Scheduling–LPs Bear Probabilities

Randomized Approximations for Min–Sum Criteria

Andreas S. Schulz and Martin Skutella

Technische Universität Berlin,
Fachbereich Mathematik, MA 6–1,
Straße des 17. Juni 136, 10623 Berlin, Germany,
E-mail {schulz,skutella}@math.tu–berlin.de

Abstract. In this paper, we provide a new class of randomized approximation algorithms for scheduling problems by directly interpreting solutions to so-called time-indexed LPs as probabilities. The most general model we consider is scheduling unrelated parallel machines with release dates (or even network scheduling) so as to minimize the average weighted completion time. The crucial idea for these multiple machine problems is not to use standard list scheduling but rather to assign jobs randomly to machines (with probabilities taken from an optimal LP solution) and to perform list scheduling on each of them.
For the general model, we give a $(2+\varepsilon)$–approximation algorithm. The best previously known approximation algorithm has a performance guarantee of 16/3 [HSW96]. Moreover, our algorithm also improves upon the best previously known approximation algorithms for the special case of identical parallel machine scheduling (performance guarantee $(2.89+\varepsilon)$ in general [CPS+96] and 2.85 for the average completion time [CMNS97], respectively). A perhaps surprising implication for identical parallel machines is that jobs are randomly assigned to machines, in which each machine is equally likely. In addition, in this case the algorithm has running time $O(n \log n)$ and performance guarantee 2. The same algorithm also is a 2–approximation for the corresponding preemptive scheduling problem on identical parallel machines.
Finally, the results for identical parallel machine scheduling apply to both the off-line and the on-line settings with no difference in performance guarantees. In the on-line setting, we are scheduling jobs that continually arrive to be processed and, for each time t, we must construct the schedule until time t without any knowledge of the jobs that will arrive afterwards.

1 Introduction

It is by now well-known that randomization can help in the design of algorithms, cf., e. g., [MR95,MNR96]. One way of guiding randomness is the use of linear programs (LPs). In this paper, we give LP-based approximation algorithms for problems which are particularly well-known for the difficulties to obtain good lower bounds: machine (or processor) scheduling problems. Because of the random choices involved, our algorithms are rather randomized approximation algorithms. A randomized ρ–approximation algorithm is a polynomial-time algorithm that produces a feasible solution whose expected value is within a factor of ρ of the optimum; ρ is also called the expected

performance guarantee of the algorithm. Actually, we always compare the output of an algorithm with a lower bound given by an optimum solution to a certain LP relaxation. Hence, at the same time we obtain an analysis of the quality of the respective LP. All our off-line algorithms can be derandomized with no difference in performance guarantee, but at the cost of increased (but still polynomial) running times.

We consider the following model. We are given a set J of n jobs (or tasks) and m unrelated parallel machines. Each job j has a positive integral processing requirement p_{ij} which depends on the machine i job j will be processed on. Each job j must be processed for the respective amount of time on one of the m machines, and may be assigned to any of them. Every machine can process at most one job at a time. In *preemptive* schedules, a job may repeatedly be interrupted and continued later on another (or the same) machine. In *nonpreemptive* schedules, a job must be processed in an uninterrupted fashion. Each job j has an integral release date $r_j \geqslant 0$ before which it cannot be processed. We denote the completion time of job j in a schedule by C_j, and for any fixed $\alpha \in (0,1]$, the α–point $C_j(\alpha)$ of job j is the first moment in time at which an α–fraction of job j has been completed; α–points were first used in the context of approximation by Hall, Shmoys, and Wein [HSW96]. We seek to minimize the total weighted completion time: a weight $w_j \geqslant 0$ is associated with each job j and the goal is to minimize $\sum_{j \in J} w_j C_j$. In scheduling, it is quite convenient to refer to the respective problems using the standard classification scheme of Graham et al. [GLLRK79]. The nonpreemptive problem $R \,|\, r_j \,|\, \sum w_j C_j$, just described, is strongly NP-hard.

Scheduling to minimize the total weighted completion time (or, equivalently, the average weighted completion time) has recently achieved a great deal of attention, partly because of its importance as a fundamental problem in scheduling, and also because of new applications, for instance, in compiler optimization [CJM+96] or in parallel computing [CM96]. In the last two years, there has been significant progress in the design of approximation algorithms for this kind of problems which led to the development of the first constant worst-case bounds in a number of settings. This progress essentially follows on the one hand from the use of preemptive schedules to construct nonpreemptive ones [PSW95,CPS+96,CMNS97,Goe97,SS97]. On the other hand, one solves an LP relaxation and then a schedule is constructed by list scheduling in a natural order dictated by the LP solution [PSW95,HSW96,Sch96,HSSW96,MSS96,Goe97,CS97,SS97].

In this paper, we utilize a different idea: random assignments of jobs to machines. To be more precise, we exploit a new LP relaxation in time-indexed variables for the problem $R \,|\, r_j \,|\, \sum w_j C_j$, and we then show that a certain variant of randomized rounding leads to a $(2+\varepsilon)$–approximation algorithm, for any $\varepsilon > 0$. At the same moment, the corresponding LP is a $(2+\varepsilon)$–relaxation, i. e., the true optimum is always within a factor of $(2+\varepsilon)$ of the optimal value of the LP relaxation; and this is tight. Our algorithm improves upon a $16/3$–approximation algorithm of Hall, Shmoys, and Wein [HSW96] that is also based on time-indexed variables which have a different meaning, however. In contrast to their approach, our algorithm does not rely on Shmoys and Tardos' rounding technique for the generalized assignment problem [ST93]. We rather exploit the LP by interpreting LP values as probabilities with which jobs are assigned to machines. For an introduction to and the application of randomized rounding to other combinatorial optimization problems, the reader is referred to [RT87,MNR96].

