INSTITUT FÜR INFORMATIK

Tight Approximation Algorithms for Scheduling with Fixed Jobs and Non-Availability

Florian Diedrich
Klaus Jansen
Ulrich M. Schwarz
Ola Svensson

Bericht Nr. 1011
September 2010

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Tight Approximation Algorithms for Scheduling with Fixed Jobs and Non-Availability*

Florian Diedrich† Klaus Jansen‡ Ulrich M. Schwarz§
Ola Svensson¶

September 1, 2010

Abstract

We study two closely related problems in non-preemptive scheduling of 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 both cases, 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. 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.

We use dual approximation, creation of a gap structure and a PTAS for the multiple subset sum problem, combined with a post-processing step to assign large jobs.

*A preliminary version of this work with the title "Improved Approximation Algorithms for Scheduling with Fixed Jobs" was presented at the 20th ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York Marriott Downtown, USA, January 4–6, 2009. The first three authors were supported in part by project AEOLUS, "Algorithmic Principles for Building Efficient Overlay Computers", EU contract 015964.

†Institut für Informatik, Universität zu Kiel, Christian-Albrechts-Platz 4, D-24098 Kiel, Germany, fdi@informatik.uni-kiel.de. Supported in part by a grant "DAAD Doktorandenstipendium" of the German Academic Exchange Service. Part of this work was done while visiting the LIG, Grenoble University.

‡Institut für Informatik, Universität zu Kiel, Christian-Albrechts-Platz 4, D-24098 Kiel, Germany, kj@informatik.uni-kiel.de. Supported in part by DFG priority program 1126, "Algorithmik großer und komplexer Netzwerke", DFG contract SR7/9.

§Institut für Informatik, Universität zu Kiel, Christian-Albrechts-Platz 4, D-24098 Kiel, Germany, ums@informatik.uni-kiel.de.

¶Nada, KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden, osven@kth.se.

1 Introduction

In parallel machine scheduling, an important issue is the scenario where either some jobs are already fixed in the system [30] or intervals of non-availability of some machines must be taken into account [5, 6, 13, 22, 24, 25]. 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 [27] and distributed 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 $p_1, \ldots, p_n \in \mathbb{N}$. The first k jobs are fixed via a list $(m_1, s_1), \ldots, (m_k, s_k)$ 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 guaranteed to be 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* [6, 22, 24, 25]. The makespan $C_{\max}$ is one of the most well-studied objectives in the field of scheduling; for this objective, most problem formulations permit good approximation algorithms. However, both problems generalize the well-known problem $P||C_{\max}$ [11] and hence are strongly NP-hard and also hard to approximate.

Results. Scheduling with fixed jobs was studied by Scharbrodt, Steger & Weisser [29, 30]. They mainly studied the problem for constant m; for this strongly NP-hard formulation (which consequently does not admit an FPTAS) they present a PTAS. They also found approximation algorithms for general m with ratios 3 [29] and $2 + \epsilon$ [30]; since the finishing time of the last fixed job is a lower bound for the optimal makespan $C^*_{\max}$, we can simply use a PTAS for the well-known problem $P||C_{\max}$ [11] 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 $\mathsf{P} = \mathsf{NP}$, for any

$\epsilon \in (0, 1/2]$. Complementing this negative result, we obtain a tight ratio with our new approach.

Theorem 1. *Scheduling with fixed jobs admits an approximation algorithm with ratio* $3/2$.

