

Scheduling Parallel Jobs with Time-Resource Tradeoff via Nonlinear Programming¹

Alexander Grigoriev, Marc Uetz

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

We consider a scheduling problem where the processing time of any job is dependent on the usage of a discrete 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 objective is to find a resource allocation and a schedule that minimizes the makespan. We explicitly allow for succinctly encodable time-resource tradeoff functions, which calls for mathematical programming techniques other than those that have been used before. Utilizing a (nonlinear) integer mathematical program, we obtain the first polynomial time approximation algorithm for the scheduling problem, with performance bound $(3 + \varepsilon)$ for any $\varepsilon > 0$. Our approach relies on a fully polynomial time approximation scheme to solve the mathematical programming relaxation. This result is interesting in itself, because we also show that the underlying mathematical program is NP-hard to solve. We also derive lower bounds for the approximation.

Key words: scheduling; mathematical programming; approximation algorithms; computational complexity

History:

1. Introduction and related work

Consider a scheduling problem where n jobs $j \in V$ need to be processed on a set of m parallel machines. Each job is dedicated to be processed on exactly one machine. There is a *renewable* discrete 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 job j can be described by some non-negative *time-resource tradeoff function* $p_j(\cdot)$ meaning that, when x resources are assigned to job j , its processing requirement becomes $p_j(x)$. At any point in time, only k units of that

¹Preliminary results were published in the Proceedings of WAOA 2005 [9].

resource are available. We may assume without loss of generality that the time-resource tradeoff functions $p_j(\cdot) : \{0, 1, \dots, k\} \rightarrow \mathbb{Z}^+$ are non-increasing positive functions. 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*, that is, 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 a lot of attention in the literature as *time-cost* tradeoff problems, e.g., [2, 12, 13, 22, 23]. 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 [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 non-increasing in the resources, the existence of a 3.75-approximation algorithm is proved in [8]. The approach presented in [8] is based upon a linear programming relaxation that uses nk variables. In this paper, however, we explicitly allow for time-resource tradeoff functions $p_j(\cdot)$ that can be encoded more succinctly. For instance, if these functions are linear, the problem input essentially consists of $O(n)$ numbers only: For each job, we have to specify its machine i and two integer coefficients a_j and \bar{p}_j , such that the time-resource tradeoff functions equal $p_j(x) = \bar{p}_j - a_j x$. Therefore, the algorithms proposed in [8] are generally not polynomial time algorithms.

In a manuscript by Grigoriev et al. [7], a restricted version of the problem at hand is addressed. They assume that the additional resource is binary, that is, 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].

The scheduling of jobs with resource dependent processing times is also known as *malleable* or *parallelizable task* scheduling; see, e.g., [11, 18, 19, 24]. In these models, independent, non-preemptive 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. [24] introduced this problem; they derive a 2-approximation algorithm. In fact, the model considered in [24] closely relates to, but also differs from the problem considered in this paper. Interpreting the parallel processors of [24] as a generic ‘resource’ that must be allocated to jobs, the problem of [24], when restricted to linear time-resource 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. [18] consider yet another restriction of the problem of [24], in that the processor allocations must be contiguous and the ‘total work functions’ sp_{js} are non-decreasing in s . For that problem, a $(\sqrt{3} + \varepsilon)$ -approximation is derived [18]. An unpublished journal version of that paper [19] claims an improved performance bound of $(3/2 + \varepsilon)$. An asymptotic fully polynomial approximation scheme for malleable task scheduling was proposed by Jansen [11].

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 [14, 15]. 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 [14]. 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.

