# Scheduling Parallel Jobs with Linear Speedup

Alexander Grigoriev and Marc Uetz

Maastricht University, Quantitative Economics, P.O.Box 616, 6200 MD Maastricht, The Netherlands.
Email: {a.grigoriev, m.uetz}@ke.unimaas.nl

**Abstract** We consider a scheduling problem where a set of jobs is distributed over parallel machines. The processing time of any job is dependent on the usage of a scarce renewable resource, e.g. personnel. An amount of $k$ units of that resource can be allocated to the jobs at any time, and the more of that resource is allocated to a job, the smaller its processing time. The dependence of processing times on the amount of resources is linear for any job. The objective is to find a resource allocation and a schedule that minimizes the makespan. Utilizing an integer quadratic programming relaxation, we show how to obtain a $(3+\varepsilon)$-approximation algorithm for that problem, for any $\varepsilon > 0$. This generalizes and improves previous results, respectively. Our approach relies on a fully polynomial time approximation scheme to solve the quadratic programming relaxation. This result is interesting in itself, because the underlying quadratic program is NP-hard to solve in general. We also briefly discuss variants of the problem and derive lower bounds.

## 1 Introduction and related work

Consider a scheduling problem where $n$ jobs $j \in V$, with processing times $p_j$ are distributed over a set of $m$ parallel, identical machines. There is a *renewable* resource, e.g. personnel, that can be allocated to jobs in order to reduce their processing requirements. We assume that the tradeoff between usage of the resource and the resulting processing requirement of a job can be described succinctly by a corresponding *linear compression rate* $b_j \geq 0$. In other words, each job has a default processing time of $\bar{p}_j$, and when $s$ resources are assigned to job $j$, its processing requirement becomes $p_{js} = \bar{p}_j - b_j\, s$. At any point in time, only $k$ units of that resource are available. Once resources have been assigned to the jobs, a schedule is called feasible if it does not consume more than the available $k$ units of the resource, at any time. The goal is to find a resource allocation and a corresponding feasible schedule that minimizes the *makespan*, the completion time of the job that finishes latest. This problem describes a typical situation in production logistics, where additional resources, such as personnel, can be utilized in order to reduce the production cycle time.

As a matter of fact, scheduling problems with a *nonrenewable* resource, such as a total budget constraint, have received quite some attention in the literature as *time-cost-tradeoff* problems, e.g., [2,10,17,18]. Surprisingly, the corresponding problems with a *renewable* resource, such as a personnel constraint, have received much less attention, although they are not less appealing from a practical viewpoint. We will refer to them as *time-resource-tradeoff* problems, in analogy to the former.

**Related work.** In a previous paper [8], we have considered the more general problem of unrelated machine scheduling with resource dependent processing times. There, jobs can be processed on *any* of the machines, and if a job is scheduled on machine $i$, using $s$ of the $k$ available units of the resource, the processing time is $p_{ijs}$. Assuming that processing times are monotone in the resources (and not necessarily linear), the existence of a $(4+2\sqrt{2})$–approximation algorithm is proved in [8]. The same paper contains a $(3+2\sqrt{2})$–approximation algorithm for the special case where the jobs are distributed over the machines beforehand. The approach presented in [8] is based upon a linear programming relaxation that essentially uses $nk$ variables. The problem

with linear resource-time tradeoff functions, however, can be encoded more succinctly by $O(n)$ numbers only; for each job, we need to specify its machine $i$, the maximum processing time $\bar{p}_j$, and the compression rate $b_j$, respectively. Therefore, the approach of [8] leads to a pseudo polynomial time $(3+\sqrt{2})$–approximation algorithm for the problem at hand.

In a manuscript by Grigoriev et al. [7], a restricted version of the problem at hand is addressed. Their model is restricted to a binary resource, thus the availability of the additional resource is $k = 1$. Any job may be processed either with or without using that resource, with 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, and for the problem with $m = 2$ machines, they derive (weak) NP-hardness and a fully polynomial time approximation scheme [7].