Unlike scheduling with fixed jobs, scheduling with non-availability without any further restriction is inapproximable within a constant ratio unless $\mathsf{P} = \mathsf{NP}$, as shown by Eyraud-Dubois, Mounié & Trystram [7]. 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. [6]. 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 [21] and Lee et al. [23] 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 [18] found an algorithm with a tight ratio of $5/4$. Furthermore, Hwang et al. [13] briefly pointed out that this problem admits a PTAS. If the beginning of the intervals is not constrained in this way, Lee [22] 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. [13] 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 $\lambda \leq m/2$ [12] which they later refined to a ratio of $1 + \lceil 1/(1 - \lambda/m) \rceil /2$ for λ arbitrary [13]. For $\lambda = m - 1$, this yields $1 + m/2$; if $\rho = (m - \lambda)/m$ 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 [24, Chapt. 22], or [28] for surveys. For the sake of completeness, some results about single-machine problems can be found in the articles [6, 17, 21]. Finally, for scheduling with non-availability, our new technique yields an improved approximation ratio independent from ρ which is tight.

Theorem 2. *Scheduling with non-availability, where the percentage* $\rho \in (0, 1)$ *of permanently available machines is constant, admits an approximation algorithm with ratio* $3/2$. *Furthermore, for this problem there is no approximation algorithm with ratio* $3/2 - \epsilon$, *unless* $\mathsf{P} = \mathsf{NP}$, *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, the complexity of this is an interesting open problem, as discussed in [9, Chapt. 2, page 67].

Techniques used in our approach. Our approach strongly relies on algorithms for the multiple subset sum problem (MSSP) from [1]: in this problem, we are given n items with sizes $w_1, \ldots, w_n$ and $m \leq n$ target capacities $C_1, \ldots, C_m$, possibly not all equal, and are asked to find a partition of the items into $m+1$ sets $S_1, \ldots, S_{m+1}$ such that $\sum_{j \in S_i} w_j \leq C_i$ for all $i \in \{1, \ldots, m\}$ and $\sum_{i=1}^{m} \sum_{j \in S_i} w_j$ is maximized.

Alternatively, a PTAS for the multiple knapsack problem (MKP), where items are additionally weighted with a profit that can be different from the item's size, can be used [3, 4, 16]. In particular, the recent EPTAS by Jansen [16] yields a runtime bound of $T_{MSSP}(n, \epsilon) = 2^{O(\log(1/\epsilon) \cdot 1/\epsilon^5)} + \text{poly}(n)$ for n items and $m \leq n$ target capacities, which can be further improved to $T_{MSSP}(n, \epsilon) = 2^{O(\log(1/\epsilon)^4 \cdot 1/\epsilon)} + \text{poly}(n)$ [15] and, if the *Modified Integer Roundup Conjecture* (MIRUP) of Scheithauer and Terno [31] holds, this even reduces to $T_{MSSP}(n, \epsilon) = 2^{O(\log(1/\epsilon)^2 \cdot 1/\epsilon)} + \text{poly}(n)$ [15]. Knapsack type problems belong to the oldest and most fundamental problems in combinatorial optimization and theoretical computer science; we refer the reader to [19, 26] for in-depth surveys or the papers [1, 4, 14, 16, 20] for literature on these problems.

The remainder of our contribution is organized as follows. In Sect. 2, we present the approximation algorithm for fixed jobs. In Sect. 3, we present the approximation algorithm and inapproximability results for the non-availibility case. As it turns out, the algorithms are very similar, so we focus on the latter. Finally we conclude in Sect. 4 with open questions.

2 Scheduling with fixed jobs

In this section we prove Theorem 1 and the approximation part of Theorem 2. We may assume that $m \leq n$. Otherwise, we have $m > n$, and in this case there are at least $m - k$ machines without fixed jobs. Since we have exactly $n - k$ non-fixed jobs, every job that has to be scheduled can be executed on a free machine of its own, solving the instance to optimality.

Our model is based on the multiple subset sum problem (MSSP) which can be formally defined as follows. We are given a set $\{1, \ldots, n\}$ of items, each item i having a positive integer weight w_i, and a set $\{1, \ldots, m\}$ 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 without exceeding the capacities.

Our algorithm is described in Fig. 1. It is based on the dual approximation paradigm [10] by using binary search on the makespan. First we set $\epsilon' := \epsilon/2$.

