

Improved Approximation Algorithms for Scheduling with Fixed Jobs

Florian Diedrich*

Klaus Jansen†

Abstract

We study two closely related problems in non-preemptive scheduling of sequential jobs on identical parallel machines. In these two settings there are either fixed jobs or non-availability intervals during which the machines are not available; in either case, the objective is to minimize the makespan. Both formulations have different applications, e.g. in turnaround scheduling or overlay computing. For both problems we contribute approximation algorithms with an improved ratio of $3/2 + \epsilon$, respectively. For scheduling with fixed jobs, a lower bound of $3/2$ on the approximation ratio has been obtained by Scharbrodt, Steger & Weisser; for scheduling with non-availability we provide the same lower bound. In total, our approximation ratio for both problems is essentially tight via suitable inapproximability results. We use dual approximation, creation of a gap structure and job configurations, and a PTAS for the multiple subset sum problem. However, the main feature of our algorithms is a new technique for the assignment of large jobs via flexible rounding. Our new technique is based on an interesting cyclic shifting argument in combination with a network flow model for the assignment of jobs to large gaps.

1 Introduction

In parallel machine scheduling, an important issue is the scenario where either some jobs are already fixed in the system [29, 30] or intervals of non-availability of some machines must be taken into account [4, 11, 21, 23, 24]. The first problem occurs since high-priority jobs are present in the system while the latter problem is due to regular maintenance of machines; both models are relevant for turnaround scheduling [26] and overlay computing where machines are donated on a volunteer basis. These two problems can be described by the same encoding of instances and only differ in the objective function. An instance consists of m , the number of machines, which is part of the input, and n jobs given by processing times p_1, \dots, p_n . The first k jobs are fixed via a list $(m_1, s_1), \dots, (m_k, s_k)$ giving a machine index and starting time for the respective job. We assume

that these fixed jobs do not overlap. A schedule is a non-preemptive assignment of the jobs to machines and starting times such that the first k jobs are assigned as encoded in the instance and that the jobs do not intersect. If the objective is to minimize the makespan for *all* jobs including the fixed ones, we call the problem *scheduling with fixed jobs*. Alternatively we can regard the k fixed jobs as intervals of non-availability which do not contribute to the makespan. Here the objective is to minimize the makespan over the *non-fixed* jobs only; this problem is called *scheduling with non-availability*. For the latter problem, we denote by $\rho \in (0, 1)$ the *percentage* of machines which are permanently available and also permit infinite length of the non-availability intervals. In the literature, scheduling with non-availability is also called *non-resumable scheduling with availability constraints* [4, 21, 23, 24]. The makespan 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. However, both problems generalize the well-known problem $P||C_{\max}$ [9] and hence are strongly NP-hard.

Results. Scheduling with fixed jobs was studied by Scharbrodt, Steger & Weisser [28, 29, 30]. They mainly studied the problem for m constant; for this easier yet still strongly NP-hard formulation they present a PTAS. They also found approximation algorithms for general m with ratios 3 [28] and $2 + \epsilon$ [30]; since the finishing time of the last fixed job is a lower bound for C_{\max}^* , we can simply use a PTAS for the well-known problem $P||C_{\max}$ [9] to schedule the remaining $n - k$ jobs after the fixed job which finishes last. Finally, Scharbrodt, Steger & Weisser [30] proved that for scheduling with fixed jobs there is no approximation algorithm with ratio $3/2 - \epsilon$, unless $P = NP$, for any $\epsilon \in (0, 1/2]$. Complementing this negative result, we obtain a basically tight ratio with our new approach.

THEOREM 1.1. *Scheduling with fixed jobs admits an approximation algorithm with ratio $3/2 + \epsilon$ for any $\epsilon \in (0, 1/2]$.*

Unlike scheduling with fixed jobs, scheduling with non-availability without any further restriction is inapproximable within a constant ratio unless $P = NP$, as shown

*Institut für Informatik, Universität zu Kiel, Christian-Albrechts-Platz 4, D-24098 Kiel, Germany, fdi@informatik.uni-kiel.de. Supported in part by project AEOLUS, “Algorithmic Principles for Building Efficient Overlay Computers”, EU contract 015964, and in part by a grant “DAAD Doktorandenstipendium” of the German Academic Exchange Service.

†Institut für Informatik, Universität zu Kiel, Christian-Albrechts-Platz 4, D-24098 Kiel, Germany, kj@informatik.uni-kiel.de. Supported in part by project AEOLUS, “Algorithmic Principles for Building Efficient Overlay Computers”, EU contract 015964, and in part by DFG priority program 1126.

by Eyraud-Dubois, Mounié & Trystram [5]. The inapproximability is circumvented by requiring at least one machine to be permanently available. The case with m constant, arbitrary non-availability intervals, and at least one machine permanently available, is strongly NP-hard but can be solved by a PTAS by Diedrich et al. [4]. For general m , researchers so far have only studied the problem where there is at most one interval of non-availability per machine. First, the even more restricted case where the intervals of non-availability start at time zero was studied. Here Lee [20] and Lee et al. [22] proved that LPT yields a ratio of $3/2 - 1/(2m)$ and can be modified to yield a ratio of $4/3$. For the same problem, Kellerer [15] found an algorithm with a tight ratio of $5/4$. Furthermore, Hwang et al. briefly pointed out that this problem admits a PTAS [11]. A more general case is the setting where the at most one interval per machine may have an arbitrary position. For this problem Lee [21] showed that general list scheduling yields a ratio of m and proved a tight ratio of $1/2 + m/2$ for LPT. Hwang et al. studied the ratio of LPT for the same scenario but assumed that at least $m - \lambda$ machines are available simultaneously. They first obtained a ratio of 2 for $\lambda \leq m/2$ [10] which they later refined to a ratio of $1 + \lceil 1/(1 - \lambda/m) \rceil / 2$ for λ arbitrary [11]. For $\lambda = m - 1$, this yields $1 + m/2$; if ρ denotes the percentage of permanently available machines, this yields $1 + \lceil 1/\rho \rceil / 2$ which depends on ρ . Concerning further results, we refer the reader to [23], Chapt. 22, or [27] for surveys and the articles [4, 14, 20] for results on single-machine problems. For scheduling with non-availability, our new technique yields an improved approximation ratio independent from ρ which is basically tight.

THEOREM 1.2. *Scheduling with non-availability, where the percentage $\rho \in (0, 1)$ of permanently available machines is constant, admits an approximation algorithm with ratio $3/2 + \epsilon$ for any $\epsilon \in (0, 1/2]$. Furthermore, for this problem there is no approximation algorithm with ratio $3/2 - \epsilon$, unless $P = NP$, for any $\epsilon \in (0, 1/2]$.*

In addition, we show that approximation of scheduling with non-availability within a constant ratio is at least as hard as approximation of Bin Packing with an additive error; however, whether this is possible is an interesting open problem, as discussed in [7], Chapt. 2, page 67.

