Machine scheduling with resource dependent processing times

Mathematical Programming manuscript No.
(will be inserted by the editor)

Alexander Grigoriev · Maxim Sviridenko · Marc Uetz

Machine scheduling with resource dependent processing times

Received: date / Revised version: date

Abstract We consider several parallel machine scheduling settings with the objective to minimize the schedule makespan. The most general of these settings is unrelated parallel machine scheduling. We assume that, in addition to its machine dependence, the processing time of any job is dependent on the usage of a scarce renewable resource. A given amount of that resource, e.g. workers, can be distributed over the jobs in process at any time, and the more of that resource is allocated to a job, the smaller is its processing time. This model generalizes classical machine scheduling problems, adding a time-resource tradeoff. It is also a natural variant of a generalized assignment problem studied previously by Shmoys and Tardos. On the basis of integer programming formulations for relaxations of the respective problems, we use LP rounding techniques to allocate resources to jobs, and to assign jobs to machines. Combined with Graham's list scheduling, we thus prove the existence of constant factor approximation algorithms. Our performance guarantee is $(4 + 2\sqrt{2}) \approx 6.83$ for the most general case of unrelated parallel machine scheduling. We improve this bound for two special cases, namely to $(3 + 2\sqrt{2}) \approx 5.83$ whenever the jobs are assigned to machines beforehand, and to $(5 + \varepsilon)$ whenever the processing times do not depend on the machine. Moreover, we discuss tightness of the relaxations, and derive inapproximability results.

1. Introduction

Unrelated parallel machine scheduling to minimize the makespan, $R||C_{\max}$ in the three-field notation of Graham et al. [7], is one of the classical problems in combinatorial optimization. Given are n jobs that have to be scheduled on m parallel machines, and the processing time of job j on machine i is p_{ij}. The goal is to minimize the latest job completion, the makespan $C_{\max}$. If the number of machines m is not fixed, the best approximation algorithm to date is a 2-approximation by Lenstra, Shmoys and Tardos [15]. Moreover, the problem cannot be approximated within a factor smaller than 3/2, unless P=NP [15]. When the processing times p_{ij} are identical on all machines i, the problem is called identical parallel machine scheduling, or $P||C_{\max}$. It is strongly NP-hard [3] and admits a polynomial time approximation scheme [10].

Shmoys and Tardos [18] consider the unrelated parallel machine scheduling problem with the additional feature of costs λ_{ij} if job j is processed on machine i. They show that, if a schedule with total cost Λ and makespan T exists, a schedule

Alexander Grigoriev, Marc Uetz: Maastricht University, Quantitative Economics, P.O.Box 616, 6200 MD Maastricht, The Netherlands. E-mail: {a.grigoriev,m.uetz}@ke.unimaas.nl

Maxim Sviridenko: IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA. E-mail: sviri@us.ibm.com

Mathematics Subject Classification (1991): 90B35, 68Q25, 68M20

with total cost Λ and makespan at most $2T$ can be found in polynomial time. The proof relies on rounding the solution of an LP relaxation. They obtain the same result even for a more general version of the problem, namely when the processing time p_{ij} of any job-machine pair is not fixed, but may be reduced linearly, in turn for a linear increase of the associated cost λ_{ij} [18]. Note that, in both versions of the problem studied in [18], the costs λ_{ij} are *nonrenewable* resources, such as a monetary budget, with a global budget Λ.

In this paper, we consider three machine scheduling settings, the most general being an unrelated parallel machine scheduling problem. This unrelated parallel machine scheduling problem is in fact a different variant of the problem considered by Shmoys and Tardos in [18]. Namely, we assume that the processing times p_{ij} of any job j, if processed on machine i, can be reduced by utilizing a *renewable* resource, such as additional workers, that can be allocated to the job. More precisely, a maximum number of k units of a resource is available at any time. It may be used to speed up the jobs, and the available amount of k units of that resource must not be exceeded at any time. In contrast to the linearity assumption on the costs and processing times in [18], the only assumption we make in this paper is that the processing times p_{ijs}, which now depend also on the number s of allocated resources, are non-increasing in s for each job-machine pair. That is, we assume that $p_{ij0} \geq p_{ij1} \geq \cdots \geq p_{ijk}$ for all jobs j and all machines i. The two other settings that we consider are special cases: We consider the problem where jobs are assigned to machines beforehand, sometimes also called *dedicated* parallel machine scheduling [13,14], and the problem where the processing time of any job is independent on the machine that processes the job, the *identical* parallel machine scheduling problem.

The practical motivation to study these *time-resource* tradeoff problems is evident; to give an example, one may think of production planning where additional (overtime) workers can be allocated to specific tasks within the production in order to reduce the production cycle time. As a matter of fact, machine scheduling problems with the additional feature of a *nonrenewable* resource constraint, such as a total budget, have received quite some attention in the literature as *time-cost tradeoff* problems. To give a few references, see [2,11,12,18,19]. Surprisingly, the corresponding time-resource tradeoff problems, that is, problems with a *renewable* resource constraint such as personnel, have received much less attention, although they are not less appealing from a practical viewpoint.

2. Results and related work

Related work. In a manuscript by Grigoriev et al. [8], a restricted version of the problem at hand is addressed. They consider the setting where jobs are assigned to machines beforehand, the *dedicated* parallel machine setting. Furthermore, their model assumes a *binary* resource, that is, there is just one unit of a renewable resource that can be used to speed up the jobs, and at any time at most one job can make use of it. Any job has a reduced processing time if the resource is used. Finally, the number of machines m in their paper is considered fixed, and

not part of the input. For that problem, they derive a $(3+\varepsilon)$–approximation algorithm, and for the problem with $m = 2$ machines, they derive (weak) NP-hardness and a fully polynomial time approximation scheme [8]. In a recent paper, Grigoriev and Uetz [9] have generalized the approximation result of [8]. The model of [9] is a dedicated machine setting as well, and assumes a *linear* time-resource tradeoff: There are k units of a renewable resource available, and the processing time p_j of any job becomes $p_{js} = \bar{p}_j - b_j\, s$ if s of the k resources are used. Using quadratic programming relaxations, a $(3+\varepsilon)$–approximation algorithm is derived in [9], for an arbitrary number of machines.