Jobs with resource dependent processing are also known as *malleable* or *parallelizable tasks*, e.g. in [14,19]. 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. [19] derive a 2–approximation algorithm for this problem. In fact, the model considered in [19] closely relates to, but also differs from the problem considered in this paper. Interpreting the parallel processors of [19] as a generic 'resource' that must be allocated to jobs, the problem of [19], when restricted to linear resource-time tradeoff functions $p_{js}$, is a special case of the problem considered in this paper: It corresponds to the case where $n$ jobs are processed on $m = n$ machines, instead of $m < n$ machines. Mounie et al. [14] consider yet another restriction of the problem of [19], in that the processor allocations must be contiguous and the 'total work functions' $sp_{js}$ are non-decreasing in $s$. For that problem, a $\sqrt{3}$–approximation is derived [14].

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 the assumption that jobs are distributed over the machines beforehand have been discussed by Kellerer and Strusevich [11,12]. They use the term *dedicated machine scheduling*. We refer to these papers for various complexity results, and note that NP-hardness of dedicated machine scheduling and a binary resource was established in [11]. 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 [11].

**Results and methodology.** We derive a $(3+\varepsilon)$-approximation algorithm for scheduling parallel jobs with linear speedup. Our result holds for an arbitrary number $m$ of machines and an arbitrary number $k$ of available resources. In that sense, our result generalizes the previous $(3+\varepsilon)$-approximation of [7] to an arbitrary number of machines, and arbitrary, linear resource dependent processing times (recall that they consider the special case $k = 1$, which may be interpreted as linear resource-time functions, too). Although we obtain the same performance bound, we stress that our result relies on a completely different approach. Moreover, restricted to linear resource-time functions, our result considerably improves upon the $(3+\sqrt{2})$–approximation from [8]. In addition, our algorithm is indeed a strongly polynomial time algorithm, while the result of [8] only yields a pseudo polynomial time algorithm.

We obtain this result by using a constrained quadratic programming formulation that constitutes a relaxation of the problem. More precisely, the mathematical program is an integer, concave minimization problem with linear constraints. Although such problems are NP-hard to solve in general [15,9], even without integrality constraints, we can show how to solve this quadratic programming relaxation with arbitrary precision in polynomial time; a result of interest in its own. Based on the solution of this mathematical program, we assign resources to the jobs. Finally, the jobs are scheduled using Graham's greedy scheduling algorithm. Making use of

the lower bound provided by the quadratic programming relaxation, we derive the performance guarantee of $(3+\varepsilon)$.

Finally, we provide a parametric example to show that our analysis cannot be improved further that a factor of 2, by showing that the allocation of resources that is computed with the quadratic program can indeed provide the 'wrong' answer, such that the greedy scheduling algorithm can provide a solution a factor $2-\varepsilon$ away from the optimum, for any $\varepsilon > 0$. The same example even shows that *any* scheduling algorithm, based on the resource allocation as suggested by the quadratic program, will be a factor 1.46 away from the optimum.

## 2 Problem definition

Let $V = \{1, \ldots, n\}$ be a set of jobs. Jobs must be processed non-preemptively on a set of $m$ 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. Each job $j$ is assigned to a given machine $i$, and $V_i$ denotes the set of jobs assigned to machine $i$, such that $V = \cup_i V_i$ forms a partition of the jobs. During its processing, a job $j$ may be assigned an amount $s \in \{0, 1, \ldots, k\}$ of an additional resource, for instance personnel, that may speed up its processing. If $s$ resources are allocated to a job $j$, the processing time of that job is $p_{js}$, $s = 0, \ldots, k$. The restriction is that in a feasible schedule, at any time not more that $k$ resources may be used.

We assume that the resource dependent processing time $p_{js}$ of any job can be encoded succinctly by the default processing time, $\bar{p}_j$, together with the linear compression rate $b_j$, such that the actual processing time becomes

$$p_{js} = \bar{p}_j - b_j\, s\,,$$

given that $s \in \{0, \ldots, k\}$ resources are assigned to job $j$, $j \in V$. Hence, the encoding of the problem just contains the $O(\,n\,)$ numbers $\bar{p}_j$ and $b_j$, as well as the given assignment of the $n$ jobs to the $m$ machines.

## 3 Quadratic programming relaxation

The approach of [8] could be used to obtain a $(3 + 2\sqrt{2})$–approximation algorithm for the problem at hand. The approach, however, is explicitly based upon an integer linear programming formulation that would require $\Theta(\,nk\,)$ variables to represent all the different processing times of jobs $p_{js}$. Obviously, this would only lead to a pseudo polynomial time algorithm for the problem at hand.