Techniques used in our approach. In contrast to previous approaches we use a new technique for rounding and assignment of large jobs which is carried out via a class of network flow problems. To bound the error incurred by this way of assignment, we use an interesting cyclic shifting technique and a redistribution argument. We believe that this approach for rounding

and assignment of suitable items will find other applications in related packing or scheduling problems. Furthermore, we use techniques like dual approximation [8], partition of the instance, linear grouping and rounding known from Bin Packing [6] or Strip Packing [17, 18], and definition of configurations. Our modelization also involves the multiple subset sum problem (MSSP) which can be formally defined as follows. We are given a set $\{1, \dots, n\}$ of items, each item i having a positive integer weight w_i , and a set $\{1, \dots, m\}$ of knapsacks, each knapsack j having a nonnegative integer capacity c_j ; the objective is to select a subset of items of maximum total weight that can be packed into the knapsacks. As an algorithmic building block we use a PTAS for MSSP from [1] where the knapsack capacities are permitted to be different; this approximation scheme is referred to by MSSPPTAS. Alternatively, a PTAS for the multiple knapsack problem (MKP) can be used [2, 3, 13]. Knapsack type problems belong to the oldest and most fundamental problems in combinatorial optimization and theoretical computer science; we refer the reader to [16, 25] for in-depth surveys or the papers [1, 3, 12, 13, 19] for literature on these problems.

The remainder of our contribution is organized as follows. In Sect. 2 we present our main result, namely an approximation algorithm with ratio $3/2 + \epsilon$ for scheduling with fixed jobs. In Sect. 3 we apply our approach to scheduling with non-availability and discuss an interesting connection to Bin Packing. Finally we conclude in Sect. 4 with open questions.

2 An Approximation Algorithm

In this section we prove Theorem 1.1. The running time of our algorithm will depend polynomially on m . However, on the one hand, if $m \leq n$, this yields a runtime bound which is polynomially bounded in the encoding length of the instance. On the other hand, if $m > n$, there are at least $m - k$ machines without fixed jobs. Since we have exactly $n - k$ non-fixed jobs, every job that is to be scheduled can be executed on a free machine of its own, solving the instance to optimality. Hence, in the latter case, it is not necessary to apply our algorithm. The algorithm is based on the dual approximation paradigm [8] by using binary search on the makespan. For any $S \subseteq \{k + 1, \dots, n\}$ let $P(S) := \sum_{i \in S} p_i$ denote the total processing time of S and for any $j \in \{1, \dots, k\}$ let $C_j := s_j + p_j$ denote the completion time of the fixed job j . It is easily seen that $\max\{C_j | j \in \{1, \dots, k\}\} \leq C_{\max}^*$ holds; furthermore, the remaining $n - k$ jobs indexed by $\{k + 1, \dots, n\}$ can be scheduled on one machine in the interval

$$[\max\{C_j | j \in \{1, \dots, k\}\}, np_{\max})$$

where $p_{\max} := \max\{p_j | j \in \{k+1, \dots, n\}\}$ denotes the maximum processing time of the non-fixed jobs. In total we obtain a search space of size at most np_{\max} for the target makespan; via binary search as in [4], we will find a suitable target makespan in $O(\log(np_{\max}))$ steps which is polynomially bounded in the encoding size of the instance. For a target makespan T , we use the technique described below which involves a PTAS for MSSP [1, 13] to schedule as much load as possible in the interval $[0, T)$. In the sequel we show that for the optimal makespan C_{\max}^* , we can algorithmically find a schedule which executes almost all load in the interval $[0, C_{\max}^*)$; the remaining load is put in the interval $[C_{\max}^*, \infty)$ via list scheduling, causing an error which will be suitably bounded however. In Subsect. 2.1–2.4 let $T \in [0, np_{\max})$ denote a candidate for the makespan; we call such a T *feasible* if there is a schedule with makespan at most T and infeasible otherwise. Furthermore, the k fixed jobs are preassigned as indicated by $(m_1, s_1), \dots, (m_k, s_k)$.

2.1 Job Classification and Gap Generation. For T we generate all intervals of availability of machines, in the following called *gaps*, within the planning horizon $[0, T)$ from the encoded fixed jobs. This can be easily achieved in time polynomially bounded in the instance size by processing the starting times and execution times of the fixed jobs. Let $q(T) \in \mathbb{N}^*$ denote the number of gaps and let $G(T) := \{G_1, \dots, G_{q(T)}\}$ denote the set of gaps. For each $i \in \{1, \dots, q(T)\}$ we also use G_i to denote the size of gap G_i . Note that $|q(T)| \leq k+m \leq 2n$ since each of the k fixed jobs induces a gap “left” to it and there are at most m gaps whose “right” limit is not created by a fixed job but by the limit of the planning horizon; in total, $|q(T)|$ is polynomially bounded in the instance size. The set of gaps is partitioned into *large* and *small* gaps via

$$\begin{aligned} G_L(T) &:= \{G \in G(T) | G > T/2\}, \\ G_S(T) &:= \{G \in G(T) | G \leq T/2\}. \end{aligned}$$

Let $q_L(T) := |G_L(T)|$, $q_S(T) := |G_S(T)|$ be the number of large and small gaps for target makespan T . Since there can be at most one large gap per machine, we have $q_L(T) \leq m$. We define

$$\begin{aligned} J_L(T) &:= \{i \in \{k+1, \dots, n\} | p_i \in (T/2, T]\}, \\ J_M(T) &:= \{i \in \{k+1, \dots, n\} | p_i \in (\epsilon T, T/2]\}, \\ J_S(T) &:= \{i \in \{k+1, \dots, n\} | p_i \in (0, \epsilon T]\} \end{aligned}$$

to partition the set of non-fixed jobs into *large*, *medium* and *small* jobs. Note $J_L(T)$ can be only packed into the at most m gaps from $G_L(T)$. Hence, if $|J_L(T)| > q_L(T)$, T is infeasible and can be safely discarded. In the sequel

we assume that there is no unnecessary idle time in the gaps, i.e. a set of jobs placed in a gap is scheduled as a continuous block which starts as early as possible and all idle time is positioned at the end of the gap.

