INSTITUT FÜR INFORMATIK

Approximation Algorithms for Scheduling with Reservations

Florian Diedrich
Klaus Jansen
Fanny Pascual
Denis Trystram

Bericht Nr. 0812
October 2008

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Approximation Algorithms for Scheduling with Reservations

Florian Diedrich
Klaus Jansen
Fanny Pascual
Denis Trystram

Bericht Nr. 0812
October 2008

e-mail: fdi@informatik.uni-kiel.de, kj@informatik.uni-kiel.de,
fanny.pascual@imag.fr, denis.trystram@imag.fr

Approximation Algorithms for Scheduling with Reservations*

Florian Diedrich† Klaus Jansen‡ Fanny Pascual§
Denis Trystram¶

October 7, 2008

Abstract

We study the problem of non-preemptively scheduling n independent sequential jobs on a system of m identical parallel machines in the presence of reservations. This setting is practically relevant because for various reasons, some machines may not be available during specified time intervals. The objective is to minimize the makespan $C_{\max}$, which is the maximum completion time.

The general case of the problem is inapproximable unless $\mathsf{P} = \mathsf{NP}$; hence, we study a suitable strongly NP-hard restriction, namely the

*An extended abstract of this work has been accepted at the 14th International Conference on High Performance Computing, HiPC 2007, Goa, India, December 18–21, 2007.

†Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Christian-Albrechts-Platz 4, D-24098 Kiel, Germany, fdi@informatik.uni-kiel.de. Research was supported in part by a grant "DAAD Doktorandenstipendium" of the German Academic Exchange Service and in part by EU research project AEOLUS, Algorithmic Principles for Building Efficient Overlay Computers, EU contract number 015964. Part of this work done while visiting the LIG, Grenoble University. Supported in part by DFG priority program 1126, "Algorithmics of Large and Complex Networks"; furthermore supported in part by a PPP funding "Scheduling in Communication Networks" D/05/06936 granted by the DAAD.

‡Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Christian-Albrechts-Platz 4, D-24098 Kiel, Germany, kj@informatik.uni-kiel.de. Supported in part by DFG priority program 1126, "Algorithmics of Large and Complex Networks" and in part by EU research project AEOLUS, Algorithmic Principles for Building Efficient Overlay Computers, EU contract number 015964.

§INRIA, LIG, Grenoble University, 51 avenue Jean Kuntzmann, F-38330 Montbonnot Saint-Martin, France, fanny.pascual@imag.fr.

¶LIG, Grenoble University, 51 avenue Jean Kuntzmann, F-38330 Montbonnot Saint-Martin, France, denis.trystram@imag.fr. Part of this work was supported by the "CoreGRID" Network of Excellence.

case where at least one machine is always available. For this setting we contribute approximation schemes, complemented by inapproximability results. The approach is based on algorithms for multiple subset sum problems; our technique yields a PTAS which is best possible in the sense that an FPTAS is ruled out unless $\mathsf{P} = \mathsf{NP}$. The PTAS presented here is the first one for the problem under consideration; so far, not even for well-known special cases approximation schemes have been proposed. Furthermore we derive a low cost algorithm with a constant approximation ratio and discuss FPTASes for special cases as well as the complexity of the problem if m is part of the input.

1 Introduction

In parallel machine scheduling, an important issue is the scenario where the machines are not continuously available but time intervals of unavailability have to be taken into account; this problem occurs due to periods of regular maintenance or because high-priority jobs have already been preallocated in the system. In either case we obtain deterministic off-line models capturing realistic industrial settings and scheduling problems in parallel computing. More precisely, we study the problem of scheduling sequential jobs on a system of m identical parallel machines, where m is constant; however, these machines may be unavailable for certain periods of time which are known a priori. The jobs must be executed non-preemptively; this setting is also called the *non-resumable* case [20, 22, 23] in the literature. The objective is to minimize the makespan $C_{\max}$, which is the maximum of the completion times of all jobs. $C_{\max}$ is one of the most well-studied objectives in the field of scheduling and usually regarded as an "easy" objective in the sense that most problem formulations permit good approximation algorithms. Quite restricted special cases of the model considered here have already been studied, as discussed in the sequel; however, on the algorithmic side, only list scheduling algorithms or similar approaches and exact exponential algorithms have been analyzed and experimentally evaluated.

Contributions. We use algorithms for multiple subset sum problems to govern the non-preemptive scheduling of jobs on identical parallel machines with reservations. On the algorithmic side we obtain a PTAS for the case of an arbitrary number m of machines which is based on dual approximation [10]; furthermore we discuss FPTASes for $m \in \{1, 2\}$ with one reservation and a fast greedy algorithm. These algorithms are complemented by inapproximability results which show that for arbitrary m no FPTAS is possible; furthermore, we show that the problem does not become easier if the number of reservations per machine is restricted to one. Finally we show

that the problem formulation where m is part of the input does not permit an approximation ratio better than 3/2; all of our inapproximability results are based on the assumption $\mathsf{P} \neq \mathsf{NP}$ and use gap creation arguments.

This article is organized as follows. In Sect. 2 we formalize the problem and discuss the inapproximability of the general case. In Sect. 3 we present a PTAS for a suitably restricted problem as well as FPTASes for $m \in \{1, 2\}$ with one reservation in Subsect. 3.1 and sketch how to obtain a fast approximation algorithm for the general problem in Subsect. 3.2; furthermore, in Subsect. 3.3 our approximation algorithms are complemented by hardness results. Finally we conclude with a summary in Sect. 4.

Related problems and previous results. Lee [19] and Lee et al. [21] studied identical parallel machines which may have different starting times; here, the LPT policy (where tasks are greedily scheduled from the largest to the smallest task) was analyzed. Lee [20] studied the case where at most one reservation per machine is permitted while one machine is always available and obtained approximation ratios for low-complexity list scheduling algorithms. Liao et al. [23] presented an experimental study of an exact algorithm for $m = 2$ within the same scenario. Hwang et al. [12] studied the LPT policy for the case where at most one interval of unavailability per machine is permitted; they proved a tight bound of $1 + \lceil m/(m-\lambda) \rceil / 2$ where at most $\lambda \in \{1, \ldots, m-1\}$ machines are unavailable simultaneously. In [22], Chapt. 22, additional problem definitions and a survey about previous results can be found. Scharbrodt et al. [27, 28] presented approximation schemes and inapproximability results for a setting where the reservations are regarded as jobs and, in contrast to our problem, also contribute to the makespan. Furthermore, Liao & Sheen [24] studied the preemptive case where the reservations are given implicitly by availability periods; they proved this problem formulation to be polynomially solvable.

