MATHEMATICS OF OPERATIONS RESEARCH
Vol. 37, No. 2, May 2012, pp. 379–398
ISSN 0364-765X (print) | ISSN 1526-5471 (online)

http://dx.doi.org/10.1287/moor.1110.0520

The Power of Preemption on Unrelated Machines and Applications to Scheduling Orders

José R. Correa
Departamento de Ingeniería Industrial, Universidad de Chile, República 701, Santiago, Chile,
jcorrea@dii.uchile.cl, http://www.dii.uchile.cl/~jcorrea

Martin Skutella, José Verschae
Fakultät II, Institut für Mathematik, Technische Universität Berlin, 10623 Berlin, Germany
{martin.skutella@tu-berlin.de, http://www.coga.tu-berlin.de/people/skutella;
verschae@math.tu-berlin.de, http://www.coga.tu-berlin.de/people/verschae}

Scheduling jobs on unrelated parallel machines so as to minimize makespan is one of the basic problems in the area of machine scheduling. In the first part of the paper, we prove that the power of preemption, i.e., the worst-case ratio between the makespan of an optimal nonpreemptive and that of an optimal preemptive schedule, is at least 4. This matches the upper bound proposed in Lin and Vitter [Lin, J.-H., J. S. Vitter. 1992. ε-approximations with minimum packing constraint violation. *Proc. 24th Annual ACM Sympos. Theory of Comput. (STOC)*, ACM, New York, 771–782] two decades ago. In the second part of the paper, we consider the more general setting in which orders, consisting of several jobs, have to be processed on unrelated parallel machines so as to minimize the sum of weighted completion times of the orders. We obtain the first constant factor approximation algorithms for the preemptive and nonpreemptive cases, improving and extending a recent result by Leung et al. [Leung, J., H. Li, M. Pinedo, J. Zhang. 2007. Minimizing total weighted completion time when scheduling orders in a flexible environment with uniform machines. *Inform. Processing Lett.* **103** 119–129]. Finally, we study this problem in a parallel machine environment, obtaining a polynomial-time approximation scheme for several special cases.

Key words: machine scheduling; approximation algorithms
MSC2000 subject classification: Primary: 90B35, 68W25; secondary: 68Q25, 90C10
OR/MS subject classification: Primary: production/scheduling, approximations/heuristic
History: Received October 22, 2010; revised August 19, 2011. Published online in *Articles in Advance* December 22, 2011.

1. Introduction. We consider the classical scheduling problem of minimizing the makespan on unrelated parallel machines. In this problem we are given a set of jobs $J = \{1, \dots, n\}$ and a set of machines $M = \{1, \dots, m\}$ to process the jobs. Each job $j \in J$ has a processing requirement of p_{ij} units of time on machine $i \in M$. Every job has to be scheduled without interruption on exactly one machine, and each machine can process at most one job at a time. The objective is to minimize the makespan $C_{\max} := \max_{j \in J} C_j$, where C_j denotes the completion time of job j. In the standard three-field scheduling notation (see, e.g., Lawler et al. [21]) this problem is denoted by $R \,\|\, C_{\max}$.

In a seminal work, Lenstra et al. [23] present a 2-approximation algorithm for $R \,\|\, C_{\max}$ and show that the problem is NP-hard to approximate within a factor better than $3/2$. On the other hand, Lawler and Labetoulle [20] consider a linear relaxation of an integer programming formulation of $R \,\|\, C_{\max}$, and show that it is equivalent to its preemptive version. In this setting, denoted $R|\text{pmtn}|C_{\max}$, jobs can be interrupted and resumed later on the same or a different machine. Naturally, the corresponding linear program can be used as a lower bound for designing approximation algorithms for $R \,\|\, C_{\max}$. Indeed, Shmoys and Tardos (cited as personal communication in Lin and Vitter [27]) present a rounding procedure showing that the integrality gap is at most 4. Equivalently, this result shows that the *power of preemption* (Canetti and Irani [6], Shachnai and Tamir [32], Schulz and Skutella [31]), the worst-case ratio between the makespan of an optimal preemptive and an optimal nonpreemptive schedule, is at most 4. Interestingly, it has been unknown whether this bound is tight. In this paper we answer this question on the positive, devising a lower bound showing that the power of preemption is exactly 4. The proof of the lower bound relies on a recursive construction, where in each iteration the gap of the instance is increased.

In the second part of the paper, we apply a variant of the rounding procedure of Shmoys and Tardos [33] to a problem of scheduling orders of jobs. In this setting, clients place orders, consisting of several products, to a manufacturer owning m unrelated parallel machines. Each product has a machine-dependent processing requirement. The manufacturer has to find an assignment of products to machines (and a schedule on each machine) so as to give the best possible service to his clients. More precisely, we are given a set of machines $M = \{1, \dots, m\}$, a set of jobs $J = \{1, \dots, n\}$, and a set of orders $O \subseteq 2^J$, such that $\bigcup_{L \in O} L = J$. Each job $j \in J$ takes p_{ij} units of time to be processed on machine $i \in M$, and each order L has a weight w_L depending on its importance. Also, job j has machine-dependent release dates r_{ij}, i.e., it can only be processed on machine i

after time r_{ij}. An order $L \in O$ is completed once all its jobs have been processed. Therefore, if C_j denotes the completion time of job j, $C_L := \max_{j \in L} C_j$ denotes the completion time of order L. The goal of the manufacturer is to find a nonpreemptive schedule on the m available machines so as to minimize the sum of weighted completion times of orders, i.e., $\min \sum_{L \in O} w_L C_L$. Let us remark that in this general framework orders might not be disjoint, and therefore one job may contribute to the completion of more than one order. Adopting the standard three-field scheduling notation, we call this problem $R|r_{ij}|\sum w_L C_L$, or $R \parallel \sum w_L C_L$ in case all release dates are zero. When we impose the additional constraint that orders are disjoint, we will add the entry part to the second field.

In this paper, we give a 27/2-approximation algorithm for $R|r_{ij}|\sum w_L C_L$, improving upon the work of Leung et al. [26], and a $(4+\varepsilon)$-approximation algorithm when preemption is allowed. This is achieved by considering the interval-indexed linear programming relaxations proposed by Dyer and Wolsey [13] and Hall et al. [16], and then applying the rounding technique used to upper bound the power of preemption in $R \parallel C_{\max}$. Finally, we design a polynomial-time approximation scheme (PTAS) for $P|\text{part}|\sum w_L C_L$, when either the number of machines, the number of orders, or the number of jobs per order is constant. Our algorithm generalizes the PTASs by Hochbaum and Shmoys [17] for minimizing the makespan and by Skutella and Woeginger [35] for minimizing the sum of weighted completion times on parallel machines. This result complements the 2-approximation algorithm for $P \parallel \sum w_L C_L$ of Leung et al. [24] and Yang and Posner [39].

1.1. Relation to other scheduling problems

1.1.1. Unrelated machines. It is easy to see that this setting generalizes several classical machine scheduling problems. In particular, our problem becomes $R \parallel C_{\max}$ if there is only one order. Thus, it follows from Lenstra et al. [23] that $R \parallel \sum w_L C_L$ cannot be approximated within a factor better than 3/2, unless $P = \text{NP}$. On the other hand, if orders are singletons, our problem becomes $R \parallel \sum w_j C_j$. As in the makespan case, this problem is known to be APX-hard (Hoogeveen et al. [18]), and therefore there is no PTAS, unless $P = \text{NP}$. Using randomized rounding techniques based on a convex quadratic relaxation, Skutella [34] proposes an approximation algorithm for this problem with performance guarantee 3/2 in the case without release dates, and 2 in the more general case. However, for the more general setting $R|r_{ij}|\sum w_L C_L$, there is no constant factor approximation algorithm known. The best-known result is due to Leung et al. [26]. They obtain an approximation algorithm for the special case of related machines without release dates, denoted $Q \parallel \sum w_L C_L$, where $p_{ij} = p_j/s_i$ and $s_i > 0$ is the speed of machine i. The performance ratio of their algorithm is $1 + \rho(m-1)/(\rho + m - 1)$, where ρ denotes the ratio of the speed of the fastest machine to that of the slowest machine. In general, this guarantee is not constant and can be as bad as $m/2$.

1.1.2. Concurrent open shop. Our problem also generalizes concurrent open-shop scheduling problems that have received attention recently (see, e.g., Chen and Hall [8, 9], Leung et al. [25], Mastrolilli et al. [29], Bansal and Khot [5]). In this setting, each job has m parts, one to be processed by each machine. Parts of jobs can be processed simultaneously on different machines. Here, p_{ij} denotes the processing time of the ith part of job j that needs to be processed on machine i. The completion time C_j of job j is the time by which all of its parts have been completed. The goal is to minimize the sum of weighted completion times $\sum w_j C_j$. Thus, in our setting, orders correspond to jobs, while jobs correspond to parts. Besides proving that the problem is NP-hard, Chen and Hall [8] and Leung et al. [25] independently give a simple 2-approximation algorithm based on a linear programming relaxation. Mastrolilli et al. [29] derive a combinatorial 2-approximation algorithm, and show that the problem is inapproximable within a factor of $6/5 - \varepsilon$, unless $P = \text{NP}$. Recently, Bansal and Khot [5] show the tightness of the previous 2-approximations, assuming the Unique Games Conjecture.

1.1.3. Parallel machines. For the special case of identical parallel machines, our problem $P \parallel \sum w_L C_L$ also generalizes $P \parallel C_{\max}$ and $P \parallel \sum w_j C_j$. These two problems are well known to be NP-hard, even for the case of only two machines. For the makespan objective, Graham [15] shows that a simple list scheduling algorithm yields a 2-approximation algorithm. Furthermore, Hochbaum and Shmoys [17] present a PTAS for the problem. On the other hand, for the sum of weighted completion times objective, several approximation algorithms were proposed, until Skutella and Woeginger [35] finally presented a PTAS (see also Afrati et al. [1]).

1.1.4. Single machine. For the even more restricted setting of a single machine, the two previously mentioned problems $1 \parallel C_{\max}$ and $1 \parallel \sum w_j C_j$ can be easily solved, the first one by any feasible solution with no idle time, and the second one by applying *Smith's rule* (Smith [36]). However, the problem $1 \parallel \sum w_L C_L$ is not

only NP-hard, but it also has the same inapproximability threshold as $1|\text{prec}|\sum w_jC_j$. For the latter problem, a partial order $\preceq$ on the set of jobs is given, meaning that job j must be processed before job k if $j \preceq k$. Indeed, Woeginger [38] shows that the approximability threshold of $1|\text{prec}|\sum w_jC_j$ equals that of the same problem when the precedence graph is bipartite and all jobs on the right-hand side of the partition have zero processing time while all jobs on the left have zero weight. This type of instances of $1|\text{prec}|\sum w_jC_j$ are in an approximation preserving one-to-one correspondence with $1\parallel\sum w_LC_L$ by simply mapping each order to a job on the right side of the partition together with its predecessors.

The scheduling problem $1|\text{prec}|\sum w_jC_j$ has attracted much attention since the sixties. Lenstra and Rinnooy Kan [22] show that this problem is strongly NP-hard, even for the special case of unit weights. On the other hand, several 2-approximation algorithms have been proposed (Hall et al. [16], Chudak and Hochbaum [10], Chekuri and Motwani [7], Margot et al. [28]). Furthermore, the results in Ambühl and Mastrolilli [2] and Correa and Schulz [11] imply that $1|\text{prec}|\sum w_jC_j$ is a special case of the *Vertex Cover Problem*. However, hardness of approximation results were unknown until Ambühl et al. [3] recently proved that there is no PTAS unless NP-hard problems can be solved in randomized subexponential time. Furthermore, Bansal and Khot [4] show that a stronger version of the Unique Games Conjecture (Khot [19]) implies that $1|\text{prec}|\sum w_jC_j$ is inapproximable within a factor $2-\varepsilon$.

1.2. Outline of the paper. In §2, we briefly present the rounding technique by Shmoys and Tardos (cited as personal communication in Lin and Vitter [27]). In §3, we exhibit our instance showing the tightness of this rounding procedure. Sections 4 and 5 are concerned with the design of approximation algorithms for $R|r_{ij},\text{pmtn}|\sum w_LC_L$ and its nonpreemptive variant $R|r_{ij}|\sum w_LC_L$. In §6, we start by giving an alternative analysis of the algorithm by Leung et al. [24] and Yang et al. [39] by using a classic linear programming framework. Then we give a PTAS for the special cases of $P|\text{part}|\sum w_LC_L$ previously described.

