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 machine scheduling on unrelated parallel machines with the objective to minimize the schedule makespan. We assume that, in addition to its machine dependence, the processing time of any job is dependent on the usage of a discrete renewable resource, e.g. workers. A given amount of that resource 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 the classical unrelated parallel machine scheduling problem by 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 an integer linear programming formulation for a relaxation of the problem, we use LP rounding techniques to allocate resources to jobs, and to assign jobs to machines. Combined with Graham's list scheduling, we show how to derive a 4-approximation algorithm for the scheduling problem. We also show how to tune our approach to yield a 3.75-approximation algorithm. This is achieved by applying the same rounding technique to a slightly modified linear programming relaxation, and by using a more sophisticated scheduling algorithm that is inspired by the harmonic algorithm for bin packing. We finally derive inapproximability results for two special cases, and discuss tightness of the integer linear programming relaxations.

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 if processed 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 part of the input, the best approximation algorithm to date is a 2-approximation by Lenstra, Shmoys and Tardos [20]. Moreover, the problem cannot be approximated within a factor smaller than 3/2, unless P=NP [20]. 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 [4] and admits a polynomial time approximation scheme [12].

Shmoys and Tardos [23] 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} [23]. Note that, in both versions of the problem studied in [23], the costs λ_{ij} are *nonrenewable* resources, such as a monetary budget, with a global budget Λ.

In this paper, we consider another generalization of the unrelated parallel machine scheduling problem $R||C_{\max}$; and at the same time a variant of the problem considered by Shmoys and Tardos [23]. Namely, we assume that the processing time of the jobs can be reduced by utilizing a discrete *renewable* resource, such as additional workers that can be allocated to the jobs. 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 [23], 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. Two other, more special machine scheduling settings will be briefly addressed as well. One is the problem where the processing time of any job is independent of the machine that processes the job, the *identical* parallel machine scheduling problem $P||C_{\max}$. The other is the problem where the jobs are assigned to machines beforehand, sometimes also called *dedicated* parallel machine scheduling [16,17].

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 [2,14,18,23,24]. Surprisingly, time-resource tradeoff problems with a *renewable* resource constraint, such as personnel, have received much less attention, although they are not less appealing from a practical viewpoint. 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. Some (restricted) versions of the problem considered in this paper were addressed in [11,13,16,21,25].

2. History, related work and results

This paper is the synthesis of our two conference papers [9] and [10]. To the best of our knowledge, the unrelated machine scheduling problem with the additional feature of processing times that depend on a renewable resource was first considered in [9]. In that paper, we prove the existence of a 6.83-approximation algorithm. Subsequently, Kumar, Marathe, Parthasarathy, and Srinivasan [19] prove that there exists a randomized 4-approximation algorithm for that problem. Then main result of their paper is a randomized rounding procedure that is applicable to several relaxations for unrelated machine scheduling problems. Inspired by their paper, in [10] we show how to obtain a deterministic 3.75-approximation for the scheduling problem at hand. To that end, we use a derandomized version of the rounding procedure of Kumar et al. [19]. This procedure considerably improves upon the weaker rounding procedure proposed in our earlier paper [9]. The improved rounding alone yields a deterministic 4-approximation algorithm for the scheduling problem. The further improvement to a 3.75-approximation algorithm is due to a modified linear programming relaxation in combination with a more sophisticated scheduling algorithm. In fact, the scheduling algorithm resembles a restricted version of the harmonic algorithm for bin packing. The present paper reflects this evolution of results by first presenting the simpler 4-approximation algorithm, and then presenting the slightly more sophisticated 3.75-approximation algorithm.

Related work. Let us next discuss further 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]. Grigoriev and Uetz [11] have generalized the approximation result of [8]. The model of [11] 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 $p_{js} = p_j - b_j \cdot s$ if s of the k resources are used. Using a quadratic programming relaxation, a $(3+\varepsilon)$–approximation algorithm is derived in [11], for an arbitrary number of machines m.

Jobs with resource dependent processing times also appear in the literature as *malleable* or *parallelizable tasks*, e.g. in [21,25]. 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. [25] derive a 2–approximation algorithm for this problem. In fact, the model considered in [25] is just a special case of the problem considered in this paper. Interpreting the parallel processors as a generic 'resource' that may be allocated to jobs, the problem of [25] 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, the number of machines m then equals the number of jobs n. Mounie et al. [21] consider yet another variant where the processor allocations must be contiguous (for that problem, [25] includes a 2.7–approximation). Moreover, in [21] it is not only assumed that the processing times p_{js} are non-increasing in s, but also the total work $s \cdot p_{js}$ are assumed to be non-decreasing in s. For that problem, a $(\sqrt{3}+\varepsilon)$–approximation is derived in [21]. An unpublished journal version of that paper [22] claims an improved performance bound of $(3/2+\varepsilon)$. Finally, an asymptotic fully polynomial approximation scheme for malleable task scheduling was proposed by Jansen [13].

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 [16,17]. 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 [16]. 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 [16].

Results and methodology. Our approach is based upon an integer linear programming formulation that defines a relaxation of the problem. It takes as input all possible processing times p_{ijs} of jobs. The main idea behind this relaxation is the utilization of an aggregate version of the resource constraints, yielding a formulation that does not require time-indexed variables. The problem and the corresponding relaxations are formally introduced in Section 3.