So far, the model under consideration in this article has not been approached with approximation schemes, not even for well-established special cases [12, 20, 23].

The approach taken in our work is based on multiple subset sum problems. These are special cases of knapsack problems, which belong to the oldest problems studied in combinatorial optimization and theoretical computer science; hence we benefit from the fact that they are relatively well understood. For the classical problem (KP) with one knapsack, besides the result by Ibarra & Kim [13], Lawler presented a sophisticated FPTAS [18] which was later improved by Kellerer & Pferschy [16]; see also the textbooks by Martello & Toth [25] and Kellerer et al. [17] for surveys. The case where the item profits equal their weights is called the subset sum problem and denoted as SSP. The problem with *multiple* knapsacks (MKP) is

a natural generalization of KP; the case with multiple knapsacks where the item profits equal their weights is called the *multiple* subset sum problem (MSSP). Various special cases and extensions of these problems have been studied [1, 2, 3, 4, 5, 6, 7, 14, 15], finally yielding PTASes for various problem formulations [2, 4, 5, 15] including the case upon which our approach is based.

2 Problem Definition and Preliminaries

Now we formally define our problem. Let $m \in \mathbb{N}^*$ denote the number of machines. An instance I consists of n jobs characterized by processing times $p_1, \ldots, p_n$, and r reservations $R_1, \ldots, R_r$. For each $k \in \{1, \ldots, r\}$, $R_k = (i_k, s_k, t_k)$ indicates unavailability of machine i_k in the time interval $[s_k, t_k)$, where $s_k, t_k \in \mathbb{N}, i_k \in \{1, \ldots, m\}$ and $s_k < t_k$. We suppose that for reservations on the same machine there is no overlap; for two reservations $R_k, R_{k'}$ such that $i_k = i_{k'}$ holds, we assume $[s_k, t_k) \cap [s_{k'}, t_{k'}) = \emptyset$. For each machine index $i \in \{1, \ldots, m\}$ let $R'_i := \{R_k \in I | i_k = i\}$ denote the set of reservations for machine i. Finally, for each $i \in \{1, \ldots, m\}$ suppose that R'_i is sorted increasingly with respect to the starting times of the reservations; more precisely, $R'_i = \{(i, s_{i1}, t_{i1}), \ldots, (i, s_{ir_i}, t_{ir_i})\}$ such that $s_{i1} < \cdots < s_{ir_i}$ where we set $r_i := |R'_i|$. These assumptions are established algorithmically in $O(r \log r)$ time by sorting $\{R_1, \ldots, R_r\}$ lexicographically with respect to the first two components of its elements and partitioning it into $R'_1, \ldots, R'_m$ and finally merging adjacent reservations in R'_i for each $i \in \{1, \ldots, m\}$. In the sequel we use $P(I) := \sum_{j=1}^{n} p_j$ to denote the total processing time of an instance I and for each $S \subseteq \{1, \ldots, n\}$ we write $P(S) := \sum_{j \in S} p_j$ for the total processing time of S. Finally let $p_{\max} := \max\{p_j | j \in \{1, \ldots, n\}\}$. A schedule is a function $\sigma : \{1, \ldots, n\} \to \{1, \ldots, m\} \times [0, \infty)$ which maps each job to its executing machine and starting time; if σ is clear from the context it may be dropped from notation. Our goal is to compute a non-preemptive schedule of the tasks such that no task is scheduled on a machine that is unavailable, and, on each machine at most one task runs at a given time; the objective is to minimize the makespan $C_{\max}$. Using the 3-field notation, we denote our problem by $\mathrm{P}m|nr\text{-}a|C_{\max}$, where the job characteristics indicate the non-resumable setting with availability constraints [20, 22]. We show that this problem is inapproximable; the proof is based on a construction by Lee [20], however there it was only remarked that LPT performs arbitrarily badly for the problem.

Theorem 1. $\mathrm{P}m|nr\text{-}a|C_{\max}$ *does not admit a polynomial time algorithm with a constant approximation ratio unless* $\mathsf{P} = \mathsf{NP}$.

(a) Structure of reservations (b) Optimal solution of I'

Figure 1: Sketch illustrating the proof of Theorem 1

Proof. Let $c \in \mathbb{R}$, $c \geq 1$; suppose there is an approximation algorithm A for $\mathrm{P}m|nr\text{-}a|C_{\max}$ with ratio c. We use a reduction from the following strongly NP-complete problem 3-Partition [9]; see Fig. 1 for a sketch of the proof.

- *Given:* Index set $S = \{1, \ldots, 3n\}$, $a_i \in \mathbb{N}^*$ for each $i \in S$ and $B \in \mathbb{N}^*$ such that $B/4 < a_i < B/2$ for each $i \in S$ and $\sum_{i=1}^{3n} a_i = nB$ holds.
- *Question:* Is there a partition of the set S into $S_1, \ldots, S_n$ such that $\sum_{i \in S_j} a_i = B$ holds for each $j \in \{1, \ldots, n\}$?

Given an instance I of 3-Partition we define an instance I' of $\mathrm{P}m|nr\text{-}a|C_{\max}$; we set $p_i := a_i$ for each $i \in \{1, \ldots, 3n\}$ and define

$$R_i := (1, i(B+1) - 1, i(B+1))$$

for each $i \in \{1, \ldots, n-1\}$, $R_n := (1, n(B+1) - 1, \lceil c \rceil n(B+1))$ and finally set $R_{n+i} := (1+i, 0, \lceil c \rceil n(B+1))$ for each $i \in \{1, \ldots, m-1\}$. I' can be generated from I in time polynomial in the encoding length of I and yields an optimal makespan $C^*_{\max} = n(B+1) - 1$ if and only if I is a yes-instance of 3-Partition; furthermore, any suboptimal schedule of I' for a yes-instance I of 3-Partition has a makespan $C_{\max} > \lceil c \rceil n(B+1)$. For any yes-instance I of 3-Partition, A generates a schedule for I' with makespan $C_{\max}$ such that

$$C_{\max} \leq cC^*_{\max} = c(n(B+1) - 1) < \lceil c \rceil n(B+1)$$

holds. Hence I is identified as a yes-instance of 3-Partition, which is impossible unless $\mathsf{P} = \mathsf{NP}$ holds. □

The inapproximability of the general case is due to the permission of intervals in which no machine is available. Hence it is reasonable to suppose that at each time step there is an available machine. This is not sufficient since we can prove in this case the same inapproximability result by considering, for example, the following instance. There is, for a given period p, a set of reservations which alternate on two machines in a such a way that there are no two reservations at the same time and the period between two consecutive reservations is smaller than the length of any task of the

instance. In this case, no task can be put during time period p and we get the same inapproximability result as in the case where there is on each of these machines a big reservation of length p. Thus we will suppose in the sequel that at least one machine is always available. If we regard reservations as preallocated high-priority jobs, then, since the machines are identical, the reservations can be put on the machines in such a way that w.l.o.g. the *first* machine is always available, hence $i_k \neq 1$ for each reservation R_k. This can be done by distributing the reservations one by one and always putting a reservation on the machine with maximum index $i \in \{1, \ldots, m\}$ among the available machines.