2. A simple rounding technique. We start by briefly discussing the rounding for $R\parallel C_{\max}$ given in Lin and Vitter [27]. This algorithm takes any preemptive schedule with makespan C, and turns it into a nonpreemptive schedule with makespan at most $4C$.

As shown by Lawler and Labetoulle [20], $R|\text{pmtn}|C_{\max}$ is equivalent to the following linear program, whose variables x_{ij} denote the fraction of job j that is being processed on machine i, and C denotes the makespan of the solution:

$$
\begin{aligned}
[\text{LL}]\quad \min\quad & C \\
\text{s.t.}\quad & \sum_{i\in M} x_{ij} = 1 \quad \text{for all } j\in J, \\
& \sum_{j\in J} p_{ij}x_{ij} \le C \quad \text{for all } i\in M, && (1)\\
& \sum_{i\in M} p_{ij}x_{ij} \le C \quad \text{for all } j\in J, && (2)\\
& x_{ij}\ge 0 \quad \text{for all } i,j.
\end{aligned}
$$

Let (x_{ij}) and C be any feasible solution to [LL]. To round this fractional solution we proceed in two steps: First, we eliminate fractional variables whose corresponding processing time is too large; then, we use the rounding technique developed by Shmoys and Tardos [33] for the generalized assignment problem. In the generalized assignment problem, we are given m machines and n jobs with machine-dependent processing times p_{ij}. We also consider a cost of assigning job j to machine i, denoted by c_{ij}. Given a total budget B and makespan C, the question is to decide whether there exists a schedule with total cost at most B and makespan at most C. The main result of Shmoys and Tardos [33] is subsumed in the next theorem.

THEOREM 2.1 (SHMOYS AND TARDOS [33]). *Given a nonnegative fractional solution to the following system of inequalities*:

$$\sum_{j\in J}\sum_{i\in M} c_{ij}x_{ij} \le B, \tag{3}$$

$$\sum_{i\in M} x_{ij} = 1, \quad \textit{for all } j\in J, \tag{4}$$

there exists an integral solution $\hat{x}_{ij} \in \{0, 1\}$ *satisfying* (3) *and* (4)*, and also,*

$$\begin{aligned} x_{ij} = 0 \implies \hat{x}_{ij} = 0 \quad &\text{for all } i \in M,\ j \in J, \\ \sum_{j \in J} p_{ij}\hat{x}_{ij} \le \sum_{j \in J} p_{ij}x_{ij} + \max\{p_{ij}\colon x_{ij} > 0\} \quad &\text{for all } i \in M. \end{aligned} \tag{5}$$

Furthermore, such integral solution can be found in polynomial time.

To proceed with our rounding, let $\beta > 1$ be a fixed parameter that we will specify later. We first define a modified solution x'_{ij} as follows:

$$x'_{ij} = \begin{cases} 0 & \text{if } p_{ij} > \beta C, \\ \dfrac{x_{ij}}{X_j} & \text{else,} \end{cases} \qquad \text{where } X_j = \sum_{i\colon p_{ij} \le \beta C} x_{ij} \text{ for all } j \in J.$$

Note that

$$1 - X_j = \sum_{i\colon p_{ij} > \beta C} x_{ij} < \sum_{i\colon p_{ij} > \beta C} x_{ij}\frac{p_{ij}}{\beta C} \le 1/\beta,$$

where the last inequality follows from [LL]. Therefore, x'_{ij} satisfies that $x'_{ij} \le x_{ij}\beta/(\beta - 1)$ for all $j \in J$ and $i \in M$, and thus $\sum_{j \in J} x'_{ij}p_{ij} \le C\beta/(\beta - 1)$ for all $i \in M$. Also, note that by construction, $\sum_{i \in M} x'_{ij} = 1$ for all $j \in J$, and $x'_{ij} = 0$ if $p_{ij} > \beta C$. Then, we can apply Theorem 2.1 to x'_{ij} (for $c_{ij} = 0$), to obtain a feasible integral solution $\hat{x}_{ij}$ to [LL], and thus a feasible solution to $R \,\|\, C_{\max}$, such that for all $i \in M$,

$$\sum_{j \in J} \hat{x}_{ij}p_{ij} \le \sum_{j \in J} x'_{ij}p_{ij} + \max\{p_{ij}\colon x'_{ij} > 0\} \le \frac{\beta}{\beta - 1}C + \beta C = \frac{\beta^2}{\beta - 1}C.$$

Therefore, by optimally choosing $\beta = 2$, the makespan of the rounded solution is at most $\beta^2/(\beta - 1) = 4$ times the makespan of the fractional solution.

3. Power of preemption for $R \,\|\, C_{\max}$. We now give a family of instances showing that the integrality gap of [LL] is arbitrarily close to 4. Surprisingly, this implies that the rounding technique shown in the last section is best possible. Note that this is equivalent to saying that the optimal nonpreemptive schedule is within a factor of 4, and no better than 4, of the optimal preemptive schedule.

Let us fix $\beta \in [2, 4)$, and $\varepsilon > 0$ such that $1/\varepsilon \in \mathbb{N}$. We now construct an instance $I = I(\beta, \varepsilon)$ such that its optimal preemptive makespan is at most $(1 + \varepsilon)C$, and that any nonpreemptive solution of I has makespan at least βC. The construction is done iteratively, maintaining at each iteration a preemptive schedule of makespan $(1 + \varepsilon)C$, and where the makespan of any nonpreemptive solution is increased at each step. Because of the equivalence between [LL] and $R|\text{pmtn}|C_{\max}$, we can use assignment variables to denote preemptive schedules.

3.1. Base case. We begin by constructing an instance I_0, which will later be our first iteration. To this end consider a set of $1/\varepsilon$ jobs $J_0 = \{j(0; 1),\ j(0; 2), \ldots, j(0; 1/\varepsilon)\}$ and a set of $1/\varepsilon + 1$ machines $M_0 = \{i(1),\ i(0; 1), \ldots,\ i(0; 1/\varepsilon)\}$. Every job $j(0; \ell)$ can only be processed on machine $i(0; \ell)$, where it takes βC units of time to process, and on machine $i(1)$, where it takes a very short time. More precisely, for all $\ell = 1, \ldots, 1/\varepsilon$, we define,

$$\begin{aligned} p_{i(0;\ell)j(0;\ell)} &:= \beta C, \\ p_{i(1)j(0;\ell)} &:= \varepsilon C\frac{\beta}{\beta - 1}. \end{aligned}$$

The rest of the processing times are defined as infinity. A sketch of the situation can be found in Figure 1. Note that a feasible fractional assignment is given by setting $x_{i(0;\ell)j(0;\ell)} = 1/\beta$ and $x_{i(1)j(0;\ell)} = f_0 := (\beta - 1)/\beta$ and setting to zero all other variables. The makespan of this fractional solution is exactly $(1 + \varepsilon)C$. Indeed, the load of each machine $i \in M_0$ is exactly C, and the load associated to each job in J_0 equals $C + \varepsilon C$. Furthermore, no nonpreemptive solution with makespan less than βC can have a job $j(0; \ell)$ processed on machine $i(0; \ell)$, and therefore all jobs must be processed on $i(1)$. This yields a makespan of $C/f_0 = \beta C/(\beta - 1)$. Therefore, the makespan of any nonpreemptive solution is at least $\min\{\beta C, C/f_0\}$. If β is chosen as 2, the makespan of any nonpreemptive solution must be at least $2C$, and therefore the gap of the instance tends to 2 when ε tends to zero.

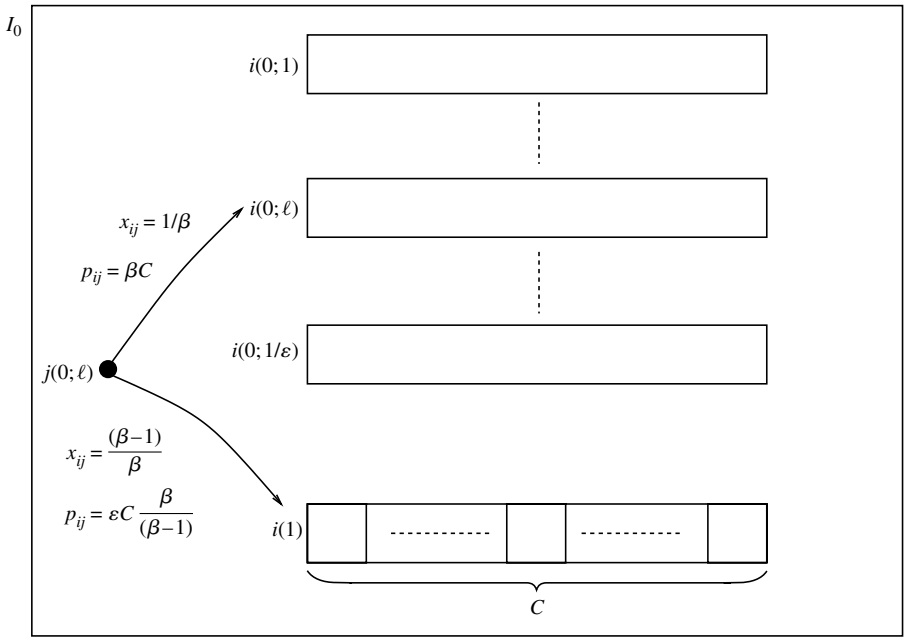

FIGURE 1. Instance I_0 and its fractional assignment.
Note. The values over the arrows x_{ij} and p_{ij} denote the fractional assignment and the processing time, respectively.

3.2. Iterative procedure. To increase the integrality gap, we proceed iteratively as follows. Starting from instance I_0, which will be the base case, we show how to construct instance I_1. As we will show later, an analogous procedure can be used to construct instance I_{n+1} from instance I_n.

Begin by making $1/\varepsilon$ copies of instance I_0, I_0^ℓ for $\ell = 1, \ldots, 1/\varepsilon$, and denote the set of jobs and machines of I_0^ℓ as J_0^ℓ and M_0^ℓ, respectively. Also, denote as $i(1;\ell)$ the copy of machine $i(1)$ belonging to M_0^ℓ. Consider a new job $j(1)$ for which $p_{i(1;\ell)j(1)} = C(\beta - \beta/(\beta-1)) = C(\beta - 1/f_0)$ for all $\ell = 1, \ldots, 1/\varepsilon$ (and ∞ otherwise), and define $x_{i(1;\ell)j(1)} = \varepsilon C / p_{i(1;\ell)j(1)}$. This way, the load of each machine $i(1;\ell)$ in the fractional solution is $(1+\varepsilon)C$, and the load corresponding to job $j(1)$ is exactly C (see Figure 2). Nevertheless, depending on the value of β, job $j(1)$ may not be completely assigned. A simple calculation shows that for $\beta = (3+\sqrt{5})/2$, job $j(1)$ is completely assigned in the fractional assignment. Furthermore, as justified before, in any nonpreemptive schedule of makespan less than βC, all jobs of instance I_0^ℓ must be processed on machine $i(1;\ell)$. Since, also, job $j(1)$ must be processed on some machine $i(1;\ell)$, then the load of that machine must be $\sum_{j \in J_0^\ell}(p_{i(1;\ell)j} + p_{i(1;\ell)j(1)}) = C\beta/(\beta-1) + C(\beta - \beta/(\beta-1)) = \beta C$. Then, the gap of the instance already constructed converges to $\beta = (3+\sqrt{5})/2$ when ε tends to 0, thus improving the gap of 2 shown before.

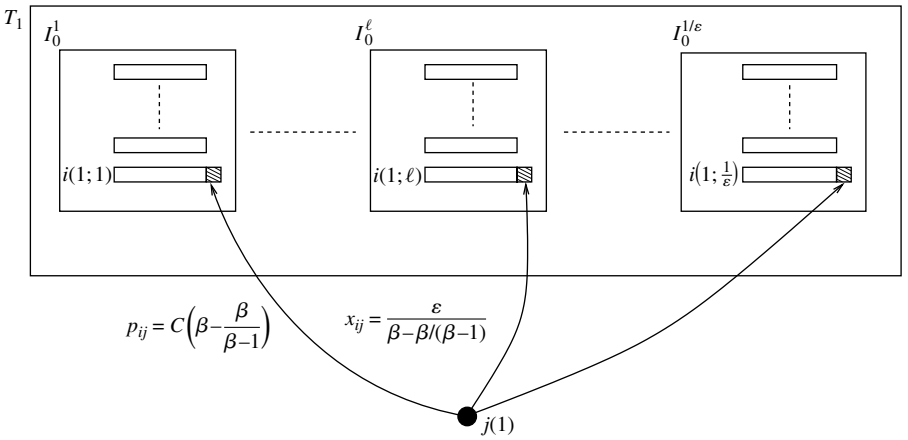

FIGURE 2. Instance T_1 and its fractional assignment.
Note. The values over the arrows x_{ij} and p_{ij} denote the fractional assignment and the processing time, respectively.