For the linear case considered in this paper, however, we can set up a polynomial size, quadratic formulation, using $O(\,n\,)$ variables $s_j \in \{0, \ldots, k\}$ that denote the number of resources allocated to job $j$. Then $p_{js} = \bar{p}_j - b_j s_j$ is the processing time of a job $j$. We also assume that $b_j$ is integral for all jobs $j$, and since any job is required to use an integral amount of resources, $p_{js}$ is integral, too. To exclude trivial solutions, we assume that $\bar{p}_j > b_j k$.

The following integer quadratic program has a solution if there is a feasible schedule with makespan $C$.

$$\sum_{j\in V_i} \bar{p}_j - b_j s_j \leq C \ , \qquad \forall\ i = 1,\dots,m\,, \tag{1}$$

$$\sum_{j\in V} \bar{p}_j s_j - b_j s_j^2 \leq k\,C \ , \tag{2}$$

$$0 \leq s_j \leq k, \qquad \forall j \in V\,, \tag{3}$$

$$s_j \in \mathbb{Z}^+, \qquad \forall j \in V\,. \tag{4}$$

The logic behind this program is the following; (1) states that the total processing on each machine is a lower bound for the makespan, and (2) states that the total resource consumption of the schedule cannot exceed the maximum value of $k\,C$. Our goal is to compute an integer feasible solution $(C^*, s^*)$ for program (1)–(4), such that $C^*$ is a lower bound for the makespan $C^{\mathrm{OPT}}$ of an optimal schedule. A candidate for $C^*$ is the smallest integer value, say $C^{\mathrm{QP}}$, for which this program is feasible. But since we do not know how to compute $C^{\mathrm{QP}}$ exactly, we will compute an approximation $C^* \leq C^{\mathrm{QP}}$.

In order to decide on feasibility for program (1)–(4), notice that we may as well solve the following constrained integer quadratic minimization problem.

$$\text{min.} \quad \sum_{j\in V} \bar{p}_j s_j - b_j s_j^2 \ , \tag{5}$$

$$\text{s. t.} \quad \sum_{j\in V_i} \bar{p}_j - b_j s_j \leq C \ , \qquad \forall\ i = 1,\dots,m\,, \tag{6}$$

$$0 \leq s_j \leq k \ , \qquad \forall j \in V \tag{7}$$

$$s_j \in \mathbb{Z}^+, \qquad \forall j \in V\,. \tag{8}$$

Obviously, (1)–(4) is feasible if and only if the constrained quadratic minimization problem (5)–(8) has a solution at most $k\,C$. It is well known that constrained quadratic programming is NP-hard in general [15], even without integrality constraints. More specifically, we have a constrained concave minimization problem, which is generally known to be NP-hard as well [9]. However, we next show that the integer quadratic program (5)–(8) can be solved with arbitrary precision in polynomial time.

**Lemma 1.** *For any $0 < \delta < 1$, we can find a solution for the constrained quadratic minimization problem (5)–(8) that is not more than a factor $(1+\delta)$ away from the optimal solution, in time polynomial in the input size and $1/\delta$.*

In other words, (5)–(8) admits an FPTAS, a fully polynomial time approximation scheme. The proof of this lemma is of interest in its own. We first show how to reduce the constrained quadratic program to a certain single machine scheduling problem, and then show that this scheduling problem admits an FPTAS, using the framework of Pruhs and Woeginger [16].

*Proof (of Lemma 1).* First observe that (5)–(8) decomposes into $m$ independent, constrained quadratic programs, one for each machine $i$:

$$\text{min.} \quad \sum_{j \in V_i} \bar{p}_j s_j - b_j s_j^2 \ , \tag{9}$$

$$\text{s. t.} \quad \sum_{j \in V_i} \bar{p}_j - b_j s_j \leq C \ , \tag{10}$$

$$0 \leq s_j \leq k \ , \qquad \forall j \in V_i \, , \tag{11}$$

$$s_j \in \mathbb{Z}^+, \qquad \forall j \in V_i \, . \tag{12}$$

We now consider an even more restrictive problem, where instead of constraints (11)-(12), we restrict the resource consumptions $s_j, j \in V_i$, to rounded powers of $(1+\varepsilon_1)$. More precisely, we set