For the special case of identical parallel machines, i. e., for each job j and all machines i we have $p_{ij} = p_j$, Chakrabarti, Phillips, Schulz, Shmoys, Stein, and Wein [CPS+96] obtained a $(2.89+\varepsilon)$–approximation by refining an on-line greedy framework of Hall et al. [HSW96]. The former best known LP-based algorithm, however, relies on an LP relaxation solely in completion time variables which is weaker than the one we propose. It has performance guarantee $(4-\frac{1}{m})$ (see [HSSW96] for the details). For the LP we use here, an optimum solution can greedily be obtained by a certain preemptive schedule on a virtual single machine which is m times as fast as any of the original machines. The idea of using a preemptive relaxation on such a virtual machine was introduced before by Chekuri, Motwani, Natarajan, and Stein [CMNS97]. They show that any preemptive schedule on this machine can be converted into a nonpreemptive schedule on the m identical parallel machines such that, for each job j, its completion time in the nonpreemptive schedule is at most $(3-\frac{1}{m})$ times its preemptive completion time. For the average completion time, they refine this to a 2.85–approximation algorithm for $P|r_j|\sum C_j$. For $P|r_j|\sum w_jC_j$, the algorithm we propose delivers in time $O(n \log n)$ a solution that is expected to be within a factor of 2 of the optimum.

Since the LP relaxation we use is also a relaxation for the corresponding preemptive problem, our algorithm is also a 2–approximation for $P|r_j, pmtn|\sum w_jC_j$. This improves upon a 3–approximation algorithm due to Hall, Schulz, Shmoys, and Wein [HSSW96].

The paper is organized as follows. In Section 2, we start with the discussion of our main result: the 2–approximation in the general context of unrelated parallel machine scheduling. In the next section, we give a combinatorial 2–approximation algorithm for $P|r_j|\sum w_jC_j$ and $P|r_j, pmtn|\sum w_jC_j$. Then, in Section 4, we discuss the derandomization of the previously given randomized algorithms. Finally, in Section 5 we give the technical details of turning the pseudo-polynomial algorithm of Section 2 into a polynomial time algorithm with performance guarantee $(2+\varepsilon)$.

2 Unrelated Parallel Machine Scheduling with Release Dates

In this section, we consider the problem $R|r_j|\sum w_jC_j$. As in [PSW94,HSW96], we will even discuss a slightly more general problem in which the release date of job j may also depend on the machine. The release date of job j on machine i is thus denoted by r_{ij}. Machine-dependent release dates are relevant to model network scheduling in which parallel machines are connected by a network, each job is located at one given machine at time 0, and cannot be started on another machine until sufficient time elapses to allow the job to be transmitted to its new machine. This model has been introduced in [DLLX90,AKP92].

The problem $R|r_{ij}|\sum w_jC_j$ is well-known to be strongly NP-hard, even for the case of a single machine [LRKB77]. The first non-trivial approximation algorithm for this problem was given by Phillips, Stein, and Wein [PSW94]. It has performance guarantee $O(\log^2 n)$. Subsequently, Hall et al. [HSW96] gave a 16/3–approximation algorithm which relies on a time-indexed LP relaxation whose optimum value serves as a surrogate for the true optimum in their estimations. We use a somewhat similar LP relaxation, but whereas Hall et al. invoke the deterministic rounding technique of Shmoys and Tar-

dos [ST93] to construct a feasible schedule we randomly round LP solutions to feasible schedules.

Let $T = \max_{i,j} r_{ij} + \sum_{j \in J} \max_i p_{ij} - 1$ be the time horizon, and introduce for every job $j \in J$, every machine $i = 1, \ldots, m$, and every point $t = 0, \ldots, T$ in time a variable y_{ijt} which represents the time job j is processed on machine i within the time interval $(t, t+1]$. Equivalently, one can say that a y_{ijt}/p_{ij}–fraction of job j is being processed on machine i within the time interval $(t, t+1]$. The LP, which is an extension of a single machine LP relaxation of Dyer and Wolsey [DW90], is as follows:

$$
\begin{aligned}
& \text{minimize} && \sum_{j \in J} w_j C_j \\
& \text{subject to} && \sum_{i=1}^{m} \sum_{t=r_{ij}}^{T} \frac{y_{ijt}}{p_{ij}} = 1 \quad \text{for all } j \in J, && (1) \\
(LP_R) \qquad & && \sum_{j \in J} y_{ijt} \leqslant 1 \quad \text{for } i = 1, \ldots, m \text{ and } t = 0, \ldots, T, && (2) \\
& && C_j = \sum_{i=1}^{m} \sum_{t=0}^{T} \left(\frac{y_{ijt}}{p_{ij}} \left(t + \tfrac{1}{2} \right) + \tfrac{1}{2} y_{ijt} \right) \quad \text{for all } j \in J, && (3) \\
& && y_{ijt} = 0 \quad \text{for } i = 1, \ldots, m,\ j \in J,\ t = 0, \ldots, r_{ij} - 1, && (4) \\
& && y_{ijt} \geqslant 0 \quad \text{for } i = 1, \ldots, m,\ j \in J,\ t = r_{ij}, \ldots, T. && (5)
\end{aligned}
$$