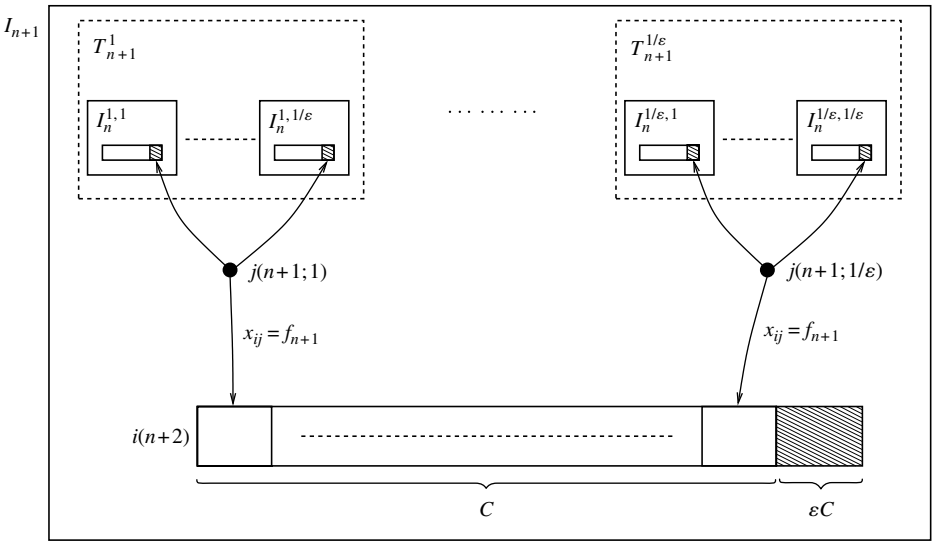

FIGURE 3. Construction of instance $I_{n+1}(\beta)$.

On the other hand, for $\beta > (3+\sqrt{5})/2$ (as we would like) there will be some fraction of job $j(1)$,

$$f_1 := 1 - \sum_{\ell=1}^{1/\varepsilon} x_{i(1;\,\ell)j(1)} = \frac{(\beta-1)f_0 - 1}{\beta f_0 - 1},$$

that must be processed elsewhere. To overcome this, we do as follows. Let us denote the instance consisting of jobs $\bigcup_{\ell=1}^{1/\varepsilon} J_0^\ell$ and machines $\bigcup_{\ell=1}^{1/\varepsilon} M_0^\ell$ as T_1, and construct $1/\varepsilon$ copies of instance T_1, T_1^k for $k = 1, \ldots, 1/\varepsilon$. Define the processing times of jobs in T_1^ℓ to infinity in all machines of T_1^k, for all $k \neq \ell$, so that jobs of T_1^ℓ can only be processed on machines of T_1^ℓ. Also, consider $1/\varepsilon$ copies of job $j(1)$, and denote them by $j(1;k)$ for $k = 1, \ldots, 1/\varepsilon$. As shown before, we can assign a fraction $1 - f_1$ of each job $j(1;k)$ to machines of T_1^k. To assign the remaining fraction f_1, we add an extra machine $i(2)$, with $p_{i(2)j(1;\,\ell)} := \varepsilon C/f_1$ (and ∞ for all other jobs), so that the fraction f_1 of each job $j(1;\ell)$ takes exactly εC units of time to process on $i(2)$. Then, defining $x_{i(2)j(1;\,\ell)} = f_1$, the total load of each job $j(1;\ell)$ does not exceed $(1+\varepsilon)C$, while the load of machine $i(2)$ is exactly C. Let us denote the instance we have constructed so far as I_1.

Following an analogous procedure to the one just described, we can construct a sequence of instances and fractional assignments (see Figure 3). Each instance I_n satisfies the following properties:

(i) The fraction of each job $j(n;1), \ldots, j(n;1/\varepsilon)$ assigned to machine $i(n+1)$ is given by $f_n = ((\beta-1)f_{n-1} - 1)/(\beta f_{n-1} - 1)$.

(ii) Job $j(n+1)$ (or any of its copies) has processing time equal to $C(\beta - 1/f_n)$ on each machine $i(n;\ell)$.

(iii) In any nonpreemptive solution of makespan less than βC, every job $j(n+1;\ell)$ must be processed on machine $i(n+2)$. Therefore, the makespan of any nonpreemptive solution is at least $\min\{\beta C, C/f_{n+1}\}$.

(iv) The makespan of the fractional solution constructed is $(1+\varepsilon)C$. In particular, the load of machine $i(n+2)$ is C, and therefore a fraction of a job which takes less than εC can still be processed on this machine without increasing the makespan.

We now show how to iteratively construct the sequence of instances I_n satisfying the properties just described. The following procedure takes β and ε as input, and if it terminates, yields an instance with gap $\beta/(1+\varepsilon)$, which equals β in the limit when ε goes to zero.

PROCEDURE I.

(i) Construct I_0 and f_0 as in §3.1, and let $n = 0$.

(ii) While $f_n > 1/(\beta-1)$, we construct instance I_{n+1} as follows.

(a) Construct an instance T_{n+1} consisting of $1/\varepsilon$ copies of instance I_n, that we denote as I_n^ℓ, for $\ell = 1, \ldots, 1/\varepsilon$, where the copy of machine $i(n+1)$ belonging to I_n^ℓ is denoted by $i(n+1;\ell)$.

(b) Create $1/\varepsilon$ copies of T_{n+1}, T_{n+1}^k for $k = 1, \ldots, 1/\varepsilon$. Denote the ℓth copy of instace I_n belonging to instance T_{n+1}^k as $I_n^{\ell k}$, and the copy of machine $i(n+1)$ that belongs to instance $I_n^{\ell k}$ as $i(n+1;\ell,k)$.

(c) Create $1/\varepsilon$ new jobs, $j(n+1;k)$, for $k = 1, \ldots, 1/\varepsilon$, and let $p_{i(n+1;\,\ell,\,k)j(n+1;\,k)} = C(\beta - 1/f_n)$ for all $k, \ell = 1, \ldots, 1/\varepsilon$ (and ∞ for all other machines).

We define the assignment variables for these new jobs as

$$x_{i(n+1;\,\ell,\,k)j(n+1;\,k)} := \frac{\varepsilon}{\beta - 1/f_n} \quad \text{for all } k, \ell = 1, \ldots, 1/\varepsilon.$$

This way, the unassigned fraction of each job $j(n+1;k)$ equals

$$f_{n+1} := 1 - \sum_{\ell=1}^{1/\varepsilon} x_{i(n+1;\,\ell,k)j(n+1;\,k)} \tag{6}$$

$$= \frac{(\beta-1)f_n - 1}{\beta f_n - 1}. \tag{7}$$

(d) To assign the remaining fraction of jobs $j(n+1;k)$ for $k = 1, \ldots, 1/\varepsilon$, we create a new machine $i(n+2)$, and define $p_{i(n+2)j(n+1;\,k)} = \varepsilon C/f_{n+1}$ for all $k = 1, \ldots, 1/\varepsilon$ (and ∞ for all other jobs).

With this we can let $x_{i(n+2)j(n+1;\,k)} = f_{n+1}$, so that this way the load of each job $j(n+1;k)$ and machine $i(n+2)$ are $(1+\varepsilon)C$ and C, respectively.

(e) Call I_{n+1} the instance constructed so far, and redefine $n \leftarrow n+1$. Observe that the defined assignment guarantees that the optimal preemptive makespan for I_{n+1} is at most $(1+\varepsilon)C$.

(iii) If $f_n \le 1/(\beta-1)$, that is, the first time the condition of step (2) is not satisfied, we do half an iteration as follows.

(a) Make $1/\varepsilon$ copies of I_n, I_n^ℓ for $\ell = 1, \ldots, 1/\varepsilon$, and call $i(n+1;\ell)$ the copy of machine $i(n+1)$ belonging to I_n^ℓ.

(b) Create a new job $j(n+1)$, and define $p_{i(n+1;\,\ell)j(n+1)} := C(\beta - 1/f_n)$ and $x_{i(n+1;\,\ell)j(n+1)} := \varepsilon$. Notice that this way job $j(n+1)$ is completely processed in the preemptive solution, and the makespan of the preemptive solution is still $(1+\varepsilon)C$, since the load of job $j(n+1)$ equals $C(\beta - 1/f_n) \le C$.

(c) Return I_{n+1}, the instance thus constructed.

LEMMA 3.1. *If* PROCEDURE I *finishes, then it returns an instance with a gap of at least* $\beta/(1+\varepsilon)$.

PROOF. It is enough to show that if the procedure finishes, then the makespan of any nonpreemptive solution is at least βC. We proceed by contradiction, assuming that instance I_{n^*} returned by PROCEDURE I has makespan strictly less than βC. Note that for the latter to hold, any job j in I_{n^*} has to be assigned to the last machine i added by PROCEDURE I for which $p_{ij} < \infty$ (this is obvious for jobs in I_0, and follows inductively for jobs in I_n, $n \le n^*$). This implies that the load of all machines $i(n^*;\ell)$ (which were the last machines included) due to jobs different from $j(n^*)$ equals C/f_{n^*-1}. Indeed, for each job j that was fractionally assigned to any of these machines $x_{i(n^*;\,\ell)j} = f_{n^*-1}$, and $i(n^*;\ell)$ was the last machine for which p_{ij} was bounded. Thus, as all machines $i(n^*;\ell)$ had load C in the fractional assignment, they will have load C/f_{n^*-1} in the nonpreemptive solution.

Furthermore, job $j(n^*)$, for which $p_{i(n^*,\ell)j(n^*)} = C(\beta - 1/f_{n^*-1})$, must be processed on some machine $i(n^*,\ell)$. It follows that there exists $\bar{\ell} \in \{1, \ldots, 1/\varepsilon\}$ such that the load of machine $i(n^*,\bar{\ell})$ is $C/f_{n^*-1} + C(\beta - 1/f_{n^*-1}) = \beta C$, which is a contradiction. □

To prove that the procedure in fact finishes, we first show a technical lemma.

LEMMA 3.2. *For each* $\beta \in [2,4)$, *if* $f_n > 1/\beta$, *then* $f_{n+1} \le f_n$.

PROOF. It follows from Equation (7) that,

$$f_{n+1} - f_n = \frac{-\beta f_n^2 + \beta f_n - 1}{\beta f_n - 1}.$$

Note that the numerator of the last expression is always negative because the square equation $-\beta x^2 + \beta x - 1$ has no real roots for $0 \le \beta < 4$. Because, by hypothesis, the denominator of this expression is always positive, the result follows. □

LEMMA 3.3. PROCEDURE I *finishes.*

PROOF. We need to show that for every $\beta \in [2,4)$, there exists $n^* \in \mathbb{N}$ such that $f_{n^*} \le 1/(\beta-1)$. If this does not hold, then $f_n > 1/(\beta-1) > 1/\beta$ for all $n \in \mathbb{N}$. Then Lemma 3.2 implies that $\{f_n\}_{n\in\mathbb{N}}$ is a decreasing sequence. Therefore, f_n must converge to some real number $L \ge 1/(\beta-1)$. Taking the limit when n goes to infinity on both sides of (7) we obtain

$$L = \frac{(\beta-1)L - 1}{\beta L - 1},$$

and therefore L is a root of equation $-\beta x^2 + \beta x - 1$, which is a contradiction since $\beta \in [2,4)$. □

We have proved the following theorem.

THEOREM 3.1. *For each $\beta \in [2,4)$ and $\varepsilon > 0$, there is an instance I of $R \| C_{\max}$, for which the optimal preemptive makespan is at most $C(1+\varepsilon)$ and the optimal nonpreemptive makespan is at least βC.*

THEOREM 3.2. *The integrality gap of relaxation [LL] is 4.*

4. A $(4+\varepsilon)$-approximation algorithm for $R|r_{ij}, \text{pmtn}|\sum w_L C_L$. In this section we adapt the rounding technique discussed in §2 to derive a $(4+\varepsilon)$-approximation algorithm for the preemptive version of $R|r_{ij}|\sum w_L C_L$. Our algorithm is based on a time-indexed linear program, whose variables correspond to the fraction of each job processed at each time on each machine. This kind of linear relaxation was originally introduced by Dyer and Wolsey [13] for $1|r_j|\sum w_j C_j$, and was extended by Schulz and Skutella [31], who used it to obtain a $(3/2+\varepsilon)$-approximation and a $(2+\varepsilon)$-approximation for $R\|\sum w_j C_j$ and $R|r_j|\sum w_j C_j$, respectively.