$$\mathcal{E} = \{0, k\} \cup \{\lceil (1+\varepsilon_1)^\ell \rceil : \ 0 \leq (1+\varepsilon_1)^\ell \leq k, \ \ell \in \mathbb{Z}^+\} \, ,$$

where $0 < \varepsilon_1 < 1$ is to be defined later. We claim that if in program (9)-(12) there exists a solution $s$ of value $X$, then in this even more restricted program there exists a solution $s'$ of value $X'$ such that $X' \leq (1+3\varepsilon_1)X$ and $s'_j \in \mathcal{E}$ for all $j \in V_i$. To see this, we consider a solution $s$ with objective value $X$. We define a new solution $s'$ by simply rounding up the values $s_j, j \in V_i$, to the nearest integer number in $\mathcal{E}$. This way all resource consumptions are rounded up, and we have that $s_j \leq s'_j$ for all $j \in V_i$, thus constraint (10) is satisfied by $s'$, too. Therefore, the obtained solution $s'$ is an integer feasible solution for program (9)-(12) with $s'_j \in \mathcal{E}$ for all $j \in V_i$.

Now consider an arbitrary $j \in V_i$ and the corresponding $\ell \in \mathbb{Z}^+$ such that $(1+\varepsilon_1)^{\ell-1} \leq s_j < (1+\varepsilon_1)^\ell$. Since $s_j$ is integer, we have that $\lceil (1+\varepsilon_1)^{\ell-1} \rceil \leq s_j < \lceil (1+\varepsilon_1)^\ell \rceil = s'_j < (1+\varepsilon_1)^\ell + 1$. Now, if $(1+\varepsilon_1)^\ell + 1 \leq (1+\varepsilon_1)^{\ell+1}$ we immediately derive that $s'_j < (1+\varepsilon_1)^2 s_j < (1+3\varepsilon_1)s_j$. If $(1+\varepsilon_1)^\ell + 1 > (1+\varepsilon_1)^{\ell+1}$, this implies that $(1+\varepsilon_1)^{\ell-1} + 1 > (1+\varepsilon_1)^\ell$, and thus $s_j = s'_j = \lceil (1+\varepsilon_1)^{\ell-1} \rceil$. Therefore, $s'_j \leq (1+3\varepsilon_1)s_j$, for all $j \in V_i$. Consequently, for the objective $X'$ we have

$$X' = \sum_{j \in V_i} s'_j(\bar{p}_j - b_j s'_j) \leq \sum_{j \in V_i} (1+3\varepsilon_1) s_j (\bar{p}_j - b_j s_j) = (1+3\varepsilon_1)X \, ,$$

as claimed before.

We next claim that the problem (9)-(12) restricted to $s_j \in \mathcal{E}, j \in V_i$, admits an FPTAS. To this end, observe that this problem is in fact a single machine scheduling problem where each job has at most $h \in O(\log_{1+\varepsilon_1} k)$ possible different processing times $\bar{p}_j - b_j s_j$ with associated costs $\bar{p}_j s_j - b_j s_j^2$, where $s_j \in \mathcal{E}$. Problem (9)-(12) thus asks for a schedule with makespan at most $C$ and minimal total cost. The proof that this problem admits an FPTAS, in terms of its input size, is presented below in Lemma 2. This input size consists of not more than $O(\log_{1+\varepsilon_1} k)$ possible processing times and costs, hence it is polynomially bounded in terms of $1/\varepsilon_1$ and the original problem size. As a consequence, we have that for any $0 < \varepsilon_1 < 1$ and for any $\varepsilon_2 > 0$ we can compute in time polynomial in the original input size, $1/\varepsilon_1$, and $1/\varepsilon_2$, a solution that is no more than a factor of $(1+3\varepsilon_1)(1+\varepsilon_2)$ away from the optimal solution. Letting $\varepsilon_1 = \delta/6$ and $\varepsilon_2 = \delta/3$, we derive $(1+3\varepsilon_1)(1+\varepsilon_2) \leq (1+\delta)$, finishing the proof. □

**Lemma 2.** *Consider a single machine scheduling problem where we have a due date $C$, and $n$ jobs, each having $h$ possible modes $s$ at which its processing time is $p_{js}$ and its cost is $w_{js}$, $s = 1, \ldots, h$. The problem is to find a mode $s$ for each job with $\sum_j p_{js} \leq C$, such that the total cost $\sum_j w_{js}$ is minimized. This problem admits an FPTAS.*

*Proof.* Utilizing the framework of Pruhs and Woeginger [16], it suffices to show that the problem admits an algorithm that solves the problem to optimality, with a computation time that is polynomially bounded in terms of $nh$, $W = \sum_{j,s} w_{js}$, and the input size of the problem. Then Theorem 1 of [16] yields that the problem admits an FPTAS.

