Discrete Optimization 6 (2009) 414–419

Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

Scheduling jobs with time-resource tradeoff via nonlinear programming

Alexander Grigoriev [a], Marc Uetz [b,*]

[a] Maastricht University, Quantitative Economics, P.O. Box 616, 6200 MD Maastricht, The Netherlands

[b] University of Twente, Applied Mathematics, P.O. Box 217, 7500 AE Enschede, The Netherlands

ARTICLE INFO

Article history:
Received 31 October 2007
Received in revised form 15 May 2009
Accepted 20 May 2009
Available online 10 June 2009

Keywords:
Scheduling
Mathematical programming
Approximation algorithms
Computational complexity
Time-resource tradeoff

ABSTRACT

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 nonlinear mathematical programming relaxation. We also derive lower bounds for the approximation.

1. Introduction and related work

Consider a scheduling problem where n jobs $j \in V$ need to be processed on a set of m 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 resource are available. We may assume without loss of generality that the time-resource tradeoff functions $p_j(\cdot) : \{0, 1, \ldots, k\} \to \mathbb{Z}^+$ 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 *non-renewable* resource, such as a total budget constraint, have received a lot of attention in the literature as *time-cost* tradeoff problems, e.g., [1–5]. 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 [6], the authors consider the more general problem of unrelated machine scheduling with resource dependent processing times, and derive a 3.75-approximation algorithm. The approach presented in [6] 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 consists

* Corresponding author.
E-mail addresses: a.grigoriev@maastrichtuniversity.nl (A. Grigoriev), m.uetz@utwente.nl (M. Uetz).

1572-5286/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2009.05.002

of $O(n)$ numbers only: For each job, we have to specify its machine i, a default processing time $\bar{p}_j$, and a compression rate a_j, such that the time-resource tradeoff functions equal $p_j(x) = \bar{p}_j - a_j x$, where x is the amount of resource assigned to job j. Then the algorithms proposed in [6] are generally not polynomial time algorithms. In [7], Kellerer considered the more restricted problem of identical parallel machine scheduling with resource dependent processing times. For this problem Kellerer derived $(3.5 + \varepsilon)$-approximation algorithm based upon an extended version of the linear program from [6]. Notice that the linear program in [7] also uses nk variables.

When we 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. [8]. More recently, such problems with the assumption that jobs are distributed over the machines beforehand have been discussed by Kellerer and Strusevich [9–11]. They use the term *dedicated machine scheduling*. We refer to these papers for various complexity results and approximation algorithms. Note that NP-hardness of dedicated machine scheduling and a binary resource was established in [9]. 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. In [10], Kellerer and Strusevich derived a $(3 + \varepsilon)$-approximation algorithm again using a linear program with nk variables. For the problem when the number of machines is constant, they derive a polynomial time approximation scheme.

Results and methodology. We derive a polynomial time $(3 + \varepsilon)$-approximation algorithm for minimizing the makespan of jobs with arbitrary time-resource tradeoffs. Our result holds for an arbitrary number m of machines, an arbitrary number k of available units of the resource, and arbitrary, polynomial time computable time-resource tradeoff functions $p_j(\cdot), j \in V$. As a special case, this comprises linear time-resource tradeoff functions. Notice that our result generalizes the $(3 + \varepsilon)$-approximation of [10]. Although we obtain the same performance bound, we stress that we derive the first polynomial time approximation algorithms for problems with succinctly encodable time-resource tradeoff functions. Previous approaches such as [6,10] generally do not yield polynomial time algorithms.

We use 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, it is a concave minimization problem with linear constraints. We use this relaxation for obtaining a lower bound on the optimal solution, and to get a clue on how to assign the resource to the jobs. The relaxation has – after a series of transformations – an interpretation as a version of the knapsack problem that describes the time-resource tradeoff on which we have to decide. Similar techniques appear in the design on algorithms for scheduling problems with controllable processing times, see for example [12,13]. Although the relaxation is NP-hard to solve in general, we show that it can be approximated arbitrarily well in polynomial time. Following the resource assignment as suggested by the solution of the relaxation, as in [6] we use simple list scheduling to obtain our result.