Let us consider a time horizon T, large enough to upper bound the greatest completion time of any reasonable schedule, for instance, $T = \max_{i\in M,\, k\in J}\{r_{ik} + \sum_{j\in J} p_{ij}\}$. We divide the time horizon into exponentially growing time intervals, so that there is only polynomially many of them. For that, let ε be a fixed parameter, and let q be the smallest integer such as $(1+\varepsilon)^{q-1} \geq T$. Then, we consider the intervals $[0,1], (1,(1+\varepsilon)], ((1+\varepsilon), (1+\varepsilon)^2], \ldots, ((1+\varepsilon)^{q-2}, (1+\varepsilon)^{q-1}]$. To simplify the notation, let us define $\tau_0 = 0$, and $\tau_\ell = (1+\varepsilon)^{\ell-1}$, for each $\ell = 1,\ldots,q$. With this, the ℓth interval corresponds to $(\tau_{\ell-1}, \tau_\ell]$.

Given any preemptive schedule, let $y_{ij\ell}$ be the fraction of job j that is processed on machine i on the ℓth interval. Then, $p_{ij}y_{ij\ell}$ is the amount of time that job j is processed on machine i on the ℓth interval. We consider the following linear program:

$$
\begin{aligned}
[\text{DW}] \quad \min \ & \sum_{L\in O} w_L C_L \\
& \sum_{i\in M}\sum_{\ell=1}^{q} y_{ij\ell} = 1 \quad \text{for all } j \in J, && (8)\\
& \sum_{j\in J} p_{ij} y_{ij\ell} \leq \tau_\ell - \tau_{\ell-1} \quad \text{for all } \ell = 1,\ldots,q \text{ and } i \in M, && (9)\\
& \sum_{i\in M} p_{ij} y_{ij\ell} \leq \tau_\ell - \tau_{\ell-1} \quad \text{for all } \ell = 1,\ldots,q \text{ and } j \in J, && (10)\\
& \sum_{i\in M}\left(y_{ij1} + \sum_{\ell=2}^{q} \tau_{\ell-1} y_{ij\ell}\right) \leq C_L \quad \text{for all } L \in O,\ j \in L, && (11)\\
& y_{ij\ell} = 0 \quad \text{for all } j, i, \ell\colon r_{ij} > \tau_\ell, && (12)\\
& y_{ij\ell} \geq 0 \quad \text{for all } i, j, \ell. && (13)
\end{aligned}
$$

It is easy to see that this is a relaxation of our problem. Indeed, Equation (8) assures that every job must be completely processed. Inequality (9) must hold since in each interval ℓ and machine i the total amount of time available is at most $\tau_\ell - \tau_{\ell-1}$. Similarly, inequality (10) holds since no job can be simultaneously processed on two machines at the same time, and therefore, for a fixed interval the total amount of time that can be used to process a job is at most the length of the interval. To see that (11) is valid, notice that $p_{ij} \geq 1$ (because, without loss of generality, we assume that $p_{ij} \in \mathbb{Z}_{>0}$), and thus $C_L \geq 1$ for all $L \in O$. Also notice that $C_L \geq \tau_{\ell-1}$ for all $L, j \in L, i, \ell$, such that $y_{ij\ell} > 0$. Thus, the left-hand side of inequality (11) is a convex combination of values at most C_L. Finally, Equation (12) must hold since no part of a job can be assigned to an interval that finishes before the release date in any given machine.

The idea behind our approximation algorithm is rather standard. We first compute the optimal solution of [DW], and then we transform this solution into a preemptive schedule whose cost is within a constant factor of the cost of the optimal solution of the linear program. To construct the schedule we do as follows. First, as was done in §2, truncate the solution given by the linear program by taking to zero all variables that assign a job to an interval that is considerably larger than the value of C_L given by [DW]. Afterward, we use the result by Lawler and Labetoulle [20] to construct a feasible schedule inside each interval, making sure that no job is processed on two machines at the same time.

More precisely, let $y^*_{ij\ell}$ and C^*_L be the optimal solution of [DW]. Let $j \in J$ and

$$L = \arg\min\{C^*_{L'} \mid j \in L' \in O\}.$$

For a given parameter $\beta > 1$, we define

$$y'_{ij\ell} = \begin{cases} 0 & \text{if } \tau_{\ell-1} > \beta C^*_L, \\ \dfrac{y^*_{ij\ell}}{Y_j} & \text{if } \tau_{\ell-1} \le \beta C^*_L, \end{cases} \tag{14}$$

where

$$Y_j = \sum_{i\in M} \sum_{\ell:\, \tau_{\ell-1} \le \beta \cdot C^*_L} y^*_{ij\ell}.$$

The modified solution y' satisfies the following lemma.

LEMMA 4.1. *The modified solution $y'_{ij\ell}$, obtained by applying Equation (14) to $y^*_{ij\ell}$ satisfies*

$$\sum_{i\in M} \sum_{\ell=1}^{q} y'_{ij\ell} = 1 \quad \textit{for all } j, \tag{15}$$

$$\sum_{j\in J} p_{ij} y'_{ij\ell} \le \frac{\beta}{\beta-1}(\tau_\ell - \tau_{\ell-1}) \quad \textit{for all } i, \ell, \tag{16}$$

$$\sum_{i\in M} p_{ij} y'_{ij\ell} \le \frac{\beta}{\beta-1}(\tau_\ell - \tau_{\ell-1}) \quad \textit{for all } j, \ell, \tag{17}$$

$$y'_{ij\ell} = 0 \qquad \textit{if } \tau_{\ell-1} > \beta C^*_L \quad \textit{for all } L \in O, \quad j \in L. \tag{18}$$

PROOF. It is clear that $y'_{ij\ell}$ satisfies (15) since

$$\sum_{i\in M} \sum_{\ell=1}^{q} y'_{ij\ell} = \sum_{i\in M} \sum_{y'_{ij\ell}>0} \frac{y^*_{ij\ell}}{Y_j} = \frac{1}{Y_j} \sum_{i\in M} \sum_{\ell:\, \tau_{\ell-1} \le \beta C^*_L} y^*_{ij\ell} = 1.$$

Furthermore, to show that inequalities (16) and (17) hold, note that

$$1 - Y_j = \sum_{i\in M} \sum_{\ell:\, \tau_{\ell-1} > \beta \cdot C^*_L} y^*_{ij\ell} < \sum_{i\in M} \sum_{\ell:\, \tau_{\ell-1} > \beta \cdot C^*_L} y^*_{ij\ell} \frac{\tau_{\ell-1}}{\beta C^*_L} \le \frac{C^*_L}{\beta C^*_L} = \frac{1}{\beta}.$$

The last inequality follows from inequality (11), and by noting that $\ell \neq 1$ whenever $\tau_{\ell-1} > \beta \cdot C^*_L$. Then, $Y_j \ge (\beta-1)/\beta$, and thus $y'_{ij\ell} \le (\beta/(\beta-1))y^*_{ij\ell}$. With this, inequalities (16) and (17) follow from inequalities (9) and (10). Finally, note that Equation (18) follows from the definition of y'. □

Equation (18) in the previous lemma implies that the variables $y'_{ij\ell}$ only assign jobs to intervals that finish before βC^*_L in case $j \in L$. On the other hand, as shown by inequalities (16) and (17), the amount of load assigned to each interval will not fit in the available space. Thus, we will have to increase the size of every interval by a factor of $\beta/(\beta-1)$. With the latter observations, we are ready to describe the algorithm.

Algorithm (GREEDY PREEMPTIVE LP)

(i) Solve [DW] to optimality and call the solution y^* and $(C^*_L)_{L\in O}$.

(ii) Define $y'_{ij\ell}$ using Equation (14).

(iii) Construct a preemptive schedule S as follows.

(a) For each $\ell = 1, \ldots, q$, define $x_{ij} = y'_{ij\ell}$ and $C = (\tau_\ell - \tau_{\ell-1})\beta/(\beta-1)$. Apply the algorithm by Lawler and Labetoulle [20] to this fractional solution, to obtain a preemptive schedule (i.e., no job is processed in parallel by two machines) of makespan C. Call the preemptive schedule obtained S_ℓ.

(b) For each job $j \in J$ that is processed by schedule S_ℓ at time $t \in [0, C]$ on machine $i \in M$, make schedule S process j on machine i at time $t + \tau_{\ell-1}\beta/(\beta-1)$.

LEMMA 4.2. *Algorithm* GREEDY PREEMPTIVE LP *constructs a feasible schedule where the completion time of each order $L \in O$ is less than $C^*_L(1+\varepsilon)\beta^2/(\beta-1)$.*

PROOF. Note that inequalities (16) and (17) imply that for each $\ell = 1, \ldots, q$, $x_{ij} = y'_{ij\ell}$ and $C = (\tau_\ell - \tau_{\ell-1})\beta/(\beta-1)$ satisfy (1) and (2). Then, the makespan of each schedule S_ℓ is less than $(\tau_\ell - \tau_{\ell-1})\beta/(\beta-1)$ (Lawler and Labetoulle [20]), and thus the schedule S_ℓ defines the schedule S in the disjoint amplified interval $[\tau_{\ell-1}\beta/(\beta-1), \tau_\ell\beta/(\beta-1))$. Also, it follows from Equation (15) that the schedule S completely processes every job.

Let us consider a fixed order $L \in O$ and job $j \in L$. Let ℓ^* be the last interval for which $y'_{ij\ell} > 0$, for some machine $i \in M$; i.e., $\ell^* = \max_{i \in M} \max\{\ell \in \{1, \dots, q\}: y'_{ij\ell} > 0\}$. Then, the completion time C_j is smaller than $\tau_{\ell^*}\beta/(\beta-1)$. To further bound C_j, we consider two cases. If $\ell^* = 1$, then

$$C_j \le \frac{\beta}{\beta-1} \le C_L^*(1+\varepsilon)\frac{\beta}{\beta-1},$$

where the last inequality follows since $C_L^* \ge 1$. On the other hand, if $\ell^* > 1$, Equation (18) implies that

$$C_j \le \tau_{\ell^*}\frac{\beta}{\beta-1} \le \tau_{\ell^*-1}(1+\varepsilon)\frac{\beta}{\beta-1} \le C_L^*(1+\varepsilon)\frac{\beta^2}{\beta-1}.$$

Thus, by taking the maximum over all $j \in L$, the completion time of the order L is upper bounded by $C_L^*(1+\varepsilon)\beta^2/(\beta-1)$. □

THEOREM 4.1. *Algorithm* GREEDY PREEMPTIVE LP *is a* $(4+\varepsilon)$*-approximation for* $\beta = 2$.

PROOF. Let C_L be the completion time of order L given by Algorithm GREEDY PREEMPTIVE LP. Taking $\beta = 2$ in the last lemma, which is the optimal choice, it follows that $C_L \le C_L^*(1+\varepsilon)\beta^2/(\beta-1) = 4(1+\varepsilon)C_L^*$. Then, multiplying C_L by its weight w_L and adding over all $L \in O$, we conclude the the cost of the schedule constructed is no larger than $4(1+\varepsilon)$ times the cost of the optimal solution for [DW], which is a lower bound on the cost of the optimal preemptive schedule. □

5. A constant factor approximation for $R|r_{ij}|\sum w_L C_L$. In this section we propose the first constant factor approximation algorithm for the nonpreemptive version of our problem, $R|r_{ij}|\sum w_L C_L$, improving the results in Leung et al. [26]. Our algorithm consists of applying the rounding shown in §2 to an adaptation of the interval-index linear programming relaxation developed by Hall et al. [16].

Let us consider a large enough time horizon T as in the last section. We divide the time horizon into exponentially growing time intervals, so that there are only polynomially many. For that, let $\alpha > 1$ be a parameter which will be determined later, and let q be the smallest integer such that $\alpha^{q-1} \ge T$. With this, consider the intervals $[1, 1], (1, \alpha], (\alpha, \alpha^2], \dots, (\alpha^{q-2}, \alpha^{q-1}]$.

To simplify the notation, let us define $\tau_0 = 1$ and $\tau_\ell = \alpha^{\ell-1}$ for each $\ell = 1, \dots, q$. With this, the ℓth interval corresponds to $(\tau_{\ell-1}, \tau_\ell]$. Note that, for technical reasons, these definitions slightly differ from the ones in the previous section.