The following dynamic program does the job. For $q = 1, \dots, n$ and $z = 0 \dots, W$, denote by $P[q, z]$ the smallest total processing time of $q$ jobs such that their total weight equals $z$. More precisely, $P[q, z]$ is the smallest number such that there exists a subset $Q$ of $q$ jobs with processing times $p_{js}$ and costs $w_{js}$, such that $\sum_{j \in Q} p_{js} = P[q, z]$ and $\sum_{j \in Q} w_{js} = z$. The initialization of $P[1, z]$ is trivial for any value $z = 0 \dots, W$, and

$$P[q+1, z] = \min\{P[q; z - w] + p \mid (p, w) = (p_{js}, w_{js}) \text{ for some } j \text{ and } s\}\,.$$

Once we completed this dynamic programming table, we find the optimum value as

$$\max\{z \mid P[n, z] \le C\}\,.$$

The total time required to run this dynamic program is polynomially bounded in $nh$, $W = \sum_{j,s} w_{js}$, and the input size of the problem. □

Now, coming back to the original problem, we can use the FPTAS of Lemma 1 in order to obtain an approximation of the smallest integer value $C^{\mathrm{QP}}$ for which (1)–(4) has a feasible solution. This is achieved as follows. For fixed $\delta > 0$, we find by binary search the smallest integer value $C$ for which the FPTAS of Lemma 1 yields a solution for (5)–(8) with value

$$z_C > (1 + \delta)\, kC\,. \tag{13}$$

Then we know by Lemma 1 that the optimal solution for (5)–(8) is larger than $kC$, and hence (1)–(4) is infeasible for $C$. Since $C$ is chosen as the smallest value with the property (13), on input $C^* = C + 1$, the FPTAS yields a solution for (5)–(8) with value $z_{C^*} \le (1+\delta)\, kC^*$. Now, we have that $C^* \le C^{\mathrm{QP}}$, since (1)–(4) is infeasible for $C^* - 1$, and $C^{\mathrm{QP}}$ was defined as the smallest integer value for which (1)–(4) has a feasible solution. Hence, $C^*$ is a lower bound on $C^{\mathrm{OPT}}$, the makespan of an optimal solution. Moreover, using the FPTAS of Lemma 1, we have an integral solution $(s_1^*, \dots, s_n^*)$ that is feasible for (1)–(4) with constraint (2) relaxed to

$$\sum_{j \in V} \bar{p}_j s_j - b_j s_j^2 \le (1 + \delta)\, k\, C^* \ . \tag{14}$$

Therefore, we conclude that we can derive an approximate solution for (1)-(4) in the following sense.

**Lemma 3.** *For any $\delta > 0$, we can find in polynomial time an integer value $C^*$ such that $C^* \le C^{OPT}$, and an integer solution $s^* = (s_1^*, \dots, s_n^*)$ for the resource consumptions of jobs such that*

$$\sum_{j \in V_i} \bar{p}_j - b_j s_j^* \le C^* \ , \qquad i = 1, \dots, m\,, \tag{15}$$

$$\sum_{j \in V} \bar{p}_j s_j^* - b_j (s_j^*)^2 \le (1 + \delta)\ kC^*\,. \tag{16}$$

## 4 QP based greedy algorithm

Our approach to obtain a constant factor approximation for the scheduling problem is now the following. We first use the solution for the quadratic programming relaxation from the previous section in order to decide on the amount of resources allocated to every individual job $j$. More precisely, job $j$ must be processed using $s_j^*$ additional resources. Then the jobs are scheduled according to the greedy list scheduling algorithm of Graham [4], in arbitrary order.

> **Algorithm** QP-Greedy: With the resource allocations as determined by the solution to the quadratic program QP, 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 given machine assignments and the resource constraint.

Now that we have allocated resources to jobs according to the above obtained solution to the quadratic program, we claim the following.