Section 4 then discusses the LP rounding procedure. We show how to round a fractional solution to an integer solution without loosing too much in terms of violation of the constraints. This rounding procedure can be seen as a derandomized version of the randomized rounding proposed by Kumar et al. [19], and it considerably improves upon the rounding used in the conference version of this paper [9]. In fact, the new rounding procedure can be viewed as an extension of the Shmoys and Tardos rounding theorem for the generalized assignment problem [23]. In this extension we consider the generalized assignment problem on a bipartite *multigraph* instead of a simple bipartite graph. Note that the techniques from [23] do not seem to be extendible to the case of a multigraph.

In Section 5, we show how to obtain a reasonably simple 4-approximation algorithm for the scheduling problem. From the rounded solution of the LP relaxation, we extract both the machine assignments and the resource allocations for the jobs. We then use Graham's greedy list scheduling [5] to generate a feasible schedule. Using the LP lower bound, we prove that this schedule is not more than a factor 4 away from the optimum. Note that this performance bound matches the one of the randomized algorithm by Kumar et al. [19].

Next, in Section 6 we show how to tune our approach in order to yield a 3.75–approximation algorithm. We achieve this –at the cost of loosing a bit of simplicity– by applying the rounding techniques from Section 4 to a slightly more sophisticated integer programming relaxation, and by modifying the scheduling algorithm appropriately. In fact, the scheduling algorithm can be interpreted as a restricted version of the harmonic algorithm for bin packing.

Section 7 addresses lower bounds. 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 an approximation algorithm with performance guarantee smaller than 3/2 cannot exist, unless P=NP [20]. In Section 7, we furthermore show that the same inapproximability threshold of 3/2 holds for both, the identical and the dedicated parallel machine settings with resource dependent processing times. Concerning tightness of our analysis, we finally provide an instance showing that, for any $\varepsilon > 0$, the integer linear programming relaxations that we use can be a factor $(2 - \varepsilon)$, respectively $(1.75 - \varepsilon)$ away from the optimum solution. Hence, our LP-based analysis cannot yield anything better than that. This lower bound holds for all three problem settings, unrelated, parallel, and dedicated machine scheduling.

We end with some concluding remarks and open problems in Section 8.

3. Problem definition and LP relaxation

Let $V = \{1, \ldots, n\}$ be a set of jobs. Jobs must be processed non-preemptively on a set $M = \{1, \ldots, m\}$ of unrelated (or identical, or dedicated) parallel machines. 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, regarding their 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). Without loss of generality, we also assume that all processing times p_{ijs} are integral. Hence, we can restrict to feasible schedules where the jobs only start (and end) at integral points in time.

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, since we assume a discrete resource, the amount of resources assigned to any job must be integral, and we require it to be the same along its processing. In other words, if $s \leq k$ 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} are *strictly* decreasing in s, the only information that is effectively required is the *breakpoints* of p_{ijs}, that is, indices s where $p_{ijs} < p_{ij,s-1}$. Hence, define the 'relevant' indices for job j on machine i as

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

Considering this 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 $s' \in S_{ij}$, without violating feasibility.

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 consider the following integer linear program, referred to as (IP).

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

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

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

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

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

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. Inequality (3) represents the aggregated resource constraint: 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 $k\,C$. Observe that we only use a single, time-aggregated resource constraint. Clearly, this constraint does not model the dynamics of a renewable resource. Yet it turns out that constraint (3) suffices to yield reasonably good lower bounds. In fact, we explicitly use this relaxed formulation of the resource constraints in order to bypass the problem to model the dynamics of a renewable resource, e.g. by using a more complicated time-indexed formulation. 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 the integer program (IP), but they will play a role later when rounding a fractional solution for the linear relaxation of (IP). Summarizing the above observations, we have:

Lemma 1. *If there is a feasible schedule with makespan C for the unrelated machine scheduling problem with resource dependent processing times, integer linear program* (1)–(5) *has a feasible solution* (C, x).

Because constraint (3) only limits the total resource consumption over the whole scheduling horizon, this integer program may have a feasible solution for some integer value of C even though a feasible schedule with makespan C does not exist; see also Example 1 in Section 7.

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

$$x_{ijs} \ge 0\,, \qquad j \in V\,,\ s \in S_{ij}\,,\ i \in M$$

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 $\sum_{j\in V}\min_{i\in M}\{p_{ijk}\}$. 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 2. *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.*

Notice that, as long as we insist on constraints (4), we can not just solve a single linear program minimizing C, since constraints (4) depend nonlinearly on C. Moreover, due to the fact that we only search for integral values C, the binary search on C does not entail any additional approximation error.

4. The rounding procedure

Given a feasible solution (C^*, x^{LP}) for the linear programming relaxation of (1)–(5), the vector x^{LP} may clearly be fractional. We aim at rounding this fractional solution to an integer one without sacrificing too much in terms of violation of the constraints (2) or (3). To this end, in the proceedings version of this paper the following lemma is proved.