To model the scheduling problem we consider the variables $y_{ij\ell}$, indicating whether job j is finished on machine i and on interval ℓ. We consider, without loss of generality, that all processing times p_{ij} are positive integers. Thus, the first interval $[\tau_0, \tau_1] = [1, 1]$ is well defined since no job finishes before time 1. The y variables allow us to write the following linear program based on that in Hall et al. [16], which is a relaxation of the scheduling problem even when integrality constraints are imposed:

$$\text{[HSSW]} \qquad \min \sum_{L \in O} w_L C_L$$

$$\sum_{i \in M}\sum_{\ell=1}^{q} y_{ij\ell} = 1 \quad \text{for all } j \in J, \tag{19}$$

$$\sum_{s=1}^{\ell}\sum_{j \in J} p_{ij} y_{ijs} \le \tau_\ell \quad \text{for all } i \in M \text{ and } \ell = 1, \dots, q, \tag{20}$$

$$\sum_{i \in M}\sum_{\ell=1}^{q} \tau_{\ell-1} y_{ij\ell} \le C_L \quad \text{for all } L \in O \text{ and } j \in L, \tag{21}$$

$$y_{ij\ell} = 0 \quad \text{for all } i, \ell, J\text{: } p_{ij} + r_{ij} > \tau_\ell, \tag{22}$$

$$y_{ij\ell} \ge 0 \quad \text{for all } i, j, \ell. \tag{23}$$

It is clear that [HSSW] is a relaxation of our problem. Indeed, (19) guarantees that each job finishes in some time interval. The left-hand side of (20) corresponds to the total load processed on machine i and interval $[0, \tau_\ell]$, and therefore the inequality is valid. The sum in inequality (21) corresponds exactly to $\tau_{\ell-1}$, where ℓ is the

interval where job j finishes, so that is at most C_j, and therefore it is upper bounded by C_L if $j \in L$. Also, it is clear that (22) must hold since no job j can finish processing on machine i before $p_{ij} + r_{ij}$.

Let $(y^*_{ij\ell})_{ij\ell}$ and $(C^*_L)_L$ be an optimal solution to [HSSW]. To obtain a feasible schedule we need to round such solution into an integral one. To this end, Hall et al. [16] used the result of Shmoys and Tardos [33] given in Theorem 2.1. If, in [HSSW], all orders are singleton (as is the situation in Hall et al. [16]), (21) becomes an equality so that one can use Theorem 2.1 to round a fractional solution to an integral solution of smaller total cost and such that the right-hand side of Equation (20) is increased to $\tau_\ell + \max\{p_{ij}\colon y_{ij\ell} > 0\} \le 2\tau_\ell$, where the last inequality follows from (22). This can be used to derive a constant factor approximation algorithm for the problem. In our setting, however, it is not possible to apply Theorem 2.1 directly, because of the nonlinearity of the objective function. To overcome this difficulty, consider $j \in J$ and $L = \arg\min\{C^*_{L'} \mid j \in L' \in O\}$, and apply (14) to y^*, thus obtaining a new fractional assignment y'. With this we obtain a solution in which job j is never assigned to an interval starting after βC^*_L. Moreover, analogously to Lemma 4.1, the following lemma holds.

LEMMA 5.1. *The modified solution $y'_{ijl} \ge 0$ satisfies*

$$\sum_{i=1}^{m}\sum_{l=1}^{q} y'_{ij\ell} = 1 \quad \text{for all } j \in J, \tag{24}$$

$$\sum_{s=1}^{\ell}\sum_{j\in J} p_{ij} y'_{ijs} \le \frac{\beta}{\beta-1}\tau_\ell \quad \text{for all } i, \ell, \tag{25}$$

$$y'_{ij\ell} = 0 \quad \text{if } p_{ij} + r_{ij} > \tau_\ell \text{ or } \tau_{\ell-1} > \beta C^*_L, \ \ \forall i, j, \ell, L\colon j \in L. \tag{26}$$

PROOF. Clearly $y'_{ij\ell}$ satisfies (24), since

$$\sum_{i\in M}\sum_{\ell=1}^{q} y'_{ij\ell} = \sum_{i\in M}\sum_{\ell\colon y'_{ij\ell}>0} \frac{y^*_{ij\ell}}{Y_j} = \sum_{i\in M}\sum_{\ell\colon \tau_{\ell-1}\le \beta C^*_L} \frac{y^*_{ij\ell}}{Y_j} = 1.$$

Furthermore, to see that (25) also holds, observe that

$$C^*_L \ge \sum_{i\in M}\sum_{\ell=1}^{q} \tau_{\ell-1} y^*_{ij\ell} \ge \sum_{i\in M}\sum_{\ell\colon \tau_{\ell-1}>\beta C^*_L} \tau_{\ell-1} y^*_{ij\ell} \ge \beta C^*_L \sum_{i\in M}\sum_{\ell\colon \tau_{\ell-1}>\beta C^*_L} y^*_{ij\ell} = \beta C^*_L (1 - Y_j).$$

Thus, we have that $Y_j \ge (\beta-1)/\beta$, which implies $y'_{ij\ell} \le (\beta/(\beta-1)) y^*_{ij\ell}$. Therefore, inequality (25) follows from (20). Finally, note that (26) is immediate from (22) and the definition of $y'_{ij\ell}$. □

With the previous lemma on hand we are in position to apply Theorem 2.1. We round $y'_{ij\ell}$ to an integral solution $\hat{y}_{ij\ell} \in \{0, 1\}$ satisfying (24), (26), and

$$\sum_{j\in J} \hat{y}_{ij\ell} p_{ij} \le \sum_{j\in J} y'_{ij\ell} p_{ij} + \max\{p_{ij}\colon y'_{ij\ell} > 0, j \in J\} \le \sum_{j\in J} y'_{ij\ell} p_{ij} + \tau_\ell, \tag{27}$$

where the last inequality follows from (5) and (26).

We are now ready to give the algorithm for $R|r_{ij}|\sum w_L C_L$.

Algorithm (GREEDY-LP)

(1) Solve [HSSW] obtaining an optimal solution $(y^*_{ij\ell})$.
(2) Modify the solution according to (14) to obtain $(y'_{ij\ell})$ satisfying (24), (25), and (26).
(3) Round $(y'_{ij\ell})$ using Theorem 2.1 to obtain an integral solution $(\hat{y}_{ij\ell})$ as above.
(4) Let $J_{i\ell} = \{j \in J\colon \hat{y}_{ij\ell} = 1\}$. Greedily schedule in each machine i, all jobs in $\bigcup_{\ell=1}^{q} J_{i\ell}$, starting from those in J_{i1} until we reach J_{iq} (with an arbitrary order inside each set $J_{i\ell}$), respecting the release dates.

To break down the analysis let us first show that GREEDY-LP is a constant factor approximation for the case in which all release dates are zero.

THEOREM 5.1. *Algorithm* GREEDY-LP *is a* (27/2)*-approximation for* $R \,\|\, \sum w_L C_L$.

PROOF. Let us fix a machine i and take a job $j \in L$ such that $\hat{y}_{ij\ell} = 1$, so that $j \in J_{i\ell}$. Clearly, C_j, the completion time of job j in Algorithm GREEDY-LP, is at most the total processing time of jobs in $\bigcup_{k=1}^{\ell} J_{ik}$. Then,

$$\begin{aligned} C_j &\le \sum_{s=1}^{\ell} \sum_{k \in J} p_{ik} \hat{y}_{iks} \\ &\le \sum_{s=1}^{\ell} \left(\sum_{k \in J} p_{ik} y'_{iks} + \tau_s \right) \\ &\le \frac{\beta}{\beta - 1} \tau_\ell + \sum_{s=1}^{\ell} \tau_s \\ &\le \left(\frac{\beta\alpha}{\beta - 1} + \frac{\alpha^2}{\alpha - 1} \right) \tau_{\ell-1} \\ &\le \beta\alpha \left(\frac{\beta}{\beta - 1} + \frac{\alpha}{\alpha - 1} \right) C^*_L. \end{aligned}$$

The second inequality follows from (27), the third from (25), and the fourth follows from the definition of τ_k. The last inequality follows since, by condition (5), $\hat{y}_{ij\ell} = 1$ implies $y'_{ij\ell} > 0$, so that by (26) we have $\tau_{\ell-1} \le \beta C^*_L$. Optimizing over the approximation factor, the best possible guarantee given by this method is attained at $\alpha = \beta = 3/2$, and thus we conclude that $C_j \le 27/2 \cdot C^*_L$ for all $L \in O$ and $j \in L$. □

THEOREM 5.2. *Algorithm* GREEDY-LP *is a* (27/2)*-approximation for* $R|r_{ij}|\sum w_L C_L$.

PROOF. Similarly to the proof of Theorem 5.1, we will show that the solution given by Algorithm GREEDY-LP satisfies $C_j \le 27/2 \cdot C^*_L$, even in the presence of release dates. Let us define $\bar{\tau}_\ell := 1/(\alpha - 1) + \sum_{s=1}^{\ell} (\sum_{k \in J} p_{ik} y'_{iks} + \tau_s)$. We will see that it is possible to schedule every set of jobs $J_{i\ell}$ on machine i between time $\bar{\tau}_{\ell-1}$ and time $\bar{\tau}_\ell$ (with an arbitrary order inside each interval), respecting all release dates. Indeed, assuming that $1 < \alpha \le 2$, it follows from (5) and (26) that for every $j \in J_{i\ell}$,

$$r_{ij} \le \tau_\ell \le \frac{1}{\alpha - 1} + \frac{\tau_\ell - 1}{\alpha - 1} = \frac{1}{\alpha - 1} + \sum_{k=1}^{\ell-1} \tau_k \le \bar{\tau}_{\ell-1}.$$

Thus, job j is available for processing on machine i at time $\bar{\tau}_{\ell-1}$. On the other hand, note that $\bar{\tau}_\ell - \bar{\tau}_{\ell-1} = \sum_{j \in J} p_{ij} y'_{ij\ell} + \tau_\ell$, so it follows from (27) that all jobs in $J_{i\ell}$ fit inside $(\bar{\tau}_{\ell-1}, \bar{\tau}_\ell]$.

We conclude that in the schedule constructed by GREEDY-LP, any job $j \in J_{i\ell}$ is processed before $\bar{\tau}_\ell$. Therefore, as in Theorem 5.1,

$$\begin{aligned} C_j &\le \frac{1}{\alpha - 1} + \sum_{s=1}^{\ell} \left(\sum_{k \in J} p_{ik} y'_{iks} + \tau_k \right) \\ &\le \left(\frac{\beta\alpha}{\beta - 1} + \frac{\alpha^2}{\alpha - 1} \right) \tau_{\ell-1} \\ &\le \beta\alpha \left(\frac{\beta}{\beta - 1} + \frac{\alpha}{\alpha - 1} \right) C^*_L. \end{aligned}$$

Again, choosing $\alpha = \beta = 3/2$ we obtain $C_j \le 27/2 \cdot C^*_L$. □

6. Parallel machines.

In what follows we study the problem of scheduling orders in a parallel machine environment. First we show an alternative analysis of a 2-approximation algorithm by Leung et al. [24] and Yang and Posner [39]. Afterward, we derive a PTAS for several special cases.

6.1. A 2-approximation algorithm.

We now show a 2-approximation algorithm for the parallel machine version of our problem $P \parallel \sum w_L C_L$. Our reasoning is based on a classical linear program first studied by Queyranne [30] and Dyer and Wolsey [13] for $1 \parallel \sum w_j C_j$ and $1|r_j|\sum w_j C_j$, respectively.