Results and methodology. We derive a polynomial time $(3 + \varepsilon)$ -approximation algorithm for minimizing the makespan of parallel jobs with arbitrary time-resource tradeoffs. More specifically, our result holds for an arbitrary number m of machines, an arbitrary number k of available units of the additional resource, and arbitrary, polynomial time computable time-resource tradeoff functions $p_j(x), j \in V$, where x is the amount of resources allocated to job j . As a special case, our approach comprises linear time-resource tradeoff functions, where $p_j(x) = \bar{p}_j - a_j x$, a_j being the slope of the linear time-resource tradeoff function, and \bar{p}_j being the default processing time. Notice that our result generalizes the

previous $(3 + \varepsilon)$ -approximation of [7] in several directions: They consider the special case $k = 1$, which can be interpreted as linear time-resource tradeoff functions, and they consider a constant number of machines m . Although we obtain the same performance bound, we stress that our result relies on a completely different approach. Also notice that we derive the first polynomial time approximation algorithms for problems with succinctly encodable time-resource tradeoff functions, as previous approaches such as [8] generally do not yield polynomial time algorithms.

Apart from improving previous results in the scheduling context, we would like to stress that the main contribution of the paper is rather on the methodology side. In fact, we obtain our result by using a mathematical programming formulation that constitutes a relaxation of the problem. This mathematical program is allowed to contain, both in the objective and in the constraints, arbitrary polynomial time computable functions. When restricted to linear time-resource tradeoff functions, for example, this mathematical program is a concave minimization problem with linear constraints. We show that the solution of the mathematical program is NP-hard even for the special case of linear time-resource tradeoff functions. Nevertheless, in general we show that it can be solved with arbitrary precision in polynomial time; a result of interest in its own.

Moreover, we provide a parametric example to show that our analysis cannot be improved further than a factor of 1.46 even for linear time-resource tradeoff functions, by showing that the allocation of resources that is computed with the mathematical program can indeed provide the ‘wrong’ answer. The same example shows that it may happen that the scheduling algorithm we use, based on the resource allocation as suggested by the mathematical program, is a factor 2 away from the optimum.

2. Problem definition

Let $V = \{1, \dots, n\}$ 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 exactly one of the machines, and V_i denotes the set of jobs assigned to machine i , such that $V = \bigcup_i V_i$ forms a partition of the jobs. During its processing, a job j may be assigned an amount $x \in \{0, 1, \dots, k\}$ of a discrete resource, for instance personnel, that may speed up its processing. The amount of resources assigned to a job must be constant throughout its processing, and

is restricted to be at most k . If x resources are allocated to a job j , the processing time of that job is p_{jx} , $x = 0, \dots, k$. The global resource constraint now consists of the fact that in a feasible solution, at any time no more than k units of the resource may be consumed by the schedule. Clearly, we may assume $k \geq 1$ since the problem is trivial otherwise.

The actual processing time p_{jx} of a job is computed via time resource tradeoff functions $p_j(\cdot) : \{0, 1, \dots, k\} \rightarrow \mathbb{Z}^+$. We assume (without loss of generality) that all time-resource tradeoff functions $p_j(\cdot)$ are non-increasing on their domains, meaning that the more resources are allocated to a job, the shorter the processing time, and $\bar{p}_j := p_j(0)$ is the default processing time, $j \in V$. We moreover assume that these functions are computable in polynomial time, that is, there is an algorithm that, for any given value $x \in \{0, 1, \dots, k\}$, returns the value $p_j(x)$ in time polynomial in the encoding length of the function $p_j(\cdot)$ and $\log k$. By definition, we have $p_{jx} = p_j(x)$ for all $j \in V$ and $x \in \{0, 1, \dots, k\}$. We make the seemingly artificial distinction between p_{jx} and $p_j(x)$ only to highlight the possible difference in the encoding length: All possible processing times of all jobs are given by the values p_{jx} , $j \in V$, $x = 0, \dots, k$. The encoding length of these values is clearly $\Omega(nk)$. But all time resource tradeoff functions $p_j(\cdot)$, $j \in V$, may in general be encoded more succinctly. Letting $p = \max_{j \in V} \bar{p}_j$ and A be the maximal encoding length of any time-resource tradeoff function, the encoding length of the problem is $O(n \log p + nA + \log k)$. For example, if we assume linear time-resource tradeoff functions where $p_j(x) = \bar{p}_j - a_j x$, the encoding length is $O(n(\log p + \log a) + \log k)$, with $a = \max_{j \in V} a_j$.