Jobs with resource dependent processing times also appear in the literature as *malleable* or *parallelizable tasks*, e.g. in [16,21]. In these models, jobs can be processed on one or more parallel processors, and they have non-increasing processing times p_{js} in the number s of processors used. Any processor can only handle one job at a time, and the goal is to minimize the schedule makespan. Turek et al. [21] derive a 2–approximation algorithm for this problem. In fact, the model considered in [21] is just a special case of the problem considered in this paper. Interpreting the parallel processors as a generic 'resource' that must be allocated to jobs, the problem of [21] corresponds to the problem considered in this paper, when letting n jobs with resource dependent processing times be processed in parallel, but each on its *own* machine. In particular, thus, the number of machines m then equals the number of jobs n. Mounie et al. [16] consider yet another variant, in that the processor allocations must be contiguous (for that problem, [21] includes a 2.7–approximation). Moreover, in [16] it is not only assumed that the processing times p_{js} are non-increasing in s, but also the processing 'areas' $s \cdot p_{js}$ are assumed to be non-decreasing in s. For that problem, a $\sqrt{3}$–approximation is derived in [16].

When we restrict even further, and assume that the decision on the allocation of resources to jobs is fixed beforehand, we are back at (machine) scheduling under resource constraints as introduced by Blazewicz et al. [1]. More recently, such problems with dedicated machines have been discussed by Kellerer and Strusevich [13,14]. We refer to these papers for various complexity results, and note that NP-hardness of the problem with dedicated machines and a binary resource was established in [13]. More precisely, they show weak NP-hardness for the case where the number of machines is fixed, and strong NP-hardness for an arbitrary number of machines [13].

Results and methodology. We derive constant-factor approximation algorithms for three time-resource tradeoff problems, namely with unrelated, dedicated, and identical machines. Our approach is based upon integer linear programming formulations that define a relaxations of the respective problems. The main idea behind these relaxations is the utilization of an aggregate version of the resource constraints, yielding a formulation that does not require time-indexed variables.

In Section 4, we address the most general, unrelated machine setting. We use a formulation that takes as input all possible processing times p_{ijs} of jobs. We then consider the linear programming relaxation of this integer program. In a first step, the solution of this LP relaxation is rounded into a (still fractional)

solution for another linear program. We then show that this in fact defines an instance (and solution) of the linear programming relaxation used by Shmoys and Tardos [18] for the generalized assignment problem. In a second step, we thus apply their rounding procedure to obtain an approximate integral solution for the original integer programming relaxation. From this solution, we extract both the machine assignments and the resource allocations for the jobs. We then use Grahams list scheduling [5] to generate a feasible schedule. Using the LP lower bounds, we prove that this schedule is not more than a factor $(4 + 2\sqrt{2}) \approx 6.83$ away from the optimum.

For the special case of dedicated parallel machines, our approach simplifies, because the machine assignments are fixed beforehand, thus the second rounding step is not even required. For that case, in Section 5, we derive a performance bound of $(3 + 2\sqrt{2}) \approx 5.83$, essentially by using the same approach as in the unrelated machine case.

For the special case of identical parallel machines, we show in Section 6 how to improve the performance bound to $(5 + \varepsilon)$, for any $\varepsilon > 0$. This approach is based upon another integer programming relaxation, and it lends some ideas from [9]. More precisely, we use a fully polynomial time approximation scheme (FPTAS), together with a sort of approximate binary search, to solve the integer programming relaxation with sufficiently large precision. The corresponding FPTAS is a consequence of a theorem of Pruhs and Woeginger on the existence of FPTAS's for subset selection problems [17]. In order to achieve the performance bound of $(5+\varepsilon)$, we also utilize as a subroutine a 2-approximation algorithm for strip packing by Steinberg [20].

Concerning lower bounds, in Section 7 we provide an instance showing that the linear integer programming relaxations can be a factor $(2 - \varepsilon)$ away from the optimum solution, hence our analysis cannot yield anything better than a 2 approximation. This holds for all three problem settings that we consider. Concerning lower bounds on the approximability, note that the unrelated parallel machine problem with resource dependent processing times is a generalization of the classical unrelated machine scheduling problem $R|\,|C_{\max}$. Therefore it cannot be approximated better than a multiplicative factor of 3/2, unless P=NP [15]. In Section 7, we furthermore show that the same inapproximability threshold of 3/2 holds for both, the dedicated and the identical parallel machine settings with resource dependent processing times.

3. Problem definition

Let $V = \{1, \ldots, n\}$ be a set of jobs. Jobs must be processed non-preemptively on a set of m unrelated (dedicated, identical) parallel machines, and the objective is to find a schedule that minimizes the makespan $C_{\max}$, that is, the time of the last job completion. During its processing, a job j may be assigned an amount $s \in \{0, 1, \ldots, k\}$ of an additional resource, for instance additional workers, that may speed up its processing. If s resources are allocated to a job j, and the job is processed on machine i, the processing time of that job is p_{ijs}. The only

assumption on the processing times in dependence on the amount of allocated resources is monotonicity, that is, we assume that

$$p_{ij0} \geq p_{ij1} \geq \cdots \geq p_{ijk}$$

for every machine-job pair (i, j). For the special case of dedicated parallel machines, the assumption is that each job j has been assigned to a machine i beforehand, and by $V_i \subseteq V$ we denote the subset of jobs that has to be processed on machine i. For the special case of identical parallel machines, the processing times of jobs do not depend on the machine. Hence, for both special cases we can denote by p_{js} the processing time of job j, if s units of the resource are used, $s = 0, \ldots, k$, and the above monotonicity condition becomes

$$p_{j0} \geq p_{j1} \geq \cdots \geq p_{jk}\,.$$

The allocation of resources to jobs is restricted as follows. At any time, no more than the available k units of the resource may be allocated to the set of jobs in process. Moreover, the amount of resources assigned to any job must be the same along its processing. In other words, if s units of the resource are allocated to some job j, t_j and t'_j denote j's starting and completion time, respectively, only $k - s$ of the resources are available for other jobs between t_j and t'_j.

We finally introduce an additional piece of notation. Since we do not assume that the functions p_{ijs} (p_{js}), in dependence on s, are *strictly* decreasing, the only information that is effectively required is the *breakpoints* of p_{ijs} (p_{js}), that is, indices s where $p_{ijs} < p_{ij,s-1}$ ($p_{js} < p_{j,s-1}$). Hence, for the unrelated parallel machine case define the 'relevant' indices for job j on machine i as

$$S_{ij} = \{0\} \cup \{s \mid s \leq k,\ p_{ijs} < p_{ij,s-1}\} \subseteq \{0, \ldots, k\}\,.$$

For the special cases of dedicated or identical parallel machines, equivalently define the 'relevant' indices for job j as

$$S_j = \{0\} \cup \{s \mid s \leq k,\ p_{js} < p_{j,s-1}\} \subseteq \{0, \ldots, k\}\,.$$

Considering these index sets obviously suffices, since in any solution, if s units of the resource are allocated to some job j, we may as well only use s' units, where $s' \leq s$ and $s' \in S_{ij}(S_j)$, without violating feasibility.