1. Set $LB := LB_0$, $UB := UB_0$, and $\epsilon' := \epsilon/2$. Let σ_{best} a schedule of makespan at most UB_0.

2. While $UB - LB \geq 1$ repeat:

   2.1 Set $T := \lceil (UB + LB)/2 \rceil$. Generate the gaps $G(T)$ and the partition into large and small jobs $J_L(T) \dot{\cup} J_S(T)$ as described in Subsect. 2.1.

   2.2 Generate a packing of $(1-\epsilon')P(J)$ into $G(T)$ by solving MSSP. If this is not possible, reject T.

   2.3 Modify the packing to include all items from $J_L(T)$. If this is not possible, reject T.

   2.4 If T was rejected, set $LB := T$, else set σ to the generated packing and $UB := T$.

3. Use the first-fit algorithm in Subsect. 2.2 to schedule the remaining jobs into an interval of length at most $UB/2$.

Figure 1: Outline of the approximation algorithm for scheduling with fixed jobs or non-availability. Unspecified parameters are given in the text.

For any subset $S \subseteq \{k+1, \ldots, n\}$ let

$$P(S) := \sum_{i \in S} p_i$$

denote the total processing time of S and for any $j \in \{1, \ldots, k\}$ let

$$C_j := s_j + p_j$$

denote the completion time of the fixed job or non-availability j, and $p_{\max} := \max\{p_j \mid j \in \{1, \ldots, n\}\}$ the maximum processing time of the jobs.

Note, then, that for fixed jobs, the optimal makespan must be at least

$$LB_0 = C^{\text{fix}}_{\max} := \max\{C_j \mid j \in \{1, \ldots, k\}\}$$

and certainly, there exists a schedule of makespan at most $UB_0 = C^{\text{fix}}_{\max} + np_{\max}$ by scheduling all other jobs after $C^{\text{fix}}_{\max}$ on a single machine.

If we use binary search as in the outermost loop in the algorithm in Fig. 1, we obtain a search space of size at most $np_{\max}$ for the target makespan; we will find a suitable target makespan (i.e. one for which we can schedule all large jobs and almost all load) in $O(\log(np_{\max}))$ steps which is polynomially bounded in the encoding size of the instance. When the algorithm in Fig. 1 reaches Step 3, the upper bound UB is the smallest target makespan for which in Steps 2.2 and 2.3 a suitable schedule can be found. As we will see in the following, $C^*_{\max}$ is also a suitable schedule, which means that if we reach Step 3, we have $UB \leq C^*_{\max}$.

For any target makespan T, we use the technique described below which involves an EPTAS for MSSP [1, 16] to schedule as much load as possible in the interval $[0, T)$. In the sequel we show that for the optimal makespan $T = C^*_{\max}$, we can thus 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 a simple scheduling heuristic, causing an error which will be suitably bounded however. In the following, let T 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 $(m_1, s_1), \ldots, (m_k, s_k)$.

2.1 Job Classification and Gap Generation

For a given target makespan 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 $G(T) := \{G_1, \ldots, G_{q(T)}\}$ denote the set of gaps. For each $i \in \{1, \ldots, q(T)\}$ we also use G_i to denote the size of gap G_i. Without loss of generality, we assume $G_1 \geq \ldots \geq G_{q(t)}$. Note that $q(T) \leq k+m \leq 2n$ since at most k fixed jobs induce a gap "left" to them 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. Furthermore, we define

$$J_L(T) := \{i \in \{k+1, \ldots, n\} \mid p_i \in (T/2, T]\},$$
$$J_S(T) := \{i \in \{k+1, \ldots, n\} \mid p_i \in (0, T/2]\}$$

to partition the set of non-fixed jobs into *large* and *small* jobs. If any unplaced job is longer than T, we can obviously immediately reject the guessed makespan of T as too small. Note that since every gap's length G_i is at most T, there is at most one large job in each gap. We now create an instance of MSSP as follows: each gap G_i corresponds to a knapsack of capacity G_i and each job of length p_i, $i = k+1 \ldots n$, corresponds to an item of size p_i. We run the EPTAS of [16] on this instance with accuracy ϵ'.