3. Computational Complexity

As a generalization of the *dedicated machine scheduling* problem as considered by Kellerer and Strusevich [14], it follows that the problem at hand is strongly NP-hard. We next derive a stronger result, namely an inapproximability result.

Theorem 1 *Unless $P = NP$, there is no approximation algorithm with a performance guarantee smaller than 1.5 for scheduling parallel jobs with time-resource tradeoff.*

Proof. Adapting a proof idea from [8], we use a reduction from the NP-complete problem PARTITION: We are given k integers a_1, \dots, a_k , with $\sum_{j=1}^k a_j = 2B$, and we are asked to decide if there exists a subset $S \subseteq \{1, \dots, k\}$ with $\sum_{j \in S} a_j = B$. We define for each item a_j

one job j , to be processed on its individual machine (so $m = n$), with time-resource tradeoff function as follows

$$p_j(x) = \begin{cases} 2a_j + 1 - 2x & x \leq a_j, \\ 1 & x > a_j. \end{cases}$$

Moreover, let the availability of the resource be $k = B$. Obviously, the encoding length of any time-resource function is in $O(\log a_j)$, hence this transformation is polynomial. Now it is easy to see that there exists a partition if and only if the optimal solution for the scheduling problem has a makespan of 2: Each job j gets assigned exactly a_j units of the resource, thus has processing time 1, and the jobs can be partitioned into two sets, each with total resource requirement B if there exists a partition. Hence the makespan is 2. Conversely, if no partition exists, any schedule must have makespan at least 3. The claimed inapproximability bound follows. \square

4. Mathematical programming relaxation

The approach of [8] could be used to obtain a 3.75-approximation algorithm for the problem at hand. The approach, however, is explicitly based upon an integer linear programming formulation that would require $\Theta(nk)$ binary variables to represent all the different processing times of jobs p_{js} . Obviously, when the time-resource tradeoff functions are encoded succinctly, this would generally not lead to a polynomial time algorithm.

To tackle the problem independent on the encoding length of the functions, we can set up a polynomial size, mathematical programming formulation, using $O(n)$ integer variables $x_j \in \{0, \dots, k\}$ that denote the number of resources allocated to job j , $j \in V$. The following integer mathematical program then has a solution if there is a feasible schedule with makespan C .

$$\sum_{j \in V_i} p_j(x_j) \leq C, \quad \forall i = 1, \dots, m, \quad (1)$$

$$\sum_{j \in V} x_j p_j(x_j) \leq kC, \quad (2)$$

$$0 \leq x_j \leq k, \quad \forall j \in V, \quad (3)$$

$$x_j \in \mathbb{Z}^+, \quad \forall j \in V. \quad (4)$$

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 kC . Our goal is to compute an integer feasible solution (C^*, x^*) for program (1)–(4), such that C^* is a lower bound for the makespan C^{OPT} of an optimal schedule. A candidate for C^* is the smallest integer value, say C^{MP} , for which this program is feasible. But since we do not know how to compute C^{MP} exactly, we will compute an approximation $C^* \leq C^{\text{MP}}$.

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

$$\min. \quad \sum_{j \in V} x_j p_j(x_j) \quad , \quad (5)$$

$$\text{s. t.} \quad \sum_{j \in V_i} p_j(x_j) \leq C \quad , \quad \forall i = 1, \dots, m, \quad (6)$$

$$0 \leq x_j \leq k \quad , \quad \forall j \in V, \quad (7)$$

$$x_j \in \mathbb{Z}^+, \quad \forall j \in V. \quad (8)$$

Obviously, (1)–(4) is feasible if and only if the problem (5)–(8) has a solution with an objective value at most kC .

Before we describe the algorithm handling the problem (5)–(8), let us briefly address the computational complexity of this problem. For example, if time-resource tradeoff functions are linear, problem (5)–(8) is a constrained quadratic optimization problem. It is well known that constrained quadratic programming is NP-hard in general [20], even without integrality constraints. More specifically, for linear time-resource tradeoff functions we have a constrained concave minimization problem, which is generally known to be NP-hard as well [10]. We next show that also the specific problem (5)–(8) is NP-hard.