Let M_j be the midpoint of job j in a given nonpreemptive schedule; in other words, $M_j = C_j - p_j/2$. Eastman et al. [14] implicitly show that for any subset of jobs $S \subseteq J$ and any feasible schedule on m parallel machines, the following inequality holds:

$$\sum_{j \in S} p_j M_j \ge \frac{p(S)^2}{2m},$$

where $p(S) := \sum_{j \in S} p_j$. These inequalities are called the *parallel inequalities*. It follows that OPT, the value of an optimal schedule, is lower bounded by the optimum solution value of the following linear program [LP]:

$$\begin{aligned} \min \quad & \sum_{L \in O} w_L C_L \\ \text{s.t.} \quad & C_L \geq M_j + \frac{p_j}{2} \quad \text{for all } L \in O \text{ and } j \in L, \\ & \sum_{j \in S} p_j M_j \geq \frac{p(S)^2}{2m} \quad \text{for all } S \subseteq N. \end{aligned}$$

Queyranne [30] shows that [LP] can be solved in polynomial time since separating the parallel inequalities reduces to submodular function minimization. Let $M_1^*, \ldots, M_n^*$ be an optimal solution to [LP] and assume without loss of generality that $M_1^* \leq M_2^* \leq \cdots \leq M_n^*$. Clearly, $C_L^* = \max_{j \in L} M_j^* + p_j/2$, so the optimal solution is completely determined by the M_j-values. Consider the algorithm that first solves (LP) and then schedules jobs greedily according to the order $M_1^* \leq M_2^* \leq \cdots \leq M_n^*$. Let C_j^A denote the completion time of job j in the schedule given by the algorithm, so that $C_L^A = \max\{C_J^A \colon j \in L\}$. It is easy to see that C_j^A equals the time at which job j is started by the algorithm, S_j^A, plus p_j. Furthermore, at any point in time before S_j^A, all machines were busy processing jobs in $\{1, \ldots, j-1\}$; thus, $S_j^A \leq p(\{1, \ldots, j-1\})/m$. It follows that

$$C_L^A \leq \max_{j \in L} \left\{ \frac{p(\{1, \ldots, j-1\})}{m} + p_j \right\}.$$

Also, $M_j^* p(\{1, \ldots, j\}) \geq \sum_{l \in \{1, \ldots, j\}} p_l M_l^* \geq p(\{1, \ldots, j\})^2/2m$. Then,

$$C_L^* \geq \max_{j \in L} \left\{ \frac{p(\{1, \ldots, j\})}{2m} + \frac{p_j}{2} \right\}.$$

We conclude that $C_L^A \leq 2C_L^*$, which implies that the algorithm returns a solution whose value is within a factor of 2 to OPT. We have proved the following.

Lemma 6.1 (Leung et al. [24], Yang and Posner [39]). *Let $M_1^*, \ldots, M_n^*$ be an optimal solution to* [LP]. *List scheduling in nondecreasing order of M_j^* yields a 2-approximation algorithm for $P \parallel \sum w_L C_L$.*

6.2. Polynomial time approximation schemes. Finally we design a PTAS for $P|\text{part}|\sum w_L C_L$ when either the number of machines is constant, the number of jobs is constant, or the number of orders is constant. First, we describe the case where the number of jobs of each order is bounded by a constant K, and then we will justify that this implies the existence of PTASs for the other cases. The results in this section closely follow the PTAS developed by Afrati et al. [1] for $P|r_j|\sum w_j C_j$. However, it is technically more involved mainly for three reasons. Firstly, the structure of optimal solutions is much more delicate than in Afrati et al. [1], and thus we must take care that each transformation we apply to an optimal solution does not brake this structure. Also, the precise localization of orders is significantly more complicated. Finally, we need to be slightly more careful in the final placing of jobs.

For the sake of brevity, we will focus on the main differences between our algorithm and the one in Afrati et al. [1]. For a more detailed exposition of these subjects, see Verschae [37].

To help us give structure to our problem, we consider the following definition: for every integer t we will denote by I_t the interval $[(1+\varepsilon)^t, (1+\varepsilon)^{t+1})$, and $|I_t|$ its length, i.e., $|I_t| = \varepsilon(1+\varepsilon)^t$. To shorten notation, in what follows if the base of a logarithm is missing, we assume that it takes base $(1+\varepsilon)$.

Besides the rounding and partitioning techniques commonly used in PTASs, a basic procedure we will use repeatedly is that of *stretching*, which consists of stretching the time axis by a factor of $1+\varepsilon$. Clearly, each time a solution is modified by such a procedure it increases its cost only by a factor of $1+\varepsilon$. The two basic stretching procedures we use are as follows:

(i) Stretch Completion Times: This procedure consists of moving all jobs to the right, so that the completion time of a job j becomes $C_j' = (1+\varepsilon)C_j$ in the new schedule. Clearly, the procedure creates an idle time of length at least εp_j before the start of each job j.

(ii) Stretch Intervals: The objective of this procedure is to create idle time in every interval, except for those having a job that completely covers them. As before, it consists of shifting jobs to the following interval. More precisely, if job j finishes in I_t and occupies d_j time units on I_t, we will move j to I_{t+1} by pushing it exactly $|I_t|$ time units, so it also uses exactly d_j time units in I_{t+1}.

Note that, if j started processing in I_s and was being processed in I_s for e_j time units, after the shifting it will be processed in I_{s+1} for at most e_j time units. Since I_{s+1} has $\varepsilon|I_s| = \varepsilon^2(1+\varepsilon)^s$ more time units than I_s, at least that much idle time will be created in I_{s+1}. Also, by moving jobs inside each interval, we can assume that this idle time is consecutive in each I_s.

Before giving a general description of the algorithm, we first state two basic lemmas concerning the structure of optimal solutions. The first lemma can also be found in Chen and Hall [8]. We give a sketch of the proof for completeness. The second lemma shows that there exists a $(1+\varepsilon)$-approximate schedule where no order crosses more than $O(\log(1/\varepsilon)) = O(1)$ intervals.

LEMMA 6.2. *For any instance of* $P|\text{part}|\sum w_L C_L$, *there exists an optimal schedule such that*
(i) *for any order* $L \in O$ *and for any machine* $i \in M$, *all jobs in* L *assigned to* i *are processed consecutively*;
(ii) *the sequence in which the orders are arranged on each machine is independent of the machine.*

PROOF. Consider an arbitrary optimal schedule with completion times C_L^*. Modify this schedule as follows: On each machine, sequence the jobs in order of nondecreasing completion times C_L^* of their respective order L (break ties consistently). The resequencing does not increase the completion times of orders and the resulting schedule is thus optimal. □

LEMMA 6.3. *There exists an* $(1+\varepsilon)$*-approximate schedule on which every order is fully processed in at most* $s+1$ *consecutive intervals, where* $s := \lceil \log(1+1/\varepsilon) \rceil$.

PROOF. Let us consider an optimal schedule as in Lemma 6.2 and apply STRETCH COMPLETION TIMES. Then we move all jobs to the right as much as possible without further increasing the completion time of any order. Note that for any order L, each job $j \in L$ increased its completion time by at least εC_L. Indeed, if this is not the case, let L be the last order (in terms of completion time) for which there exists a $j \in L$ that increased its completion time by less than εC_L. Let i be the machine processing j. Lemma 6.2 implies that all jobs processed on i after job j belong to orders that finish later than C_L, and thus they increase their completion time by at least εC_L. As the completion time of order L was also increased by εC_L, we conclude that job j could be moved to the right by εC_L contradicting the assumption.

This implies that after moving jobs to the right, the starting point of an order L, S_L, will be at least εC_L, and therefore $C_L - S_L < C_L \le S_L/\varepsilon$. Let I_x and I_y be the intervals where L starts and finishes respectively; then $(1+\varepsilon)^y - (1+\varepsilon)^{x+1} \le C_L - S_L < S_L/\varepsilon \le (1/\varepsilon)(1+\varepsilon)^{x+1}$, which implies that $y - x - 1 < \log(1+1/\varepsilon)$, and thus $y - x \le s$. □

6.3. Algorithm overview. In the following we describe the general idea of the PTAS. Let us divide the time horizon into blocks of $s+1 = \lceil \log(1+1/\varepsilon) \rceil + 1$ intervals, and denote as B_ℓ the block $[(1+\varepsilon)^{\ell(s+1)}, (1+\varepsilon)^{(\ell+1)(s+1)})$. Lemma 6.3 suggests optimizing over each block separately, and later joining these partial schedules to construct a global optimum.

Because there may be orders that cross from one block to the next, it will be necessary to perturb the "shape" of blocks. For that we introduce the concept of a *frontier*. The outgoing frontier of block B_ℓ is a vector that has m entries. Its ith coordinate is a guess on the completion time of the last job scheduled on machine i among jobs that belong to orders that began processing in B_ℓ (see Figure 4). On the other hand, the incoming frontier of a block is the outgoing frontier of the previous block. For a given block B_ℓ and its incoming and outgoing frontiers, we will say that an order is scheduled in block B_ℓ if on each machine all jobs in that order begin processing after the incoming frontier, and finish processing before the outgoing frontier.

Assume that we know how to compute a near-optimal solution for a given subset of orders $V \subseteq O$ inside a block B_ℓ, with fixed incoming and outgoing frontiers F' and F, respectively. Let $W(\ell, F', F, V)$ be the cost (sum of weighted completion times) of this solution.

Let $\mathcal{F}_\ell$ be the set of possible outgoing frontiers of block B_ℓ. Using dynamic programming, we can fill a table $T(\ell, F, U)$ containing the cost of a near-optimal schedule for the subset of orders $U \subseteq O$ in block B_ℓ or before, respecting the outgoing frontier F of B_ℓ. To compute this quantity, we can use the recursive formula:

$$T(\ell+1, F, U) = \min_{F' \in \mathcal{F}_\ell,\, V \subseteq U} \{T(\ell, F', V) + W(\ell+1, F', F, U \setminus V)\}.$$

Unfortunately, table T is not of polynomial size, or even finite. Then, it will be necessary to reduce its size as done in Afrati et al. [1]. Summarizing, the outline of the algorithm, which we simply call ALGORITHM PTAS, is the following.

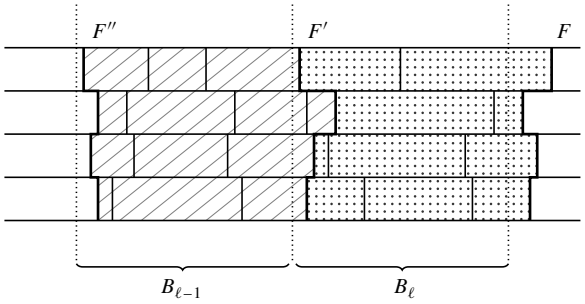

FIGURE 4. Incoming and outgoing frontiers of blocks.
Note. Hatched jobs belong to orders scheduled in $B_{\ell-1}$, and dotted jobs belong to orders scheduled in B_ℓ.

Outline of ALGORITHM PTAS

(i) *Localization*: In this step we bound the time-span of the intervals in which each order may be processed. We give extra structure to the instance and define a release date r_L for each order L, such that there exists a near-optimal solution where each order begins processing after r_L and ends processing no later than a constant number of intervals after r_L. More precisely, we prove that each order L is scheduled in the interval $[r_L, r_L \cdot (1+\varepsilon)^{g(\varepsilon,K)}]$, for some function g. This plays a crucial role in the next step.

(ii) *Polynomial representation of subsets of orders*: The goal of this step is to reduce the number of subsets of orders needed to consider in the dynamic program. To do this, for all ℓ, we find a polynomial size collection $\Theta_\ell \subseteq 2^O$ (where O is the set of orders) of possible subsets of orders that are processed in B_ℓ or before in some near-optimal schedule.

(iii) *Polynomial representation of frontiers*: In this step we reduce the number of frontiers we need to try in the dynamic program. For all ℓ, we find $\hat{\mathcal{F}}_\ell$, a set of polynomial size that describes all relevant frontiers in a compact manner.

(iv) *Dynamic programming*: For all ℓ, frontiers $F \in \hat{\mathcal{F}}_{\ell+1}$, and subset of orders $U \in \Theta_\ell$, compute

$$T(\ell, F, U) = \min_{F' \in \hat{\mathcal{F}}_\ell,\, V \subseteq U,\, V \in \Theta_{\ell-1}} \left\{ T(\ell-1, F', V) + W(\ell, F', F, U \backslash V) \right\}.$$

It is clear that it is not necessary to compute exactly $W(\ell, F', F, U\backslash V)$; it suffices to find a $(1+\varepsilon)$-approximation of this value, that moves the frontiers by at most a $1+\varepsilon$ factor. To compute this approximation, we partition orders into small and large. For large orders we use enumeration and essentially try all possible schedules, while for small orders we greedily schedule them using Smith's rule.

One of the main difficulties of this approach is that all the modifications applied to the optimal solution must conserve the properties given by Lemma 6.2. This is necessary to describe the interaction between one block and the following by using only the frontier. In other words, if this is not true, it could happen that some jobs of an order that begins processing in a block B_ℓ are processed after a job of an order that begins processing in block $B_{\ell+1}$. This would greatly increase the complexity of the algorithm, since we would need to consider this interaction in the dynamic program, which would become too large. This is also the main reason why our result does not directly generalize to the case when we have release dates, because then, Lemma 6.2 does not hold. In the sequel we will analyze each of the previous steps separately.

6.4. Localization. Lemma 6.3 shows that we can restrict to schedules where each order is completely being processed within at most a constant number s of consecutive intervals. However, we do not know a priori when each order is scheduled. In what follows, we refine this result by explicitly finding a constant number of consecutive intervals within which each order is being processed by a near-optimal schedule. This property will be helpful for guessing the specific block on which each order is being processed, and thus it allows us to reduce the number of subsets of orders we need to try in the dynamic programming.

