN PROBLEMS

• Can we use our (both off-line and on-line) techniques to derive non-preemptive algorithms (with good bounds)?

• Does our analysis give algorithms with tight bounds? Especially, in section 7, can we reduce the factor in the result $\sum C_j^{**} \leq 3 \cdot \sum C_j^*$? This obviously will improve the constant competitive ratios of our on-line algorithms.

• Finding lower bounds on the performance of any on-line algorithms for the (weighted) general model. Obviously, any lower bound for the identical parallel machine applies here, but we believe that stronger bounds do exist.

• Designing algorithms while running times of the arriving job is not known until the job completes its running. Our simulations (section 7) are no longer valid for this paradigm. However, it is obvious that in order to minimize the sum of completion times in this case we must use preemptions.

• Can our online frameworks (section 7) be adjusted to approximate other objectives, especially total flow times?

REFERENCES

1. 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 Annual IEEE Symposium on Foundations of Computer Science", 1999.

2. K. R. Baker. "Introduction to sequencing and scheduling". Wiley, New York, 1974.

3. C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, Approximation Techniques for Average Completion Time Scheduling, *in* "Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)", 1997.

4. M. Drozdowski, On complexity of multiprocessor tasks scheduling, *Bulletin of the Polish Academy of Sciences, Technical Science 42/3*, pp. 437-445, 1994.

5. M. Drozdowski, Scheduling multiprocessor tasks - An overview, *European Journal of Operational Research* **94** (1996) 215–230.

6. M. E. Dyer and L. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics* **26** (1990) 255–270.

7. P. Erdős and J. L. Selfridge. On a combinatorical game. *Journal of Combinatorical Theory A* **14** (1990) 298–301.

8. M. Garey and R. Graham. Bounds for Multiprocessor Scheduling with Resource Constraints. SIAM *Journal on Computing* **4(2)** (1975) 187–200.

9. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* **5** (1979) 287–326.

10. M. X. Goemans, J. Wein, and D. P. Williamson. Personal Communication, August 1997. (Quoted in [16].)

11. L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms. *Mathematics of Operations Research*, August 1997.

12. L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms. *in* "Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms", pp. 142-151, 1996.

13. J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. *in* "Progress in Combinatorial Optimization" (W. R. Pulleyblank, Ed.), pp. 245-261, 1984.

14. C. Phillips, C. Stein, J. Wein. Minimizing Average Completion Time in the Presence of Release Date. To appear in *Mathematical Programming.* Preliminary version appeared as Scheduling Jobs That Arrive Over Time. *in* "Proceedings of 1995 Workshop on Algorithms and Data Structures", pp. 86–97, Aug 1995.

15. M. Queyranne. Structure of a simple scheduling polyhedron. *Math. Programming* **58** (1993) 263–285.

16. A. S. Schulz, and M. Skutella. The Power of alpha-Points in Preemptive Single Machine Scheduling. Submitted to publication. Postscript located at: http://www.math.TU-Berlin.DE/~skutella/publications.html.

17. A. S. Schulz, and M. Skutella. Scheduling-LPs Bear Probabilities: Randomized Approximations for Min-Sum Criteria. *in* "Proceedings of the 5th Annual European Symposium on Algorithms" (R. Burkard and G. Woeginger, Eds.), pp. 416-429, Lecture Notes in Computer Science 1284, Springer: Berlin, 1997.

18. A. S. Schulz and M. Skutella. Random-Based Scheduling: New Approximations and LP Lower Bounds. *in* "Randomization and Approximation Techniques in Computer Science", (J. Rolim, Ed.), pp. 119-133, Lecture Notes in Computer Science 1269, Springer: Berlin, 1997.

19. U. Schwiegelshohn. Preemptive Weighted Completion Time Scheduling of Parallel Jobs. *in* "Proceedings of the 4th Annual European Symposium on Algorithms", pp. 39-51, Lecture Notes in Computer Science 1136, Springer: Berlin, 1996.

20. U. Schwiegelshohn, W. Ludwig, J. L. Wolf, J. J. Turek, and P. S. Yu. Smart SMART Bounds for Weighted Response Time Scheduling. *SIAM Journal on Computing* **28(1)** (1999) 237–253.

21. J. Sgall. On-line scheduling - A Survey *in* "Online Algorithms: The State of the Art", (A. Fiat and G. J. Woeginger, Eds.),pp. 196-231, Lecture Notes in Computer Science 1442, Springer:Berlin, 1998.

22. J. Spencer. *Ten Lectures on the Probabilistic Method.* Number 52 in CBMS-NSF Reg. Conf. Ser. Appl. Math. SIAM, 1987.

23. C. Stein, and J. Wein. On the Existence of Schedules that are Near-Optimal for both Makespan and Total Weighted Completion Time. To appear in *Operations Research Letters.*

24. J. Turek, J. L. Wolf, and P. S. Yu, Approximate algorithms for scheduling parallelizable tasks, *in* "4th Symp. Parallel Algorithms & Architectures", pp. 323-332, Jun 1992.

25. J. Turek, U. Schwiegelshohn, J. Wolf and P. S. Yu, Scheduling Parallel Tasks to Minimize Average Response Time, *in* "Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms", Arlington, VA, pp. 112-121, Jan 1994.