Theorem 2 *The integer mathematical programming problem (5)–(8), and also its linear relaxation (5)–(7) are NP-hard to solve to optimality, even if all time-resource tradeoff functions $p_j(\cdot)$, $j \in V$, are linear.*

Proof. We again use a reduction from the NP-complete problem PARTITION. Recall that we are given ℓ nonnegative integers a_1, \dots, a_ℓ , with $\sum_{j=1}^{\ell} a_j = 2B$, and we are asked to decide if there exists a subset $S \subseteq \{1, \dots, \ell\}$ with $\sum_{j \in S} a_j = B$. Define $m = 3$ machines. For any $j = 1, \dots, \ell$ we introduce three identical jobs, each of which is to be scheduled on one of the three machines, each with a linear time-resource tradeoff function $p_j(x) = 2a_j - a_j x$,

$x \in \{0, 1\}$. Thus in total there are $n := 3\ell$ jobs each of which can be processed either in its default mode with processing time $2a_j$, or in fast mode with processing time a_j . Only one unit of the resource is available, thus $k = 1$. Using default processing times the makespan on each machine is exactly $4B$. We ask for a makespan at most $3B$. For this problem, the linear relaxation (5)–(7) becomes

$$\begin{aligned} \min. \quad & \sum_{i=1}^3 \sum_{j=1}^{\ell} a_j x_j^i (2 - x_j^i) , \\ \text{s. t.} \quad & \sum_{j=1}^{\ell} a_j (2 - x_j^i) \leq 3B , & i = 1, 2, 3 , \\ & 0 \leq x_j^i \leq 1 , & \forall j = 1, \dots, \ell, i = 1, 2, 3 . \end{aligned}$$

Now it is not hard to see that there is a partition if and only if the optimal solution to this concave quadratic program equals $3B$: If there exists a partition induced by $S \subseteq \{1, \dots, \ell\}$, just let $x_j^i = 1$ if $j \in S$ and $x_j^i = 0$ otherwise, for all $i = 1, 2, 3$. Conversely, assume a solution with value $3B$. We claim that any optimal solution must be in $\{0, 1\}^n$. To see this, first notice that $\sum_{j=1}^{\ell} a_j (2 - x_j^i) = 3B$ for $i = 1, 2, 3$ for any optimal solution, for otherwise at least one variable x_j^i could be decreased, thereby decreasing also the objective value. Now assume there is a fractional variable $0 < x_j^i < 1$, then by the previous observation there must be another fractional variable $0 < x_{j'}^i < 1$, and we could trade-off one for another, while improving the objective value. Next, in order to fulfill the constraint set, we must have that $\sum_j a_j x_j^i \geq B$ for all $i = 1, 2, 3$. Now if $\sum_j a_j x_j^i > B$ for some i , by integrality this yields that $\sum_i \sum_j a_j x_j^i = \sum_i \sum_j a_j x_j^i (2 - x_j^i) > 3B$, a contradiction. Thus $\sum_j a_j x_j^i = B$ for all $i = 1, 2, 3$, and therefore a partition exists. \square

We next show that the (integer) mathematical program (5)–(8) can nevertheless be solved with arbitrary precision in polynomial time.

Lemma 1 *For any $0 < \delta < 1$, we can find a solution for the mathematical program (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 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 [21].