2.2 Definition of Configurations for Medium Jobs.

We obtain few distinct job sizes in $J_M(T)$ via rounding. This construction is lossy in the sense that some jobs will not be executed in $[0, T)$ anymore; however, the loss of jobs will be suitably bounded. Fix a schedule σ with makespan T . Since $p_i > \epsilon T$ for each $i \in J_M(T)$ and the interval $[0, T)$ provides an amount of total processing time of at most mT ,

$$n' := |J_M(T)| \leq \lfloor \frac{mT}{\epsilon T} \rfloor = \lfloor \frac{m}{\epsilon} \rfloor \leq \frac{m}{\epsilon}$$

holds. We apply linear grouping as in [6] to the medium jobs in $J_M(T)$ by setting $c_1 := \lceil 1/\epsilon^2 \rceil$ and creating $c_1 + 1$ groups. We sort the medium jobs in $J_M(T)$ in non-increasing order of processing time and in this order create groups of cardinality $\lfloor n'/c_1 \rfloor$ where the last group is possibly of smaller cardinality. We denote the resulting groups by $C_1^M, \dots, C_{c_1+1}^M$. Next for each $i \in \{1, \dots, c_1 + 1\}$ the processing times of medium jobs in C_i^M are rounded up to the largest processing time occurring in the respective group, i.e. we define $q_i := \max\{p_j | j \in C_i^M\}$ and the resulting rounded groups are denoted by $\tilde{C}_1^M, \dots, \tilde{C}_{c_1+1}^M$. We remove C_1^M from σ , resulting in a *partial* schedule with makespan at most T and some free space. Note that by embedding \tilde{C}_i^M into the space for \tilde{C}_{i-1}^M for each $i \in \{2, \dots, c_1\}$, we can reschedule the medium jobs in $J_M(T) \setminus C_1^M$ based on the assignment in σ . We use σ_1 to denote the resulting partial schedule. Using this approach we have limited the number of distinct item sizes in $J_M(T)$ to c_1 at the cost of removing C_1^M from the schedule. In total, we have $|C_1^M| \leq n'/c_1$ and each job in C_1^M is no larger than $T/2$. Hence, via

$$P(C_1^M) \leq \frac{Tn'}{2c_1} \leq \frac{Tm}{2\epsilon c_1} \leq \frac{\epsilon Tm}{2}$$

the total size of C_1^M is suitably bounded. Note that $G \leq T$ for each gap G and $p_i > \epsilon T$ for each medium job $i \in J_M(T)$, hence consequently at most $\lfloor 1/\epsilon \rfloor$ medium jobs from $J_M(T)$ can occur in each gap in σ_1 . Now a *configuration* is a c_1 -tuple over $\{0, \dots, \lfloor 1/\epsilon \rfloor\}$ in which the i -th component denotes the number of items of size q_i . Each configuration naturally corresponds to a choice of rounded items from $J_M(T)$ which can occur together in a gap. Note that this definition also includes the “empty” configuration in which all entries are zero. For convenience, we will also refer to the corresponding set of medium jobs as a configuration.

Let c_2 denote the number of configurations which is bounded by $c_2 \leq (\lfloor 1/\epsilon \rfloor + 1)^{c_1}$ and hence independent from the encoding size of the input. We denote all configurations by $\kappa^{(1)}, \dots, \kappa^{(c_2)}$. Furthermore, for each $\ell \in \{1, \dots, c_2\}$, we denote by

$$s_\ell := \sum_{i=1}^{c_1} \kappa_i^{(\ell)} q_i$$

the total size of the ℓ -th configuration. So far, by removing C_1^M from σ , we have defined a partial schedule σ_1 with makespan at most T and simplified structure in which only a small amount of total processing time is not scheduled; furthermore, each job which is not scheduled is a medium job. We have established Lemma 2.1 and later use enumeration to find a suitable choice of configurations for a target makespan T .

LEMMA 2.1. *For every feasible makespan T there is a partial schedule σ_1 with makespan at most T and the following properties. Every large job from $J_L(T)$ is scheduled in a gap from $G_L(T)$. Every small job from $J_S(T)$ is scheduled. In each gap in $G_L(T)$ there are at most three objects, namely a large job from $J_L(T)$, a configuration and a set of small jobs from $J_S(T)$. There are only medium jobs from $J_M(T)$ which are not scheduled, and the total processing time of these is at most $\epsilon T m/2$.*

2.3 Discretization of Large Jobs. We discretize the large jobs in $J_L(T)$ which are packed together in a gap with a non-empty configuration in σ_1 from Lemma 2.1. The large jobs in $J_L(T)$ which are packed into a gap from $G_L(T)$ with the empty configuration in σ_1 are not discretized. The construction described here leaves the small jobs from $J_S(T)$ untouched and modifies only the large jobs and configurations in gaps in $G_L(T)$. We assume that in σ_1 in each large gap from $G_L(T)$ there is a job from $J_L(T)$; otherwise we introduce an artificial “large” job of size 0 for such a gap. Hence we obtain $|J_L(T)| = |G_L(T)| \leq m$, i.e. there are as many large jobs as large gaps. Now let $c_3 := \lceil 1/\epsilon \rceil - 1$, and for each $k \in \{1, \dots, c_3\}$ let $I_k(T) := (k\epsilon T, (k+1)\epsilon T]$. Finally for each $k \in \{1, \dots, c_3\}$, $\ell \in \{1, \dots, c_2\}$ let

$$J_L(T, k) := \{j \in J_L(T) | p_j \in I_k(T)\}$$

denote the set of large jobs with processing times in the interval $I_k(T)$. Note that in the above definition and the following construction, it is sufficient to have $k \geq \lceil 1/(2\epsilon) \rceil$ since every large job has a processing time larger than $T/2$; however, to keep the notation as simple as possible, we use $k \in \{1, \dots, c_3\}$. In each gap in $G_L(T)$ there are exactly three objects, namely a large

job, a configuration, and a set of small jobs which may be empty however. Let $\mathcal{I} := \{1, \dots, c_3\} \times \{1, \dots, c_2\}$ to denote the set of indices for intervals and configurations; let $(k, \ell) \in \mathcal{I}$. We define

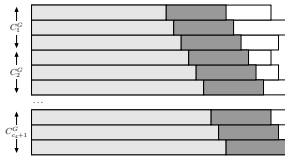
$$G_L(T, k, \ell) := \{G \in G_L(T) | \text{in } \sigma_1 \text{ gap } G \text{ contains a job from } J_L(T, k) \text{ and a non-empty } \kappa^{(\ell)}\}$$

and present a construction to round the large jobs from $J_L(T)$ contained in $G_L(T, k, \ell)$ under a small loss of total size of the medium jobs in $J_M(T)$; the approach is illustrated in Fig. 1. There, light grey areas indicate the large jobs, dark grey areas indicate the configurations, and white areas indicate small jobs or idle time. Conceptually arrange the gaps in $G(T, k, \ell)$ along with their contents in σ_1 in a vertical stack, starting at the top with a gap containing a job from $J_L(T, k)$ of smallest size to the bottom finishing with a gap containing a job from $J_L(T, k)$ of largest size. Except for the small jobs from $J_S(T)$, each of these gaps contains exactly two objects, namely a job from $J_L(T, k)$ and a configuration $\kappa^{(\ell)}$. If two such objects occur together in a large gap we will call them *associated*. Similar as in [6], we apply linear grouping to the stack. More precisely, we set $c_4 := \lceil 1/\epsilon \rceil$; beginning from the top, we create $c_4 + 1$ groups $C_1^G, \dots, C_{c_4+1}^G$ of size $\lfloor |G_L(k, \ell)|/c_4 \rfloor$ where the last group is possibly of smaller cardinality. In each group C_i^G , each large job is rounded up to the size of the largest large job occurring in C_i^G , namely to