Equations (1) ensure that the whole processing requirement of every job is satisfied. The machine capacity constraints (2) simply express that each machine can process at most one job at a time. Now, for (3), consider an arbitrary feasible schedule, where job j is being continuously processed between time $C_j - p_{hj}$ and C_j on machine h. Then the expression for C_j in (3) corresponds to the real completion time, if we assign the values to the LP variables y_{ijt} as defined above, i.e., $y_{ijt} = 1$ if $i = h$ and $t \in \{C_j - p_{hj}, \ldots, C_j - 1\}$, and $y_{ijt} = 0$ otherwise. Hence, (LP_R) is a relaxation of the scheduling problem under consideration.

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

Algorithm R

1) Compute an optimum solution y to (LP_R).
2) Assign job j to a machine-time pair (i,t) at random with probability y_{ijt}/p_{ij}; draw t_j from the chosen time interval $(t, t+1]$ at random with uniform distribution.
3) Schedule on each machine i the jobs that were assigned to it nonpreemptively as early as possible in nondecreasing order of t_j.

Note that all the random assignments need to be performed independently from each other (at least pairwise).

Lemma 1. *Let y be the optimum solution to (LP_R) which is computed in Step 1 of Algorithm* R. *Then, for each job $j \in J$ the expected processing time of j in the schedule constructed by Algorithm* R *equals $\sum_{i=1}^{m} \sum_{t=0}^{T} y_{ijt}$.*

Proof. First, we fix a machine-time pair (i,t) job j has been assigned to. Then, the expected processing time of j under these conditions is p_{ij}. By adding these conditional expectations over all machines and time intervals, weighted by the corresponding probabilities y_{ijt}/p_{ij}, we get the claimed result. □

Note that the lemma remains true if we start with an arbitrary, not necessarily optimal solution y to (LP_R) in Step 1 of Algorithm R. This also holds for the following theorem. The optimality of the LP solution is only needed to get a lower bound on the value of an optimal schedule.

Theorem 2. *The expected completion time of each job j in the schedule constructed by Algorithm* R *is at most $2 \cdot C_j^{LP}$, where C_j^{LP} is the LP completion time (defined by* (3)*) of the optimum solution y we started with in Step* 1.

Proof. We consider an arbitrary, but fixed job $j \in J$. To analyze the expected completion time of job j, we first also consider a fixed assignment of j to a machine-time pair (i,t). Then, the expected starting time of job j under these conditions precisely is the conditional expected idle time plus the conditional expected amount of processing of jobs that come before j on machine i.

Observe that there is no idle time on machine i between the maximum release date of jobs on machine i which start no later than j and the starting time of job j. It follows from the ordering of jobs and constraints (4) that this maximum release date and therefore the idle time of machine i before the starting time of j is bounded from above by t.

On the other hand, we get the following bound on the conditional expected processing time on machine i before the start of j:

$$\sum_{k \neq j} p_{ik} \cdot \Pr(k \text{ on } i \text{ before } j) \leqslant \sum_{k \neq j} p_{ik} \cdot \sum_{\ell=0}^{t} \frac{y_{ik\ell}}{p_{ik}} \leqslant t+1 \ .$$

The last inequality follows from the machine capacity constraints (2). Putting the observations together we get an upper bound of $2 \cdot (t + \frac{1}{2})$ for the expected starting time of j. Unconditioning the expectation together with Lemma 1 and (3) yields the theorem. □

Theorem 2 implies a performance guarantee of 2 for Algorithm R. In the course of its proof, we also have shown that (LP_R) is a 2–relaxation. Moreover, even for the case of identical parallel machines without release dates there are instances for which this bound is asymptotically attained, see Section 3. Thus our analysis is tight.

Unfortunately, (LP_R) has exponentially many variables. Consequently, Algorithm R only is a pseudo-polynomial-time algorithm. However, we can overcome this drawback by introducing new variables which are not associated with time intervals of length 1, but rather with intervals of geometrically increasing size. The idea of using interval-indexed variables to get polynomial-time solvable LP relaxations was introduced earlier by Hall, Shmoys, and Wein [HSW96]. In order to get polynomial-time approximation algorithms in this way, we have to pay for with a slightly worse performance guarantee. For any constant $\varepsilon > 0$, we get an approximation algorithm with performance guarantee $2+\varepsilon$ for $\mathrm{R}\,|\,r_{ij}\,|\sum w_j C_j$. The rather technical details can be found in Section 5.

It is shown in [SS97] that in the absence of (non-trivial) release dates, the use of a slightly stronger LP relaxation improves the performance guarantee of Algorithm R to 3/2. Independently, this has also been observed by Fabián A. Chudak (communicated to us by David B. Shmoys, March 1997) after reading a preliminary version of the paper on hand which only contained the 2–approximation for $R|r_{ij}|\sum w_jC_j$.