Observe that if (and only if) our current guessed makespan T is at least the optimal makespan OPT, it is possible to pack all items into the gaps, so the EPTAS will leave items of total area at most $\epsilon'(\sum_{i=k+1}^{n} p_i) \leq \epsilon' mT$ unpacked. (We will see in the end that a choice of $\epsilon = 1/4$, hence $\epsilon' = 1/8$ is sufficient.) Here, we use that mT is a natural upper bound on $\sum_{i=1}^{n} p_i$ if $T \geq \text{OPT}$. Hence, if more than total length $\epsilon' mT$ is not packed, we can immediately reject our guessed T.

At this stage, the unpacked jobs may still include up to $\lfloor (\epsilon' mT)/(T/2) \rfloor = \lfloor \epsilon m \rfloor$ large jobs. Obviously, if large jobs, which have length $> T/2$, are not packed in the gaps in the period $[0, T)$, we cannot hope to find an overall schedule of length at most $\frac{3}{2}T$. Hence, we modify the packing using the following construction.

Lemma 3. *Given a packing of some jobs into the gaps such that jobs of total length δ are unpacked, amongst them a large job of some length $p_{j_1} > T/2$, we can either find in polynomial time a modified packing such that the total length of unpacked jobs is at most $\delta + p_{j_1}$ and the additional large job j_1 is packed as well as all previously packed large jobs or else prove that no packing of all jobs into the gaps exists at all.*

Proof. Let p_{j_1} denote the size of a large unpacked job, i.e. $p_{j_1} > T/2$. Let t_1 be the largest index such that $G_{t_1} \geq p_{j_1}$. (Recall that $T \geq G_1 \geq \cdots \geq G_{q(T)}$.) Clearly, p_{j_1} can only possibly be scheduled in one of the gaps $G_1, \ldots, G_{t_1}$,

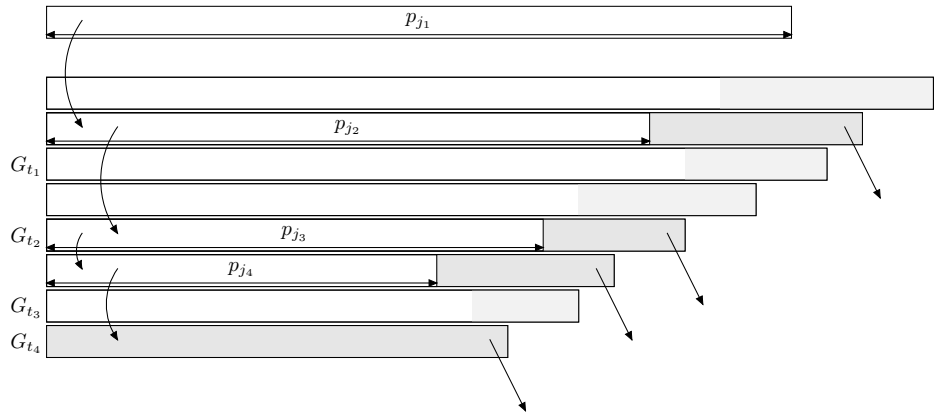


Figure 2: Choice of gaps $G_{j_1}, \ldots$ in the proof of Lemma 3. Shaded areas indicate possible small jobs; the darker areas are actually unpacked.

so if each one of these already contains a job at least as large as p_{j_1}, no packing can exist at all. Otherwise, we select one gap G_{j_1} among the gaps $G_1, \ldots, G_{t_1}$ that contains a large job of minimal size. (For this purpose, a gap without large job contains a 'dummy large job' of size 0; see also Fig. 2 for the following construction.) Denote this job j_2, of size $p_{j_2} < p_{j_1}$. We temporarily unpack j_2, and permanently unpack all small jobs that might have been in its gap as well, which have total length $\ell_1 \leq G_{j_1} - p_{j_2} \leq T - p_{j_2}$.