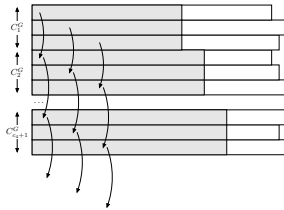
$$q'_j := \max\{p_j | j \in J_L(T), j \text{ occurs in a gap from } C_i^G\}$$

and it remains to show that the rounded large jobs in the stack can be packed together with nearly all of their associated configurations. To this end, we use an elegant cyclic shifting argument which is sketched in Fig. 1. Each rounded job from $J_L(T, k)$ is shifted downwards $\lfloor |G(k, \ell)|/c_4 \rfloor$ gaps in the stack into at least the next larger group, where it can be safely packed together with the configuration of size s_ℓ . The large jobs in the $\lfloor |G(T, k, \ell)|/c_4 \rfloor$ gaps at the bottom are pushed out of the stack. Hence, a total number of $\lfloor |G(T, k, \ell)|/c_4 \rfloor$ large jobs from $J_L(T)$ not packed. We remove the configurations in group C_1^G and denote the set of non-packed associated configurations by $I(T, k, \ell)$; the set $I(T, k, \ell)$ is removed from the schedule. Now the uppermost $\lfloor |G(T, k, \ell)|/c_4 \rfloor$ gaps in the stack are empty. Every non-packed configuration has a total size of at most $T/2$ since it was packed together with a large job in σ_1 ; consequently, we can use

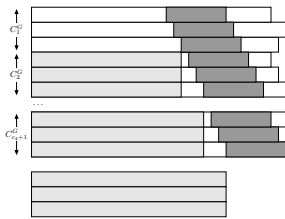
$$(1) \quad P(I(T, k, \ell)) \leq \frac{T}{2} \lfloor |G(T, k, \ell)|/c_4 \rfloor \leq \frac{T |G(T, k, \ell)|}{2c_4}$$



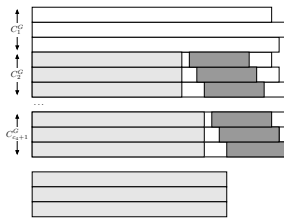
(a) Create a stack for $G(T, k, \ell)$ by starting at the top with a gap containing a large job of smallest size and finishing at the bottom with a gap containing a large job of largest size. Then we create $c_4 + 1$ groups of equal size, except for the last group.



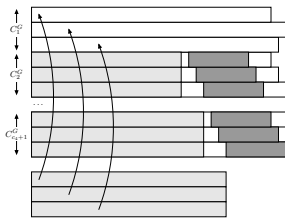
(b) The size of the large jobs is rounded up to the largest job size occurring in the respective group; the configurations are not shown due to overlap. The large jobs are rearranged by shifting them in the next lower group; the bottommost large jobs are pushed out of the stack.



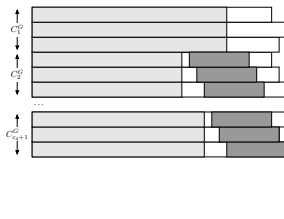
(c) In the resulting arrangement there is no overlap between rounded large jobs in the stack and configurations. Here the configurations are shown again.



(d) To accommodate the rounded large jobs from below the stack, the configurations in the topmost group C_1^G are removed. The space created there is large enough to contain the rounded large jobs from below the stack.



(e) The rounded large jobs from below the stack are shifted in the gaps of the topmost group C_1^G .



(f) In the final arrangement all rounded large jobs from $J_L(T, k)$ are packed. The only jobs which are not packed are the configurations from the first group C_1^G .

Figure 1: This sketch illustrates the construction from Subject. 2.3; light grey areas indicate the large jobs from $G(T, k, \ell)$, dark grey areas indicate the associated configurations of type $\kappa^{(\ell)}$ and white areas indicate small jobs or idle time.

to bound the total processing time of the non-packed configurations. Concerning the $\lfloor |G(T, k, \ell)|/c_4 \rfloor$ large jobs from $J_L(T, k)$ which are not packed, we note that the size of each of them is at most $(k + 1)\epsilon T$. The size of each of the empty gaps in the upper parts of the stack is at least $k\epsilon T + \epsilon T = (k + 1)\epsilon T$. Consequently, the large jobs which have been pushed out of the stack can be feasibly put in the uppermost gaps, i.e. the gaps in C_1^G .

LEMMA 2.2. *By applying the rounding and cyclic shifting for all interval indices and configuration types $(k, \ell) \in \mathcal{I}$, only medium jobs from $J_M(T)$ are lost. The total processing time of these is bounded by $\epsilon T m/2$.*

Proof. By construction only medium jobs from $J_M(T)$ are not packed. Now we use the bound (1), the estimation $\sum_{(k, \ell) \in \mathcal{I}} |G(T, k, \ell)| \leq |G_L(T)| \leq m$, and the definition of c_4 to obtain the chain of inequalities

$$\begin{aligned} P(\cup_{(k, \ell) \in \mathcal{I}} I(T, k, \ell)) &= \sum_{(k, \ell) \in \mathcal{I}} P(I(T, k, \ell)) \\ &\leq T/(2c_4) \sum_{(k, \ell) \in \mathcal{I}} |G(T, k, \ell)| \leq \frac{T}{2c_4} |G_L(T)| \\ &\leq \frac{Tm}{2c_4} \leq \frac{\epsilon Tm}{2} \end{aligned}$$

which yields the claim. \square

We use σ_2 to denote the resulting partial schedule in which all jobs which are now removed, more precisely the medium jobs in $\cup_{(k, \ell) \in \mathcal{I}} I(T, k, \ell)$, do not occur. For each $(k, \ell) \in \mathcal{I}$ let

$$c(k, \ell) := \{ |G \in G_L(T)| \text{ in } \sigma_2 \text{ gap } G \text{ contains a job from } J_L(T, k) \text{ and a non-empty } \kappa^{(\ell)} \} \leq m$$

denote the number of large jobs from $J_L(T)$ with processing times in the interval $I_k(T)$ which are packed together with the ℓ -th configuration in σ_2 . In total we obtain the following result.