Remark 3. The reader might wonder whether the seemingly artificial random choice of the t_j's in Algorithm R is really necessary. Indeed, it is not, which also implies that we could work with a discrete probability space: From the proof of Theorem 2 one can see that we could simply set $t_j = t$ if job j is assigned to a machine-time pair (i,t) — without loosing the performance guarantee of 2. Ties are simply broken (as before) by preferring jobs with smaller indices, or even randomly. We mainly chose this presentation for the sake of giving a different, perhaps more intuitive interpretation in the special case of identical parallel machine scheduling; this will become apparent soon.

3 Identical Parallel Machine Scheduling with Release Dates

We now consider the special case of m *identical* parallel machines. Each job must be nonpreemptively processed on one of these machines, and may be assigned to any of them. The processing requirement and the release date of job j no longer depend on the machine job j is processed on and are thus denoted by p_j resp. r_j. Already the problem $P2||\sum w_jC_j$ is NP-hard, see [BCS74,LRKB77]. We consider $P|r_j|\sum w_jC_j$. There are several good reasons to explicitly investigate this special case:

- There is a purely combinatorial algorithm to solve the LP relaxation. This leads to a randomized approximation algorithm with running time $O(n\log n)$.
- The previous use of randomness obtains another interpretation in terms of scheduling by α–points (and vice versa).
- The same algorithm also is a 2–approximation algorithm for the preemptive variant $P|r_j, pmtn|\sum w_jC_j$.
- The algorithm can easily be turned into a randomized on-line algorithm, with no difference in performance guarantee.

We give an approximation algorithm that converts a preemptive schedule on a virtual single machine into a nonpreemptive one on m identical parallel machines. The single machine is assumed to be m times as fast as each of the original m machines, i. e., the (virtual) processing time of any job j on this (virtual) machine is p_j/m (w. l. o. g., we may assume that p_j is a multiple of m). The weight and the release date of job j remain the same. This kind of single machine relaxation has been used before in [CMNS97]. The algorithm is as follows:

Algorithm P

1) Construct a preemptive schedule on the virtual single machine by scheduling at any point in time among the available jobs the one with largest w_j/p_j ratio. Let C_j be the completion time of job j in this preemptive schedule.
2) Independently for each job $j \in J$, draw α_j randomly and uniformly distributed from $(0,1]$ and assign j randomly (with probability $1/m$) to one of the m machines.
3) Schedule on each machine i the jobs that were assigned to it nonpreemptively as early as possible in nondecreasing order of $C_j(\alpha_j)$.

Notice that in Step 1 whenever a job is released, the job being processed (if any) will be preempted if the released job has a higher w_j/p_j ratio.

In the analysis of Algorithm P, we use a reformulation of (LP_R) for the special case of identical parallel machines. We therefore combine for each job j and each time interval $(t,t+1]$ the variables y_{ijt}, $i=1,\dots,m$, to a single variable y_{jt} with the interpretation $y_{jt} = y_{1jt} + \cdots + y_{mjt}$. This leads to the following simplified LP:

$$
\begin{aligned}
& \text{minimize} && \sum_{j\in J} w_j C_j && \\
& \text{subject to} && \sum_{t=r_j}^{T} y_{jt} = p_j \quad \text{for all } j \in J, && (6)\\
(LP_P) \qquad & && \sum_{j\in J} y_{jt} \leqslant m \quad \text{for } t = 0,\dots,T, && (7)\\
& && C_j = \frac{p_j}{2} + \frac{1}{p_j}\sum_{t=0}^{T} y_{jt}\left(t+\tfrac{1}{2}\right) \quad \text{for all } j \in J, && (8)\\
& && y_{jt} = 0 \quad \text{for all } j \in J \text{ and } t = 0,\dots,r_j-1, && (9)\\
& && y_{jt} \geqslant 0 \quad \text{for all } j \in J \text{ and } t = r_j,\dots,T. && (10)
\end{aligned}
$$

For the special case $m = 1$, this LP was introduced by Dyer and Wolsey [DW90]. One crucial insight for the analysis of Algorithm P is that the preemptive schedule on the fast machine that is constructed in Step 1 of Algorithm P defines an optimum solution to (LP_P). If we set $y_{jt} = m$ whenever job j is being processed on the virtual machine in the period $(t,t+1]$ by the preemptive schedule, it essentially follows from the work of Goemans [Goe96] that y is an optimum solution to (LP_P).

The prior discussion especially implies that Step 1 of Algorithm P simply computes an optimum solution to the LP relaxation, as does Step 1 of Algorithm R. Moreover, the optimum solution of (LP_R) that corresponds to the preemptive schedule in Step 1 of Algorithm P is symmetric for the m machines and therefore for a job j each machine i is chosen with probability $1/m$ in Algorithm R. The symmetry also yields that for each job j the choice of t_j is not correlated with the choice of i. It can easily be seen that the probability distribution of the random variable t_j in Algorithm R exactly equals the probability distribution of $C_j(\alpha_j)$ in Algorithm P. For this, observe that the probability that $C_j(\alpha_j) \in (t,t+1]$ for some t equals the fraction y_{jt}/p_j of job j that is being processed in this time interval. Moreover, since α_j is uniformly distributed in $(0,1]$ each point in $(t,t+1]$ is equally likely to be obtained for $C_j(\alpha_j)$. Therefore, the random choice of $C_j(\alpha_j)$ in Algorithm P is an alternative way of choosing t_j as it is done in Algorithm R. Consequently, Algorithm P is a reformulation of Algorithm R for the identical parallel machine case. In particular, the expected completion time of each job is bounded from above by twice its LP completion time and Algorithm P is a 2–approximation algorithm.