If $p_{j_2} = 0$, we have now scheduled one more large job. Otherwise, we need to re-schedule j_2. As for j_1, let $t_2 \geq t_1$ be the largest index such that $G_{t_2} \geq p_{j_2}$. Furthermore, we already know that gaps $G_1, \ldots, G_{t_1}$ all carry large jobs at least as large as j_2, since j_2 was chosen to be of minimal size. Hence, we can restrict our attention to the gaps $G_{t_1+1}, \ldots, G_{t_2}$. Again, if all these gaps already contain jobs at least as large as j_2, no feasible packing exists for this choice of T at all. Otherwise, we select a gap G_{j_2} with a large job j_3 of minimal size p_{j_3} (possibly 0) and iterate as above, discarding small jobs of total size $\ell_2 \leq G_{j_2} - p_{j_3} \leq G_{t_1+1} - p_{j_3} \leq p_{j_1} - p_{j_3}$.

After some number $r \leq m$ of iterations, we have either rejected our guess of T, or $p_{j_{r+1}} = 0$, i.e. we did not need to unpack another large job, and the number of packed large jobs has increased by one. Finally, the total size of small jobs that were unpacked can be bounded by

$$\begin{aligned}\sum_{i=1}^{r} \ell_i &\leq (T - p_{j_2}) + (p_{j_1} - p_{j_3}) + \cdots + (p_{j_{r-1}} - p_{j_{r+1}}) \\ &= T + \sum_{i=1}^{r-1} p_{j_i} - \sum_{i=2}^{r+1} p_{j_i} = T + p_{j_1} - p_{j_r} \leq 2p_{j_1}\,.\end{aligned}$$

The final inequality holds since we know that j_r is a large job, so $T - p_{j_r} < T/2 < p_{j_1}$. Since we have now additionally packed p_{j_1}, the net loss incurred is bounded by p_{j_1}. □

Note that the total size of all unpacked large jobs is initially bounded by $\epsilon' mT$, so we immediately obtain:

Corollary 4. *Given a packing of some jobs into the gaps such that jobs of total length $\epsilon' mT$ are unpacked, we obtain after at most $\lfloor \epsilon' mT/(T/2) \rfloor$ applications of Lemma 3 a packing that includes all large jobs and has unpacked jobs with total size at most $2\epsilon' mT = \epsilon mT$.*

The running time of this procedure is bounded by $O(n^2)$.

2.2 Packing remaining jobs

After the construction of the previous subsection, we are left with the minimal value T such that we first have successfully packed almost all jobs, (all but total processing time $\epsilon' mT$), which we have modified by Lemma 3 to a packing of all but total processing time $2\epsilon' mT = \epsilon mT$. Since the construction is valid for makespan $T = \mathrm{OPT}$, we know that the final $T \leq \mathrm{OPT}$. (Recall that ϵ is a fixed constant which we will shortly set explicitly.)

We will now schedule the remaining jobs in the interval $[T, \frac{3}{2}T)$ using a Next Fit Decreasing heuristic as follows: for convenience, denote these jobs $j_1 \ldots, j_{n'}$ ordered such that $T/2 \geq p_{j_1} \geq \cdots \geq p_{j_{n'}}$. Denote with $m' > 0$ the number of machines that is permanently available in the interval $[T, \frac{3}{2}T)$; without loss of generality, these are machines $M_1, \ldots, M_{m'}$. For every of these machines, we greedily assign jobs to it until its extra load is at most $\frac{1}{2}T$ or we run out of jobs. Clearly, the running time of this procedure is then bounded by the $O(n \log n)$ needed to sort the jobs by size.