LEMMA 2.3. *For every feasible makespan T there is a partial schedule σ_2 with makespan at most T and the following properties. Every large job from $J_L(T)$ is scheduled in a gap from $G_L(T)$. Every small job from $J_S(T)$ is scheduled. In each large gap from $G_L(T)$ there are exactly three objects, namely a large job from $J_L(T)$, a configuration and a subset of small jobs from $J_S(T)$. For each $(k, \ell) \in \mathcal{I}$ there is an integer $c(k, \ell) \leq m$ which indicates how often a large job from $I_k(T)$ is packed together with a non-empty configuration $\kappa^{(\ell)}$. There are only medium jobs from $J_M(T)$ which are not scheduled, and the total processing time of these is at most $\epsilon T m/2 + \epsilon T m/2 = \epsilon T m$.*

Clearly, there are at most $m^{c_2 c_3}$ choices for the values $c(k, \ell)$, hence these can be found by enumeration. However, algorithmically we have to deal with the problem that, even if the values $c(k, \ell)$ are known, it is difficult to find the assignment of large jobs in $J_L(T)$ and associated configurations exactly as in σ_2 . However, with Lemma 2.4, we will show that by using a straightforward greedy argument for the small jobs in $J_S(T)$, any feasible assignment of the large jobs in $J_L(T)$ and the associated configurations to the gaps in $G_L(T)$ can be extended to a partial schedule under a small loss of processing time. To this end we define a multiset of large jobs and configurations; more precisely let

$$J_{LC}(T) := \{j \in J_L(T) \mid \text{job } j \text{ is rounded as in } \sigma_2\} \\ \cup \{\kappa^{(\ell)} \mid \kappa^{(\ell)} \text{ occurs in a gap in } G_L(T) \text{ in } \sigma_2\}$$

denote the large jobs and configurations as they occur in the large gaps in σ_2 . We obtain the following result.

LEMMA 2.4. *Let T be a feasible makespan. Let σ_3 be a partial schedule of makespan at most T which assigns exactly the large and medium jobs in $J_{LC}(T)$ to the large gaps in $G_L(T)$ in any feasible way. Then σ_3 can be extended to a partial schedule σ_4 with makespan at most T and the following properties. Every large job from $J_L(T)$ is scheduled in a large gap from $G_L(T)$. There are only medium and small jobs from $J_M(T) \cup J_S(T)$ which are not scheduled of total processing time at most $\epsilon T m / 2 + \epsilon T m / 2 + \epsilon T m = 2\epsilon T m$.*

Proof. Let σ_2 denote the schedule from Lemma 2.3. Let

$$I_S := \{j \in J_S(T) \mid \text{job } j \text{ is scheduled in a gap from } G_L(T) \text{ in } \sigma_2\}.$$

Remove the small jobs from I_S in σ_2 and let

$$P_{LG} := \sum_{i=1}^{q_L(T)} G_i - P(J_{LC}(T))$$

denote the remaining free processing time in the gaps from $G_L(T)$ after I_S is removed. Clearly we have $P(I_S) \leq P_{LG}$. From the resulting schedule remove all jobs from $J_{LC}(T)$ and reschedule them again as in σ_3 . Clearly, this does not change the total load in $G_L(T)$, i.e. in $G_L(T)$ there is still an amount of P_{LG} of free processing time. Since $P(I_S) \leq P_{LG}$, we can distribute the small jobs in I_S to the gaps in $G_L(T)$ in a first fit manner, fractionalizing jobs which cannot be accommodated completely. In this way, at most $|G_L(T)| - 1 \leq m$ small jobs are fractionalized. The set of these is called S and is removed from the schedule; the resulting schedule is denoted by σ_4 . Since

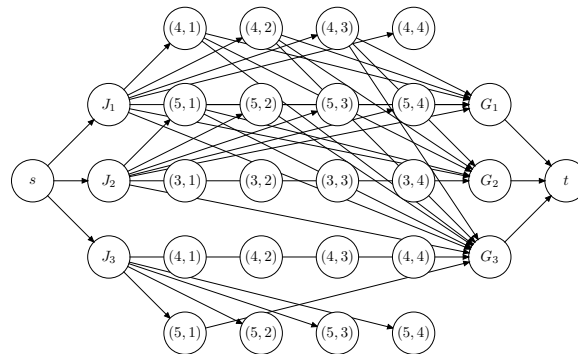


Figure 2: Sketch of the network used for assignment of large jobs and configurations. The example for $\epsilon = 1/9$ shows three large jobs with processing times $J_1 = 5/9$, $J_2 = 6/9$, and $J_3 = 9/9$. The configurations are of sizes $s_1 = 1/9$, $s_2 = 3/9$, $s_3 = 5/9$, and $s_4 = 6/9$. The three large gaps are of sizes $G_1 = 7/9$, $G_2 = 8/9$, and $G_3 = 9/9$. Interval-configuration nodes with interval index smaller than 4 are not shown since they are not connected to any job node by construction.

for each $j \in S$ we have $p_j \leq \epsilon T$ and $|S| \leq m$, we have $P(S) \leq \epsilon T m$. Furthermore, except for the gaps in $G_L(T)$, σ_4 is identical to σ_2 and the jobs in $J_{LC}(T)$ are scheduled as in σ_3 . \square

2.4 Assignment of Jobs to Large Gaps via Network Flow. In Lemma 2.4 we have argued that basically it is not important how the large and medium jobs in $J_{LC}(T)$ are packed into the gaps in $G_L(T)$ once the set $J_{LC}(T)$ is known. In this subsection we show how both the selection of suitable configurations and the assignment to the gaps can be done via enumeration of a class of network flow models. Given a makespan T we use a network flow model to find a feasible assignment, if one exists, for $J_L(T)$ and the associated configurations, implicitly given by the values $c(k, \ell) \leq m$ for $(k, \ell) \in \mathcal{I}$. We define a directed acyclic graph $N(T) = (V(T), E(T))$ where $V(T)$ consists out of five layers; the construction is sketched in Fig. 2. In the first layer there is only s , the *source* node. In the second layer there are the at most m large jobs from $J_L(T)$ (*job* nodes). In the third layer there is a set of nodes to govern the assignment of large jobs and configurations, namely exactly \mathcal{I} (*interval-configuration* nodes). In the fourth layer there are the at most m large gaps from $G_L(T)$ (*gap* nodes). In the fifth layer there is only t , the *terminal* node. The *arc set* $E(T)$ is defined as follows. The source s is connected to each large job node, i.e. $(s, j) \in E(T)$ for each $j \in J_L(T)$ (arcs connect layers 1 and 2). Each job node is connected to an interval-