We use $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ to denote this restricted problem; $1up$ means that at least one machine is always available. This problem is still strongly NP-hard for $m \geq 2$, as we will see later in Theorem 5.

3 Algorithms and Hardness Results

We present approximation algorithms and complexity results. In Subsect. 3.1 we obtain approximation schemes; in Subsect. 3.2 we discuss fast greedy algorithms that are based on the same idea. We close the section with complexity results in Subsect. 3.3.

3.1 Polynomial Time Approximation Schemes

We explain the multiple subset sum approach for $m \geq 2$ in detail to obtain a PTAS for $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$. Later we discuss the cases $m \in \{1, 2\}$, which admit FPTASes for the case where only one reservation is permitted. Our idea is based on obtaining a complementary representation for the periods of availability in order to reduce the problem to MSSP which however admits a PTAS [2, 4, 5]; we derive a dual approximation algorithm [10] by using binary search on the makespan where a PTAS for MSSP serves as a relaxed decision procedure, as illustrated in Fig. 2. In Sect. 2 we argued how to obtain sorted sets R'_i of reservations for each $i \in \{2, \ldots, m\}$. We use the algorithm in Fig. 3 to obtain sets of inclusionwise maximal availability intervals A_i for each $i \in \{1, \ldots, m\}$, each one containing elements (i, s, t) indicating that machine i is available in $[s, t)$ where $s \in \mathbb{N}, t \in \mathbb{N} \cup \{\infty\}$. Below we discuss the single steps in detail.

Step 1 in Fig. 3 defines all time available on the first machine as an interval of availability. Step 2.1 checks if there is no reservation on machine i; in this case the entire processing time $[0, \infty)$ on machine i is added to the set A_i of availability intervals for machine i. In the innermost loop,

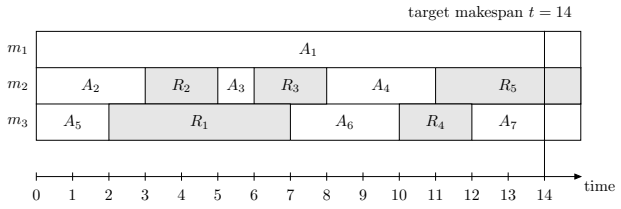


Figure 2: Sketch illustrating the approach of the algorithm in Fig. 5. The grey zones $R_1, \ldots, R_5$ are the reservations. If the target makespan is 14, we try to fill all the jobs in knapsacks of sizes corresponding to $A_1, \ldots, A_7$; zones A_1 and A_7 end at time 14

1. Set $A_1 := \{(1, 0, \infty)\}$ and for each $i \in \{2, \ldots, m\}$ set $A_i := \emptyset$.
2. For each $i \in \{2, \ldots, m\}$ execute Steps 2.1–2.3.
   - 2.1. If $r_i = 0$, set $A_i := \{(i, 0, \infty)\}$ and proceed with the next iteration of the loop started in Step 2.
   - 2.2. Set $t := 0$.
   - 2.3. For each $r \in \{1, \ldots, r_i\}$ execute Steps 2.3.1–2.3.2.
     - 2.3.1 If $s_{ir} = 0$ then proceed with the next iteration of the loop started in Step 2.3, otherwise set $A_i := A_i \cup \{(i, t, s_{ir})\}$ and $t := t_{ir}$.
     - 2.3.2 If $r = r_i$ then set $A_i := A_i \cup \{(i, t, \infty)\}$.

Figure 3: Algorithm GenAvail

the variable t serves as a pointer indicating the next time step in which the current machine becomes potentially available. Step 2.3.1 covers the case in which the current reservation starts at time zero, which can only be the case for the first reservation on the current machine; in this case no interval is generated, otherwise the processing time before the current reservation is added to A_i. Finally, Step 2.3.2 checks if the current reservation is the last one on the current machine and handles the case correspondingly. The running time of the algorithm in Fig. 3 is linear in m, r and independent from n; furthermore at most $2r$ intervals of availability are generated. For a fixed machine $i \in \{1, \ldots, m\}$, we use the initial sorting of R'_i to obtain that the intervals of availability for machine i are sorted with respect to their starting times.

1. For each $i \in \{1, \ldots, m\}$ execute Steps 1.1–1.2.

   1.1. Set $A'_i(t) = \{(i, s', t') \in A_i | s' < t)\}$ and $a_i := |A'_i(t)|$.

   1.2. If $a_i > 0$ set $t_{ia_i} := \min\{t_{ia_i}, t\}$.

Figure 4: Algorithm GenAvailFinite

However more important is the subroutine in Fig. 4 that uses $A_1, \ldots, A_m$ to generate the finite intervals of *availability* for a *fixed finite* planning horizon $[0, t)$ where $t \in \mathbb{N}$. Step 1.1 in the algorithm in Fig. 4 removes all intervals of availability that begin outside of $[0, t)$ while Step 1.2, if necessary, truncates the last interval on a machine to fit exactly into the planning horizon. The running time of the algorithm in Fig. 4 is independent from n and linear in m, r. In the sequel we denote $A(t) := \cup_{i=1}^{m} A'_i(t)$ and will use the at most $2r$ intervals stored in $A(t)$ as knapsacks in which we like to pack the jobs in $\{1, \ldots, n\}$. To this end, we use a PTAS for MSSP and for each job $j \in \{1, \ldots, n\}$ define an item j with weight p_j to obtain an instance of MSSP. The algorithm is described in Fig. 5, where MSSPPTAS is a PTAS for MSSP where the knapsack capacities are permitted to be different [2, 4, 5]. We suppose that MSSPPTAS does not only select a desired $S \subseteq \{1, \ldots, n\}$ but also stores the feasible assignment to the knapsacks as a byproduct. In total, we obtain the algorithm in Fig. 5; the approach is sketched in Fig. 2.