Lemma 3 (Grigoriev et al. [9]). *Let C^* be the lower bound on the makespan of an optimal solution as defined in Lemma 2, then for any $\varepsilon > 0$ we can find a feasible solution $x^* = (x^*_{ijs})$ for the following integer linear program in polynomial time.*

$$\begin{aligned}
&\sum_{i\in M}\sum_{s\in S_{ij}} x_{ijs} = 1\ , && \forall\, j\in V,\\
&\sum_{j\in V}\sum_{s\in S_{ij}} x_{ijs}\,p_{ijs} \le \left(1+\frac{1}{1-\varepsilon}\right) C^*\ , && \forall\, i\in M\,,\\
&\sum_{j\in V}\sum_{i\in M}\sum_{s\in S_{ij}} x_{ijs}\,s\,p_{ijs} \le \frac{k}{\varepsilon} C^*\ , &&\\
&x_{ijs}\in\{0,1\}\ , && \forall\, i,j,s\,.
\end{aligned}$$

The proof relies on a 2-phase rounding procedure. In the first rounding phase, a fractional solution x^{LP} is rounded 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. This new fractional solution $\bar{x}$ then defines a solution for an LP relaxation for the generalized assignment problem discussed by Shmoys and Tardos [23]. Therefore, one can utilize the rounding procedure proposed in [23] as a second rounding phase, and Lemma 3 can be derived. We refer to [9] for a proof of Lemma 3, and note that it allows us to derive an approximation algorithm with performance guarantee $(4+2\sqrt{2}) \approx 6.83$ for the unrelated parallel machine problem, and $(3+2\sqrt{2}) \approx 5.83$ for the dedicated parallel machine problem [9].

The result of Lemma 3 can be further improved by observing that in an optimal LP-solution, there is at most one fractional assignment of each job to any machine. This was observed by Kellerer [15]. However, we next derive an even stronger result, which will be used to obtain an approximation algorithm with performance guarantee 4 for the unrelated parallel machine setting. We present a rounding procedure that is inspired by a recent paper by Kumar et al. [19]. In fact, it can be seen as a deterministic version of the randomized rounding algorithm of [19]. In the following lemma, we replace the total resource consumptions of jobs, $s\,p_{ijs}$, by arbitrary (nonnegative) coefficients c_{ijs}.

Lemma 4. *Let C^* be the minimal integer for which the following linear program has a feasible solution*

$$\sum_{i\in M}\sum_{s\in S_{ij}} x_{ijs} = 1\ , \qquad \forall\, j \in V, \tag{6}$$

$$\sum_{j\in V}\sum_{s\in S_{ij}} x_{ijs}\, p_{ijs} \le C^*, \qquad \forall\, i \in M\,, \tag{7}$$

$$\sum_{j\in V}\sum_{i\in M}\sum_{s\in S_{ij}} x_{ijs} c_{ijs} \le kC^*\ , \tag{8}$$

$$x_{ijs} = 0\ , \qquad \textit{if } p_{ijs} > C, \tag{9}$$

$$x_{ijs} \ge 0\ , \qquad \forall\, i,j,s\,, \tag{10}$$

and let (C^*, x^{LP}) be the corresponding feasible solution, then we can find a feasible solution $x^* = (x^*_{ijs})$ for the following integer linear program in polynomial time.

$$\sum_{i\in M}\sum_{s\in S_{ij}} x_{ijs} = 1\ , \qquad \forall\, j \in V, \tag{11}$$

$$\sum_{j\in V}\sum_{s\in S_{ij}} x_{ijs}\, p_{ijs} \le C^* + p_{\max}, \qquad \forall\, i \in M\,, \tag{12}$$

$$\sum_{j\in V}\sum_{i\in M}\sum_{s\in S_{ij}} x_{ijs} c_{ijs} \le kC^*\ , \tag{13}$$

$$x_{ijs} \in \{0,1\}\ , \qquad \forall\, i,j,s\,, \tag{14}$$

where $p_{\max} = \max\{p_{ijs} \mid x^{\mathrm{LP}}_{ijs} > 0\}$ and $c_{ijs} \ge 0$ are arbitrary fixed coefficients.

One option to prove the lemma is to derandomize the corresponding randomized rounding algorithm of [19], using the method of conditional probabilities. For reasons of self-containedness and accessibility, we nevertheless prefer to present a direct proof here. Notice, however, that the basic elements of the proof are indeed the same as in [19].

Proof. The rounding algorithm works in stages. Let x denote the current fractional solution in a given stage. In the first stage, define $x = x^{\mathrm{LP}}$, and notice that x^{LP} fulfills (11)–(13). Subsequently, we alter the current solution x, while maintaining validity of constraints (11) and (13).

In each stage, we consider a bipartite multigraph $G(x) = (V \cup M, E)$, where the set E of edges is defined as follows. For every pair $i \in M$ and $j \in V$, E contains a set of parallel edges, namely one for each fractional value $0 < x_{ijs} < 1$, $s = 0, \dots, k$. Therefore, we could have up to $k+1$ parallel edges between every machine-job pair (i, j). Notice that the degree of any non-isolated vertex $v \in V$ is at least 2, due to constraint (11). We furthermore eliminate isolated vertices from graph $G(x)$.

We will encode each edge $e \in E$ by the triplet (i, j, s). For every vertex $w \in V \cup M$ let d_w denote its degree in $G(x)$, and let E_w be the edges incident to w. We define a variable ϵ_{ijs} for every edge $(i, j, s) \in E$ and a set of linear equations:

$$\sum_{(i,j,s)\in E_j} \epsilon_{ijs} = 0,\ \ j \in V\,, \tag{15}$$

$$\sum_{(i,j,s)\in E_i} p_{ijs}\epsilon_{ijs} = 0,\ \ i \in M \text{ and } d_i \geq 2\,. \tag{16}$$

Let c_1 and c_2 be the number of constraints in (15) and (16), respectively. Let $r \leq \min\{c_1 + c_2, |E|\}$ be the rank of that system. Now observe that $c_1 \leq |V| \leq |E|/2$, because of constraint (11). Moreover, $c_2 \leq |E|/2$ by definition. Thus we obtain that either $r \leq c_1 + c_2 \leq |E| - 1$ or $c_1 = c_2 = |E|/2$. In the latter case, constraints (11), the degree condition in (16), and the fact that there are no isolated vertices, imply that there are exactly $|E|$ vertices in $G(x)$. Hence, the degree of each vertex must equal 2 (and graph $G(x)$ is a collection of even cycles).