Finally, we provide a parametric example that yields lower bounds on the performance guarantee that can be achieved with our approach.

2. Problem definition

Let $V = \{1, \ldots, n\}$ be a set of jobs. Jobs must be processed non-preemptively on a set of m 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 pre-assigned to exactly one of the machines, and V_i denotes the set of jobs assigned to machine i. Thus, subsets $V_i,\ i \in \{1, \ldots, m\}$ form a partition of V. During its processing, a job j may be assigned an amount $x \in \{0, 1, \ldots, k\}$ 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}, $x = 0, \ldots, k$. 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. Clearly, we 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 $p_j(\cdot) : \{0, 1, \ldots, k\} \to \mathbb{Z}^+$. We assume (without loss of generality) that all time-resource tradeoff functions $p_j(\cdot)$ are non-increasing. By $\bar{p}_j := p_j(0)$ we denote the default processing time. We assume that these functions are computable in polynomial time, that is, there is an algorithm that, for any given value $x \in \{0, 1, \ldots, k\}$, 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 $x \in \{0, 1, \ldots, k\}$. 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 $p_{jx}, j \in V, x = 0, \ldots, k$. The encoding length of these values is $\Omega(nk)$. 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, for linear 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 [9], it follows that the problem at hand is strongly NP-hard. Using standard gap-reduction techniques (see, e.g., [14]) we next derive a stronger result.

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

Proof. We use a reduction from the NP-complete problem PARTITION: We are given ℓ integers $a_1, \ldots, a_\ell$, with $\sum_{j=1}^{\ell} a_j = 2B$, and we are asked to decide if there exists a subset $S \subseteq \{1, \ldots, \ell\}$ 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

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

Let the availability of the resource be $k = B$. 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. □

4. Mathematical programming relaxation

The approach of [6] 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}. In general, this does 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 $x_j \in \{0, \ldots, k\}$ 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.

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

$$\sum_{j\in V} x_j p_j(x_j) \le k\,C, \tag{2}$$

$$0 \le x_j \le k, \quad \forall j \in V, \tag{3}$$

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

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^*, 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 $C^* \le C^{\mathrm{MP}}$.

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

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

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

$$0 \le x_j \le k, \quad \forall j \in V, \tag{7}$$

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

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

In [15], Moré and Vivasis show that the problems (5)–(7) and (5)–(8) are NP-hard. We next prove that the integer mathematical program (5)–(8) can nevertheless be solved with arbitrary precision in polynomial time, i.e., (5)–(8) admits a fully polynomial time approximation scheme (FPTAS).

Lemma 1. *For any $\delta > 0$, 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 $1/\delta$.*

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

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

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

$$0 \le x_j \le k, \quad \forall j \in V_i, \tag{11}$$

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

Notice that the input size of program (11) and (12) depends on the encoding length of functions p_j, $j \in V_i$. At a small cost to the objective function we can avoid this dependency. To achieve this, we use a geometric scaling of the resource consumption. Consider an even more restrictive problem, where instead of constraints (11) and (12), we restrict the resource consumptions x_j, $j \in V_i$, to rounded powers of $(1+\varepsilon)$. More precisely, we set

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

where $\varepsilon > 0$ is an arbitrary precision. It is straightforward to verify 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 $X' \le (1+3\varepsilon)X$ and $x'_j \in \mathcal{E}$ for all $j \in V_i$. Since $|\mathcal{E}| \in O(\log k)$, the input size of the restricted problem is $O(n \log k)$.