**Theorem 1.** *For any $\varepsilon > 0$, there exists an algorithm* QP-Greedy *that is a $(3+\varepsilon)$–approximation algorithm for scheduling parallel jobs with linear speedup. The computation time of the algorithm is polynomial in the input size and the precision $1/\varepsilon$.*

Note that the result of Theorem 1 improves considerably on the performance bound of $(3+2\sqrt{2})$ from [8] for the more general case of nonlinear resource-time tradeoff functions. Moreover, also recall that the approach of [8] only yields a pseudo polynomial time algorithm for the linear problem at hand.

*Proof.* In order to do the binary search for the integer value $C^*$ in the quadratic programming relaxation (1)–(4), we first use the FPTAS of Lemma 1, with $\delta = \varepsilon/2$. As described previously, this yields a lower bound $C^*$ on the makespan $C^{\mathrm{OPT}}$ of an optimal schedule, together with an integer solution $s^*$ for (1),(3),(4), and (14). We then fix the assignments of resources to the jobs as suggested by the solution $s^*$, and apply the greedy algorithm. The analysis of the greedy algorithm itself is based on the same basic idea as in our previous paper [8]. For convenience, we present the complete proof here.

Consider some schedule $\mathcal{S}$ produced by algorithm QP-Greedy, and denote by $C^{\mathrm{QPG}}$ the corresponding makespan. Denote by $C^{\mathrm{OPT}}$ the makespan of an optimal solution. For schedule $\mathcal{S}$, let $t(\beta)$ be 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 $\beta = C^{\mathrm{QPG}} - t(\beta)$ be the length of the period in which only big jobs are processed (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 $\alpha$ the total length of busy periods on machine $i$ between 0 and $t(\beta)$, and by $\gamma$ the total length of idle periods on machine $i$ between 0 and $t(\beta)$. We then have that

$$C^{\mathrm{QPG}} = \alpha + \beta + \gamma\,. \tag{17}$$

Due to (15), we get that for machine $i$

$$\alpha \leq \sum_{j \in V_i} \bar{p}_j - b_j s_j^* \leq C^* \ . \tag{18}$$

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 $i$. We claim that

$$\beta+\gamma \leq 2(1+\delta)\, C^* \,. \tag{19}$$

To see this, 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 $(1+\delta)\,kC^*$ is an upper bound on the total resource consumption of the jobs, due to (16). Hence, we obtain

$$(1+\delta)kC^* \geq \beta\,\frac{k}{2}+\gamma\,\frac{k}{2}\,.$$

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

Now we are ready to prove the performance bound of Theorem 1. First, use (17) together with (18) and (19) to obtain

$$C^{\mathrm{QPG}} \;\leq C^* + 2(1+\delta)C^* \;=\; (3+2\delta)\, C^* \,.$$

Eventually, because $C^*$ is a lower bound on $C^{\mathrm{OPT}}$, this yields a performance bound for QP-Greedy of $3+2\delta = 3+\varepsilon$, due to the choice of $\delta=\varepsilon/2$.

The claim on the polynomial computation time follows from the fact that we use an FPTAS in Lemma 1, and since the greedy algorithm obviously runs in polynomial time. □

## 5 Lower bounds

Concerning lower bounds on approximation, we know that the problem at hand is a generalization of the dedicated machine scheduling problem as considered by Kellerer and Strusevich [11], hence it follows that it is strongly NP-hard. Unlike for the nonlinear problem, where an inapproximability result of 3/2 is known [8], we did not succeed to derive a stronger negative result without further generalizing the problem. See Section 6 for a brief discussion of this issue. We next show, however, that our approach may yield a solution that is a factor $2-\varepsilon$ away from the optimal solution, for any $\varepsilon>0$.