Theorem 2. *The algorithm in Fig. 5 is a PTAS for* $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$.

Proof. Since the first machine is available at each time step $t \in [0, \infty)$, the sum of processing times $P(I)$ is an upper bound for the optimal makespan $C^*_{\max}$; hence in Step 2, the lower bound LB and the upper bound UB are initialized to have the following properties.

1. $LB < C^*_{\max}$.
2. There is a set $S \subseteq \{1, \ldots, n\}$ such that the jobs in S permit a feasible schedule into the time horizon $[0, UB)$ and $P(S) \geq (1 - \epsilon/m)P(I)$.

The second property is due to the fact that, since $C^*_{\max} \leq UB$, all jobs can be scheduled in $[0, UB)$ and thus it is impossible that the algorithm MSSPPTAS returns a set $S \subseteq \{1, \ldots, n\}$ such that $P(S) < (1 - \epsilon/m)P(I)$ holds; both properties are invariant under the update of LB and UB in Step 3.3. The number of iterations of the binary search in Step 3 is bounded by $\log P(I) \leq \log(np_{\max}) = \log n + \log p_{\max}$ which is polynomially bounded in the encoding length of I. On termination of the binary search in Step 3,

1. Use the algorithm in Fig. 3 to generate A_i for each $i \in \{1, \ldots, m\}$.

2. Set $LB := 0$ and $UB := P(I)$.

3. While $UB - LB > 1$ repeat Steps 3.1–3.3.

   3.1 Set $t := \lfloor (UB - LB)/2 \rfloor$. Use the algorithm in Fig. 4 to generate $A(t)$, the set of availability intervals for fixed planning horizon $[0, t)$.

   3.2 Use MSSPPTAS with accuracy ϵ/m to select a set of jobs

   $$S \subseteq \{1, \ldots, n\}$$

   such that

   $$P(S) \geq (1 - \epsilon/m) \max\{P(S') | S' \subseteq \{1, \ldots, n\}, \\ S' \text{ permits a feasible packing into the intervals in } A(t)\}.$$

   3.3 If $P(S) < (1 - \epsilon/m)P(I)$ then set $LB := t$ else store S and set $UB := t$.

4. Schedule the jobs in the last stored set S into the interval $[0, UB)$ as indicated by the solution generated by MSSPPTAS when S was returned; schedule the jobs in $\{1, \ldots, n\} \setminus S$ in the interval $[UB, \infty)$ on the first machine without unnecessary idle time.

Figure 5: Algorithm MultiSubsetSumScheduler

$LB + 1 = UB$ holds, hence $UB \leq C^*_{\max}$ since $LB < C^*_{\max}$ is satisfied. This means that the set S selected in Step 4 can be scheduled in $[0, UB)$ and satisfies $P(S) \geq (1 - \epsilon/m)P(I)$; hence $P(\{1, \ldots, n\} \setminus S) \leq \epsilon P(I)/m$ holds. Furthermore the jobs in $\{1, \ldots, n\} \setminus S$ can be scheduled on the first machine in $[UB, \infty)$ since the first machine is available. We have $P(I)/m \leq C^*_{\max}$; in total, the makespan of the schedule generated by the algorithm in Fig. 5 is bounded by $UB + \epsilon P(I)/m \leq C^*_{\max} + \epsilon C^*_{\max} = (1 + \epsilon)C^*_{\max}$ and we obtain the desired approximation ratio. Since the running time of MSSPPTAS is polynomially bounded in r and n the claim is proved. □

However, since the running time of MKPPTAS may grow exponentially in $1/\epsilon$, the running time of the algorithm in Fig. 5 may also grow exponentially in m. MSSP does not admit an FPTAS even for the special case of two knapsacks of equal capacity, unless $\mathsf{P} = \mathsf{NP}$ holds, as discussed in [17], Subsect. 10.4. Hence it is impossible for the approach used above to yield an FPTAS for $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ by replacing MSSPPTAS with a better algorithm, which is not surprising in the light of Corollary 6 in Subsect. 3.3.

For $m = 1$ the situation is different. Lee [20] remarked that $1|nr\text{-}a|C_{\max}$ is strongly NP-hard via reduction from 3-Partition. The problem is inapproximable in the general case by Theorem 1 and remains inapproximable if the number of reservations is restricted to two, as can be seen in Lemma 8 in Subsect. 3.3. However, if there is only one reservation, an FPTAS can be obtained since SSP admits an FPTAS [15, 17]. This case corresponds to a simple knapsack problem – if all tasks can be scheduled before the reservation, we get an optimal solution; otherwise we use the FPTAS for SSP to schedule as much load as possible before the reservation.

As in [23] we study the case $m = 2$ with one reservation $R_1 = (2, s, t)$ and show how to obtain an FPTAS based on dynamic programming and scaling the state space. Here $C' := P(I)$ yields a 2-approximation, hence we have $C^*_{\max} \leq C' \leq 2C^*_{\max}$; Furthermore we denote by A the interval $[0, \infty)$ on machine 1, by B the interval $[0, s)$ on machine 2 and by C the interval $[t, \infty)$ on machine 2. For a (partial) schedule σ we use $A(\sigma)$ to denote its load in A, $B(\sigma)$ to denote its load in B and $C(\sigma)$ to denote its load in C. The states of the dynamic program can be organized as a table by defining

$$F[k, x, y] := \min\{\infty, \min\{B(\sigma) | \sigma \text{ is a schedule for the jobs in } \{1, \ldots, k\} \text{ such that } A(\sigma) = x \text{ and } C(\sigma) = y\}\}$$

for each $k \in \{1, \ldots, n\}$ and $x, y \in \{0, \ldots, C'\}$, where ∞ indicates the nonexistence of such a schedule. We obtain the recurrence relation