We next claim that the problem (9)–(12) restricted to $x_j \in \mathcal{E}, j \in V_i$, admits an FPTAS. Observe that this problem is in fact a special case of the knapsack problem with separable nonlinear functions, which is reducible to a 0-1 multiple-choice knapsack problem with $n|\mathcal{E}|$ items and n equivalence classes; see Lawler [16]. This problem can be solved, for any prescribed relative error $\delta > 0$, in $O(n|\mathcal{E}| \log |\mathcal{E}| + n^2|\mathcal{E}|/\delta)$ time [16]. Since $|\mathcal{E}| \in O(\log k)$, the run time is polynomial in the input size and $1/\delta$, finishing the proof. □

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

$$z_{C^*} \le (1+\delta)\, k\, C^*. \tag{13}$$

Consider $C := C^* - 1$. By the 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 $C^* \le C^{\mathrm{MP}}$. 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 $(x_1^*, \ldots, x_n^*)$ that is feasible for (1)–(4) with constraint (2) relaxed to

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

We conclude that we can derive an approximate solution for (1)–(4) in the following sense.

Lemma 2. *For any $\delta > 0$, we can find in polynomial time an integer value C^* such that $C^* \le C^{OPT}$, and an integer solution $x^* = (x_1^*, \ldots, x_n^*)$ for the resource consumptions of jobs such that*

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

$$\sum_{j \in V} x_j^* p_j(x_j^*) \le (1+\delta)\ kC^*. \tag{16}$$

5. Greedy algorithm

We use the solution of the mathematical programming relaxation from the previous section in order to decide on the amount of resources – namely, x_j^* – allocated to every individual job j. Then the jobs are scheduled according to an adaptation of the greedy list scheduling algorithm of Graham [17,18], 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 $t' > t$.

Theorem 2. *For any $\varepsilon > 0$, algorithm* MP-Greedy *is a $(3+\varepsilon)$-approximation algorithm for scheduling jobs with time-resource tradeoff. The computation time of the algorithm is polynomial in the input size and the precision $1/\varepsilon$.*

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 similar ideas as that of our previous paper [6]. Therefore, we only present the main idea here.

Denote by C^{MPG} the makespan of the schedule produced by the algorithm. The schedule is split up into three periods: Let $t(\beta)$ be the point in time from which only jobs with resource consumption strictly larger than $k/2$ are processed on all

machines. Let $\beta = C^{\mathrm{MPG}} - t(\beta)$ (which might be 0). Next, select a machine i where a job with resource consumption at most $k/2$ finishes at $t(\beta)$. Define I as the total length of idle periods on machine i up to $t(\beta)$, and B as the total length of busy periods on machine i up to $t(\beta)$. Clearly, $C^{\mathrm{MPG}} = B + I + \beta$.

Due to (15), we get that for machine i, $B \leq \sum_{j \in V_i} p_j(x_j^*) \leq C^*$. Moreover, using (16), exactly as in [6, Lemma 5], one can show that $I + \beta \leq 2(1+\delta)\, C^*$. Putting all this together we obtain

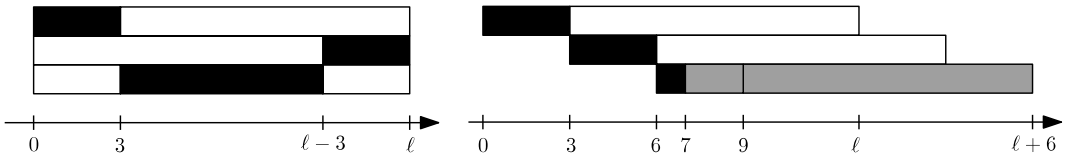
$$C^{\mathrm{MPG}} = B + I + \beta \leq C^* + 2(1+\delta)C^* = (3 + 2\delta)\, C^*.$$

Because C^* is a lower bound on C^{OPT}, this yields a performance bound for MP-Greedy of $3 + 2\delta = 3 + \varepsilon$, as $\delta = \varepsilon/2$. The claim on the computation time follows from the fact that we use an FPTAS in Lemma 1, and since the greedy algorithm runs in polynomial time. □

6. Lower bounds

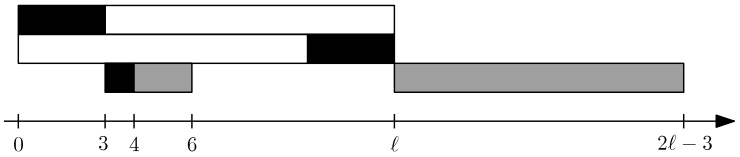
We next show that our approach may yield a solution that is a factor $2 - \varepsilon$ away from the optimal solution, for any $\varepsilon > 0$.