*Example 1.* Consider an instance with $m=3$ machines and $k=2$ units of the additional resource. Let an integer $\ell$ be fixed. The first two machines are assigned two jobs each, symmetrically. One of these two jobs has a compression rate of 0, thus a constant processing time $p_{js}=\ell-3$, for any $s=0,\ldots,2$. The other job has a processing time $p_{js}=3+2\ell-\ell s$ if assigned $s$ units of the resource, thus the only way to get this job reasonably small is to assign all 2 resources, such that $p_{j2}=3$. On the third machine, we have three jobs. Two identical *short* jobs with processing times $p_{js}=3-s$, and one *long* job with processing time $p_{js}=\ell-3s$, $s=0,\ldots,2$. See Figure 1 for an example. □

**Proposition 1.** *There exists an instance where the assignment of resources to the jobs as proposed by the solution to the quadratic programming relaxation is wrong in the sense that any scheduling algorithm yields a solution that is a factor at least* $19/13\approx 1.46$ *away from the optimum. Moreover, for any* $\varepsilon>0$*, there exist instances where algorithm* QP-Greedy *yields a solution that is a factor* $2-\varepsilon$ *away from the optimum.*

*Proof.* Consider the parametric instance defined in Example 1, with parameter $\ell \geq 13$. The assignment of resources to the jobs on the first two machines is essentially fixed by construction of the instance, for any reasonable makespan (i.e., less than $2\ell$): the two jobs with the high compression rate consume 2 units of the resource, yielding a total processing time of $\ell$ on the first two machines. In the optimal solution, the makespan is exactly $\ell$, by assigning 2 resources to the long job on the third machine, and no resources to the small jobs. The corresponding schedule is depicted in Figure 1(a). The smallest value $C$ such that the quadratic programming relaxation (1)–(4) is feasible is $C = \ell$, too. We claim that our solution to the quadratic programming relaxation would assign one unit of the resource to both, the big and one of the small jobs, and two units of the resource to the remaining small job. This is due to the fact that, in solving the QP, we actually minimize the total resource consumption of the schedule, subject to the constraint that the total processing time on each machine stays below the makespan bound of $C = \ell$. On the third machine, the minimal resource consumption, subject to the condition that the makespan is at most $\ell$ is achieved as explained, yielding a total resource consumption of $\ell + 1$. All other assignments of resources to the jobs on the third machine either violate the makespan bound of $\ell$, or require more resources (in fact, at least $2(\ell - 6) \geq \ell + 1$). Now, it is straightforward to verify that any schedule with this resource assignment will provide a solution that has a makespan of at least $3 + 3 + (\ell - 3) + 1 + 2 = \ell + 6$, since no two resource consuming jobs can be processed in parallel. Figure 1(b) depicts such a schedule. Since $\ell$ would be optimal,

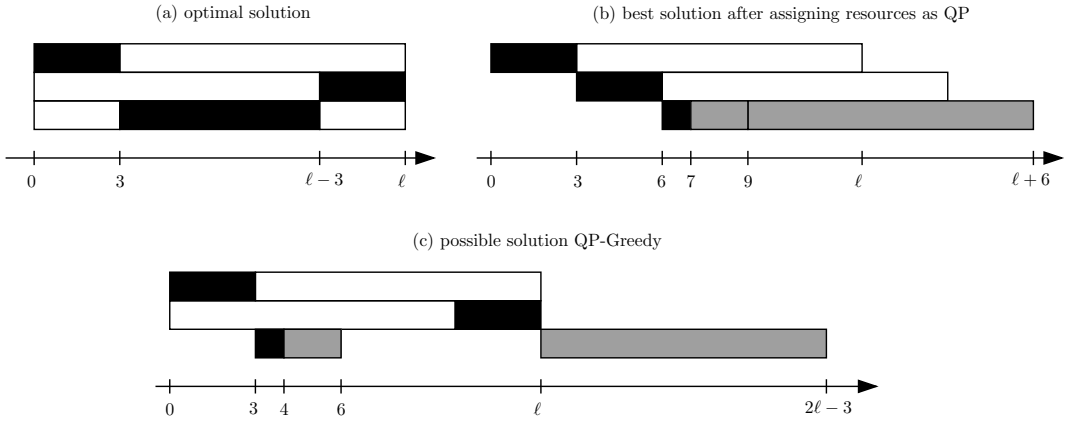


**Figure 1.** Instance Example 1. Black jobs consume 2 resources, gray jobs 1, and white jobs 0 resources.

this yields the claimed ratio of 19/13 when utilizing $\ell = 13$. On the other hand, if the scheduling algorithm fails to compute this particular solution, the makespan becomes $2\ell - 3$, as depicted in Figure 1(c). This yields a ratio of $(2\ell - 3)/\ell$, which is arbitrarily close to 2 for large $\ell$. □

## 6 Conclusions

It remains open whether there exist instances of the problem on which algorithm QP-Greedy outputs a solution with performance ratio worse than 2. Even more interesting, however, would be a lower bound on the approximability for the scheduling problem considered in this paper, since the so far strongest result is NP-hardness [11].