At this point, it is appropriate to briefly compare this result in the single machine case ($m = 1$) with the result of Goemans [Goe97]. In Step 2, if we only work with one α for all jobs instead of individual and independent α_j's and if we draw α uniformly from $(0,1]$, then we precisely get Goemans' randomized 2–approximation algorithm

RANDOM$_\alpha$ [Goe97]. In particular, for arbitrary m, Algorithm P has the same running time $O(n \log n)$ as RANDOM$_\alpha$ since it is dominated by the effort to compute the preemptive schedule in Step 1. Finally, whereas Goemans' algorithm has a small sample space — the algorithm can only output $O(n)$ different schedules — Algorithm P can construct exponentially many different schedules.

Let us continue with some additional remarks on Algorithm P. Since (LP_P) is also a relaxation for $P|r_j, pmtn|\sum w_jC_j$ it follows that the (nonpreemptive) schedule constructed by Algorithm P is not worse than twice the optimum preemptive schedule. This improves upon a 3–approximation algorithm due to Hall et al. [HSSW96].

The analysis implies as well that (LP_P) is a 2–relaxation for $P|r_j|\sum w_jC_j$ and $P|r_j, pmtn|\sum w_jC_j$. In fact, this bound is best possible, for (LP_P). For this, consider an instance with m machines and one job of length m, unit weight, and release date 0. The optimum LP completion time is $\frac{m+1}{2}$, whereas the optimum completion time is m.

Furthermore, notice that Algorithm P can easily be turned into an on-line algorithm with competitive ratio 2. In particular, the preemptive schedule in Step 1 can be constructed until time t without any knowledge of jobs that are released afterwards. The random assignment of a job to a machine and the random choice of α_j can be done as soon as the job is released. Moreover, it follows from the analysis in the proof of Theorem 2 that we get the same performance guarantee if job j is not started before time t_j resp. $C_j(\alpha_j)$.

Finally, by a nonuniform choice of the α_j's one can improve the analysis for the on-line and the off-line algorithm to get a performance guarantee better than 2. This improvement, however, depends on m. For the single machine case, Goemans, Queyranne, Schulz, Skutella, and Wang elaborate on this and give a 1.6853–approximation algorithm [GQS+97].

The perhaps most appealing aspect of Algorithm P is that the random assignment of jobs to machines does not depend on job characteristics; any job is put with probability $1/m$ to any of the machines. This technique also proves useful for the problem without (non-trivial) release dates. The very same random machine assignment followed by list scheduling in order of nonincreasing ratios w_j/p_j on every machine is a randomized $3/2$–approximation algorithm (see [SS97] for details). Quite interestingly, its derandomized variant precisely coincides with the WSPT-rule analyzed by Kawaguchi and Kyan [KK86]: list the jobs according to nonincreasing ratios w_j/p_j and schedule the next job whenever a machine becomes available.

4 Derandomization

Up to now we have presented randomized algorithms that compute a feasible solution the *expected* value of which can be bounded from above by twice the optimum solution to the scheduling problem under consideration. This means that our algorithms will perform well on average; however, we cannot give a firm guarantee for the performance of any single execution. From a theoretical point of view it is perhaps more desirable to have (deterministic) algorithms that obtain a certain performance in all cases rather than merely with high probability.

If we can bound the expected value of the solution computed by a randomized algorithm we know that there exists at least one particular fixed assignment of values to random variables such that the value of the corresponding solution is at least as good as the expected value and can thus be bounded by the same factor. The issue of derandomization is to find such a good assignment deterministically in reasonable time.

One of the most important techniques in this context is the method of conditional probabilities. This method is implicitly contained in a paper of Erdös and Selfridge [ES73] and has been developed in a more general context by Spencer [Spe87]. The idea is to consider the random decisions in a randomized algorithm one after another and always to choose the most promising alternative. This is done by assuming that all the remaining decisions will be made randomly. Thus, an alternative is said to be most promising if the corresponding conditional expectation for the value of the solution is as small as possible.

The purpose of this section is to derandomize Algorithm R by the method of conditional probabilities. Using Remark 3 we consider the variant of Algorithm R where we set $t_j = t$ if job j is assigned to a machine-time pair (i,t). Thus, we have to construct a deterministic assignment of jobs to machine-time pairs. Unfortunately, the analysis of Algorithm R in the proof of Theorem 2 does not give a precise expression for the expected value of the solution but only an upper bound. However, in order to apply the method of conditional probabilities we have to compute in each step exact conditional expectations. Hence, for the sake of a more accessible derandomization, we modify Algorithm R by replacing Step 3 with

3') Schedule on each machine i the jobs that were assigned to it nonpreemptively in nondecreasing order of t_j, where ties are broken by preferring jobs with smaller indices. At the starting time of job j the amount of idle time on its machine has to be exactly t_j.

Since $r_{ij} \leqslant t_j$ for each job j that has been assigned to machine i and $t_j \leqslant t_k$ if job k is scheduled after job j, Step 3' defines a feasible schedule. In the proof of Theorem 2 we have bounded the idle time before the start of job j on its machine from above by t_j. Thus, the analysis still works for the modified Algorithm R which therefore has expected performance guarantee 2. The main advantage of the modification of Step 3 is that we can now give precise expressions for the expectations resp. conditional expectations of completion times.