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 $p_j(x) = \ell - 3x$, $x = 0, \ldots, 2$. See Fig. 1. □



(a) Optimal solution.

(b) Best solution after assigning resources as suggested by MP.



(c) Possible solution MP-Greedy.

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

Proposition 1. *There exists an instance where the solution to the mathematical programming relaxation yields a resource assignment such that any schedule is 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 instance defined in Example 1, with parameter $\ell \geq 13$. The assignment of resources to the jobs on the first two machines is 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 Fig. 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. Fig. 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 Fig. 1(c). This yields a ratio of $(2\ell - 3)/\ell$, which is arbitrarily close to 2 for large ℓ. □

Acknowledgements

We thank Gerhard Woeginger and Frits Spieksma for several helpful suggestions on a previous version of this paper [19], and the anonymous referees for their helpful comments and useful references.

References

[1] Z.-L. Chen, Simultaneous job scheduling and resource allocation on parallel machines, Ann. Oper. Res. 129 (2004) 135–153.
[2] K. Jansen, M. Mastrolilli, R. Solis-Oba, Approximation schemes for job shop scheduling problems with controllable processing times, European J. Oper. Res. 167 (2005) 297–319.
[3] J.E. Kelley, M.R. Walker, Critical Path Planning and Scheduling: An Introduction, Mauchly Associates, Ambler (PA), 1959.
[4] D.B. Shmoys, E. Tardos, An approximation algorithm for the generalized assignment problem, Math. Prog. 62 (1993) 461–474.
[5] M. Skutella, Approximation algorithms for the discrete time-cost tradeoff problem, Math. Oper. Res. 23 (1998) 909–929.
[6] A. Grigoriev, M. Sviridenko, M. Uetz, Machine scheduling with resource dependent processing times, Math. Prog. 110 (2007) 209–228.
[7] H. Kellerer, An approximation algorithm for identical parallel machine scheduling with resource dependent processing times, Oper. Res. Lett. 36 (2008) 157–159.
[8] J. Blazewicz, J.K. Lenstra, A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, Discrete Appl. Math. 5 (1983) 11–24.
[9] H. Kellerer, V.A. Strusevich, Scheduling parallel dedicated machines under a single non-shared resource, European J. Oper. Res. 147 (2003) 345–364.
[10] H. Kellerer, V.A. Strusevich, Scheduling parallel dedicated machines with the speeding-up resource, Naval Res. Logist. 55 (2008) 377–389.
[11] H. Kellerer, V.A. Strusevich, Scheduling problems for parallel dedicated machines under multiple resource constraints, Discrete Appl. Math. 133 (2004) 45–68.
[12] K. Jansen, M. Mastrolilli, Approximation schemes for parallel machine scheduling problems with controllable processing times, Comput. Oper. Res. 31 (2004) 1565–1581.
[13] D. Shabtay, G. Steiner, A survey of scheduling with controllable procesing times, Discrete Appl. Math. 155 (2007) 1643–1666.
[14] V.V. Vazirani, Approximation Algorithms, Springer, Berlin, 2001.
[15] J.J. Moré, S.A. Vivasis, On the solution of concave knapsack problems, Math. Prog. 49 (1991) 397–411.
[16] E. Lawler, Fast approximation algorithms for knapsack problems, Math. Oper. Res. 4 (1979) 339–356.
[17] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell System Technical Journal 45 (1966) 1563–1581.
[18] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 17 (1969) 416–429.
[19] A. Grigoriev, M. Uetz, Scheduling parallel jobs with linear speedup, in: Approximation and Online Algorithms, in: T. Erlebach, P. Persiano (Eds.), Lecture Notes in Computer Science, vol. 3879, Springer, 2006, pp. 203–215.