Consider the first case when $r \leq |E| - 1$. Since the system of linear equations (15)–(16) is underdetermined, by Gaussian elimination we can find a general solution of this system in the form $\epsilon_{ijs} = \sum_{t=1}^{|E|-r} \alpha_{tijs}\delta_t$ in polynomial time. Here, δ_t, $t = 1 \dots, |E| - r$, are the real valued parameters representing the degrees of freedom of the linear system, and $\alpha_{tijs} \neq 0$ are the corresponding coefficients. Hence, by fixing $\delta_2 = \delta_3 = \dots = \delta_{|E|-r} = 0$, we obtain a solution $\epsilon_{ijs} = \alpha_{1ijs}\delta_1$. For convenience of notation we just write $\epsilon_{ijs} = \alpha_{ijs}\delta$, and note that δ is an arbitrary parameter.

Next, we can define a new fractional solution of the original linear program by letting

$$\bar{x}_{ijs} = \begin{cases} x_{ijs} + \alpha_{ijs}\delta & \text{if } x_{ijs} \text{ is fractional,} \\ x_{ijs} & \text{otherwise.} \end{cases}$$

Due to constraints (15) and (16) we obtain that constraints (11) are satisfied for all $j \in V$, and constraints (12) are satisfied for all $i \in M$ except those vertices (machines) $i \in M$ that have $|E_i| = 1$. Finally, since $\sum_{j\in V}\sum_{i\in M}\sum_{s\in S_{ij}} \bar{x}_{ijs}c_{ijs}$ is a linear function of δ we obtain that constraint (13) is satisfied either for positive or for negative δ. Therefore, by choosing δ either maximal or minimal such that $0 \le \bar{x}_{ijs} \le 1$ and such that constraint (13) is still satisfied, we obtain a new solution with one more integral variable satisfying constraints (11) and (13).

Repeating the above procedure we either end up with an integral solution x, fulfilling constraints (11) and (13), together with an empty graph $G(x)$, or we end up with some fractional solution x such that the degree of each vertex in $G(x)$ is at most 2 (even exactly 2). This means that at most two fractional jobs are assigned to any machine, and each fractional job is assigned to at most two machines. If that happens, we continue with with a rounding procedure that is akin to the dependent rounding that was proposed by Gandhi et al. [3]. Let us therefore call the following rounding stages *late* stages, and the previous ones *early* stages.

In a late stage, the maximum vertex degree in $G(x)$ is 2. Moreover, since $G(x)$ is bipartite, we can partition $G(x)$ into two matchings M_1 and M_2. Thus we can define a new fractional solution

$$\bar{x}_{ijs} = \begin{cases} x_{ijs} & \text{for } x_{ijs} \text{ integral,} \\ x_{ijs} + \delta & \text{for } (i,j,s) \in M_1, \\ x_{ijs} - \delta & \text{for } (i,j,s) \in M_2, \end{cases}$$

for some δ. Again, since $\sum_{j\in V}\sum_{i\in M}\sum_{s\in S_{ij}} \bar{x}_{ijs}c_{ijs}$ is a linear function of δ we obtain that constraint (13) is satisfied either for positive or for negative δ. Therefore, by choosing δ either maximal or minimal such that $0 \le \bar{x}_{ijs} \le 1$ and such that constraint (13) is still satisfied, we obtain a new solution with at least one more integral variable, still satisfying constraint (13). Moreover, since the two edges incident to any vertex $v \in V$ must belong to different matchings M_1 and M_2, the assignment constraint (11) remains valid, too. Notice that the resulting graph $G(\bar{x})$ still has vertex degrees at most 2, since only edges are dropped due to the rounding. Hence, we can iterate the rounding until all variables are integral.

At the end of the rounding algorithm we obtain an integral solution that obviously satisfies constraints (11). Since in every step we have chosen a solution minimizing a linear function corresponding to constraint (13), we obtain that this constraint is satisfied too.

To show that constraints (12) are satisfied for each $i \in M$, we have to show that the left hand side of the original constraint (2) increases by at most $p_{\max} = \max\{p_{ijs} \mid x_{ijs}^{\mathrm{LP}} > 0\}$. We consider the rounding stage when (2) is violated for machine $i \in M$.

On the one hand, this might happen in an early stage when $d_i = 1$. In this case, however, since there is exactly one fractional edge incident to i, we could add at most $p_{\max}$ in any future rounding stages to the total load of machine i. Hence, constraint (12) is fulfilled by machine i.

On the other hand, the violation of the original constraint (2) might happen in a late stage, where all vertices in $G(x)$ have degree at most 2. When $d_i = 1$ we argue as before. So assume $d_i = 2$. Consider machine i together with its two incident edges (i, j, s) and (i, j', s'). Whenever $x_{ijs} + x_{ij's'} \geq 1$ before the rounding, we claim that the total load of machine i increases by at most $p_{\max}$ by any possible further rounding. This because the total remaining increase in the left hand side of (2) for machine i is at most $(1-x_{ijs})p_{ijs}+(1-x_{ij's'})p_{ij's'} \leq p_{\max}$. So assume that $x_{ijs} + x_{ij's'} < 1$. We claim that at most one of the jobs j and j' will finally be assigned to machine i. To see why, consider the stage where one of these variables was rounded to an integer. Recalling that edges (i, j, s) and (i, j', s') must belong to different matchings M_1 and M_2, we may assume that x_{ijs} is rounded up, and $x_{ij's'}$ is rounded down. Clearly, $x_{ijs} + x_{ij's'} < 1$ holds before that rounding stage. Assuming that x_{ijs} is rounded to 1, it must hold that $x_{ij's'} \geq 1 - x_{ijs}$, because otherwise $x_{ij's'}$ would become negative. In other words, $x_{ijs} + x_{ij's'} \geq 1$, a contradiction. Hence, the only way to round one of the variables x_{ijs} or $x_{ij's'}$ to an integer, is to round $x_{ij's'}$ down to 0. Therefore edge $(i, j's')$ disappears, and indeed, at most job j can be assigned to machine i. Clearly, the resulting increase in the left hand side of (2) for machine i is again at most $p_{\max}$. Hence, constraint (12) is fulfilled after the rounding. □