Let y be the optimum solution we started with in Step 1 of Algorithm R. Using the same arguments as in the proof of Theorem 2 we get the following expected completion time of job j in the schedule constructed by our modified Algorithm R

$$E(C_j) = \sum_{i=1}^{m} \sum_{t=0}^{T} \frac{y_{ijt}}{p_{ij}} \left(p_{ij} + t + \sum_{k \neq j} \sum_{\ell=0}^{t-1} y_{ik\ell} + \sum_{k<j} y_{ikt} \right) .$$

Moreover, we are also interested in the conditional expectation of j's completion time if some of the jobs have already been assigned to a machine-time pair. Let $K \subseteq J$ be such a subset of jobs. For each job $k \in K$ the 0/1–variable x_{ikt} for $t \geqslant r_{ik}$ indicates if k has been assigned to the machine time-pair (i,t) ($x_{ikt} = 1$) or not ($x_{ikt} = 0$). This enables

us to give the following expressions for the conditional expectation of j's completion time. If $j \notin K$ we get

$$E_{K,x}(C_j) = \sum_{i=1}^{m} \sum_{t=0}^{T} \frac{y_{ijt}}{p_{ij}} \Big(p_{ij} + t + \sum_{k \in K} \sum_{\ell=0}^{t-1} x_{ik\ell} p_{ik} + \sum_{k \in K, k<j} x_{ikt} p_{ik} + \sum_{k \in J \setminus (K \cup \{j\})} \sum_{\ell=0}^{t-1} y_{ik\ell} + \sum_{k \in J \setminus K, k<j} y_{ikt} \Big) , \tag{11}$$

and, if $j \in K$, we get

$$E_{K,x}(C_j) = p_{ij} + t + \sum_{k \in K} \sum_{\ell=0}^{t-1} x_{ik\ell} p_{ik} + \sum_{k \in K, k<j} x_{ikt} p_{ik} + \sum_{k \in J \setminus K} \sum_{\ell=0}^{t-1} y_{ik\ell} + \sum_{k \in J \setminus K, k<j} y_{ikt} , \tag{12}$$

where (i,t) is the machine-time pair job j has been assigned to, i. e., $x_{ijt} = 1$. The following lemma is the most important part of the derandomization of Algorithm R by the method of conditional probabilities.

Lemma 4. *Let y be the optimum solution we started with in Step 1 of Algorithm R, $K \subseteq J$ and x a fixed assignment of the jobs in K to machine-time pairs. Furthermore let $j \in J \setminus K$. Then, there exists an assignment of j to a machine time pair (i,t) (i. e., $x_{ijt} = 1$) with $r_{ij} \leqslant t$ such that*

$$E_{K \cup \{j\},x}(\textstyle\sum_\ell w_\ell C_\ell) \leqslant E_{K,x}(\textstyle\sum_\ell w_\ell C_\ell) \ . \tag{13}$$

Proof. The conditional expectation on the right hand side of (13) can be written as a convex combination of conditional expectations $E_{K \cup \{j\},x}(\sum_\ell w_\ell C_\ell)$ over all possible assignments of job j to machine-time pairs (i,t) with coefficients y_{ijt}/p_{ij}. □

We therefore get a good derandomized version of Algorithm R if we replace Step 2 by

2') Set $K = \emptyset$; x:=0; for all $j \in J$ do
  i) for all possible assignments of job j to machine-time pairs (i,t) (i. e., $x_{ijt} = 1$) compute $E_{K \cup \{j\},x}(\sum_\ell w_\ell C_\ell)$;
  ii) determine the machine time pair (i,t) that minimizes $E_{K \cup \{j\},x}(\sum_\ell w_\ell C_\ell)$;
  iii) set $K := K \cup \{j\}$; $x_{ijt} = 1$;

Notice that we have replaced Step 3 of Algorithm R by 3' only to give a more accessible analysis of its derandomization. Since the value of the schedule constructed in Step 3 of Algorithm R is always at least as good as the one constructed in Step 3', the following theorem can be formulated for Algorithm R with the original Step 3.

Theorem 5. *If we replace Step 2 in Algorithm R with 2' we get a deterministic algorithm with performance guarantee 2. Moreover, the running time of this algorithm is polynomially bounded in the number of variables of the LP relaxation.*

Proof. Since $E(\sum_\ell w_\ell C_\ell) \leq 2\sum_\ell w_\ell C_\ell^{LP}$ by Theorem 2, an inductive use of Lemma 4 yields the performance guarantee 2 for the derandomized algorithm. The computation of (11) and (12) is polynomially bounded in the number of variables. Therefore, the running time of each of the n iterations in Step 2' is polynomially bounded in this number. □

Since Algorithm P can be seen as a special case of Algorithm R, it can be derandomized in the same manner. Notice that, in contrast to the situation for the randomized algorithms, we can no longer give job-by-job bounds for the derandomized algorithms.

5 Interval-Indexed Formulation

As mentioned earlier, the Algorithm R for scheduling unrelated parallel machines suffers from the exponential number of variables in the corresponding LP relaxation. However, we can overcome this drawback by using new variables which are not associated with exponentially many time intervals of length 1, but rather with a polynomial number of intervals of geometrically increasing size. This idea was earlier introduced by Hall et al. [HSW96].