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

$$\min. \quad \sum_{j \in V_i} x_j p_j(x_j) \quad , \quad (9)$$

$$\text{s. t.} \quad \sum_{j \in V_i} p_j(x_j) \leq C \quad , \quad (10)$$

$$0 \leq x_j \leq k \quad , \quad \forall j \in V_i \quad , \quad (11)$$

$$x_j \in \mathbb{Z}^+ \quad , \quad \forall j \in V_i \quad . \quad (12)$$

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

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

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

Now consider an arbitrary $j \in V_i$ and the corresponding $\ell \in \mathbb{Z}^+$ such that $(1 + \varepsilon_1)^{\ell-1} \leq x_j < (1 + \varepsilon_1)^\ell$. Since x_j is an integer, we have that $[(1 + \varepsilon_1)^{\ell-1}] \leq x_j < [(1 + \varepsilon_1)^\ell] = x'_j < (1 + \varepsilon_1)^\ell + 1$. Now, if $(1 + \varepsilon_1)^\ell + 1 \leq (1 + \varepsilon_1)^{\ell+1}$ we immediately derive that $x'_j < (1 + \varepsilon_1)^2 x_j < (1 + 3\varepsilon_1)x_j$. 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 $x_j = x'_j = [(1 + \varepsilon_1)^{\ell-1}]$. Therefore, $x'_j \leq (1 + 3\varepsilon_1)x_j$, for all $j \in V_i$. Consequently, from the observation above and the monotonicity of the time-resource tradeoff functions, we have

$$X' = \sum_{j \in V_i} x'_j p_j(x'_j) \leq \sum_{j \in V_i} (1 + 3\varepsilon_1)x_j p_j(x_j) = (1 + 3\varepsilon_1)X \quad ,$$

as claimed before.

We next claim that the problem (9)–(12) restricted to $x_j \in \mathcal{E}, j \in V_i$, 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 processing times $p_j(x_j)$ with associated costs $x_j p_j(x_j)$, where $x_j \in \mathcal{E}$. 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 $O(\log_{1+\varepsilon_1} k)$ processing times and costs per job, 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. \square

Lemma 2 *Consider a single machine scheduling problem where we have a due date C , and n jobs, each having h possible modes $s = 1, \dots, h$ at which its processing time is p_{js} and its cost is w_{js} , $s = 1, \dots, h$. 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 a fully polynomial time approximation scheme (FPTAS).*

Proof. Utilizing the framework of Pruhs and Woeginger [21], 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 [21] 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

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

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

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

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. \square

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^{MP} 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

$$z_{C^*} \leq (1 + \delta) k C^* . \quad (13)$$

Consider $C := C^* - 1$. By definition of C^* as the smallest integer with property (13), on value C the FPTAS yields a solution with $z_C > (1 + \delta) k C$, and by Lemma 1, the optimal solution for (5)–(8) is larger than $k C$, and hence (1)–(4) is infeasible for C . Hence, the smallest integer value for which (1)–(4) has a feasible solution is at least $C^* = C + 1$, or $C^* \leq C^{\text{MP}}$. Therefore, C^* is a lower bound on C^{OPT} , the makespan of an optimal solution. Moreover, using the FPTAS of Lemma 1 and (13), we have an integral solution (x_1^*, \dots, x_n^*) that is feasible for (1)–(4) with constraint (2) relaxed to

$$\sum_{j \in V} x_j p_j(x_j) \leq (1 + \delta) k C^* . \quad (14)$$

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 $C^* \leq C^{\text{OPT}}$, and an integer solution $x^* = (x_1^*, \dots, x_n^*)$ for the resource consumptions of jobs such that*

$$\sum_{j \in V_i} p_j(x_j^*) \leq C^* , \quad i = 1, \dots, m , \quad (15)$$

$$\sum_{j \in V} x_j^* p_j(x_j^*) \leq (1 + \delta) k C^* . \quad (16)$$

5. 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 mathematical 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 x_j^* additional resources. Then the jobs are scheduled according to (an adaptation of) the greedy list scheduling algorithm of Graham [4], in arbitrary order.

Algorithm MP-GREEDY: Let the resource allocations be fixed as determined by the solution to the mathematical program. 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 $t' > t$.

Obviously, this algorithm can be implemented in polynomial time. Now we claim the following.

Theorem 3 *For any $\varepsilon > 0$, algorithm MP-GREEDY is a $(3+\varepsilon)$ -approximation algorithm for scheduling parallel jobs with time-resource tradeoff. The computation time of the algorithm is polynomial in the input size and the precision $1/\varepsilon$.*

Note that the result of Theorem 3 improves considerably on the performance bound of 3.75 from [8]. Moreover, also recall that the approach of [8] does not yield polynomial time algorithms for succinctly encoded time-resource tradeoff functions.