5. An LP-based 4-approximation algorithm

Our approach to obtain a constant factor approximation for the scheduling problem is now the following. We first use the rounded integral solution from Lemma 4 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 (11)–(14) obtained after the rounding of Lemma 4. To that end, in Lemma 4 we let coefficients c_{ijs} be equal to the total resource consumption of job j when assigned to machine i and using s units of additional resources, $c_{ijs} = s\, p_{ijs}$. After the machine assignments and resource allocations are fixed, the jobs are scheduled according to the greedy list scheduling algorithm of Graham [5], in arbitrary order.

Algorithm LP-GREEDY: Let the resource allocations and machine assignments be fixed as determined by the solution of the LP-based rounding procedure. The algorithm iterates over time epochs t, starting at $t = 0$. We do the following until all jobs are scheduled.

- Check if some yet unscheduled job can be started at time t on an idle machine without violating the resource constraint. If yes, schedule the job to start at time t; ties are broken arbitrarily.
- If no job can be scheduled on any of the machines at time t, update t to the next smallest job completion time $t' > t$.

Theorem 1. *Algorithm* LP-GREEDY *is a 4–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 greedy scheduling, can be implemented in polynomial time.

To verify the performance bound, we first need some additional notation. Consider some schedule produced by algorithm LP-Greedy, and denote by C^{LPG} the corresponding makespan. Denote by C^{OPT} the makespan of an optimal schedule. For the schedule of LP-Greedy let C_β 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 $k/2$. Denote by $\beta = C^{\mathrm{LPG}} - C_\beta$ 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 C_β which is not a big job. Due to the definition of C_β such a machine must exist, because otherwise all machines were idle right before C_β, contradicting the definition of the greedy algorithm. Between time 0 and C_β, periods may exist where machine i is idle. Denote by B the total length of busy periods on machine i in interval $[0, C_\beta]$, and by I the total length of idle periods on machine i in interval $[0, C_\beta]$. We then have that

$$C^{\mathrm{LPG}} = B + I + \beta\,. \tag{17}$$

Due to (12), we get that for machine i

$$B \le \sum_{j\in V} \sum_{s\in S_{ij}} x^*_{ijs}\, p_{ijs} \le C^* + p_{\max} \;\le\; 2\,C^* \;. \tag{18}$$

Here, notice that the last inequality follows from constraint (4) in the relaxation, since $p_{\max} \le C^*$.

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

Lemma 5. *We have that*

$$I + \beta \le 2\,C^*\,.$$

Proof. First, observe that the total resource consumption of the schedule is at least $I\,\frac{k}{2} + \beta\,\frac{k}{2}$. This because, on the one hand, all jobs after C_β are big jobs and they require at least $k/2$ resources by definition. On the other hand, during idle periods on machine i between 0 and C_β, at least $k/2$ of the resources must be consumed by the schedule 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 C_β 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 $k\,C^*$ is an upper bound on the total resource consumption of the schedule, due to (13). Hence, we obtain

$$k\,C^* \ge I\,\frac{k}{2} + \beta\,\frac{k}{2}\,.$$

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

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

Proof (of Theorem 1). Use (17) together with (18) and Lemma 5 to obtain

$$C^{\mathrm{LPG}} \;=\; B + I + \beta \;\le\; 2\,C^{*} + 2\,C^{*} \;\le\; 4\,C^{\mathrm{OPT}}\,,$$

where the last inequality follows from Lemma 2. That completes the proof. □

6. An LP-based 3.75-approximation algorithm

We next show how to tune the techniques presented in Section 5 to yield a 3.75–approximation algorithm. To achieve this result, we apply the same rounding as in Lemma 4 to another integer programming relaxation. Moreover, we modify the greedy scheduling algorithm appropriately.

Let $B_{ij} \subseteq S_{ij}$ be the set of breakpoints that lie in the interval $(k/2, k]$, i.e., $B_{ij} = \{s \in S_{ij} \mid k/2 < s \le k\}$. If any two jobs are processed using s resources, where $s \in B_{ij}$, these two jobs cannot be processed in parallel. Then consider the following integer linear program.

$$\sum_{i\in M}\sum_{s\in S_{ij}} x_{ijs} = 1\ , \qquad \forall\, j \in V\,, \tag{19}$$

$$\sum_{j\in V}\sum_{s\in S_{ij}} x_{ijs}\,p_{ijs} \le C\ , \qquad \forall\, i \in M\,, \tag{20}$$

$$\sum_{j\in V}\sum_{i\in M}\left(1.5\sum_{s\in S_{ij}} x_{ijs}\,\frac{s}{k}\,p_{ijs} + 0.25\sum_{s\in B_{ij}} x_{ijs}p_{ijs}\right) \le 1.75C\ , \tag{21}$$

$$x_{ijs} = 0\ , \qquad \text{if } p_{ijs} > C, \tag{22}$$

$$x_{ijs} \in \{0,1\}\ , \qquad \forall\, i, j, s. \tag{23}$$

Lemma 6. *If there is a feasible schedule with makespan C for the unrelated machine scheduling problem with resource dependent processing times, integer linear program* (19)–(23) *has a feasible solution* $(C, \tilde{x})$.