Lemma 5. *If the total size of jobs to be scheduled in this way is at most $m'T/4$, all jobs can be assigned in the interval $[T, \frac{3}{2}T)$.*

Proof. The algorithm assigns jobs to the machines in a greedy fashion, and once a machine is considered "full", it is closed and never reopened again, and the next machine is considered. We show that a machine is not closed unless its load larger than $T/4$: then, assuming a job would need to be assigned to a $(m'+1)$st machine, the total length of the jobs would be strictly larger $m'T/4$.

Clearly, the first job j_1 can always be assigned to m_1, since $p_{j_1} \leq T/2$. Assume now that some machine m_i is closed because it cannot accomodate job j for some $j > 1$. This means m_i's current load is in the interval $(T/2 -$

$p_j, T/2]$. If $p_j \leq T/4$, our claim is true since $T/2 - p_j \geq T/4$. If $p_j > T/4$, m_i''s current load must be non-zero, because $p_j \leq T/2$. In particular, p_{j-1} was assigned to m_i, and since $p_{j-1} \geq p_j > T/4$, m_i's load is at least $T/4$. □

Hence, for purpose of our algorithm, if we set ϵ such that $\epsilon mT \leq m'T/4$, the overall construction is valid.

To complete the proof of Theorem 1 for fixed jobs, it is now sufficient to note that since T is at least $C_{\max}^{\text{fix}}$, the time the last fixed job terminates, all m machines are available afterwards, i.e. $m = m'$, such that we may set $\epsilon = 1/4$ and $\epsilon' = 1/8$. The total running time is then bounded by $O(n \log n) + T_{MSSP}(n, 1/8)$.

3 Scheduling with Non-Availability

In this section, we consider the case of Scheduling with Non-Availability, i.e. the fixed jobs do not count towards the objective value. We require that a constant fraction $\rho \in [1/m, 1)$ of the machines does not contain any fixed jobs. As we will see in Subsect. 3.2, the problem becomes hard to approximate otherwise.

3.1 A 3/2-Approximation for Scheduling with Non-Availability

In this subsection, we describe the (small) changes that are needed to apply the algorithm in Fig. 1. The three key parameters of the algorithm are the initial values for lower and upper bound, LB_0 and LB_0, and the accuracy parameter ϵ. As to the bounds, note that the optimal makespan OPT certainly satisfies OPT ≥ 0; on the other hand, there exists a trivial schedule of length $\sum_{j=1}^{n} p_j$ which just uses one of the permanently-available machines. Hence, we can set $LB_0 := 0$ and $UB_0 := \sum_{j=1}^{n} p_j$. Since $UB_0 - LB_0 = \sum_{j=1}^{n} p_j$ as in the fixed-job case, the number of iterations is polynomial in the input size.

We can then continue in the same way as described above, creating gaps, filling them with an EPTAS for MSSP, and changing this packing to accomodate all jobs of length more than $T/2$. The only other change concerns the accuracy ϵ needed for the MSSP: by Lemma 5, it is sufficient that $\epsilon mT = m'T/4$, where m' denotes the number of permanently available machines. Hence, we have $m' = \rho m$ in this case, so it is easy to see that by choosing $\epsilon = \rho/4$, so $\epsilon' = \rho/8$, the claim is obtained. The total running time is then bounded by $O(n \log n) + T_{MSSP}(n, \rho/8)$.

3.2 Lower Bounds for Scheduling with Non-Availability

Here we describe why our approach needs stronger preconditions for scheduling with non-availability; as we have seen, the idea is basically the same as for scheduling with fixed jobs, but the construction 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 $\mathsf{P} = \mathsf{NP}$ holds. This follows from the fact that scheduling parallel jobs on parallel machines with non-availability is inapproximable unless $\mathsf{P} = \mathsf{NP}$ [7]. Earlier, Lee [22] 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, even the restriction to instances where for each time step there is at most one unavailable machine is not sufficient to obtain a constant approximation ratio, as can be seen via a reduction from Equal Cardinality Partition.