$$F[k, x, y] = \min\{F[k-1, x - p_k, y], F[k-1, x, y - p_k]\}$$

if $F[k-1,x,y]+p_k > s$ (job k can not be placed in B) and

$$F[k,x,y] = \min\{F[k-1,x-p_k,y], F[k-1,x,y-p_k], F[k-1,x,y]+p_k\}$$

if $F[k-1,x,y]+p_k \leq s$ (job k can be placed in B); this recurrence relation can be proved in detail by induction on k. Hence, either inductively by iterating over $k \in \{1,\dots,n\}$ or recursively using lazy evaluation, we can solve the problem $\mathrm{P2},1up|nr\text{-}a|C_{\max}$ with one reservation to optimality within the pseudopolynomial runtime bound $O(nC'^2) = O(n^3 p_{\max}^2)$ by selecting coordinates $x,y \in \{0,\dots,C'\}$ in order to minimize the value

$$f(x,y) := \begin{cases} \max\{x, t+y\} & : F[n,x,y] \neq \infty, y > 0 \\ \max\{x, F[n,x,y]\} & : F[n,x,y] \neq \infty, y = 0 \\ \infty & : F[n,x,y] = \infty \end{cases}$$

which, in the case $f(x,y) \neq \infty$, is the makespan of a corresponding schedule. A suitable schedule can either be found by backtracking or maintaining suitable auxiliary data structures while evaluating the states; both approaches can be implemented within the given runtime bound.

Now we discretize the state space of the dynamic program by defining a scaling factor $K := \epsilon C'/(2n)$ and introducing scaled job sizes $q_j := \lceil p_j/K \rceil$ for each $j \in \{1,\dots,n\}$. The values q_j are used for computation of the indices on the x and y axes while the values p_j are still used to compute the values for the states of the dynamic program, where now $x,y \in \{0,\dots,\lceil C'/K \rceil\}$. Hence, the discretized makespans of schedules for the jobs in $\{1,\dots,n\}$ now have the load values Kx and Ky for the intervals A and C, respectively. In total, the values of f defined above are modified by replacing x by Kx and y by Ky in the maximum expressions; finally, the described algorithm yields the following result.

Theorem 3. $\mathrm{P2},1up|nr\text{-}a|C_{\max}$ *with one reservation admits an FPTAS.*

Proof sketch. We obtain $\lceil C'/K \rceil \in O(n/\epsilon)$, hence the runtime bound of the sketched algorithm is bounded by $O(n^3/\epsilon^2)$ which is polynomial in both $1/\epsilon$ and the encoding length of the instance. Furthermore the inequality

$$Kq_j \geq p_j > K(q_j - 1)$$

is valid for each $j \in \{1,\dots,n\}$; with calculations similar to those in [18], we obtain $K\sum_{j\in S} q_j \leq \sum_{j\in S} p_j + \epsilon C^*_{\max}$ for each $S \subseteq \{1,\dots,n\}$. In particular, this inequality is satified for suitable job sets $S_1, S_2 \subseteq \{1,\dots,n\}$ which constitute the machine loads in A and C in an optimal schedule; in total this yields the desired approximation ratio. □

1. Sort items by size in non-increasing order yielding $p_1 \geq \cdots \geq p_n$; sort knapsacks by capacity in non-decreasing order yielding $c_1 \leq \cdots \leq c_m$.

2. Iterate items in the order generated in Step 1; at each step, assign the current item to the knapsack with minimum index it can be feasibly packed into, if any. Discard the current item otherwise.

Figure 6: Algorithm GreedyMSSP

3.2 Greedy Algorithms

In [6] a greedy 2-approximation algorithm for MSSP with running time $O(n^2)$ is briefly mentioned; the subject is also discussed in [17], Subsect. 10.4.1, with a slightly different approach. Here we present the algorithm from [6] in Fig. 6.

Theorem 4. *The algorithm in Fig. 6 is a 2-approximation algorithm for MSSP; furthermore this approximation ratio is asymptotically attained.*

Proof. By the sorting generated in Step 1, w.l.o.g. we have $p_j \leq c_m$ for each $j \in \{1, \ldots, n\}$ since other items can not occur in any feasible solution. Let A be the assignment generated by the algorithm in Fig. 6; we denote by $A(i)$ the total load that A assigns to knapsack i for each $i \in \{1, \ldots, m\}$. Let $U \subseteq \{1, \ldots, n\}$ be the set of items which are not packed by A. If $U = \emptyset$ all items are packed and A is an optimal assignment; hence suppose $U \neq \emptyset$. For each $j \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, m\}$ we call j *admissible* to i iff $p_j \leq c_i$; furthermore a knapsack i is called *half-full* iff $A(i) \geq c_i/2$. If there are only half-full knapsacks the claim follows; hence suppose there are knapsacks which are not half-full and let $i' := \max\{i \in \{1, \ldots, m\} | i \text{ is not half-full}\}$. Aiming at a contradiction, assume $i' = m$, hence m is not half-full. A assigns at least one item to m, since otherwise $U \neq \emptyset$ is violated. Consequently because m is not half-full, A assigns only items of size at most $c_m/2$ to m. Since U contains the items which are not packed by A, U contains only items of size at most $c_m/2$, which A would assign to m, a contradiction; hence $i' < m$ holds. Let $c := c_{i'}$ and note that $c < c_{i'+1}$ holds; aiming at a contradiction, assume $c = c_{i'+1}$. Let j be the last item assigned to $i' + 1$ by A, which must exist since $i' + 1$ is half-full. If $i' + 1$ contains at least two items, they cannot be both larger than $c/2$, hence A would assign p_j to i' which is not half-full; this yields a contradiction. If $i' + 1$ contains only the item j, every item that A assigns to i' must be smaller than p_j; consequently, the algorithm in Fig. 6 tries to pack j *before* every item packed in i'; since

$c_{i'} = c_{i+1}$ item j is admissible to i' and packed there by A, a contradiction. In total, $c < c_{i'+1}$ holds.

A knapsack $i \in \{1, \ldots, m\}$ will be called *small* iff $i \in \{1, \ldots, i'\}$ and will be called *large* iff $i \in \{i'+1, \ldots, m\}$; in a similar way, an item $j \in \{1, \ldots, n\}$ is called *small* iff $p_j \leq c$ and called *large* iff $p_i > c$. By this definition, a large item is not admissible to a small knapsack and every large bin is half-full.

We show that A packs every small item of the instance into a small knapsack. Aiming at a contradiction, assume that there is a small item $j \in U$. Item j is admissible to knapsack i', so A tries to pack it there; i' is not half-full, so every item packed there prior to j is smaller than $c/2$. Consequently $p_j < c/2$, so A assigns j to knapsack i', a contradiction. Hence, every small item in the instance is packed. Next suppose that there is a small item j which A assigns to a large knapsack. However, it is tried to be packed in knapsack i' first. Since i' is not half-full, every item packed there prior to j is smaller than $c/2$. Consequently $p_j < c/2$ and A assigns j to a knapsack with index at most i', a contradiction. In total, every small item of the instance is packed into a small knapsack.