Proof. Fix some feasible schedule with makespan C and let $\tilde{x}$ be the solution corresponding to that schedule, i.e. we have $\tilde{x}_{ijs} = 1$ if job j is processed on machine i and uses s additional resources and $\tilde{x}_{ijs} = 0$ otherwise. To prove the lemma we only have to verify validity of the new total resource constraint (21). For any feasible schedule, two jobs with resource consumption larger then $k/2$ cannot be processed in parallel, so

$$\sum_{j\in V}\sum_{i\in M}\sum_{s\in Bij} \tilde{x}_{ijs}p_{ijs} \le C. \tag{24}$$

Combining (24) with valid inequality (3) we derive inequality (21). □

As before, by binary search on C while using Lemma 6 instead of Lemma 1, we can find a lower bound C^* on the makespan of an optimal solution for the unrelated machine scheduling problem.

Lemma 7. *Let C^* be the lower bound on the makespan of an optimal solution, and let (C^*, x^{LP}) be the corresponding feasible solution of the LP-relaxation of* (19)–(23)*, then we can find a feasible solution $x^* = (x^*_{ijs})$ for the following integer linear program in polynomial time.*

$$\sum_{i\in M}\sum_{s\in S_{ij}} x_{ijs} = 1\ , \qquad \forall\, j \in V, \tag{25}$$

$$\sum_{j\in V}\sum_{s\in S_{ij}} x_{ijs}\, p_{ijs} \le C^* + p_{\max}, \qquad \forall\, i \in M\,, \tag{26}$$

$$\sum_{j\in V}\sum_{i\in M}\left(1.5\sum_{s\in S_{ij}} x_{ijs}\frac{s}{k}p_{ijs} + 0.25\sum_{s\in B_{ij}} x_{ijs}p_{ijs}\right) \le 1.75C^*\ , \tag{27}$$

$$x_{ijs} \in \{0,1\}\ , \qquad \forall\, i,j,s\,, \tag{28}$$

where $p_{\max} = \max\{p_{ijs} \mid x^{\mathrm{LP}}_{ijs} > 0\}$.

Proof. The proof follows from Lemma 4 with

$$c_{ijs} = \begin{cases} \left(\frac{1.5s}{1.75k} + \frac{0.25}{1.75}\right) p_{ijs} & \text{for all } s \in B_{ij}\,, \\ \frac{1.5s}{1.75k}\, p_{ijs} & \text{for all } s \in S_{ij}\setminus B_{ij}\,. \end{cases}$$

□

Now, we are ready to present an algorithm with an improved approximation guarantee. To that end, we first partition the set of jobs J into three groups J_1, J_2, and J_3 according to the amount of resources consumed. Define $J_1 = \{j \mid k/2 < s^* \le k \text{ and } x_{ijs^*} = 1\}$, $J_2 = \{j \mid k/3 < s^* \le k/2 \text{ and } x_{ijs^*} = 1\}$, and $J_3 = \{j \mid s^* \le k/3 \text{ and } x_{ijs^*} = 1\}$.

Algorithm Improved-LP-Greedy: Let the resource allocations and machine assignments be determined by the rounded LP solution as in Lemma 7. The algorithm schedules jobs group by group. In the first phase it schedules jobs from J_1 one after another (they cannot be processed in parallel since they consume too much resources). Let C_1 be the completion time of the last job from J_1. In the second phase the algorithm schedules jobs from J_2 starting at time C_1. The algorithm always tries to run two jobs from J_2 in parallel. Let C_2 be the first time when the algorithm fails to do so. This could happen either because J_2 is empty or all remaining jobs must be processed on the same machine, say M_1. In the last case the algorithm places all remaining jobs on M_1 without idle time between them. In the third phase the algorithm greedily schedules jobs from J_3, starting no earlier than time C_2. So if some job from J_3 can be started at the current time C_2, we start processing this job. When no jobs can start at the current time we increment the current time to the next job completion time and repeat until all jobs are scheduled. Let C_3 be the completion time of the last job from the set $J_2 \cup J_3$.

We now estimate the makespan C^{LPG} of the schedule. Consider the machine i with the job that finishes last in the schedule. Let B be the total time when machine i is busy and I be the total time when machine i is idle in the interval $[0, C^{\mathrm{LPG}}]$, i.e., $C^{\mathrm{LPG}} = B + I$. By constraint (26) in Lemma 7, we have $B \leq C^* + p_{max} \leq 2C^*$, where the last inequality follows from (22).