The localization will be done by introducing artificial release dates; i.e., for each order L we will give a point in time r_L such that, by decreasing the objective function by at most a factor of $1+\varepsilon$, order L starts processing after r_L. Naturally, it is enough to consider release dates that are powers of $1+\varepsilon$. The release dates are chosen so that the following crucial property is satisfied.

PROPERTY 6.1. *For any $t \in \mathbb{Z}$, the total processing time of orders released at $(1+\varepsilon)^t$ is $O(m(1+\varepsilon)^t)$.*

We first define release dates satisfying this property. At the end of this section we show that this is enough to conclude that each order L is completely processed before $r_L \cdot (1+\varepsilon)^g$, for some constant g.

To define the release dates r_L, we do as follows. Begin by defining $r_j := (1+\varepsilon)^{\lfloor \log_{1+\varepsilon}(\varepsilon p_j)\rfloor}$ for every job j. It is easy to see that these are valid release dates, since applying Stretch Completion Times ensures that no job starts processing before εp_j. Then, for every order $L \in O$, we initialize a release date $r_L := \max_{j\in L} r_j(1+\varepsilon)^{-s}$. This follows since for every order L at least one of its jobs begins processing after $\max_{j\in J} r_j$, and Lemma 6.3 assures that every order is processed in a time span of at most $s+1$ consecutive intervals.

Clearly, this initial definition of the release dates may not be enough to assure Property 6.1. To amend this, we delay the release dates of orders that are not able to start before the next integer power of $1+\varepsilon$. We first classify orders by the size of their jobs: we say that two orders are of the same *type* if, for any $p \in \mathbb{Z}$, the number of jobs of size p is the same in both orders. It is not hard to see that among two orders of the same type, jobs of the order with larger weight will always have priority over the jobs of the other one. This is the key argument to justify the delaying of release dates.

Intuitively, if there are too many orders of the same type released at $(1+\varepsilon)^t$, only the ones with larger weight will be able to start processing before $(1+\varepsilon)^{t+1}$, and thus we can increase the release date of the remaining orders to $(1+\varepsilon)^{t+1}$. This will allow us to prove that for a given type of order the total processing time of jobs released at $(1+\varepsilon)^t$ is $O(m(1+\varepsilon)^t)$. We can then conclude Property 6.1 if we are able to show that there are only a constant number of types of orders released at $(1+\varepsilon)^t$. Although this is not true in general, we can work around it by treating orders that are too small separately. For this, consider the following definition.

Definition 6.1. (i) A job $j \in L$ is *small* if $p_j \le \varepsilon^3 r_L$. Otherwise, we say that j is *big*.

(ii) An order L is *small* if $p(L) \le \varepsilon^2 r_L$. Otherwise, we say that L is *big*.

Observe that for the initial definition of the release dates, all orders are big with respect to their release dates.

We reduce the number of types of big orders in two steps. First, we show that we can group small jobs into larger ones, and thus assure that big orders contain no small job. Then we round the processing times to powers of $1+\varepsilon$, thus bounding the number of values a processing time can take. The grouping of jobs of an order $L \in O$ and the rounding is done as follows:

(i) Sort jobs in L by nonincreasing size. Then greedily assign jobs to groups until the total processing time of each group just surpasses $\varepsilon^2 r_L$. After this process, there may be at most one group of size smaller than $\varepsilon^2 r_L$.

(ii) If the smallest group has total processing time at most $\varepsilon^3 r_L$, add it to the group with largest total processing time. Otherwise, leave it untouched. (Note that this is possible since there always is a group—consisting of only one job—whose size is larger than $\varepsilon^2 r_L$.)

(iii) Redefine the jobs of L as the newly created groups, and round the processing times to $p_j := (1+\varepsilon)^{\lceil \log_{1+\varepsilon} p_j\rceil}$.

It is important to remark that after the grouping is done, all jobs are big. Moreover, we obtain the following important property.

Property 6.2. *For any order L, we have*

$$\frac{\max_{j\in L} p_j}{\min_{j\in L} p_j} \in O(1).$$

Proof. Note that the grouping procedure does not touch the largest job of L. Then, by the definition of the release dates,

$$\max_{j\in L} p_j \le \frac{(1+\varepsilon)^{s+1}}{\varepsilon} r_L.$$

The result follows since the grouping procedure guarantees that $p_j \ge \varepsilon^3 r_L$ for all $j \in L$. □

The correctness of the grouping procedure is justified by the following lemma.

Lemma 6.4. *There exists a $(1+O(\varepsilon))$-approximate schedule where all jobs in a group are being processed consecutively on the same machine.*

Proof (Sketch). Consider a $(1+O(\varepsilon))$-approximate schedule of the instance before grouping the jobs. Fix a group and consider the machines and intervals on which the jobs that belong to it are being processed. Interpreting a machine-interval pair as a virtual machine, the group can be seen as a virtual job that is fractionally assigned to the virtual machines containing its jobs. Now we can apply Theorem 2.1 to round this fractional solution so that each virtual job is processed completely inside one virtual machine. The rounding guarantees that the total processing time assigned to each virtual machine is increased by at most an additive factor, $2\varepsilon^2 r_L$.

By applying Stretch Intervals twice, we create the extra idle time needed. Note that the completion time of orders is not increased by more than a $1+O(\varepsilon)$ factor.

If the size of each group is larger than $\varepsilon^3 r_L$, we are done. Otherwise, we get rid of the single small group by merging it with the largest group in L. This can be done since there always is a group of size larger than $\varepsilon^2 r_L$, and thus we can fit the small group before it by using Stretch Completion Times again. □

As discussed before, we now delay all orders of a given type that are not able to start at their current release dates. Starting from left to right, consider the set of orders released at time $(1+\varepsilon)^t$ of type α, and let p_α be the smallest processing time of a job in such an order. Then, it is clear that at most $m(|I_t|/p_\alpha+1)$ of these orders can start processing before time $(1+\varepsilon)^{t+1}$. We thus leave untouched the $m(|I_t|/p_\alpha+1)$ orders with largest weight, and update r_L to $(1+\varepsilon)^{t+1}$ for the rest of the orders. After doing this, if P_α denotes the largest processing time of a job in an order of type α, the total processing time of orders of type α released at time $(1+\varepsilon)^t$ is upperbounded by

$$KP_\alpha m\left(\frac{|I_t|}{p_\alpha}+1\right)=\frac{P_\alpha}{p_\alpha}(1+\varepsilon)^t O(m)=O(m(1+\varepsilon)^t), \tag{28}$$

where the last equality follows from Property 6.2.

With the latter observations it is easy to prove the next theorem.

Theorem 6.3. *After delaying orders, there is at most $O(m(1+\varepsilon)^t)$ total processing time corresponding to big orders released at $(1+\varepsilon)^t$.*

Proof (Sketch). Thanks to (28), it is enough to show that the number of types of big orders released at $(1+\varepsilon)^t$ is constant. To prove this, consider a big order of type α released at $(1+\varepsilon)^t$, and note that P_α can only take a constant number of values. Indeed, since L is big, $P_\alpha \geq p(L)/K \geq (1+\varepsilon)^t\varepsilon^2/K$. Also, if r_L is the initial release date of L, it follows from the definition of r_L that $P_\alpha \leq r_L(1+\varepsilon)^{-s-1}/\varepsilon \leq (1+\varepsilon)^t(1+\varepsilon)^{-s-1}/\varepsilon$. The two latter observations plus the fact that P_α can only take values of the form $(1+\varepsilon)^k$ imply that P_α takes at most a constant number of different values. Furthermore, by Property 6.2, we can conclude that the amount of different processing times any job in L can have is constant. Therefore, there are at most $K^{O(1)}$ different types of big orders that can be released at time $(1+\varepsilon)^t$. □

With this we have completely dealt with big orders, but the delaying procedure created several small orders. To cope with them, we first notice that by using the grouping procedure once more, each order becomes a single larger job. With this we can show that processing these jobs greedily by nondecreasing order of $w_L/p(L)$ (Smith's rule) yields a near-optimal solution. Following a similar procedure as explained before, we can delay orders released at $(1+\varepsilon)^t$ that cannot be processed within I_t. We skip the details since they are analogous to the work of Afrati et al. [1].

Having defined release dates that satisfy Property 6.1, we can thus conclude the main objective of this section.

Theorem 6.4. *Consider release dates r_L satisfying Property 6.1. Then there exists a near-optimal schedule where every order $L \in O$ is processed between r_L and $r_L(1+\varepsilon)^g$, for some constant $g=g(\varepsilon,K)$.*

(Proof Sketch). Let g be a constant that will be chosen later. For a fixed t, consider all orders released at $(1+\varepsilon)^t$ that are scheduled completely after time $(1+\varepsilon)^{t+g-s}$. We will move these orders to the left, so that they are completely processed inside interval I_{t+g-s}. The rest of the orders can be left untouched since they are processed in at most $s+1$ consecutive intervals, and then they are completely processed before $(1+\varepsilon)^{t+g}$. Since the amount of processing time released at time $(1+\varepsilon)^t$ is $O(m(1+\varepsilon)^t)$, by choosing g large enough, we can make this quantity smaller than the idle time created in I_{t+g-s} by Stretch Intervals, that is, $\varepsilon^2(1+\varepsilon)^{t+g-s}m$. Furthermore, for any large enough g, every order released at time $(1+\varepsilon)^t$ is small with respect to $(1+\varepsilon)^{t+g-s}$. It is thus possible to accommodate all necessary orders released at time $(1+\varepsilon)^t$ in interval I_{t+g-s} greedily.

Importantly, the structure of the near-optimal solution given in Lemma 6.2 is preserved because all jobs of orders that are moved can be processed consecutively on the same machine. □

6.5. Polynomial representation of subsets and frontiers. We now briefly explain how to reduce the number of possible subsets processed in some block B_ℓ or before. Equivalently, we can decide which are the orders that are going to be processed after $B_{\ell+1}$ among those released before time $(1+\varepsilon)^{(s+1)(\ell+1)}$. Moreover, by Theorem 6.4, each order is completed within at most $g(\varepsilon,K)$ intervals after its release date. We then only consider orders released between $(1+\varepsilon)^{(s+1)(\ell+1)-g(\varepsilon,K)}$ and $(1+\varepsilon)^{(s+1)(\ell+1)}$. Note that these are only a constant number of intervals, and thus, for a given $t \in \{(s+1)(\ell+1)-g(\varepsilon,K),\ldots,(s+1)(\ell+1)\}$, it is enough to

guess which orders are processed after B_ℓ among the ones released at $(1+\varepsilon)^t$. We can then combine these guesses among all $t \in \{(s+1)(\ell+1) - g(\varepsilon, K), \ldots, (s+1)(\ell+1)\}$ without loosing polynomiality.

Among all orders of a particular type, the k orders with smallest weight will be processed after $B_{\ell+1}$, for some $k \in \{0, \ldots, n\}$. In other words, for a particular type α, there are only n different subsets of orders of that type that we need to try. Analogously, small orders can be processed by Smith's rule, and thus we only have to check the sets of k orders with smallest $w_L/p(L)$, for some $k \in \{0, \ldots, n\}$. Thus, among all orders released at time $(1+\varepsilon)^t$, it suffices to consider sets of orders of the form $A_s \cup_\alpha A_\alpha$, where A_s is a subset of small orders and A_α is a set of orders of type α. Since for A_s and for each A_α we have n different choices, and in the last section we showed that there are only a constant number of types of big orders released at time $(1+\varepsilon)^t$ (see proof of Theorem 6.3), we conclude that we have to try at most a polynomial number $n^{O(1)}$ of subsets of orders.

To deal with frontiers, we use standard rounding techniques. Consider a block B_ℓ, and note that by using Stretch Completion Times we can restrict to outgoing frontiers whose entries take only a constant number of different values. We can thus easily describe a frontier as follows: Specify, for each possible value a frontier can take, the number of machines having that frontier. In particular, the number of outgoing frontiers that we have to consider is $m^{O(1)}$. We omit further details.

6.6. A PTAS for a specific block. To conclude our algorithm we need to give the subroutine of Algorithm PTAS that computes the entry $W(\ell, F', F, U)$. In the other words, for a given block B_ℓ, incoming and outgoing frontiers F' and F, and subset of orders U, we need to find a $(1+\varepsilon)$-approximate schedule of jobs in U inside B_ℓ. Note that it is possible to move the frontiers by factors of at most $1+\varepsilon$ without increasing the cost of a global solution by more than a $1+\varepsilon$ factor.