4. Unrelated parallel machines

Integer programming relaxation. Let x_{ijs} denote binary variables, indicating that an amount of s resources is used for processing job j on machine i. Then the following integer linear program, referred to as (IP), has a feasible solution

if there is a feasible schedule of length C for the original scheduling problem.

$$\sum_{i=1}^{m} \sum_{s \in S_{ij}} x_{ijs} = 1 \ , \qquad \forall\, j \in V\,, \tag{1}$$

$$\sum_{j \in V} \sum_{s \in S_{ij}} x_{ijs}\, p_{ijs} \le C \ , \qquad \forall\, i = 1, \ldots, m\,, \tag{2}$$

$$\sum_{j \in V} \sum_{i=1}^{m} \sum_{s \in S_{ij}} x_{ijs}\, s\, p_{ijs} \le k\, C \ , \tag{3}$$

$$x_{ijs} = 0 \ , \qquad \text{if } p_{ijs} > C, \tag{4}$$

$$x_{ijs} \in \{0, 1\} \ , \qquad \forall\, i, j, s. \tag{5}$$

Here, C represents the schedule makespan. Equalities (1) make sure that every job is assigned to one machine and uses a constant amount of resources during its processing. Inequalities (2) express the fact that the total processing on each machine is a lower bound on the makespan. Inequalities (3) represent the aggregated resource constraints: In any feasible schedule, the left-hand side of (3) is the total resource consumption of the schedule. Because no more than k resources may be consumed at any time, the total resource consumption cannot exceed $k\,C$. Finally, constraints (4) make sure that we do not use machine-resource pairs such that the job processing time exceeds the schedule makespan. These constraints are obviously redundant for (IP), but they will be used later in rounding a solution for the linear relaxation of (IP). Notice that this integer program may have a feasible solution for some integer value of C, although no feasible schedule with makespan C exists; see Example 1 further below.

Linear programming relaxation. The integer linear program (IP) with the 0/1-constraints on x relaxed to

$$x_{ijs} \ge 0\,, \qquad j \in V\,,\ s \in S_{ij}\,,\ i = 1, \ldots, m$$

also has a solution for value C if there is a feasible schedule for the original scheduling problem with makespan C. We note that it can be solved in polynomial time, because it has a polynomial number of variables and constraints. Since we assume integrality of data, we are actually only interested in integral values C. Moreover, an upper bound for C is given by $\sum_{j \in V} \min_{i=1,\ldots,m}\{p_{ijk}\}$. Therefore, by using binary search on possible values for C, we can find in polynomial time the smallest integral value C^* such that the linear programming relaxation of (1)–(5) has a feasible solution x^{LP}. We therefore obtain the following.

Lemma 1. *The smallest integral value value C^* such that the linear programming relaxation of* (1)–(5) *has a feasible solution is a lower bound on on the makespan of any feasible schedule, and it can be computed in polynomial time.*

Rounding the LP solution. Given a pair (C^*, x^{LP}) we next define an integer solution x^* from x^{LP} by the following, 2-phase rounding procedure. In the first rounding phase, we transform a fractional solution x^{LP} to another fractional solution $\bar{x}$, in such a way that for every machine-job pair (i, j) there is exactly one index s (amount of resource) such that $\bar{x}_{ijs}$ is nonzero. Intuitively, we decide for every machine-job pair on the amount of resources it may consume. By doing this, we effectively get rid of the index s. This new fractional solution in fact defines a fractional solution for an LP relaxation for the generalized assignment problem discussed by Shmoys and Tardos [18]. Therefore, we will be able to use their rounding procedure as our second rounding phase, and thus we eventually obtain an integral solution x^* from x^{LP}.

First, let us choose an arbitrary ε such that $0 \leq \varepsilon \leq 1$. Then, for every machine i and job j individually, define

$$\tilde{y}_{ij} = \sum_{s \in S_{ij}} x^{\mathrm{LP}}_{ijs} \tag{6}$$

as the total fractional value allocated by the LP solution x^{LP} to the machine-job pair (i, j). Then let index $t^{ij} \in S_{ij}$ be chosen minimal with the property that

$$\sum_{s \in S_{ij}, s \leq t^{ij}} x^{\mathrm{LP}}_{ijs} \geq (1 - \varepsilon)\, \tilde{y}_{ij}\,. \tag{7}$$

Then, for every machine i and job j define index $s^{ij} \geq t^{ij}$ as the minimizer of $s \cdot p_{ijs}$, for $s \geq t^{ij}$,

$$s^{ij} = \arg\min_{s \geq t^{ij}} \; s \cdot p_{ijs}\,. \tag{8}$$

By definition, it follows that $s^{ij} \in S_{ij}$. We now consider a fractional solution $\bar{x}$ defined by

$$\bar{x}_{ijs} = \begin{cases} \tilde{y}_{ij} & s = s^{ij}\,, \\ 0 & \text{otherwise}\,. \end{cases} \tag{9}$$

By definition, this solution fulfills (1). Moreover, we claim that it is an approximate solution for inequalities (2) and (3) in the following sense.

Lemma 2. *Let C^* be the lower bound on the makespan of an optimal solution as defined in Lemma 1, and let $\bar{x} = (\bar{x}_{ijs})$ be the fractional solution obtained by the above described rounding procedure. Then*

$$\sum_{j \in V} \sum_{s \in S_{ij}} \bar{x}_{ijs}\, p_{ijs} \leq \frac{1}{1 - \varepsilon}\, C^* \;, \qquad i = 1, \ldots, m\,, \tag{10}$$

$$\sum_{j \in V} \sum_{i=1}^{m} \sum_{s \in S_{ij}} \bar{x}_{ijs}\, s\, p_{ijs} \leq \frac{k}{\varepsilon}\, C^* \;. \tag{11}$$

Proof. The proof of both claims is based on proving the statement for every machine-job pair. First, recall that x^{LP} denotes the optimal fractional solution for the linear programming relaxation of (1)–(5), for $C = C^*$. Validity of (10) can be seen as follows. We know that

$$(1-\varepsilon)\,\bar{x}_{ijs^{ij}} = (1-\varepsilon)\,\tilde{y}_{ij} \leq \sum_{s\in S_{ij}, s\leq t^{ij}} x^{\mathrm{LP}}_{ijs}$$

by definition of t^{ij} in (7). By the fact that $p_{ijt^{ij}} \leq p_{ijs}$ for all $s \leq t^{ij}$, we therefore have

$$(1-\varepsilon)\,\bar{x}_{ijs^{ij}} p_{ijt^{ij}} \leq \sum_{s\in S_{ij}, s\leq t^{ij}} x^{\mathrm{LP}}_{ijs}\, p_{ijs} \leq \sum_{s\in S_{ij}} x^{\mathrm{LP}}_{ijs}\, p_{ijs}$$

for every machine i and job $j \in V$. Again due to monotonicity, $p_{ijs^{ij}} \leq p_{ijt^{ij}}$ for all $j \in V$ and $i = 1,\ldots,m$, and we obtain

$$\begin{aligned}\sum_{s\in S_{ij}} \bar{x}_{ijs}\, p_{ijs} \;&=\; \bar{x}_{ijs^{ij}} p_{ijs^{ij}} \;\leq\; \bar{x}_{ijs^{ij}} p_{ijt^{ij}} \\ &\leq\; \frac{1}{1-\varepsilon} \sum_{s\in S_{ij}} x^{\mathrm{LP}}_{ijs}\, p_{ijs}\end{aligned}$$

for all jobs $j \in V$ and machines $i = 1,\ldots,m$. Summing over $j \in V$, and using (2), inequalities (10) follow for any machine i.