Proof. In order to do the binary search for the integer value C^* in the mathematical 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^{OPT} of an optimal schedule, together with an integer solution x^* for (1),(3),(4), and (14). We then fix the assignments of resources to the jobs as suggested by the solution x^* , 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 MP-GREEDY, and denote by C^{MPG} the corresponding makespan. Denote by C^{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 $\beta = C^{\text{MPG}} - t(\beta)$ 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 α the total length of busy periods on machine i between 0 and $t(\beta)$, and by γ the total length of idle periods on machine i between 0 and $t(\beta)$. We then have that

$$C^{\text{MPG}} = \alpha + \beta + \gamma. \quad (17)$$

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

$$\alpha \leq \sum_{j \in V_i} p_j(x_j^*) \leq C^*. \quad (18)$$

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

$$\beta + \gamma \leq 2(1 + \delta) C^*. \quad (19)$$

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

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

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

Now we are ready to prove the performance bound of Theorem 3. First, use (17) together with (18) and (19) to obtain

$$C^{\text{MPG}} \leq C^* + 2(1 + \delta) C^* = (3 + 2\delta) C^*.$$

Eventually, because C^* is a lower bound on C^{OPT} , this yields a performance bound for MP-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. \square

6. Lower bounds

Theorem 1 shows that the problem at hand does not allow for an approximation algorithm with performance bound smaller than 1.5. We next show 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, and linear time-resource tradeoff functions. Let an integer ℓ be fixed. The first two machines are assigned two jobs each, symmetrically. One of these two jobs has a constant processing time $p_j(x) = \ell - 3$, for any $x = 0, 1, 2$. The other job has a processing time $p_j(x) = 3 + 2\ell - \ell x$ if assigned x units of the resource, thus the only way to get this job reasonably small is to assign all 2 resources, such that $p_j(2) = 3$. On the third machine, we have three jobs. Two identical short jobs with processing times $p_j(x) = 3 - x$, and one long job with processing time $p_j(x) = \ell - 3x$, $x = 0, \dots, 2$. See Figure 1 for an example. \square

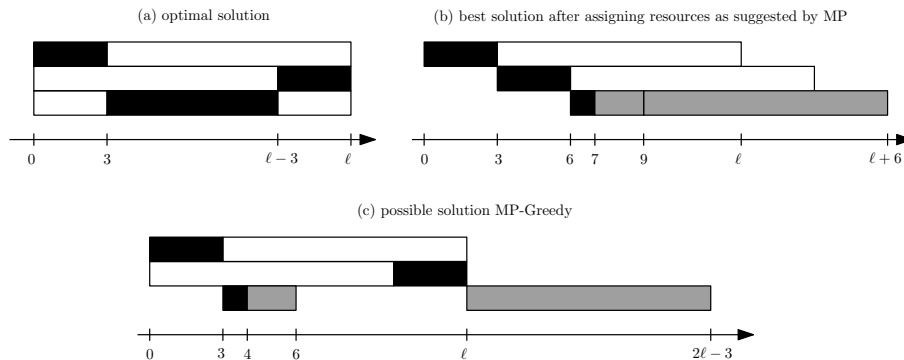


Figure 1: Black jobs consume 2 resources, gray jobs 1, and white jobs 0 resources.

Proposition 1 There exists an instance where the assignment of resources to the jobs as proposed by the solution to the mathematical 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 MP-GREEDY may yield 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 2ℓ): the two jobs with the high compression rate consume 2 units of the resource, yielding a total processing time of ℓ on the first two machines. In the optimal solution, the makespan is exactly ℓ , 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 mathematical programming relaxation (1)–(4) is feasible is $C = \ell$, too. We claim that our solution to the mathematical 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 MP, we minimize the total resource consumption of the schedule, subject to the constraint that the total processing time on each machine is bounded by $C = \ell$. On the third machine, the minimal resource consumption, subject to the condition that the makespan is at most ℓ 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 ℓ , 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 ℓ would be optimal, 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 ℓ . \square

It remains open at this point whether there exist instances of the problem on which algorithm MP-GREEDY outputs a solution with performance ratio worse than 2.