In the sequel, we consider orders and jobs as big or small with respect to the beginning of block B_ℓ. In other words, a job is small if its processing time is smaller than $\varepsilon^3(1+\varepsilon)^{(s+1)\ell}$, and big otherwise. Additionally, an order is small if its total processing time is smaller than $\varepsilon^2(1+\varepsilon)^{(s+1)\ell}$, and big otherwise. Following the ideas of the previous sections, we enumerate over schedules of big orders, and apply Smith's rule to greedily assign small orders.

For the enumeration we first need to find a suitable compact description of schedules of big orders. For this, the following definition will be of use: a *type of job* j is a pair (p, α), where p denotes the processing time of job j and α denotes the type of order that j belongs to. Importantly, among jobs belonging to a big order, we can reduce the number of different types of jobs to a constant. Indeed, applying the grouping technique of §6.4, we obtain that p will be larger than $\varepsilon^3(1+\varepsilon)^{(s+1)\ell}$. Also, since j must be processed in B_ℓ, $p \le (1+\varepsilon)^{(\ell+2)(s+1)} = O((1+\varepsilon)^{\ell(s+1)})$. Thus, if (p, α) is the type of job j belonging to a big order L, we conclude that p can only take a constant number of different values, and thus also α. Let $\mathcal{T}$ be the set of all such types of jobs.

It is not hard to see, by Stretch Completion Times, that we can restrict ourselves to schedules in which big jobs only start processing at times belonging to a set

$$\Omega_\ell := \left\{(1+\varepsilon)^{(s+1)\ell} + k\delta \colon k = 0, \ldots, \omega\right\},$$

where $\delta = \varepsilon^4(1+\varepsilon)^{\ell(s+1)}$, and ω is a suitably large—but still constant—integer.

We define a *single machine configuration* as a vector S with ω entries. For $k = 1, \ldots, \omega$, the kth entry of S denotes the type of job that is processed at $(1+\varepsilon)^{(s+1)\ell} + k\delta$. We consider only types of jobs in $\mathcal{T}$, and thus the number of different single machine configurations is bounded by $|\mathcal{T}|^\omega = O(1)$. Furthermore, we define a *parallel machine configuration* as a vector M that tells, for each single machine configuration S, how many machines follow S. It is thus clear that the number of different vectors M is at most $m^{O(1)}$, and thus we can enumerate them in polynomial time. Additionally, we will only consider vectors M that correspond to schedules that can be processed between the frontiers F and F'.

For any given parallel machine configuration M, it is thus enough to find the schedule of lowest cost following this configuration. This can be done by a greedy approach using M as a template schedule. Indeed, starting from left to right, we greedily assign jobs to machines at times $(1+\varepsilon)^{(s+1)\ell} + k\delta$, following M. We break ties among jobs of the same type by giving priority to jobs belonging to orders with larger weight. It is then clear that this yields the optimal schedule of large orders that follow M. For finding the schedule for small orders, we can simply apply Smith's rule. We thus conclude the main result of this section.

Theorem 6.5. Algorithm PTAS *is a polynomial-time approximation scheme for the restricted version of* $P|\text{part}|\sum w_L C_L$ *when the number of jobs per order is bounded by a constant* K.

Note that, since $n > m$ for any nontrivial instance, a straightforward calculation shows that the running time of this algorithm is given by

$$n^{K^{O(\log(K/\varepsilon))}/\varepsilon^2} m^{K^{O(\log(1/\varepsilon)/\varepsilon^6)}} = n^{K^{O(\log(K/\varepsilon))}} m^{K^{O(\log(1/\varepsilon))}} = n^{K^{O(\log(K/\varepsilon)/\varepsilon^6)}},$$

which is polynomial for fixed K and ε.

6.7. Variations. In the last section we showed a PTAS for minimizing the sum of weighted completion times of orders on parallel machines, when the number of jobs per order is constant. Now we show how to bypass the last assumption by assuming that the number of machines m is a constant independent of the input. Indeed, we will show that a slight modification of the algorithm gives a PTAS for this case.

THEOREM 6.6. *There exists a PTAS for* $Pm|\text{part}|\sum w_L C_L$.

PROOF. Consider the same algorithm, with the only modification that we apply the grouping technique of §6.4 one more time after the delaying of release dates. Then, every small order consists of only one job, and every big order only contains jobs larger than $\varepsilon^3 r_L$. Since every order finishes within s intervals, it cannot contain more than

$$\frac{m r_L (1+\varepsilon)^s}{\varepsilon^3 r_L} = O(m) = O(1)$$

jobs. Thus, in the modified instance every order contains a constant number of jobs. □

Finally, we consider the case when the number of orders is constant. Note that this variant contains the minimum makespan problem as a special subcase. Having a constant number of orders makes the problem considerably simpler for two reasons. First, the number of possible subsets of orders is also constant, and therefore the localization and the polynomial representation of subsets of orders are not needed: simply define Θ_ℓ as the power set of O. Moreover, the number of possible types of orders is also constant, and therefore the PTAS for each block takes polynomial time. Let us call this modified version ALGORITHM PTAS II.

THEOREM 6.7. ALGORITHM PTAS II *is a PTAS for the restricted version of* $P|\text{part}|\sum w_L C_L$, *with a constant number of orders.*

A simple, though careful, calculation shows that the running time of ALGORITHM PTAS II is $n \cdot m^{C^{O(1/\varepsilon^6)}}$, where C is the number of orders.

Acknowledgments. The authors thank Nikhil Bansal for stimulating discussions on the material in §3 and David Shmoys for bringing Lin and Vitter [27] to their attention. They also thank the anonymous referees for helpful comments and suggestions on the presentation of this paper. A preliminary version of this paper appeared in Correa et al. [12]. This work was partially supported by Berlin Mathematical School, by DFG (Deutsche Forschungsgemeinschaft) Research Center MATHEON *Mathematics for Key Technologies* in Berlin, and by CONICYT (Comisión Nacional de Investigación Científica y Tecnológica) through Grant FONDECYT 1090050.

References

[1] Afrati, F., E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, M. Sviridenko. 1999. Approximation schemes for minimizing average weighted completion time with release dates. *Proc. 40th Annual IEEE Sympos. Foundations Comput. Sci. (FOCS)*, IEEE Computer Society, Washington, DC, 32–43.
[2] Ambühl, C., M. Mastrolilli. 2009. Single machine precedence constrained scheduling is a vertex cover problem. *Algorithmica* **53** 488–503.
[3] Ambühl, C., M. Mastrolilli, O. Svensson. 2011. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM J. Comput.* **40** 567–596.
[4] Bansal, N., S. Khot. 2009. Optimal long code test with one free bit. *Proc. 50th Annual IEEE Sympos. Foundations Comput. Sci. (FOCS)*, IEEE Computer Society, Washington, DC, 453–462.
[5] Bansal, N., S. Khot. 2010. Inapproximability of hypergraph vertex cover and applications to scheduling problems. *Proc. 37th Internat. Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, Vol. 6198. Springer-Verlag, Berlin, 250–261.
[6] Canetti, R., S. Irani. 1998. Bounding the power of preemption in randomized scheduling. *SIAM J. Comput.* **27** 993–1015.
[7] Chekuri, C., R. Motwani. 1999. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Appl. Math.* **98** 29–38.
[8] Chen, Z., N. G. Hall. 2001. Supply chain scheduling: Assembly systems. Technical report, Ohio State University, Columbus.
[9] Chen, Z., N. G. Hall. 2007. Supply chain scheduling: Conflict and cooperation in assembly systems. *Oper. Res.* **55** 1072–1089.

[10] Chudak, F., D. S. Hochbaum. 1999. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Oper. Res. Lett.* **25** 199–204.
[11] Correa, J. R., A. S. Schulz. 2005. Single machine scheduling with precedence constraints. *Math. Oper. Res.* **30** 1005–1021.
[12] Correa, J. R., M. Skutella, J. Verschae. 2009. *Proc. 12th Internat. Workshop and 13th Internat. Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX-RANDOM)*, Lecture Notes in Computer Science, Vol. 5687. Springer-Verlag, Berlin, 84–97.
[13] Dyer, M. E., L. A. Wolsey. 1999. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Appl. Math.* **26** 255–270.
[14] Eastman, W. L., S. Even, I. M. Isaacs. 1964. Bounds for the optimal scheduling of n jobs on m processors. *Management Sci.* **11** 268–279.
[15] Graham, R. L. 1966. Bounds for certain multiprocessing anomalies. *AT&T Tech. J.* **45** 1563–1581.
[16] Hall, L. A., A. S. Schulz, D. B. Shmoys, J. Wein. 1997. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.* **22** 513–544.
[17] Hochbaum, D., D. B. Shmoys. 1987. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. ACM* **34** 144–162.
[18] Hoogeveen, H., P. Schuurman, G. J. Woeginger. 2001. Nonapproximability results for scheduling problems with minsum criteria. *INFORMS J. Comput.* **13** 157–168.
[19] Khot, S. 2002. On the power of unique 2-prover 1-round games. *Proc. 34th Annual ACM Sympos. Theory Comput. (STOC)*, ACM, New York, 767–775.
[20] Lawler, E. L., J. Labetoulle. 1978. On preemptive scheduling of unrelated parallel processors by linear programming. *J. ACM* **25** 612–619.
[21] Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys. 1993. Sequencing and scheduling: Algorithms and complexity. S. C. Graves, A. H. G. Rinnooy Kan, P. H. Zipkin, eds. *Handbooks in Operations Research and Management Science*, Logistics of Production and Inventory, Vol. 4. North-Holland, Amsterdam, 445–522.
[22] Lenstra, J. K., A. H. G. Rinnooy Kan. 1978. Complexity of scheduling under precedence constraints. *Oper. Res.* **26** 22–35.
[23] Lenstra, J. K., D. B. Shmoys, É. Tardos. 1990. Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming* **46** 259–271.
[24] Leung, J., H. Li, M. Pinedo. 2006. Approximation algorithm for minimizing total weighted completion time of orders on identical parallel machines. *Naval. Res. Logist.* **53** 243–260.
[25] Leung, J., H. Li, M. Pinedo. 2007. Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Appl. Math.* **155** 945–970.
[26] Leung, J., H. Li, M. Pinedo, J. Zhang. 2007. Minimizing total weighted completion time when scheduling orders in a flexible environment with uniform machines. *Inform. Processing Lett.* **103** 119–129.
[27] Lin, J.-H., J. S. Vitter. 1992. ε-approximations with minimum packing constraint violation. *Proc. 24th Annual ACM Sympos. Theory of Comput. (STOC)*, ACM, New York, 771–782.
[28] Margot, F., M. Queyranne, Y. Wang. 2003. Decompositions, network flows, and a precedence constrained single machine scheduling problem. *Oper. Res.* **51** 981–992.
[29] Mastrolilli, M., M. Queyranne, A. S. Schulz, O. Svensson, N. A. Uhan. 2010. Minimizing the weighted sum of completion times in concurrent open shops. *Oper. Res. Lett.* **38** 390–395.
[30] Queyranne, M. 1993. Structure of a simple scheduling polyhedron. *Math. Programming* **58** 263–285.
[31] Schulz, A. S., M. Skutella. 2002. Scheduling unrelated machines by randomized rounding. *SIAM J. Discrete Math.* **15** 450–469.
[32] Shachnai, H., T. Tamir. 2002. Multiprocessor scheduling with machine allotment and parallelism constraints. *Algorithmica* **32** 651–678.
[33] Shmoys, D. B., É. Tardos. 1993. An approximation algorithm for the generalized assignment problem. *Math. Programming* **62** 461–474.
[34] Skutella, M. 2001. Convex quadratic and semidefinite programming relaxations in scheduling. *J. ACM* **48** 206–242.
[35] Skutella, M., G. J. Woeginger. 2000. Minimizing the total weighted completion time on identical parallel machines. *Math. Oper. Res.* **25** 63–75.
[36] Smith, W. E. 1956. Various optimizers for single-stage production. *Naval Res. Logist. Quart.* **3** 59–66.
[37] Verschae, J. 2008. Approximation algorithms for scheduling orders on parallel machines. Mathematical engineering thesis, Universidad de Chile, Santiago, Chile.
[38] Woeginger, G. J. 2003. On the approximability of average completion time scheduling under precedence constraints. *Discrete Appl. Math.* **131** 237–252.
[39] Yang, J., M. E. Posner. 2005. Scheduling parallel machines for the customer order problem. *J. Sched.* **8** 49–74.