To see (11), first observe that $\varepsilon\,\bar{x}_{ijs^{ij}} = \varepsilon\,\tilde{y}_{ij} \leq \sum_{s\in S_{ij}, s\geq t^{ij}} x^{\mathrm{LP}}_{ijs}$ by definition of t^{ij}, since t^{ij} is the *minimal* index with property (7). Therefore,

$$\varepsilon\,\bar{x}_{ijs^{ij}} s^{ij} p_{ijs^{ij}} \leq \sum_{s\in S_{ij}, s\geq t^{ij}} x^{\mathrm{LP}}_{ijs}\, s\, p_{ijs} \leq \sum_{s\in S_{ij}} x^{\mathrm{LP}}_{ijs}\, s\, p_{ijs}$$

for every machine i and job $j \in V$, where the first inequality follows because s^{ij} was chosen among all $s \geq t^{ij}$ such as to minimize $s\, p_{ijs}$. Hence, we obtain

$$\sum_{s\in S_{ij}} \bar{x}_{ijs}\, s\, p_{ijs} = \bar{x}_{ijs^{ij}} s^{ij}\, p_{ijs^{ij}} \leq \frac{1}{\varepsilon} \sum_{s\in S_{ij}} x^{\mathrm{LP}}_{ijs}\, s\, p_{ijs}\,,$$

for all jobs $j \in V$ and machines $i = 1,\ldots,m$. Summing over $j \in V$ and all machines $i = 1,\ldots,m$, and using (3), eventually yields (11). □

Next, we want to use a rounding procedure by Shmoys and Tardos [18] in order to end up with an integer solution.

Lemma 3 (Shmoys & Tardos [18, Theorem 2.1]). *Given a feasible fractional solution $\tilde{y} = (\tilde{y}_{ij})$ to the linear program*

$$\sum_{i=1}^{m} y_{ij} = 1 \ , \qquad \forall\ j \in V\,, \tag{12}$$

$$\sum_{j \in V} y_{ij}\tau_{ij} \leq T \ , \qquad \forall\ i = 1, \ldots, m\,, \tag{13}$$

$$\sum_{j \in V} \sum_{i=1}^{m} y_{ij}\lambda_{ij} \leq \Lambda \ , \tag{14}$$

$$y_{ij} \geq 0 \ , \qquad \forall\ i, j. \tag{15}$$

with nonnegative parameters $T, \Lambda, \tau = (\tau_{ij})$, and $\lambda = (\lambda_{ij})$, there is a polynomial time algorithm which computes an integral solution $\bar{y}$ to (12), (14), (15), *and*

$$\sum_{j \in V} \bar{y}_{ij}\tau_{ij} \leq T + \tau_{max} \ , \qquad \forall\ i = 1, \ldots, m\,, \tag{16}$$

where $\tau_{\max} = \max_{i,j}\{\tau_{ij} \mid \tilde{y}_{ij} > 0\}$. □

The fractional solution $\tilde{y}$ defined in (6), however, is nothing but a feasible fractional solution for linear program (12)–(15), namely with parameters $T = 1/(1-\varepsilon)\,C^*$, $\Lambda = k/\varepsilon\,C^*$, $\tau_{ij} = p_{ijs^{ij}}$, and $\lambda_{ij} = s^{ij}p_{ijs^{ij}}$ for all job-machine pairs (i, j). Therefore, combining Lemma 2, the above result of Shmoys and Tardos, and the fact that

$$\tau_{\max} = \max_{i,j} p_{ijs^{ij}} \leq \max_{i,j,s}\{p_{ijs} \mid x^{\mathrm{LP}}_{ijs} > 0\} \leq C^* \tag{17}$$

by constraints (4), we can show the following.

Lemma 4. *Let C^* be the lower bound on the makespan of an optimal solution as defined in Lemma 1, then we can find a feasible solution $x^* = (x^*_{ijs})$ for the following integer linear program in polynomial time.*

$$\sum_{i=1}^{m} \sum_{s \in S_{ij}} x_{ijs} = 1 \ , \qquad \forall\ j \in V, \tag{18}$$

$$\sum_{j \in V} \sum_{s \in S_{ij}} x_{ijs}\, p_{ijs} \leq \left(1 + \frac{1}{1-\varepsilon}\right) C^* \ , \qquad \forall\ i\,, \tag{19}$$

$$\sum_{j \in V} \sum_{i=1}^{m} \sum_{s \in S_{ij}} x_{ijs}\, s\, p_{ijs} \leq \frac{k}{\varepsilon}\, C^* \ , \tag{20}$$

$$x_{ijs} \in \{0, 1\} \ , \qquad \forall\ i, j, s\,. \tag{21}$$

Proof. We briefly summarize the previously described steps. First, recall that x^{LP} denotes the optimal fractional solution for the linear programming relaxation of (1)–(5), for $C = C^*$. Using $x^{\mathrm{LP}} = (x^{\mathrm{LP}}_{ijs})$, define $\tilde{y} = (\tilde{y}_{ij})$ as in (6), and apply the rounding defined by (9). This yields a fractional solution $\bar{x} = (\bar{x}_{ijs})$ that is nonzero only for one resource index $s = s^{ij}$, for any pair of i and j, as defined in (8). Interpreting $\tilde{y}$ as fractional solution for the generalized assignment problem (12)–(15), use Lemma 3 to round it to an integral solution $\bar{y} = (\bar{y}_{ij})$. Now define the integral solution x^* by

$$x^*_{ijs} = \begin{cases} \bar{y}_{ij} & s = s^{ij}\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

With the help of Lemmas 2 and 3, and utilizing (17), it is now straightforward to verify that x^* fulfills (18)–(21). □

LP based greedy algorithm. Our approach to obtain a constant factor approximation for the scheduling problem is now the following. We first use the rounded 0/1-solution from the previous section in order to decide both, on the amount of resources allocated to every individual job j, and on the machine where this job must be executed. More precisely, job j must be processed on machine i and use s additional resources iff $x^*_{ijs} = 1$, where x^* is the feasible integral solution of (18)–(21) obtained after the 2-phase rounding. Then the jobs are scheduled according to the greedy list scheduling algorithm of Graham [5], in arbitrary order.