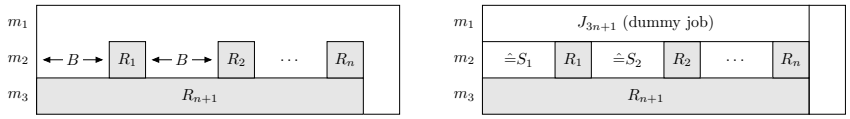
Let OPT be an optimal packing of the instance and let P_{OPT} denote its total profit. Let P^S_{OPT} be the total profit of small items in OPT and P^L_{OPT} be the total profit of large items in OPT; let P_A denote the total profit obtained by A and P^S_A, P^L_A denote the total profit of small and large items in A, respectively. In total we obtain

$$P_A = P^S_A + P^L_A \geq P^S_{\mathrm{OPT}} + P^L_A \geq P^S_{\mathrm{OPT}} + P^L_{\mathrm{OPT}}/2 \geq P_{\mathrm{OPT}}/2$$

which yields the approximation ratio. The bound is tight even for one knapsack which can be seen by defining an instance with capacity $B \in \mathbb{N}^*$, n even, and 3 items $p_1 := B/2 + 1$ and $p_2 := p_3 := B/2$. Here the choice of items 2 and 3 yields an optimal profit of B, while the algorithm in Fig. 6 selects item 1; since $\lim_{B\to\infty}(B/2+1)/B = 2$, the ratio is tight. □

By using the algorithm from Fig. 6 instead of MSSPPTAS and changing the bound $1 - \epsilon/m$ to $1/2$ in Step 3 of the algorithm in Fig. 5 we obtain an approximation algorithm with ratio $1 + m/2$ for $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ by following the lines of the proof of Theorem 2. On the other hand, scheduling all jobs on the first machine here yields an m-approximation algorithm; hence the algorithm sketched above yields a better bound than this approach only if $m > 2$ holds.

In [20], Lee studied the case where at most one reservation per machine is permitted and one machine is always available; an approximation ratio of $(m+1)/2$ for LPT is proved. For our generalization $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ we obtain the same asymptotic behavior in m with our greedy approach.



(a) Structure of reservations (b) Optimal solution of I'

Figure 7: Sketch illustrating the proof of Theorem 5

Comparing our result here with the bound $1 + \lceil m/(m-\lambda) \rceil/2$ for LPT [12] where $\lambda \in \{1, \ldots, m-1\}$ is the maximum number of machines which are permitted to be unavailable at the same time, we basically get the same ratio for our case $\lambda = m - 1$. In total, we obtain similar approximation ratios for more general problems, which comes at the cost of increased computational effort, however.

3.3 Hardness Results

We present an inapproximability result which shows that the PTAS for $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ is close to best possible; hence $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ is substantially harder than $\mathrm{P}m||C_{\max}$ which permits an FPTAS [26].

Theorem 5. $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ *is strongly* NP*-hard for* $m \geq 2$.

Proof. We use reduction from 3-Partition which is strongly NP-complete [9]; see Fig. 7 for a sketch of the construction. Given an instance I of 3-Partition we define an instance I' of the problem $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ for $m \geq 2$. We set $p_i := a_i$ for each $i \in \{3n\}$ (*small* jobs), $p_{3n+1} := n(B+1)$ (*dummy* job) and define suitable reservations $R_i := (2, i(B+1) - 1, i(B+1))$, for each $i \in \{1, \ldots, n\}$, $R_{n+i} := (2+i, 0, n(B+1))$ for each $i \in \{1, \ldots, m-2\}$. I' can be generated from I in time polynomial in the length of I and has an optimal makespan of $C^*_{\max} = n(B+1)$ if and only if I is a yes-instance of 3-Partition by putting the small jobs according to the existing partition $S_1, \ldots, S_n$ in the intervals $[0, B), \ldots, [(n-1)(B+1), n(B+1)-1)$ on machine 2 and putting the dummy job on machine 1; conversely in a schedule with makespan exactly $n(B+1)$ the dummy job must be put on machine 1 and hence the small jobs run on machine 2 which indicates the partition of S into $S_1, \ldots, S_n$ since no more than 3 small jobs can fit into an interval of length B. In total, $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ is strongly NP-hard. □

Since the objective values of feasible schedules for $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ are integral and $C^*_{\max} \leq P(I)$, the next result immediately follows from [8].

Corollary 6. $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ *does not admit an FPTAS for* $m \geq 2$ *unless* $\mathsf{P} = \mathsf{NP}$.

It is a natural question whether the problem becomes easier if the number of reservations per machine is restricted to one. Surprisingly, this is not the case, which can be shown by adaptation of a construction from [1]. The following result implies that $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ with at most one reservation per machine for $m \geq 3$ is strongly NP-hard.

Theorem 7. $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ *does not admit an FPTAS, even if there is at most one reservation per machine, for* $m \geq 3$ *unless* $\mathsf{P} = \mathsf{NP}$.

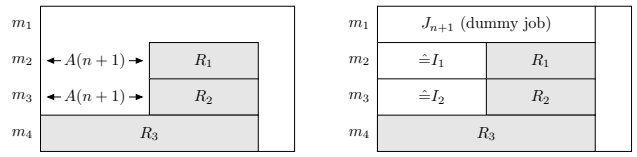
Proof. We use a reduction from the following problem, Equal Cardinality Partition or ECP for short, which is NP-complete [9]; see Fig. 8 for a sketch of the construction.

- *Given:* Finite list $I = (a_1, \ldots, a_n)$ of even cardinality with $a_i \in \mathbb{N}^*$ for each $i \in \{1, \ldots, n\}$, $A \in \mathbb{N}^*$ such that $\sum_{i=1}^{n} a_i = 2A$ holds.
- *Question:* Is there a partition of the list I into lists I_1 and I_2 such that $|I_1| = n/2 = |I_2|$ and $\sum_{i \in I_1} a_i = A = \sum_{i \in I_2} a_i$ holds?