Another interesting generalization of the problem discussed in this paper is obtained when each job has an individual upper bound on the maximal resource consumption, so $p_{js} = \bar{p}_j - b_j s_j$, and $0 \leq s_j \leq k_j$ for each job $j$. The problem discussed in this paper then corresponds to the

special case where $k_j = k$ for all jobs $j$. As a matter of fact, it is not hard to see that our approximation result still holds for that generalized version of the problem. Moreover, this generalized version does not admit an approximation algorithm with a performance ratio better than 3/2, which follows by a simple adaption of the gap-reduction from PARTITION in Theorem 3 of [8]: Given $n$ integers $a_j$ with $\sum_{j=1}^{n} a_j = 2B$, we define for each item $a_j$ one job $j$ with linear function $p_{js} = 2a_j + 1 - 2s_j$, let $s_j \leq k_j = a_j$, and $k = B$. Then there exists a partition if and only if the optimal solution for the scheduling problem has a makespan of 2.

**Acknowledgements.** We thank Gerhard Woeginger for several helpful suggestions. In particular, he pointed us to the paper [16], and proposed the proof for the FPTAS for the single machine scheduling problem in Lemma 2.

## 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), pp. 11–24.
2. Z.-L. Chen, Simultaneous Job Scheduling and Resource Allocation on Parallel Machines, *Annals of Operations Research*, **129** (2004), pp. 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. R. L. Graham, Bounds for certain multiprocessing anomalies, *Bell System Technical Journal*, **45** (1966), pp. 1563–1581. See also [5].
5. R. L. Graham, Bounds on multiprocessing timing anomalies, SIAM *Journal on Applied Mathematics*, **17** (1969), pp. 416–429.
6. 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), pp. 287–326.
7. A. Grigoriev, H. Kellerer, and V. A. Strusevich, Scheduling parallel dedicated machines with the speeding-up resource, manuscript (2003). Extended abstract in: *Proceedings of the 6th Workshop on Models and Algorithms for Planning and Scheduling Problems*, Aussois, France, 2003, pp. 131–132.
8. A. Grigoriev, M. Sviridenko, and M. Uetz, Unrelated Parallel Machine Scheduling with Resource Dependent Processing Times, *Proceedings of the 11th Conference on Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, to appear 2005.
9. R. Horst and P. M. Pardalos, Editors, *Handbook of Global Optimization*, volume 2 of Nonconvex Optimization and Its Applications, Springer, 1995.
10. J. E. Kelley and M. R. Walker, *Critical path planning and scheduling: An introduction*, Mauchly Associates, Ambler (PA), 1959.
11. H. Kellerer and V. A. Strusevich, Scheduling parallel dedicated machines under a single non-shared resource, *European Journal of Operational Research*, **147** (2003), pp. 345–364.
12. H. Kellerer and V. A. Strusevich, Scheduling problems for parallel dedicated machines under multiple resource constraints, *Discrete Applied Mathematics*, **133** (2004), pp. 45–68.
13. J. K. Lenstra, D. B. Shmoys and E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming*, Series A, **46** (1990), pp. 259–271.
14. G. Mounie, C. Rapine, and D. Trystram, Efficient Approximation Algorithms for Scheduling Malleable Tasks, *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1999, pp. 23–32.
15. P. M. Pardalos and G. Schnitger, Checking Local Optimality in Constrained Quadratic Programming is NP-hard, *Operations Research Letters*, **7** (1988), pp. 33–35.
16. K. Pruhs and G. J. Woeginger, Approximation Schemes for a Class of Subset Selection Problems, *Proceedings of the 6th Latin American Symposium on Theoretical Informatics*, Lecture Notes in Computer Science 2976, 2004, pp. 203–211.
17. D. B. Shmoys and E. Tardos, An approximation algorithm for the generalized assignment problem, *Mathematical Programming*, Series A, **62** (1993), pp. 461–474.
18. M. Skutella, Approximation algorithms for the discrete time-cost tradeoff problem, *Mathematics of Operations Research*, **23** (1998), pp. 909–929.
19. 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, pp. 323–332.