To bound the total idle time on machine i we consider two cases. We first consider the case where the job that finishes last belongs to J_1 or J_2. If the last job is from J_1, intervals $[C_1, C_2]$ and $[C_2, C_3]$ have length 0. If the last job is from J_2, there is no idle time on machine i in the interval $[C_2, C_3]$. Thus in both cases, there is no idle time on machine i in the interval $[C_2, C_3]$. Let I_1 be the total idle time on machine i during $[0, C_1]$ and I_2 be the total idle time during $[C_1, C_2]$. Then $I = I_1 + I_2$ is the total idle time on machine i. Since we process one job at a time from J_1 during the time interval $[0, C_1]$ and two jobs at a time from J_2 during $[C_1, C_2]$, the total resource consumption of the schedule during idle times on machine i is at least

$$I_1 \frac{k}{2} + I_2 \frac{2k}{3} .$$

Letting $R_I := I_1/2 + 2I_2/3$, the total resource consumption of the schedule during idle times on machine i is at least $R_I k$, and we therefore get

$$\begin{aligned} I \;=\; I_1 \;+\; I_2 \;=\; 1.5\, R_I \;+\; 0.25\, I_1 & \\ \leq \sum_{j \in V} \sum_{i \in M} \left(1.5 \sum_{s \in S_{ij}} x^*_{ijs} \frac{s}{k} p_{ijs} + 0.25 \sum_{s \in B_{ij}} x^*_{ijs} p_{ijs} \right) & \\ \leq 1.75 C^* . & \qquad (29) \end{aligned}$$

Here, the first inequality holds since $\sum_{j \in V} \sum_{i \in M} \sum_{s \in S_{ij}} x^*_{ijs} \frac{s}{k} p_{ijs}$ equals the total resource consumption of the schedule divided by k, and since $I_1 \leq C_1 = \sum_{j \in V} \sum_{i \in M} \sum_{s \in B_{ij}} x^*_{ijs} p_{ijs}$. The second inequality follows from (27).

Similarly, if the last job in the schedule belongs to J_3, let I_1 be the total idle time on machine i during $[0, C_1]$, I_2 be the total idle time on machine i during $[C_1, C_2]$ and I_3 be the total idle time on machine i during $[C_2, C_3]$. Then $I = I_1 + I_2 + I_3$ is the total idle time on machine i. Again, we process one job at a time from J_1 during the time interval $[0, C_1]$, and two jobs at a time from J_2 during $[C_1, C_2]$. Moreover, due to the resource constraint the last job –which is from J_3– could not be scheduled at idle times on machine i during $[C_2, C_3]$, so the total resource consumption of the schedule during idle times on machine i in $[C_2, C_3]$ is at least $2/3\, k$. Hence, the total resource consumption of the schedule during idle times on machine i is at least

$$I_1 \frac{k}{2} + (I_2 + I_3)\, \frac{2k}{3} .$$

Again, letting $R_I := I_1/2 + 2(I_2 + I_3)/3$, the total resource consumption of the schedule during idle times on machine i is at least $R_I k$, and we get

$$I \;=\; I_1 \;+\; (I_2 + I_3) \;=\; 1.5\, R_I \;+\; 0.25\, I_1\,.$$

Exactly as before in (29) we conclude that $I \leq 1.75 C^*$. Therefore, in either of the two cases we have $C^{\mathrm{LPG}} = B + I \leq 2\; C^* + 1.75\; C^* = 3.75\; C^*$, and we have proved the following theorem.

Theorem 2. *Algorithm* Improved-LP-Greedy *is a* 3.75*–approximation algorithm for unrelated parallel machine scheduling with resource dependent processing times.*

7. Lower bounds and special cases

In identical parallel machine scheduling, the processing time of a job does not depend on the machine it is processed on. It is obviously a special case of the unrelated parallel machine model considered in Section 5. Moreover, in dedicated machine scheduling [16,17], the jobs are assigned to machines *beforehand*. By letting all but one machine assignment result in very large processing times, this is a special case of the unrelated parallel machine model as well. Therefore, Theorems 1 and 2 yield the same results for these two special cases.

Corollary 1. *Algorithm* LP-Greedy *is a* 4*-approximation algorithm, and Algorithm* Improved-LP-Greedy *is a* 3.75*-approximation algorithm for identical or dedicated parallel machine scheduling with resource dependent processing times.*

Lower bounds on approximation. The problem with unrelated machines does not allow for an approximation algorithm with performance guarantee smaller than 3/2 (unless P=NP), as it generalizes the classical unrelated machine scheduling problem [20]. We next show that the same inapproximability threshold exists for the problems with identical or dedicated parallel machines.

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

Proof. The proof relies on a gap-reduction from Partition [4]: 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 identical or dedicated). 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 functions p_{js} can be encoded in $\mathrm{O}(\log a_j)$, for all jobs j. Therefore 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 identical or dedicated parallel machines, even if the resource consumption of any job j is fixed beforehand to a_j (with a processing time of 1).

Corollary 2. *There is no polynomial time approximation algorithm for identical or dedicated parallel machine scheduling with an additional* (*renewable*) *resource constraint that has a performance guarantee smaller 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 [17], and a PTAS if the number of machines m is fixed and the resource is binary (i.e., $k = 1$) [16].

Lower bound for the (*integer*) *linear program.* We next give an instance to show that the integer linear programs that we use can be a factor $(2-\varepsilon)$, respectively $(1.75-\varepsilon)$ away from the optimal solution. Hence, our LP-based analysis cannot yield anything better than that. This even holds for all three versions of the problem, the unrelated, identical, and dedicated machine setting.