Given an instance I of ECP we define an instance I' of $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ for $m \geq 3$ as follows. We set $p_i := 2A + a_i$ for each $i \in \{1, \ldots, n\}$ (*small* jobs), $p_{n+1} := 2A(n+1)$ (*dummy* job) and $R_k := (k, A(n+1), 2A(n+1))$ for $k \in \{2, 3\}$ and $R_k := (k, 0, 2A(n+1))$ for each $k \in \{4 \ldots, m\}$. I' is generated from I in running time polynomial in the length of I. Furthermore I' has an optimal makespan of $C^*_{\max} = 2A(n+1)$ if and only if I is a yes-instance by executing the small jobs according to the partition I_1 and I_2 on machines 2 and 3 and putting the dummy job on machine 1; conversely in a schedule with makespan $2A(n+1)$ the dummy job is put on machine 1 and hence the small jobs run on machines 2 and 3 which indicates the partition of I into I_1 and I_2 since no more than $n/2$ jobs fit into an availability interval of length $A(n+1)$. Let I be a yes-instance of ECP and consider a suboptimal schedule of I'; the makespan of a suboptimal schedule of I' must be at least $2A(n+1) + A$ since every job in I' has a processing time larger than A and is scheduled either on machine $i \in \{2, \ldots, m\}$ or on machine 1 together with the dummy job, unless the dummy job is scheduled on a machine other than the first one. Given an FPTAS for $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$, choose $\epsilon \in (0, 1)$ such that

$$1 + \epsilon < \frac{2A(n+1) + A}{2A(n+1)} = \frac{2n+3}{2n+2}$$

holds, which is equivalent to $\epsilon < 1/(2n+2)$; consequently ϵ can be chosen in such a way that $1/\epsilon$ is polynomially bounded in n and hence polynomially



(a) Structure of reservations (b) Optimal solution of I'

Figure 8: Sketch illustrating the proof of Theorem 7

bounded in the encoding length of I. Then, the FPTAS generates a schedule with makespan $C_{\max}$ such that

$$C_{\max} \leq (1+\epsilon)C^*_{\max} < \frac{2A(n+1)+A}{2A(n+1)} 2A(n+1) = 2A(n+1) + A$$

holds. Hence I' is solved to optimality in polynomial time and I is identified as a yes-instance of ECP, which is impossible unless $\mathsf{P} = \mathsf{NP}$. □

Theorem 7 does not cover the case $m = 2$ for which there is an FPTAS, see Theorem 3; however, for the case $m = 2$, we obtain a similar result if we permit an arbitrary constant number of reservations as in Theorem 5.

Next, we discuss the hardness of $1|nr\text{-}a|C_{\max}$; more precisely, if more than one reservation is permitted, we obtain the following inapproximability result.

Lemma 8. $1|nr\text{-}a|C_{\max}$*, if more than one reservation is permitted, does not admit a constant approximation ratio unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Let $c \in \mathbb{R}$, $c \geq 1$; suppose there is an approximation algorithm A for $1|nr\text{-}a|C_{\max}$ with ratio c. For an instance I of Partition, which is known to be NP-complete [9], given by $I = \{a_1, \ldots, a_n\}$ such that $\sum_{i \in I} a_i = 2B$, define an instance I' of $1|nr\text{-}a|C_{\max}$ by setting $p_i := a_i$ for each $i \in \{1, \ldots, n\}$, furthermore $R_1 := (1, B, B+1)$ and $R_2 := (1, 2B+1, \lceil c \rceil (2B+2))$. Then I is a yes-instance of Partition if and only if I' has an optimal makespan of $C^*_{\max} = 2B + 1$. However, any suboptimal schedule of I' for a yes-instance I of Partition has a makespan $C_{\max} > \lceil c \rceil (2B+2)$. For any yes-instance I of Partition, A generates a schedule for I' with makespan $C_{\max}$ such that $C_{\max} \leq cC^*_{\max} = c(2B+1) < \lceil c \rceil (2B+2)$ holds. Hence, I is identified as a yes-instance of Partition, which is impossible unless $\mathsf{P} = \mathsf{NP}$ holds. □

Finally we show that the problem formulation where the number m of machines is part of the input does not permit an approximation ratio of 3/2

or better, even if there is at most one reservation per machine. The proof is based on a construction from [28]; again the problem $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ is harder than $\mathrm{P}||C_{\max}$ which is strongly NP-hard but permits a PTAS [11].

Lemma 9. *$\mathrm{P}, 1up|nr\text{-}a|C_{\max}$, even if there is at most one reservation per machine, does not admit a polynomial time approximation algorithm with ratio $3/2 - \epsilon$, unless $\mathsf{P} = \mathsf{NP}$, for any $\epsilon \in (0, 1/2]$.*

Proof. We use a reduction from the following version of 3-Partition which is NP-complete; the NP-completeness can be proved via a reduction from the problem Numerical Matching with Target Sums [9].

- *Given:* Disjoint sets A, B containing n respectively $2n$ elements of sizes $a_i \in \mathbb{N}$ for each $i \in \{1, \ldots, n\}$, $b_i \in \mathbb{N}$ for each $i \in \{1, \ldots, 2n\}$ and $L \in \mathbb{N}$ such that $\sum_{i=1}^{n} a_i + \sum_{i=1}^{2n} b_i = nL$ holds.
- *Question:* Is there a $\pi \in S_{2n}$ such that $a_i + b_{\pi(2i-1)} + b_{\pi(2i)} = L$ holds for each $i \in \{1, \ldots, n\}$?

Given an instance I of the above problem we define an instance I' of the problem $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ as follows. We choose a constant $K \in \mathbb{N}$ such that we have $K > (1/2-\epsilon)L/(2\epsilon)$; we use $n+1$ machines and suitable reservations $R_i := (i+1, 2K+L-a_i, 2K+L)$ for each $i \in \{1, \ldots, n-1\}$. Furthermore we introduce small jobs by $p_i := b_i + K$ for each $i \in \{1, \ldots, 2n\}$ and a dummy job $p_{2n+1} := 2K + L$. Note that I' can be generated from I in running time polynomial in the encoding length of I. Finally I' has an optimal makespan of $C^*_{\max} = 2K + L$ if and only if I is a yes-instance of the above problem by executing the dummy job on machine 1 and executing the small jobs on machines $2, \ldots, n+1$ according to the existing permutation π; conversely, in a schedule with makespan $2K + L$ the dummy job is put on machine 1 and hence the small jobs run on machines $2, \ldots, n+1$ which indicates the desired permutation π. Let I be a yes-instance of the above problem and consider a suboptimal schedule of I'; the makespan of a suboptimal schedule of I' must be at least $3K+L$ since every job in I' has processing time larger than K and is either scheduled on machine $i \in \{2, \ldots, n\}$ or on machine 1 together with the dummy job, unless the dummy job is scheduled on a machine other than the first one. Suppose there is a polynomial-time approximation algorithm A with approximation ratio $3/2-\epsilon$; then, A executed on I' generates a schedule with makespan $C_{\max}$ such that