For a given $\varepsilon > 0$, L is chosen to be the smallest integer such that $(1+\varepsilon)^L \geqslant T+1$. Consequently, L is polynomially bounded in the input size of the considered scheduling problem. Let $I_0 = [0,1]$ and for $1 \leqslant \ell \leqslant L$ let $I_\ell = \left((1+\varepsilon)^{\ell-1}, (1+\varepsilon)^\ell\right]$. We denote with $|I_\ell|$ the length of the ℓ–th interval, i. e., $|I_\ell| = \varepsilon(1+\varepsilon)^{\ell-1}$ for $1 \leqslant \ell \leqslant L$. To simplify notation we define $(1+\varepsilon)^{\ell-1}$ to be $\frac{1}{2}$ for $\ell = 0$. We introduce variables $y_{ij\ell}$ for $i = 1,\ldots,m$, $j \in J$, and $\ell = 0,\ldots,L$ with the following interpretation: $y_{ij\ell} \cdot |I_\ell|$ is the time job j is processed on machine i within time interval I_ℓ, or, equivalently: $(y_{ij\ell} \cdot |I_\ell|)/p_{ij}$ is the fraction of job j that is being processed on machine i within I_ℓ. Consider the following linear program in these interval-indexed variables:

$$(LP_R^\varepsilon) \qquad \text{minimize} \quad \sum_{j\in J} w_j C_j$$

$$\text{subject to} \quad \sum_{i=1}^{m} \sum_{\substack{\ell=0 \\ (1+\varepsilon)^\ell > r_{ij}}}^{L} \frac{y_{ij\ell} \cdot |I_\ell|}{p_{ij}} = 1 \quad \text{for all } j \in J, \tag{14}$$

$$\sum_{j\in J} y_{ij\ell} \leqslant 1 \quad \text{for } i = 1,\ldots,m \text{ and } \ell = 0,\ldots,L, \tag{15}$$

$$C_j = \sum_{i=1}^{m} \sum_{\ell=0}^{L} \left(\frac{y_{ij\ell} \cdot |I_\ell|}{p_{ij}} (1+\varepsilon)^{\ell-1} + \tfrac{1}{2} \cdot y_{ij\ell} \cdot |I_\ell| \right) \quad \text{for all } j \in J, \tag{16}$$

$$y_{ij\ell} = 0 \quad \text{for } i = 1,\ldots,m,\ j \in J,\ (1+\varepsilon)^\ell \leqslant r_{ij}, \tag{17}$$

$$y_{ij\ell} \geqslant 0 \quad \text{for } i = 1,\ldots,m,\ j \in J,\ \ell = 0,\ldots,L. \tag{18}$$

Consider a feasible schedule and assign the values to the variables $y_{ij\ell}$ as defined above. This solution is clearly feasible: constraints (14) are satisfied since a job j consumes p_{ij} time units if it is processed on machine i; constraints (15) are satisfied since

the total amount of processing on machine i of jobs that are processed within the interval I_ℓ cannot exceed its size. Finally, if job j is continuously being processed between $C_j - p_{hj}$ and C_j on machine h, then the right hand side of equation (16) is a lower bound on the real completion time. Thus, (LP_R^ε) is a relaxation of the scheduling problem $R|r_{ij}|\sum w_jC_j$.

Since for fixed $\varepsilon > 0$ (LP_R^ε) is of polynomial size, an optimum solution can be computed in polynomial time. The following algorithm is an adaptation of Algorithm R to the new LP:

Algorithm R^ε

1) Compute an optimum solution y to (LP_R^ε).
2) Assign job j to a machine-interval pair (i, I_ℓ) at random with probability $\frac{y_{ij\ell}\cdot|I_\ell|}{p_{ij}}$; draw t_j from the chosen time interval at random with uniform distribution.
3) Schedule on each machine i the jobs that were assigned to it nonpreemptively as early as possible in nondecreasing order of t_j.

Again, all the random assignments need to be performed independently from each other. The following lemma is a reformulation of Lemma 1 for Algorithm R^ε and can be proved analogously.

Lemma 6. *The expected processing time of each job $j \in J$ in the schedule constructed by Algorithm R^ε equals $\sum_{i=1}^{m}\sum_{\ell=0}^{L} y_{ij\ell}\cdot|I_\ell|$.*

Theorem 7. *The expected completion time of each job j in the schedule constructed by Algorithm R^ε is at most $2\cdot(1+\varepsilon)\cdot C_j^{LP}$, where C_j^{LP} is the LP completion time (defined by (16)) of the optimum solution y we started with in Step 1 of Algorithm R^ε.*

Proof. We argue almost exactly as in the proof of Theorem 2, but rather use Lemma 6 instead of Lemma 1. We consider an arbitrary, but fixed job $j \in J$. First, we also consider a fixed assignment of j to machine i and time interval I_ℓ. Again, the conditional expectation of j's starting time equals the expected idle time plus the expected processing time on machine i before j is started.

With similar arguments as in the proof of Theorem 2, we can bound the sum of the idle time plus the processing time by $2\cdot(1+\varepsilon)\cdot(1+\varepsilon)^{\ell-1}$. This, together with Lemma 6 and (16) yields the theorem. □

Theorem 7 implies that Algorithm R^ε is a $(2+2\varepsilon)$–approximation algorithm. Furthermore, (LP_R^ε) is a $(2+2\varepsilon)$–relaxation of the problem $R|r_{ij}|\sum w_jC_j$.