7. Discussion

We proved the existence of a $(3 + \varepsilon)$ -approximation algorithm for scheduling problems where the processing times of jobs can be reduced by investing a renewable, discrete resource, such

as additional personal. The strength of this result, in contrast to previous approaches such as [8], is that we explicitly allow for any type of succinctly encodable time-resource tradeoff functions. This comprises important special cases, such as (piecewise) linear time-resource tradeoff functions, for example. For such problems, no polynomial time (approximation) algorithm was known before. Apart from this result, we see our main contribution on the methodology side, as we believe that there might be more applications where the FPTAS for solving NP-hard (integer) mathematical programs can be useful. Investigating further into possible generalizations of this result seems to us an interesting direction for future research; see for example also [16].

Acknowledgements

We thank Gerhard Woeginger for several helpful suggestions. In particular, Gerhard pointed us to the paper [21], and proposed the proof for the FPTAS for the single machine scheduling problem in Lemma 2. We also thank Frits Spieksma for some remarks, and the anonymous referees of an earlier version of this paper for their constructive comments.

References

- [1] J. BLAZEWICZ, J. K. LENSTRA AND A. H. G. RINNOOY KAN, Scheduling subject to resource constraints: Classification and complexity, *Discr. Appl. Math.* **5** (1983), pp. 11–24.
- [2] Z.-L. CHEN, Simultaneous Job Scheduling and Resource Allocation on Parallel Machines, *Ann. Oper. Res.* **129** (2004), pp. 135–153.
- [3] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 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 J. Applied Math.* **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, *Ann. Discr. Math.* **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, Machine Scheduling with Resource Dependent Processing Times, *Math. Prog.* **110** (2007), pp. 209–228.
- [9] 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, pp. 203–215.
- [10] R. HORST AND P. M. PARDALOS, Editors, *Handbook of Global Optimization*, volume 2 of Nonconvex Optimization and Its Applications, Springer, 1995.
- [11] K. JANSEN, Scheduling Malleable Parallel Tasks: An Asymptotic Fully Polynomial Time Approximation Scheme, *Algorithmica* **39** (2004), pp. 59-81.
- [12] K. JANSEN, M. MASTROLILLI AND R. SOLIS-OBA, Approximation Schemes for Job Shop Scheduling Problems with Controllable Processing Times, *European Journal of Operational Research* **167** (2005), pp. 297-319.
- [13] J. E. KELLEY AND M. R. WALKER, *Critical path planning and scheduling: An introduction*, Mauchly Associates, Ambler (PA), 1959.
- [14] H. KELLERER AND V. A. STRUSEVICH, Scheduling parallel dedicated machines under a single non-shared resource, *Europ. J. Oper. Res.* **147** (2003), pp. 345–364.
- [15] H. KELLERER AND V. A. STRUSEVICH, Scheduling problems for parallel dedicated machines under multiple resource constraints, *Discr. Appl. Math.* **133** (2004), pp. 45–68.
- [16] W. KERN AND G. WOEGINGER, Quadratic programming and combinatorial minimum weight product problems, *Math. Prog.* **110** (2007), pp. 641–649.

- [17] J. K. LENSTRA, D. B. SHMOYS AND E. TARDOS, Approximation algorithms for scheduling unrelated parallel machines, *Math. Prog.* **46** (1990), pp. 259–271.
- [18] 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.
- [19] 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>
- [20] P. M. PARDALOS AND G. SCHNITGER, Checking Local Optimality in Constrained Quadratic Programming is NP-hard, *Oper. Res. Lett.* **7** (1988), pp. 33–35.
- [21] 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*, M. Farach-Colton (ed.), Lecture Notes in Computer Science 2976, Springer, 2004, pp. 203–211.
- [22] D. B. SHMOYS AND E. TARDOS, An approximation algorithm for the generalized assignment problem, *Math. Prog.* **62** (1993), pp. 461–474.
- [23] M. SKUTELLA, Approximation algorithms for the discrete time-cost tradeoff problem, *Math. Oper. Res.* **23** (1998), pp. 909–929.
- [24] 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.