Theorem 6. *Scheduling with non-availability, even if for each time step there is only one unavailable machine, does not admit a polynomial time algorithm with a constant approximation ratio unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Let $c \in \mathbb{R}$, $c \geq 1$. We aim at a contradiction and suppose that there is an approximation algorithm B with constant ratio c for scheduling with non-availability where for each time step there is only one unavailable machine. Without loss of generality, we assume that c is integer. We use a reduction from the following NP-complete problem Equal Cardinality Partition (ECP) [8]. The construction is sketched in Fig. 3.

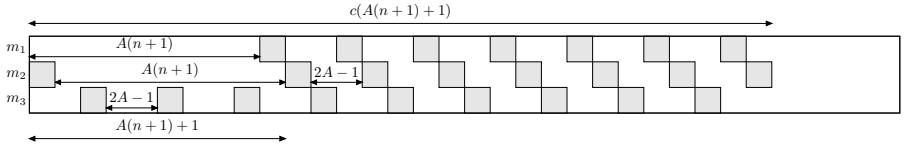


Figure 3: In the structure of intervals of non-availability of the generated instance I', for every time step there is at most one unavailable machine.

- *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 $\sum_{i=1}^{n} a_i = 2A$.

- *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 $\sum_{i \in I_1} a_i = A = \sum_{i \in I_2} a_i$?

Given an instance I of ECP we define an instance I' of scheduling with non-availability for arbitrary $m \geq 2$ where for each time step there is only at most one unavailable machine as follows. We may assume $m \leq A$ by suitable scaling of all a_i.

Each item a_i is copied to a job $k + i$ of length $2A + a_i$. Furthermore, the k periods of non-availability are defined as follows (also see Fig. 3): for all $t = 0, \ldots, c(A(n+1)+1)$, there is a non-availability interval $[t, t+1)$ on machine m_1 iff $t \equiv 0 \pmod{2A}$ and $t > A(n+1)$, on machine m_2 iff $t \equiv 1 \pmod{2A}$ and $t > A(n+1)+1$, on machine m_i for $3 \leq i \leq m$ iff $t \equiv i-1 \pmod{2A}$. Additionally, the interval $[0, 1)$ on machine m_2 is not available. It is easily seen that at any point, at most one machine is unavailable.

Note also that $k \leq cm(n+1)$, since there are at most m non-availabilities per time interval of length A, so k is polynomial in the size of the instance I.

By construction, on no machine, there is an available gap of length $\geq 2A$ that is at the same time entirely in the interval $[A(n+1)+1, c(A(n+1)+1))$, so no job of I' can be scheduled there. For machines $3, \ldots, m$, there even is no gap of size $\geq 2A$ in the interval $[0, c(A(n+1)+1))$. In particular, the makespan of any schedule of this instance is either at most $A(n+1)+1$ or strictly larger than $c(A(n+1)+1)$.

If I is a yes-instance to ECP, then there is a schedule in I' that has makespan (at most) $A(n+1)+1$: let $I_1 \cup I_2$ a suitable partition of I, then the corresponding partition $I'_1 \cup I'_2$ of I' satisfies that

$$\sum_{i \in I'_1} p_{i'} = \sum_{i \in I'_1} 2A + a_i = \frac{n}{2} 2A + \sum_{i \in I'_1} a_i = (n+1)A = \sum_{i \in I'_2} p_{i'} \,,$$

so these two sets can be scheduled on m_1 and m_2 respectively in the intervals $[0, A(n+1))$ and $[1, A(n+1)+1)$. Also, our c-approximation B will deliver a schedule of length at most $c(A(n+1)+1)$.

On the other hand, if there is an optimal schedule of length at most $A(n+1)+1$, it must schedule all jobs either on machine m_1 in the interval $[0, A(n+1))$ or on m_2 in the interval $[1, A(n+1)+1)$. Since the total length of all jobs is $2(An+1)$, both intervals are filled exactly. Also, since $n/2+1$ jobs would have total length more than $(n/2+1)2A = (n+2)A$, neither interval can contain more than $n/2$ jobs, so both contain exactly $n/2$ jobs, which means they induce a solution to I. So, for no-instances to ECP, the optimal makespan is at least $c(A(n+1)+1)+1$, and algorithm B must return a solution that has at least this length.