configuration node if it is contained in the interval, i.e. $(j, (k, \ell)) \in E(T) \Leftrightarrow p_j \in I_k(T)$ for $j \in J_L(T)$, $(k, \ell) \in \mathcal{I}$ (arcs connect layers 2 and 3). Each interval-configuration node is connected to a gap node if each possibly rounded job from the respective interval can be packed together with the configuration into the gap, i.e. $((k, \ell), G) \in E(T) \Leftrightarrow (k+1)\epsilon T + s_\ell \leq G$ for each $(k, \ell) \in \mathcal{I}$ and $G \in G_L(T)$ (arcs connect layers 3 and 4). Each gap node is connected to the terminal node, i.e. $(G, t) \in E(T)$ for each $G \in G_L(T)$ (arcs connect layers 4 and 5). Each job node is connected to each gap node it fits into, i.e. precisely $(j, G) \in E(T) \Leftrightarrow p_j \leq G$ for each $j \in J_L(T)$, $G \in G_L(T)$ (arcs connect layers 2 and 4). The arcs in $E(T)$ are endowed with a lower capacity 0 and an upper capacity of 1. Finally for each $(k, \ell) \in \mathcal{I}$ we require the flow in the interval-configuration node (k, ℓ) to be $c(k, \ell)$ which can be done by expanding a node to two nodes connected by an artificial edge with suitable flow constraints. Note that the encoding size of $N(T)$ is polynomially bounded in the encoding size of the instance. Furthermore $N(T)$ has an optimal s - t -flow of value $|J_L(T)|$ if and only if $J_L(T)$ together with the selected configurations, implicitly given by the values $c(k, \ell)$, can be packed into the gaps in $G_L(T)$; a corresponding packing is then given in a natural way via the network flow. However, the values $c(k, \ell)$ for each $(k, \ell) \in \mathcal{I}$ have to be enumerated. We have $|G_L(T)| \leq m$ and hence at most m large jobs from $J_L(T)$ can be packed together with a certain configuration; consequently $c(k, \ell) \leq m$ holds for each $(k, \ell) \in \mathcal{I}$. Furthermore there is only a constant number of at most $c_2 c_3$ nodes in the third layer. Since each of these nodes gets assigned a capacity $c(k, \ell) \in \{0, \dots, m\}$, the quantity $(m+1)^{c_2 c_3}$ is an upper bound for the number of possible assignments of flow restrictions on configuration-interval nodes.

2.5 Packing of Medium and Small Jobs. Let $I := \{k+1, \dots, n\}$. Given the large jobs $J_L(T)$, all configurations $\kappa^{(1)}, \dots, \kappa^{(c_2)}$, and a suitable choice of values $c(k, \ell)$ for each $(k, \ell) \in \mathcal{I}$ we can use the network flow model from Subsect. 2.4 to find a feasible assignment, if one exists, of the large jobs and associated configurations to the gaps in $G_L(T)$. Let I_1 denote the set of jobs assigned in this way and denote by σ_3 the corresponding partial schedule; let $I_2 := I \setminus I_1$. The assignment in σ_3 is done without unnecessary idle time in the large gaps and is fixed in the candidate solution, i.e. we aim at extending σ_3 . Consequently, the large gaps in $G_L(T)$ become smaller since σ_3 assigns some jobs there, as sketched in Fig. 3. More precisely, we denote by S_i the set of jobs scheduled in the large gap G_i for $i \in \{1, \dots, q_L(T)\}$ and introduce new gaps \tilde{G}_i of

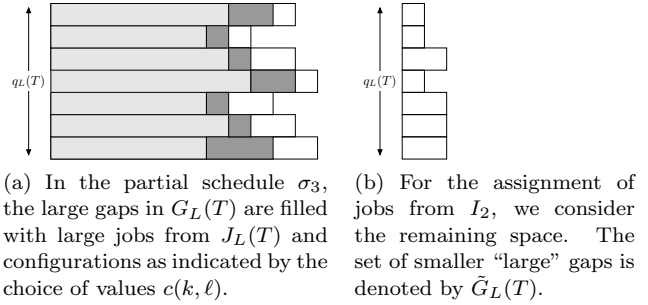


Figure 3: This sketch illustrates the large gaps in the schedule σ_3 . Light grey areas indicate large jobs, dark grey areas indicate configurations, and white areas indicate idle time.

sizes

$$\tilde{G}_i := G_i - \sum_{j \in S_i} p_j$$

for each $i \in \{1, \dots, q_L(T)\}$. Furthermore we use $\tilde{G}_L(T) := \{\tilde{G}_1, \dots, \tilde{G}_{q_L(T)}\}$ to denote the set of new gaps and let $G'(T) := \tilde{G}_L(T) \cup G_S(T)$. Now G' is to be algorithmically filled with jobs from I_2 . If T is feasible, by Lemma 2.4 there is a subset $I_3 \subseteq I_2$ such that I_3 can be scheduled in $G'(T)$ and $P(I_1) + P(I_3) \geq P(I) - 2\epsilon Tm$ holds; let I_3 be chosen as such. We use MSSPPTAS to select $I_4 \subseteq I_2$ such that $P(I_4) \geq (1 - \epsilon)P(I_3)$. In total we obtain

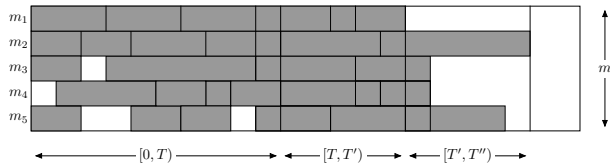
$$\begin{aligned} P(I_1) + P(I_4) &\geq P(I_1) + (1 - \epsilon)P(I_3) \\ &\geq (1 - \epsilon)(P(I_1) + P(I_3)) \geq (1 - \epsilon)(P(I) - 2\epsilon Tm) \end{aligned}$$

unless T is infeasible and can safely be discarded. In total, for the optimal makespan $T = C_{\max}^*$ and a suitable choice of values $c(k, \ell)$ we can schedule a total load of at least $(1 - \epsilon)(P(I) - 2\epsilon Tm)$ in $[0, T)$. Hence after T there remains a total processing time of at most

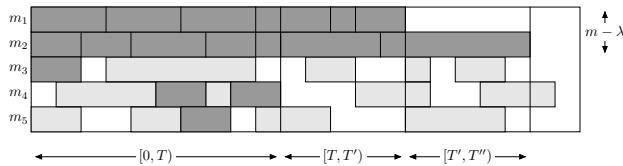
$$P(I) - (1 - \epsilon)(P(I) - 2\epsilon Tm) \leq 2\epsilon Tm + \epsilon P(I)$$

to schedule. We can use any list scheduling algorithm to execute this small load in the interval $[T, \infty)$; the following analysis is sketched in the upper part of Fig. 4. Let T' denote the last step in $[T, \infty)$ where there is no idle machine and let T'' denote the last time step in $[T', \infty)$ where there is a busy machine. Now we use the well-known Graham bounds. Here we obtain $|[T', T'']| \leq T/2 \leq C_{\max}^*/2$ for the last part of the schedule and

$$\begin{aligned} |[T, T']| &\leq \frac{2\epsilon Tm + \epsilon P(I)}{m} \leq \frac{2\epsilon Tm + \epsilon m C_{\max}^*}{m} \\ &\leq \frac{2\epsilon m C_{\max}^* + \epsilon m C_{\max}^*}{m} = 3\epsilon C_{\max}^* \end{aligned}$$



(a) Illustration of the list scheduling from Sect. 2. The jobs in the interval $[0, T]$ are scheduled via the technique described in Subsect. 2.1–2.4 and MSSPPTAS; the jobs in $[T, T'']$ are assigned via list scheduling, hence in $[T, T']$ there is no idle time.