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

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

for both machines i and both jobs j. □

Consider the integer solution x for the integer linear program (1)–(5), where $x_{11s} = x_{22s} = 1$ for $s = \lceil k/2 \rceil$, and $x_{ijs} = 0$ otherwise. Clearly, constraints (1) and (5) are satisfied. Moreover, the machine inequalities (2) yield $k \le C$, and inequality (3) yields $k(k+1) = 2k\lceil k/2 \rceil \le kC$. Thus define $C = (k+1)$, and these two constraints are satisfied as well. With $C = (k+1)$, constraint (4) is fulfilled too. Hence, with $C = k+1$ there exists a feasible solution for the integer linear program (1)–(5). A fortiori, we know that for the corresponding *linear* programming relaxation, $C^* \le k+1$. But in the optimal solution of the scheduling problem, $C^{\mathrm{OPT}} = 2k$ by letting the two jobs be processed sequentially, each with $s > k/2$ units of the resource. Hence, the ratio between C^{OPT} and C^* can be as large as $(2-\varepsilon)$, for any $\varepsilon > 0$. With the same reasoning (and the same instance) one can verify that the ratio between C^{OPT} and C^* can be as large as $(1.75-\varepsilon)$ for the stronger integer linear programming relaxation (19)–(23).

Observation 1 *There are instances where the lower bound provided by the integer linear programming relaxation* (1)–(5) *has a feasible solution that is a factor* $(2-\varepsilon)$ *away from the optimum, for any* $\varepsilon > 0$*. The same holds for integer linear program* (19)–(23) *with an optimality gap of* $(1.75-\varepsilon)$*, for any* $\varepsilon > 0$*.*

The bad quality of the lower bounds provided by the relaxations is obviously a consequence of the fact that we only use an aggregate formulation of the resource constraints, whereas any schedule has to respect the resource constraints at any time.

8. Concluding Remarks

Our results use a class of formulations that only use one aggregate resource constraint. An example shows that this leads to an optimality gap of 2 (1.75, respectively). However, we have not been able to prove tightness of our analysis. Hence it can be conjectured that even for the class of formulations we use, stronger approximation results might be obtained by using other scheduling algorithms and a more sophisticated analysis. Stronger results might also be obtainable using other formulations of the problem that better reflect the dynamics of a renewable resource constraint.

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 the lower bound 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. Alexander Grigoriev was partially supported by the Netherlands Organization for Scientific Research (NWO), within the project “Treewidth and Combinatorial Optimization” (TACO). Maxim Sviridenko was partially supported by METEOR, the Maastricht Research School of Economics of Technology and Organizations. We thank Hans Kellerer [15] for sharing with us several nice ideas, and we thank the anonymous referees for several helpful remarks.

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. R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan, Dependent Rounding in Bipartite Graphs, in Proc. 43rd Annual IEEE Symp. on Foundations of Computer Science, 2002, 323–332.

4. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completenes*, W. H. Freeman, New York, 1979.
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: Proc. 6th Workshop on Models and Algorithms for Planning and Scheduling Problems, 2003, 131–132.
9. A. Grigoriev, M. Sviridenko and M. Uetz, Unrelated Parallel Machine Scheduling with Resource Dependent Processing Times, in: Integer Programming and Combinatorial Optimization, M. Jünger and V. Kaibel (eds.), *Lecture Notes in Computer Science* 3509, Springer, 2005, 182–195.
10. A. Grigoriev, M. Sviridenko and M. Uetz, LP Rounding and an Almost Harmonic Algorithm for Scheduling with Resource Dependent Processing Times, in: Approximation, Randomization and Combinatorial Optimization, J. Diaz, K. Jansen, J. D. P. Rolim, and U. Zwick (eds.), *Lecture Notes in Computer Science* 4110, Springer, 2006, 140–151.
11. A. Grigoriev and M. Uetz, Scheduling Parallel Jobs with Linear Speedup, in: Approximation and Online Algorithms, T. Erlebach and P. Persiano (eds.), *Lecture Notes in Computer Science* 3879, Springer, 2006, 203–215.
12. 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.
13. K. Jansen, Scheduling Malleable Parallel Tasks: An Asymptotic Fully Polynomial Time Approximation Scheme, *Algorithmica* **39** (2004), pp. 59-81.
14. K. Jansen and M. Mastrolilli, Approximation schemes for parallel machine scheduling problems with controllable processing times, *Computers and Operations Research* **31** (2004), 1565–1581.
15. H. Kellerer, *Personal Communication*, 2005.
16. 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.
17. H. Kellerer and V. A. Strusevich, Scheduling problems for parallel dedicated machines under multiple resource constraints, *Discrete Applied Mathematics* **133** (2004), 45–68.
18. J. E. Kelley and M. R. Walker, *Critical path planning and scheduling: An introduction*, Mauchly Associates, Ambler (PA), 1959.
19. V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, Approximation Algorithms for Scheduling on Multiple Machines, Proc. 46th Annual IEEE Symp. on Foundations of Computer Science, 2005, 254–263.
20. J. K. Lenstra, D. B. Shmoys and É. Tardos, Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming*, Series A **46** (1990), 259–271.
21. G. Mounie, C. Rapine, and D. Trystram, Efficient Approximation Algorithms for Scheduling Malleable Tasks, Proc. 11th ACM Symp. on Parallel Algorithms and Architectures, 1999, 23–32.
22. G. Mounie, C. Rapine, and D. Trystram, A 3/2-Dual Approximation Algorithm for Scheduling Independent Monotonic Malleable Tasks, Manuscript, Retrieved from `http://citeseer.csail.mit.edu/558879.html`
23. D. B. Shmoys and É. Tardos, An approximation algorithm for the generalized assignment problem, *Mathematical Programming*, Series A **62** (1993), 461–474.
24. M. Skutella, Approximation algorithms for the discrete time-cost tradeoff problem, *Mathematics of Operations Research* **23** (1998), pp. 909–929.
25. J. Turek, J. L. Wolf, and P. S. Yu, Approximate Algorithms for Scheduling Parallelizable Tasks, Proc. 4th ACM Symp. on Parallel Algorithms and Architectures, 1992, 323–332.