Of course, as suggested by Remark 3, t_j need also not be chosen at random in Step 2 of Algorithm R^ε. Moreover, Algorithm R^ε can be derandomized with the same technique as described in Section 4. In particular, the running time of the derandomized version is again polynomially bounded in the number of variables in the corresponding LP relaxation and therefore, for fixed $\varepsilon > 0$, polynomial in the input size.

Acknowledgements: The authors are grateful to Chandra S. Chekuri, Michel X. Goemans, and David B. Shmoys for helpful comments.

References

[AKP92] B. Awerbuch, S. Kutten, and D. Peleg. Competitive distributed job scheduling. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 571 – 581, 1992.

[BCS74] J. L. Bruno, E. G. Coffman Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the Association for Computing Machinery*, 17:382 – 387, 1974.

[CJM+96] C. S. 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. December 1996. Proceedings of the 29th Annual International Symposium on Microarchitecture (MICRO–29), Paris, France.

[CM96] S. Chakrabarti and S. Muthukrishnan. Resource scheduling for parallel database and scientific applications. June 1996. Proceedings of the 8th ACM Symposium on Parallel Algorithms and Architectures.

[CMNS97] C. S. 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*, pages 609 – 618, 1997.

[CPS+96] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In F. Meyer auf der Heide and B. Monien, editors, *Automata, Languages and Programming*, number 1099 in Lecture Notes in Computer Science, pages 646 – 657. Springer, Berlin, 1996. Proceedings of the 23rd International Colloquium (ICALP'96).

[CS97] F. A. Chudak and D. B. 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*, pages 581 – 590, 1997.

[DLLX90] X. Deng, H. Liu, J. Long, and B. Xiao. Deterministic load balancing in computer networks. In *Proceedings of the 2nd Annual IEEE Symposium on Parallel and Distributed Processing*, pages 50 – 57, 1990.

[DW90] M. E. Dyer and L. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26:255 – 270, 1990.

[ES73] P. Erdös and J. L. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory A*, 14:298 – 301, 1973.

[GLLRK79] 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*, 5:287 – 326, 1979.

[Goe96] M. X. Goemans. A supermodular relaxation for scheduling with release dates. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, number 1084 in Lecture Notes in Computer Science, pages 288 – 300. Springer, Berlin, 1996. Proceedings of the 5th International IPCO Conference.

[Goe97] M. X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th ACM–SIAM Symposium on Discrete Algorithms*, pages 591 – 598, 1997.

[GQS+97] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang, 1997. In preparation.

[HSSW96] 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. Preprint

516/1996, Department of Mathematics, Technical University of Berlin, Berlin, Germany, 1996. To appear in *Mathematics of Operations Research.*

[HSW96] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off–line and on–line algorithms. In *Proceedings of the 7th ACM–SIAM Symposium on Discrete Algorithms*, pages 142 – 151, 1996.

[KK86] T. Kawaguchi and S. Kyan. Worst case bound of an LRF schedule for the mean weighted flow–time problem. *SIAM Journal on Computing*, 15:1119 – 1129, 1986.

[LRKB77] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343 – 362, 1977.

[MNR96] R. Motwani, J. Naor, and P. Raghavan. Randomized approximation algorithms in combinatorial optimization. In D. S. Hochbaum, editor, *Approximation Algorithms for NP–Hard Problems*, chapter 11. Thomson, 1996.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[MSS96] R. H. Möhring, M. W. Schäffter, and A. S. Schulz. Scheduling jobs with communication delays: Using infeasible solutions for approximation. In J. Diaz and M. Serna, editors, *Algorithms – ESA'96*, volume 1136 of *Lecture Notes in Computer Science*, pages 76 – 90. Springer, Berlin, 1996. Proceedings of the 4th Annual European Symposium on Algorithms.

[PSW94] C. Phillips, C. Stein, and J. Wein. Task scheduling in networks. In *Algorithm Theory — SWAT'94*, number 824 in Lecture Notes in Computer Science, pages 290 – 301. Springer, Berlin, 1994. Proceedings of the 4th Scandinavian Workshop on Algorithm Theory.

[PSW95] C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures*, number 955 in Lecture Notes in Computer Science, pages 86 – 97. Springer, Berlin, 1995.

[RT87] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365 – 374, 1987.

[Sch96] A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP–based heuristics and lower bounds. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, number 1084 in Lecture Notes in Computer Science, pages 301 – 315. Springer, Berlin, 1996. Proceedings of the 5th International IPCO Conference.

[Spe87] J. Spencer. *Ten Lectures on the Probabilistic Method.* Number 52 in CBMS-NSF Reg. Conf. Ser. Appl. Math. SIAM, 1987.

[SS97] A. S. Schulz and M. Skutella. Random–based scheduling: New approximations and LP lower bounds. Preprint 549/1997, Department of Mathematics, Technical University of Berlin, Berlin, Germany, February 1997. To appear in Springer Lecture Notes in Computer Science, Proceedings of the 1st International Symposium on Randomization and Approximation Techniques in Computer Science (Random'97).

[ST93] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461 – 474, 1993.