(b) Illustration of the list scheduling from Sect. 3. The jobs in the interval $[0, T]$ are scheduled via the technique described in Subsect. 2.1–2.4 and MSSPPTAS; the jobs in $[T, T'']$ are assigned via list scheduling, hence in $[T, T']$ there is no idle time on the first $m - \lambda$ machines.

Figure 4: Illustration of both analyses carried out in Sect. 2 & 3 for the respective list scheduling which finally arranges the hitherto non-scheduled jobs. The jobs are indicated by dark grey areas while light grey areas indicate the periods of non-availability; white areas indicate idle time.

for the middle part of the schedule, hence the makespan of our algorithmically generated schedule can be bounded by

$$\begin{aligned}
 & |[0, T]| + |[T, T']| + |[T', T'']| \\
 & \leq C_{\max}^* + 3\epsilon C_{\max}^* + \frac{1}{2}C_{\max}^* = (3/2 + 3\epsilon)C_{\max}^*.
 \end{aligned}$$

By carrying out the entire construction from Subsect. 2.1–2.5 with $\epsilon' := \epsilon/3$ instead of ϵ , we have established Theorem 1.1 and shown our main result.

3 Scheduling with Non-Availability

Here we describe how our approach can be applied to scheduling with non-availability; the idea is basically the same as for scheduling with fixed jobs, but results in a construction which is slightly more technical in nature. The main reason for this is that, in terms of complexity, scheduling with fixed jobs and non-availability behave differently. The general problem of scheduling with non-availability without any further restriction does not admit a constant approximation ratio unless $P = NP$ holds; this follows from the fact that scheduling with non-availability for m constant is also inapproximable

unless $P = NP$ [4] or, alternatively, from the fact that scheduling parallel jobs on parallel machines with non-availability is inapproximable unless $P = NP$ [5]. Earlier, Lee [21] only pointed out that LPT performs arbitrarily badly. In either case the inapproximability is due to the permission of time steps where no machine is available. Since the periods of non-availability do not contribute to the makespan, scheduling with non-availability admits a gap-creating reduction which separates the objective values of optimal solutions and suboptimal solutions of yes-instances. However, the restriction to instances where for each time step there is an available machine is not sufficient to obtain a constant approximation ratio, as can be seen via a reduction from Equal Cardinality Partition; the proof is omitted for space reasons.

THEOREM 3.1. *Scheduling with non-availability, even if for each time step there is an available machine, does not admit a polynomial time algorithm with a constant approximation ratio unless $P = NP$.*

Consequently we assume that at least one machine is always available. The algorithm we are about to present will use the assumption that the percentage ρ of permanently available machines is constant. Surprisingly, even this severe restriction is algorithmically hard to approximate. Lemma 3.1 yields the inapproximability result from Theorem 1.2; the proof is omitted due to lack of space.

LEMMA 3.1. *Scheduling with non-availability, even if the ratio $\rho \in (0, 1)$ of permanently available machines is constant, does not admit a polynomial time approximation algorithm with absolute approximation ratio $3/2 - \epsilon$, unless $P = NP$, for any $\epsilon \in (0, 1/2)$.*

Without the restriction of a constant percentage of machines being permanently available, scheduling with non-availability yields an interesting connection to the well-known problem Bin Packing; the existence of an approximation algorithm for scheduling with non-availability with constant ratio implies the existence of an approximation algorithm for Bin Packing with additive error. However, this is an open problem, as discussed in [7], Chapt. 2, page 67. Theorem 3.2 can be seen as an informal reason for scheduling with non-availability being hard to approximate.

THEOREM 3.2. *Suppose there is a polynomial time algorithm for scheduling with non-availability with absolute approximation ratio $c \in \mathbb{N} \setminus \{1\}$. Then there is a polynomial time algorithm for Bin Packing with additive error $2(c - 1)$.*

Proof. Let A be an algorithm for scheduling with non-availability with approximation ratio $c \in \mathbb{N} \setminus \{1\}$. For each instance I of Bin Packing with n items and bin size b we define n instances I'_i for $i \in \{1, \dots, n\}$ of scheduling with non-availability by setting $m = i$ and defining intervals of non-availability $(j + 1, b)$ of size ∞ for each $j \in \{1, \dots, i - 1\}$. Note each I'_i can be generated from I within a polynomial runtime bound. For each instance I of Bin Packing, n is an upper bound for $\text{OPT}(I)$, the minimum number of bins in which the items of I can be packed. Let I be an instance of Bin Packing. Let $n' := \min\{i \in \{1, \dots, n\} \setminus \{1\} \mid A(I'_i) \leq cb\}$ which can be found in polynomial time by enumeration. Hence $A(I'_{n'-1}) > cb$, from which follows $C_{\max}^*(I'_{n'-1}) > b$. This means that it is impossible to pack the items of I in less than n' bins of size b , hence $\text{OPT}(I) \geq n'$ holds. Consider the schedule for $I'_{n'}$ generated by A . The schedule for the machines $2, \dots, n'$ yields $n' - 1$ bins. Furthermore, the jobs scheduled on machine 1 can be packed in $1 + 2(c - 1)$ bins by packing all jobs from intervals of the form $[\ell b, (\ell + 1)b)$ into one bin and packing each job crossing the boundaries of such adjacent intervals into a separate bin. In total, the number of bins needed for this packing can be bounded by $n' - 1 + 1 + 2(c - 1) \leq \text{OPT}(I) + 2(c - 1)$, hence the approach yields an algorithm for Bin Packing which uses at most $2(c - 1)$ additional bins. \square

Now we present the approximation algorithm for scheduling with non-availability where the ratio of permanently available machines is constant. Similar to [11], we use $\lambda \in \{1, \dots, m - 1\}$ to denote the number of machines which are permitted to be temporarily unavailable. Since the machines are identical, we assume that the *first* $m - \lambda$ machines are permanently available; in total, $\rho = (m - \lambda)/m = 1 - \lambda/m$ is the percentage of permanently available machines. As for scheduling with fixed jobs, the running time of our algorithm will depend polynomially on m , which yields a polynomially bounded running time with the same argument as before. Next we obtain the algorithm mentioned in Theorem 1.2. Let $I := \{1, \dots, n\}$; for scheduling with non-availability, $P(I) \leq np_{\max}$ yields an upper bound for the optimal makespan since all jobs can be scheduled on the permanently available machine in the time interval $[0, P(I))$. Similar as before we perform binary search for the makespan in $[0, P(I))$, which yields a suitable makespan in $O(\log(np_{\max}))$ steps. The gap classification for a target makespan T is done as in Subsect. 2.1, yielding $G_L(T)$ and $G_S(T)$; likewise, the partition into large, medium and small gaps is done as in Subsect. 2.1. We proceed as before by defining configurations for medium jobs as in Subsect. 2.2; the rounding results in a loss of processing time of at most $\epsilon Tm/2$, but still