> **Algorithm** LP-Greedy: With the resource allocations and machine assignments as determined by the LP based rounding, do until all jobs are scheduled: Starting at time 0, iterate over completion times of jobs, and schedule as many jobs as allowed, obeying the machine assignments and the resource constraints.

Theorem 1. *Algorithm* LP-Greedy *is a* $(4 + 2\sqrt{2})$*–approximation algorithm for unrelated parallel machine scheduling with resource dependent processing times.*

The fact that the algorithm requires only polynomial time follows directly from the fact that both, solving and rounding the LP relaxation, as well as the list scheduling, can be implemented in polynomial time.

To verify the performance bound, we first need some additional notation. Consider some schedule $\mathcal{S}$ produced by algorithm LP-Greedy, and denote by C^{LPG} the corresponding makespan. Denote by C^{OPT} the makespan of an optimal solution. For schedule $\mathcal{S}$, let $t(\beta)$ denote the earliest point in time after which only *big jobs* are processed, big jobs being defined as jobs that have a resource consumption larger than $k/2$. Moreover, let $\beta = C^{\mathrm{LPG}} - t(\beta)$ be the length of the period in which only big jobs are processed (note that possibly $\beta = 0$).

Next, we fix a machine, say machine i, on which some job completes at time $t(\beta)$ which is not a big job. Due to the definition of $t(\beta)$, such a machine

must exist, because otherwise all machines were idle right before $t(\beta)$, contradicting the definition of the greedy algorithm. Note that, between time 0 and $t(\beta)$, periods may exist where machine i is idle. Denote by α the total length of busy periods on machine i between 0 and $t(\beta)$, and by γ the total length of idle periods on machine i between 0 and $t(\beta)$. We then have that

$$C^{\mathrm{LPG}} = \alpha + \beta + \gamma\,. \tag{22}$$

Due to (19), we get that for machine i

$$\alpha \leq \sum_{j \in V} \sum_{s \in S_{ij}} x^*_{ijs}\, p_{ijs} \leq \left(1 + \frac{1}{1-\varepsilon}\right) C^* \ . \tag{23}$$

The next step is an upper bound on $\beta + \gamma$, the length of the final period where only big jobs are processed, together with the length of idle periods on machine i.

Lemma 5. *We have that*

$$\beta + \gamma \leq \frac{2}{\varepsilon}\, C^*\,.$$

Proof. First, observe that the total resource consumption of schedule $\mathcal{S}$ is at least $\beta\,\frac{k}{2} + \gamma\,\frac{k}{2}$. This because, on the one hand, all jobs after $t(\beta)$ are big jobs and require at least $k/2$ resources, by definition of $t(\beta)$. On the other hand, during all idle periods on machine i between 0 and $t(\beta)$, at least $k/2$ of the resources must be in use as well. Assuming the contrary, there was an idle period on machine i with at least $k/2$ free resources. But after that idle period, due to the selection of $t(\beta)$ and machine i, some job is processed on machine i which is not a big job. This job could have been processed earlier during the idle period, contradicting the definition of the greedy algorithm. Next, recall that $(k/\varepsilon)\, C^*$ is an upper bound on the total resource consumption of the jobs, due to (20). Hence, we obtain

$$\frac{k}{\varepsilon}\, C^* \geq \beta\,\frac{k}{2} + \gamma\,\frac{k}{2}\,.$$

Dividing by $2/k$ yields the claimed bound on $\beta + \gamma$. □

Now we are ready to prove the performance bound of Theorem 1.

Proof (of Theorem 1). First, use (22) together with (23) and Lemma 5 to obtain

$$C^{\mathrm{LPG}} \ \leq \left(1 + \frac{1}{1-\varepsilon}\right) C^* + \frac{2}{\varepsilon}\, C^* \ \leq \ \left(1 + \frac{1}{1-\varepsilon} + \frac{2}{\varepsilon}\right) C^{\mathrm{OPT}}\,.$$

Solving for the best possible value for ε gives $\varepsilon = 2 - \sqrt{2} \approx 0.5858$, which yields the claimed performance bound of $4 + 2\sqrt{2} \approx 6.83$. □

5. Dedicated parallel machines

As a special case of the unrelated machine scheduling model considered so far, let us assume that the jobs are assigned to machines *beforehand*. Recall that the set of jobs V is partitioned into m subsets $V_1, \dots, V_m$ a priori, V_i being the jobs that must be processed on machine i, and p_{js}, $s = 0, \dots, k$, denotes the resource dependent processing time of job j. Such problems are also called *dedicated* parallel machine scheduling problems [13,14].

By letting all but one machine assignment result in very large processing times, this is obviously a special case of the unrelated machine scheduling model. Hence, our above analysis also yields a $(4+2\sqrt{2})$–approximation for this model. However, noting that the machine index i can be eliminated from the formulation (1)–(5), we can use variables x_{js} where $x_{js} = 1$ if job j is processed with s units of the resource. We thus obtain the following LP relaxation.

$$\text{min.} \quad C \tag{24}$$

$$\text{s. t.} \quad \sum_{s \in S_j} x_{js} = 1 \ , \qquad \forall\, j \in V\,, \tag{25}$$

$$\sum_{j \in V_i} \sum_{s \in S_j} x_{js}\, p_{js} \le C \ , \qquad \forall\, i = 1, \dots, m\,, \tag{26}$$

$$\sum_{j \in V} \sum_{s \in S_j} x_{js}\, s\, p_{js} \le k\, C \ , \tag{27}$$

$$x_{js} \ge 0 \ , \qquad \forall j, s\,. \tag{28}$$

Here, recall that S_j are the breakpoints of the function p_{js} as defined in Section 3. The interpretation of the constraints is the same as in the previous section. The only difference is that we do not need the explicit constraints (4), and therefore can just minimize C instead of using binary search on C. Now, the accordingly adapted first phase rounding of (9) already yields an integral solution. More precisely, given a fractional solution (C^*, x^{LP}) for the above LP relaxation, we choose index t^j minimal with the property that $\sum_{s \in S_j, s \le t^j} x^{\mathrm{LP}}_{js} \ge 1 - \varepsilon$ and define index $s^j = \arg\min_{s \ge t^j} s \cdot p_{js}$. Again, it follows that $s^j \in S_j$. Then the solution $\bar{x}$, defined by $\bar{x}_{js} = 1$ if $s = s^j$ and $\bar{x}_{js} = 0$ otherwise, is already integral. Hence, Shmoys and Tardos' rounding is not required, and instead of using the bounds (19) and (20), we now have an integral solution $\bar{x} = (\bar{x}_{js})$ which fulfills the constraints

$$\sum_{j \in V_i} \sum_{s \in S_j} \bar{x}_{js}\, p_{js} \le \frac{1}{1-\varepsilon}\, C^* \ , \qquad \forall\, i = 1, \dots, m\,,$$

$$\sum_{j \in V} \sum_{s \in S_j} \bar{x}_{js}\, s\, p_{js} \le \frac{k}{\varepsilon}\, C^* \ .$$

Validity of these bounds is proved along the same lines as Lemma 2. This eventually yields an improved performance bound for the dedicated machine model.