$$C_{\max} \leq (3/2 - \epsilon)C^*_{\max} = (3/2 - \epsilon)(2K + L) < 3K + L$$

Table 1: Complexity results

Problem	$m = 1$	$m = 2$	$m \geq 3$
$\mathrm{P}m\|nr\text{-}a\|C_{\max}$ arbitrary reservations	no polynomial time algorithm with constant approximation ratio unless $\mathsf{P} = \mathsf{NP}$		
$\mathrm{P}m\|nr\text{-}a\|C_{\max}$ at most one reservation per machine	NP-hard, FPTAS	no polynomial time algorithm with constant approximation ratio unless $\mathsf{P} = \mathsf{NP}$	
$\mathrm{P}m, 1up\|nr\text{-}a\|C_{\max}$ arbitrary reservations	P ($r = 0$)	strongly NP-hard, PTAS, no FPTAS unless $\mathsf{P} = \mathsf{NP}$	
$\mathrm{P}m, 1up\|nr\text{-}a\|C_{\max}$ at most one reservation per machine	P ($r = 0$)	NP-hard, FPTAS	strongly NP-hard PTAS, no FPTAS unless $\mathsf{P} = \mathsf{NP}$
$\mathrm{P}, 1up\|nr\text{-}a\|C_{\max}$ at most one reservation per machine	no polynomial time algorithm with approximation ratio better than 3/2 unless $\mathsf{P} = \mathsf{NP}$		

holds, where the last inequality follows from the equivalence

$$\frac{3K + L}{2K + L} > \frac{3}{2} - \epsilon \Leftrightarrow K > \frac{(1/2 - \epsilon)L}{2\epsilon}$$

which can be proved by elementary calculation. Hence A solves I' to optimality in polynomial time and I is identified as a yes-instance of the above problem, which is impossible unless $\mathsf{P} = \mathsf{NP}$ holds. □

4 Conclusion

We studied scheduling on a constant number of identical parallel machines with reservations and have shown that a restriction to $\mathrm{P}m, 1up|nr\text{-}a|C_{\max}$ is necessary to obtain a bounded approximation ratio. On the algorithmic side we have taken an approach that is based on using approximation algorithms for SSP and MSSP. For the case of arbitrary constant m our approach yields a PTAS and we have shown that no FPTAS exists unless $\mathsf{P} = \mathsf{NP}$ holds, even if the number of reservations per machine is restricted to one. In total, for the general problem as well as various special cases we have settled the approximability; the results are summarized in Tab. 1. Furthermore we have shown that $\mathrm{P}, 1up|nr\text{-}a|C_{\max}$ can not be approximated arbitrarily close unless $\mathsf{P} = \mathsf{NP}$; we propose the investigation of this problem more closely in order to obtain a tight approximation ratio.

Acknowledgements. The authors thank Érik Saule and Ulrich M. Schwarz for many fruitful discussions.

References

[1] A. Caprara, H. Kellerer, and U. Pferschy. The multiple subset sum problem. Technical report, Technische Universität Graz, 1998.

[2] A. Caprara, H. Kellerer, and U. Pferschy. A PTAS for the multiple subset sum problem with different knapsack capacities. *Inf. Process. Lett.*, 73(3-4):111–118, 2000.

[3] A. Caprara, H. Kellerer, and U. Pferschy. A 3/4-approximation algorithm for multiple subset sum. *J. Heuristics*, 9(2):99–111, 2003.

[4] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *SODA*, pages 213–222, 2000.

[5] C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005.

[6] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi. Approximation algorithms for the multiple knapsack problem with assignment restrictions. Technical report, IBM Research Division, 1998.

[7] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi. Approximation algorithms for the multiple knapsack problem with assignment restrictions. *J. Comb. Optim.*, 4(2):171–186, 2000.

[8] M. R. Garey and D. S. Johnson. "strong" NP-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, 1978.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[10] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. ACM*, 34(1):144–162, 1987.

[11] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988.

[12] H.-C. Hwang, K. Lee, and S. Y. Chang. The effect of machine availability on the worst-case performance of LPT. *Disc. App. Math.*, 148(1):49–61, 2005.

[13] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.

[14] H. Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In D. S. Hochbaum, K. Jansen, J. D. P. Rolim, and A. Sinclair, editors, *RANDOM-APPROX*, volume 1671 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 1999.

[15] H. Kellerer, R. Mansini, U. Pferschy, and M. G. Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *J. Comput. Syst. Sci.*, 66(2):349–370, 2003.

[16] H. Kellerer and U. Pferschy. A new fully polynomial time approximation scheme for the knapsack problem. *J. Comb. Optim.*, 3(1):59–71, 1999.

[17] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[18] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979.

[19] C.-Y. Lee. Parallel machines scheduling with non-simultaneous machine available time. *Disc. App. Math.*, 30:53–61, 1991.

[20] C.-Y. Lee. Machine scheduling with an availability constraint. *J. Global Optimization, Special Issue on Optimization of Scheduling Applications*, 9:363–384, 1996.

[21] C.-Y. Lee, Y. He, and G. Tang. A note on "parallel machine scheduling with non-simultaneous machine available time". *Disc. App. Math.*, 100(1-2):133–135, 2000.

[22] J. Y.-T. Leung, editor. *Handbook of Scheduling*. Chapman & Hall, 2004.

[23] C.-J. Liao, D.-L. Shyur, and C.-H. Lin. Makespan minimization for two parallel machines with an availability constraint. *European J. of Operational Research*, 160:445–456, 2003.

[24] L.-W. Liao and G.-J. Sheen. Parallel machine scheduling with machine availability and eligibility constraints. *European J. of Operational Research*, 184:458–467, 2008.

[25] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.

[26] S. Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23(1):116–127, 1976.

[27] M. Scharbrodt, A. Steger, and H. Weisser. Approximability of scheduling with fixed jobs. In *SODA*, pages 961–962, 1999.

[28] M. Scharbrodt, A. Steger, and H. Weisser. Approximability of scheduling with fixed jobs. *J. Scheduling*, 2:267–284, 1999.