all large jobs are scheduled. The discretization of large jobs is carried out as in Subsect. 2.3 which again results in an additional loss of total load $\epsilon Tm/2$ by using the enumeration of network flow models as in Subsect. 2.4; by guessing the assignment of large jobs to large gaps we lose again an amount of ϵTm of processing time. In the innermost loop of our algorithm, we pack the remaining small jobs using MSSPPTAS from [1, 13]. In total, for the optimal target makespan $T = C_{\max}^*$ a total amount of processing time of at least $(1 - \epsilon)(P(I) - 2\epsilon Tm)$ can be scheduled in the interval $[0, T)$; consequently, we have only medium and small jobs with total processing time of at most $2\epsilon Tm + \epsilon P(I)$ to schedule. We use list scheduling to pack the remaining jobs in the time interval $[T, \infty)$ on the first $m - \lambda$ machines which are free by assumption; let $M := \{1, \dots, m - \lambda\}$ denote the set of these. The analysis is illustrated in the lower part of Fig. 4. Similar as in Subsect. 2.5, let T' denote the last step in $[T, \infty)$ where there is no idle machine in M and let T'' denote the last time step in $[T', \infty)$ where there is a busy machine in M . Again we use the Graham bounds and obtain $||[T', T'']| \leq T/2 \leq C_{\max}^*/2$ and

$$\begin{aligned}
 |[T, T']| &\leq \frac{2\epsilon Tm + \epsilon P(I)}{m - \lambda} \leq \frac{2\epsilon Tm + \epsilon m C_{\max}^*}{m - \lambda} \\
 &\leq \frac{2\epsilon C_{\max}^* + \epsilon C_{\max}^*}{1 - \lambda/m} = \frac{3\epsilon}{1 - \lambda/m} C_{\max}^*,
 \end{aligned}$$

hence the makespan of our algorithmically generated schedule can be bounded by

$$\begin{aligned}
 |[0, T]| + |[T, T']| + |[T', T'']| \\
 \leq C_{\max}^* + \frac{3}{1 - \lambda/m} \epsilon C_{\max}^* + \frac{1}{2} C_{\max}^* \\
 = (3/2 + \frac{3}{1 - \lambda/m} \epsilon) C_{\max}^*.
 \end{aligned}$$

In total, we carry out the entire construction with $\epsilon' := \epsilon(1 - \lambda/m)/3 = \epsilon\rho/3$ instead of ϵ ; here we use the assumption that $1 - \lambda/m$ is constant to obtain a polynomial runtime bound. Finally, we have established Theorem 1.2 and shown our second main result.

4 Conclusion

We have studied non-preemptive scheduling with fixed jobs and non-availability where the objective is to minimize the makespan. For scheduling with fixed jobs we obtained a polynomial time algorithm with ratio $3/2 + \epsilon$; for this problem, an approximation ratio of $3/2 - \epsilon$ is impossible unless $P = NP$ holds. For our algorithm we have developed a novel technique of discretization and flow-based assignment of large jobs. This technique can also be used for scheduling

with non-availability where a constant percentage of the machines is permanently available; there it also yields an approximation algorithm with ratio $3/2 + \epsilon$ while a ratio of $3/2 - \epsilon$ is impossible unless $P = NP$ holds. In total, in a certain sense our approach yields two tight approximation results. However, for both problems under consideration, it is an open problem whether an approximation ratio of *exactly* $3/2$ can be obtained.

References

- [1] A. Caprara, H. Kellerer, and U. Pferschy. A PTAS for the multiple subset sum problem with different knapsack capacities. *Information Processing Letters*, 73(3-4):111–118, 2000.
- [2] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *SODA*, pages 213–222, 2000.
- [3] C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.
- [4] F. Diedrich, K. Jansen, F. Pascual, and D. Trystram. Approximation algorithms for scheduling with reservations. In S. Aluru, M. Parashar, R. Badrinath, and V. K. Prasanna, editors, *HiPC*, volume 4873 of *LNCS*, pages 297–307. Springer, 2007.
- [5] L. Eyraud-Dubois, G. Mounie, and D. Trystram. Analysis of scheduling algorithms with reservations. In *IPDPS*, pages 1–8. IEEE, 2007.
- [6] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.
- [7] D. S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1996.
- [8] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.
- [9] 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.
- [10] H.-C. Hwang and S. Y. Chang. Parallel machines scheduling with machine shutdowns. *Computers and Mathematics with Applications*, 36(3):21–31, 1998.
- [11] H.-C. Hwang, K. Lee, and S. Y. Chang. The effect of machine availability on the worst-case performance of LPT. *Discrete Applied Mathematics*, 148(1):49–61, 2005.
- [12] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.
- [13] K. Jansen. Parameterized approximation scheme for the multiple knapsack problem. To appear in Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, USA, January 4–6, 2009.
- [14] I. Kacem. Approximation algorithms for the makespan minimization with positive tails on a single machine with a fixed non-availability interval. *Journal of Combinatorial Optimization*, 2007.
- [15] H. Kellerer. Algorithms for multiprocessor scheduling with machine release times. *IIE Transactions*, 30(11), 1998.
- [16] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.
- [17] C. Kenyon and E. Rémila. Approximate strip packing. In *FOCS*, pages 31–36, 1996.
- [18] C. Kenyon and E. Rémila. A near-optimal solution to a two dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.
- [19] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.
- [20] C.-Y. Lee. Parallel machines scheduling with non-simultaneous machine available time. *Discrete Applied Mathematics*, 30:53–61, 1991.
- [21] C.-Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization, Special Issue on Optimization of Scheduling Applications*, 9:363–384, 1996.
- [22] C.-Y. Lee, Y. He, and G. Tang. A note on “parallel machine scheduling with non-simultaneous machine available time”. *Discrete Applied Mathematics*, 100(1-2):133–135, 2000.
- [23] J. Y.-T. Leung, editor. *Handbook of Scheduling*. Chapman & Hall, 2004.
- [24] C.-J. Liao, D.-L. Shyur, and C.-H. Lin. Makespan minimization for two parallel machines with an availability constraint. *European Journal of Operational Research*, 160:445–456, 2003.
- [25] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.
- [26] N. Megow, R. H. Möhring, and J. Schulz. Turnaround scheduling in chemical manufacturing. In *In Proceedings of the 8th Workshop on Models and Algorithms for Planning and Scheduling Problems, MAPSP 2007, Istanbul, Turkey*, 2007.
- [27] E. Sanlaville and G. Schmidt. Machine scheduling with availability constraints. *Acta Informatica*, 35(9):795–811, 1998.
- [28] M. Scharbrodt. *Produktionsplanung in der Prozessindustrie: Modelle, effiziente Algorithmen und Umsetzung*. PhD thesis, Fakultät für Informatik, Technische Universität München, 2000.
- [29] M. Scharbrodt, A. Steger, and H. Weisser. Approximability of scheduling with fixed jobs. In *SODA*, pages 961–962, 1999.
- [30] M. Scharbrodt, A. Steger, and H. Weisser. Approximability of scheduling with fixed jobs. *Journal of Scheduling*, 2:267–284, 1999.