Theorem 2. *Algorithm* LP-Greedy *is a* $(3+2\sqrt{2})$*–approximation algorithm for dedicated parallel machine scheduling with resource dependent processing times.*

6. Identical parallel machines

We can further improve the performance guarantee for the case that jobs are not assigned to machines beforehand, but each jobs can be processed on any machine, and all machines are identical. Hence, any job j has a processing time p_{js} whenever s units of the resource are assigned to the job, independent of the machine where the job is processed on. As the problem with dedicated parallel machines, this identical parallel machine problem is a special case of the problem with unrelated parallel machines, hence Section 4 yields an approximation ratio of $(4+2\sqrt{2})$. For the identical parallel machine case, however, we next derive a performance guarantee of $(5+\varepsilon)$, for any $\varepsilon > 0$.

Like in the previous section, let x_{js} denote binary variables, indicating that an amount of s resources is used for processing job j. Then the following integer linear program, referred to as (IPM), has a feasible solution if there is a feasible schedule of length C for the original scheduling problem.

$$\sum_{s\in S_j} x_{js} = 1 \ , \qquad \forall\ j \in V\,, \tag{29}$$

$$\sum_{j\in V}\sum_{s\in S_j} x_{js}\, p_{js} \le m\, C \ , \tag{30}$$

$$\sum_{j\in V}\sum_{s\in S_j} x_{js}\, s\, p_{js} \le k\, C \ , \tag{31}$$

$$x_{js} = 0 \ , \qquad \text{if } p_{js} > C, \tag{32}$$

$$x_{js} \in \{0,1\} \ , \qquad \forall\ j, s. \tag{33}$$

Here, as in the previous section, S_j are the breakpoints of the function p_{js} as defined in Section 3. Again, C represents the schedule makespan. Equalities (29) make sure that every job uses a constant amount of resources during its processing. Inequalities (30) express the fact that the average total processing time per machine cannot be more than the makespan C. Inequalities (31) again represent the aggregated resource constraints, as in the two previous sections, and condition (32) is required later in the proof.

In order to decide on feasibility for the above program (IPM) for any given value of C, we can equivalently solve the following linear integer program

$$\text{min.} \sum_{j\in V}\sum_{s\in S_j} x_{js}\, s\, p_{js} \tag{34}$$

$$\text{s.t.} \sum_{s\in S_j} x_{js} = 1 \ , \qquad \forall\, j \in V\,, \tag{35}$$

$$\sum_{j\in V}\sum_{s\in S_j} x_{js}\, p_{js} \le m\, C \ , \tag{36}$$

$$x_{js} = 0 \ , \qquad \text{if } p_{js} > C, \tag{37}$$

$$x_{js} \in \{0,1\} \ , \qquad \forall\, j, s\,. \tag{38}$$

Clearly, if the optimal objective of (34)–(38) does not exceed $k\,C$, (29)–(33) is feasible, and otherwise infeasible.

Observe that we could as well eliminate constraints (37) by redefining $p_{js} = mC + 1$ whenever $p_{js} > C$. After this transformation, the problem is in fact equivalent to the *multiple-choice knapsack problem*, for which Gens and Levner derived a 4/5-approximation algorithm [4].

We next claim that problem (34)–(38) admits an FPTAS, a fully polynomial time approximation scheme. Notice that this also implies an FPTAS for the multiple-choice knapsack problem of [4]. Indeed, after the described transformation, problem (34)–(38) can also be interpreted as a single machine scheduling problem with n jobs, each of them can be processed in a mode $s \in S_j$, with associated processing time p_{js} and cost $w_{js} := s\, p_{js}$. The problem is to select exactly one mode s for each job such that the due date constraint (36) is fulfilled, and the goal is to minimize total costs $\sum_{j\in V} w_{js}$. It follows from Lemma 2 of Grigoriev and Uetz [9] that such problems admit an FPTAS. In fact, this lemma is just an application of a more general theorem about subset selection problems by Pruhs and Woeginger [17]. We summarize as follows.

Lemma 6. *For any $\delta > 0$, a solution for problem* (34)–(38) *can be computed that is not more than a factor $(1+\delta)$ away from the optimum, in time polynomial in the input size and $1/\delta$.*

Now, for any $\delta > 0$ we can use this FPTAS to design an approximate binary search for the smallest integer value, say C^{IPM}, for which the linear integer program (IPM) has a feasible solution. To this end, we compute by binary search the smallest integer value C^* such that the above FPTAS yields a solution for (34)–(38) with objective z_{C^*}, such that

$$z_{C^*} \le (1+\delta) k\, C^*\,. \tag{39}$$

By minimality of C^* for property (39), for $C' := C^* - 1$, the FPTAS yields a solution with $z_{C'} > (1+\delta) k\, C'$. Hence, by Lemma 6, the integer linear program (IPM) has no feasible solution for value C'. Therefore, the minimal integral value for which the linear integer program (IPM) has a feasible solution is at least $C'+1 = C^*$, or in other words, $C^* \le C^{\text{IPM}}$, and thus C is a lower bound on the makespan of an optimal solution for the original scheduling problem. Using the fact that the FPTAS delivers a solution with property (39), we conclude the following.

Lemma 7. *For any $\delta > 0$ we can compute a lower bound C^* on the makespan of an optimal schedule, and a solution x^* of integer program* (29)–(33) *where constraint* (31) *is relaxed to*

$$\sum_{j \in V} \sum_{s \in S_j} x^*_{js}\, s\, p_{js} \leq (1+\delta)\, k\, C^* \quad . \tag{40}$$

The computation time is polynomial in the input size of the problem and $1/\delta$.

The next step is a modification of the greedy algorithm. For any job j fix the resource consumption to s_j, where s_j is the amount of resources that the solution x^* of Lemma 7 assigns to job j, i.e., $x_{js_j} = 1$. Then the jobs are scheduled in any order according to the traditional greedy list scheduling algorithm of Graham [5]. Finally, the obtained schedule is corrected as follows.

Algorithm IPM-Greedy:
Scheduling phase. With the resource allocations as determined above, do until all jobs are scheduled: Starting at time 0, iterate over completion times of jobs, and at any time schedule as many jobs as allowed, obeying the machine and resource constraints.
Correction phase. Find all time periods when the total resource consumption is less than $k/2$ and at least one machine is idle, say T is the union of all such time periods. Now, each machine has at most one job in total that is processed during one of the periods of T, since otherwise the later job would have been scheduled earlier by the greedy algorithm on the idle machine: it requires less than $k/2$ resources and these resources are available on the idle machine. Next remove all jobs that are processed in T, say this is set R. By the observation above, $|R| \leq m$. Concatenate the remaining parts of the schedule. To schedule the at most m remaining jobs in R, we can now solve the minimum height two dimensional strip packing problem with strip width k and rectangle sizes $s_j \times p_{js_j}$. This is feasible since each job can be scheduled on an individual machine. To solve the strip packing problem, use a polynomial time 2-approximation algorithm of Steinberg [20]. Concatenate the obtained two schedules.

Theorem 3. *For any $\varepsilon > 0$, algorithm* IPM-Greedy *is a $(5+\varepsilon)$–approximation algorithm for identical parallel machine scheduling with resource dependent processing times. The computation time is polynomial in the input size of the problem and $1/\varepsilon$.*

Proof. The fact that the algorithm requires only polynomial time follows directly from the fact that for deciding on the resource assignment we use Lemma 7, and the list scheduling, finding the time periods T in the correction phase, and eventually the approximate solution of the strip packing problem can be implemented in polynomial time.

To verify the performance bound, fix $\delta = \varepsilon/3$, and consider some schedule $\mathcal{S}$ produced by algorithm IPM-Greedy. Denote by C^{IPMG} the corresponding

makespan of schedule $\mathcal{S}$, by γ the makespan for the job set R in the solution of the strip packing problem obtained by the algorithm from [20], and by C^{OPT} the (overall) makespan of an optimal solution. For the first part of schedule $\mathcal{S}$, let α be the total length of the intervals without idle machines, and β be the total length of the intervals where at least $k/2$ units of the resource are assigned to jobs. By construction of the first part of the schedule (i.e., by definition of the periods T), all time periods belong either to α or β. From (30) and (40), we have that

$$\alpha \leq C^*\,, \quad \text{and} \quad \beta \leq 2\,(1+\delta)\,C^*\,,$$

where C^* is the lower bound on C^{OPT} from Lemma 7.

Next let us consider the second part of the schedule. Since in the strip packing problem there are at most m rectangles, the corresponding schedule of the jobs from R does not violate the constraint that the number of machines is m. By Theorem 2.3 in [20], we get for the makespan $\gamma \leq L + \max\{L, p_{\max}\}$, where $L = \sum_{j\in R} s_j p_{js_j}/k$ and $p_{\max} = \max_{j\in R}\{p_{js_j}\}$. Now by (40), we have

$$L = \sum_{j\in R} s_j p_{js_j}/k \leq \sum_{j\in V} s_j p_{js_j}/k \leq (1+\delta)C^*\,,$$

and by(32), $p_{\max} = \max_{j\in R}\{p_{js_j}\} \leq C^*$. Therefore, the makespan γ of the jobs in R is bounded by $\gamma \leq 2C^* + \delta C^*$.

Hence, by choice of $\delta = \varepsilon/3$ we get that

$$C^{\mathrm{IPMG}} = \alpha+\beta+\gamma \leq C^*+2(1+\delta)C^*+(2C^*+\delta C^*) = (5+\varepsilon)C^* \leq (5+\varepsilon)C^{\mathrm{OPT}},$$

as required. □

7. Lower bounds

Lower bound for the (integer) linear programs. We next give an instance to show that the integer linear programs we use can be a factor $(2-\varepsilon)$ away from the optimal solution, for any $\varepsilon > 0$. Hence, our LP-based analysis cannot yield anything better than a 2-approximation, for all three versions, the unrelated, dedicated, and parallel machine case.

Example 1 *Consider a problem with $m = 2$ (dedicated or parallel) machines and k units of the additional resource, where k is odd. There are 2 jobs, with resource-dependent processing times*

$$p_{js} = \begin{cases} 2k+1 & \text{if } s < \frac{k}{2} \\ k & \text{if } s > \frac{k}{2} \end{cases}$$

for both jobs j. □

Consider the integer solution x, where $x_{js} = 1$ if $s = \lceil k/2 \rceil$, and $x_{js} = 0$ otherwise, for both jobs j. With this setting of variables consider the program (24)–(28), then the two inequalities (26) yield $k \leq C$, and inequality (27) yields $k(k+1) = 2k\lceil k/2 \rceil \leq kC$. With the same setting of variables, consider program (29)–(33), then inequality (30) yields $2k \leq 2C$, and from inequality (31), we get $k(k+1) = 2k\lceil k/2 \rceil \leq kC$, as above. Therefore, with $C = (k+1)$, there exists a feasible, even integral solution for both, integer programming relaxation (24)–(28) for the dedicated machine case, and integer programming relaxation (29)–(33) for the identical machine case. A fortiori, we know that for the corresponding *linear* programming relaxations, $C^* \leq k+1$. But in the optimal solution, $C^{\mathrm{OPT}} = 2k$. Hence, in both settings the gap between C^* and C^{OPT} can be as large as $2-\varepsilon$, for any $\varepsilon > 0$. The bad quality of the LP lower bound is obviously a consequence of the fact that we only use an aggregate formulation of the resource constraints in (27) (or (31), respectively), whereas any schedule has to respect the resource constraint at any time. We summarize as follows.

Observation 1 *There are instances where the respective lower bounds provided by the integer programming relaxations* (24)–(28)*,* (29)–(33)*, and a fortiori also* (1)–(5)*, have a feasible solution that is a factor* $2-\varepsilon$ *away from the optimum, for any* $\varepsilon > 0$*.*

Lower bounds on approximation. The problem with unrelated machines cannot be approximated within a factor smaller than 3/2 as a generalization of the classical unrelated machine scheduling problem [15], as mentioned earlier. We next show that the same inapproximability result holds for the problems with dedicated or identical parallel machines.

Theorem 4. *There is no polynomial time approximation algorithm for dedicated or identical parallel machine scheduling with resource dependent processing times that has a performance guarantee less than* 3/2*, unless* P=NP*.*

Proof. The proof relies on a gap-reduction from Partition [3]: Given n integers a_j, with $\sum_{j=1}^{n} a_j = 2k$, it is NP-complete to decide if there exists a subset $W \subseteq \{1,\ldots,n\}$ with $\sum_{j\in W} a_j = k$. Let us define an instance of the machine scheduling problem (either dedicated or identical machines) as follows. Each a_j gives rise to one job j with an individual machine. Hence, we have n jobs and $m = n$ machines (and in fact it does not matter for what follows if the machines are dedicated or identical). There are k units available of the additional resource. Any job j has a processing time defined by

$$p_{js} = \begin{cases} 3 & \text{if } s < a_j \\ 1 & \text{if } s \geq a_j\,. \end{cases}$$

Hence, the a_j's are the only breakpoints in the functions p_{js}, and the index set $S_j = \{0, a_j\}$ for all jobs j. In other words, the functions p_{js} can be encoded in $\mathrm{O}(\log a_j)$ for all jobs j, and the transformation is indeed polynomial. We claim that there exists a feasible schedule with makespan $C_{\max} < 3$ if and only

if there exists a solution for the PARTITION problem. Otherwise, the makespan is at least 3. To this end, observe that in any solution with makespan $C_{\max} < 3$, we may assume that each job j consumes exactly a_j units of the resource: If it was less than a_j for some jobs j, the makespan would be at least 3; if it was more than a_j for some job j, letting the resource allocation equal a_j does not violate feasibility, while maintaining the same processing time. Now, if and only if there is a solution, say W, for the PARTITION problem, there exists a resource feasible schedule with makespan 2, namely where jobs $j \in W$ start at time 0, and all jobs $j \notin W$ start at time 1. □

Finally, it is not difficult to see that the above proof yields the same inapproximability result for the problems with dedicated or parallel machines, even if the resource consumption of jobs is fixed beforehand.

Corollary 1. *There is no polynomial time approximation algorithm for dedicated or identical parallel machine scheduling with an additional resource constraint that has a performance guarantee less than* 3/2*, unless* P=NP*.*

In contrast to this corollary, note that for dedicated machines, there exists a polynomial time algorithm if the number of machines is 2 [14], and a PTAS if the number of machines m is fixed and the resource is binary (i.e., $k = 1$) [13].

8. Concluding Remarks

The integer programming relaxations that we use may be a factor 2 away from the respective optimal solutions, but we have not been able to design tighter worst case examples for either of the three settings. It is not too difficult to construct instances where the individual steps of the algorithms perform suboptimal, yet it seems difficult to design instances that indeed 'withstand' the concatenations of the individual steps of the algorithms.

Moreover, it remains open at this point if the unrelated parallel machine scheduling problem with resource dependent processing times admits a stronger inapproximability results than 3/2. Since the problem adds an additional time-resource tradeoff to the classical problem $R||C_{\max}$, one could conjecture it to be more difficult. Yet, the inapproximability results of 3/2 already holds for the problem without additional resources, $R||C_{\max}$.

Acknowledgements. This work was partially supported by METEOR, the Maastricht Research School of Economics of Technology and Organizations.

References

1. J. Blazewicz, J. K. Lenstra and A. H. G. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, Discrete Applied Mathematics, **5** (1983), 11–24.
2. Z.-L. Chen, Simultaneous Job Scheduling and Resource Allocation on Parallel Machines, Annals of Operations Research, **129** (2004), 135–153.

3. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completenes*, W. H. Freeman, New York, 1979.
4. G. Gens and E. Levner, An approximate binary search algorithm for the multiple-choice knapsack problem, Information Processing Letters, **67** (1998), 261-265.
5. R. L. Graham, Bounds for certain multiprocessing anomalies, Bell System Technical Journal, **45** (1966), 1563–1581. See also [6].
6. R. L. Graham, Bounds on multiprocessing timing anomalies, SIAM Journal on Applied Mathematics, **17** (1969), 416–429.
7. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, Annals of Discrete Mathematics, **5** (1979), 287–326.
8. A. Grigoriev, H. Kellerer and V. A. Strusevich, Scheduling parallel dedicated machines with the speeding-up resource, manuscript (2003). Extended abstract in: Proceedings 6th Workshop on Models and Algorithms for Planning and Scheduling Problems, Aussois, France, 2003, 131–132.
9. A. Grigoriev and M. Uetz, Scheduling Jobs with Linear Speedup, Proceedings 3rd Workshop on Approximation and Online Algorithms, T. Erlebach and P. Persiano (eds.), Lecture Notes in Computer Science, Springer, 2006, to appear.
10. D. S. Hochbaum and D. B. Shmoys, Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results, Journal of the ACM, **34** (1987), 144–162.
11. K. Jansen and M. Mastrolilli, Approximation schemes for parallel machine scheduling problems with controllable processing times, Computers and Operations Research **31** (2004), 1565–1581.
12. J. E. Kelley and M. R. Walker, *Critical path planning and scheduling: An introduction*, Mauchly Associates, Ambler (PA), 1959.
13. H. Kellerer and V. A. Strusevich, Scheduling parallel dedicated machines under a single non-shared resource, European Journal of Operational Research, **147** (2003), 345–364.
14. H. Kellerer and V. A. Strusevich, Scheduling problems for parallel dedicated machines under multiple resource constraints, Discrete Applied Mathematics, **133** (2004), 45–68.
15. J. K. Lenstra, D. B. Shmoys and E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, Mathematical Programming, Series A, **46** (1990), 259–271.
16. G. Mounie, C. Rapine, and D. Trystram, Efficient Approximation Algorithms for Scheduling Malleable Tasks, Proceedings 11th Annual ACM Symposium on Parallel Algorithms and Architectures, 1999, 23–32.
17. K. Pruhs and G. J. Woeginger, Approximation Schemes for a Class of Subset Selection Problems, Proceedings 6th Latin American Symposium on Theoretical Informatics, M. Farach-Colton (ed.), Lecture Notes in Computer Science 2976, Springer, 2004, 203–211.
18. D. B. Shmoys and E. Tardos, An approximation algorithm for the generalized assignment problem, Mathematical Programming, Series A, **62** (1993), 461–474.
19. M. Skutella, Approximation algorithms for the discrete time-cost tradeoff problem, Math. Oper. Res. **23** (1998), pp. 909–929.
20. A. Steinberg, A Strip-Packing Algorithm with Absolute Performance Bound 2, SIAM Journal on Computing, **26** (1997), 401–409.
21. J. Turek, J. L. Wolf, and P. S. Yu, Approximate Algorithms for Scheduling Parallelizable Tasks, Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 1992, 323–332.