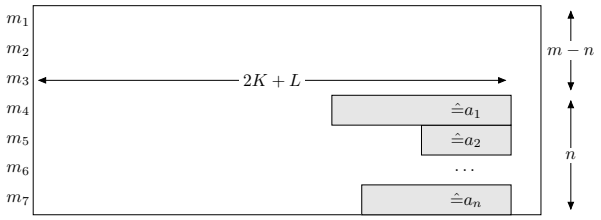


Figure 4: This sketch illustrates the proof of Theorem 7.

In total, B decides in polynomial time whether I is a yes-instance or not, which is impossible unless $\mathsf{P} = \mathsf{NP}$. □

This result motivies our assumption that at least one fixed machine is *always* available. The algorithm we have presented used the stronger assumption that the percentage ρ of permanently available machines is constant. Surprisingly, even this restriction is algorithmically hard to approximate. Theorem 7 yields the inapproximability result from Theorem 2.

Theorem 7. *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 an absolute approximation ratio of $3/2-\epsilon$, unless $\mathsf{P} = \mathsf{NP}$, for any $\epsilon \in (0, 1/2]$.*

Proof. We aim at a contradiction and suppose there is a polynomial-time approximation algorithm A for our scheduling problem with approximation ratio $3/2 - \epsilon$. We use a reduction from the following version of 3-Partition which is strongly NP-complete; the strong NP-completeness can be proved via a reduction from the problem Numerical Matching with Target Sums, which is strongly NP-complete, as discussed in [8, SP 17].

- *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 $\sum_{i=1}^{n} a_i + \sum_{i=1}^{2n} b_i = nL$.
- *Question:* Is there a $\pi \in S_{2n}$ such that $a_i + b_{\pi(2i-1)} + b_{\pi(2i)} = L$ for each $i \in \{1, \ldots, n\}$?

Given an instance I of the above problem we define an instance I' of scheduling with non-availability where a percentage of at least $\rho \in (0, 1)$ machines is permanently available as follows; the construction is sketched in Fig. 4. We choose $K \in \mathbb{N}$ such that $K > \max\{L, (1/2 - \epsilon)L/(2\epsilon)\}$; furthermore we use

$$m := \lceil \frac{n}{1 - \rho} \rceil$$

machines and define n suitable intervals of non-availability by setting $p_i := a_i$ for each $i \in \{1, \ldots, n\}$ which are fixed via $(i+m-n, 2K+L-a_i)$. As sketched in Fig. 4, these jobs are fixed to finish at time $2K + L$. Note that

$$\frac{m-n}{m} = 1 - \frac{n}{m} = 1 - \frac{n}{\lceil \frac{n}{1-\rho} \rceil} \geq 1 - \frac{n}{n/(1-\rho)} = 1 - (1-\rho) = \rho$$

holds. In the further presentation of the proof, we assume that the first $m-n$ machines are permanently available. Furthermore we introduce small jobs by defining

$$p_{n+i} := b_i + K$$

for each $i \in \{1, \ldots, 2n\}$. Finally we define $m - n$ dummy jobs

$$p_{3n+i} := 2K + L$$

for each $i \in \{1, \ldots, m-n\}$. Note that I' can be generated from I in running time polynomial in the encoding length of I. Note that for a yes-instance I of the above problem, we can execute the dummy jobs of I' on the machines $1, \ldots, m - n$. Finally we use the existing permutation π; since

$$a_i + b_{\pi(2i-1)} + b_{\pi(2i)} = L$$

for each $i \in \{1, \ldots, n\}$, we have

$$(b_{\pi(2i-1)} + K) + (b_{\pi(2i)} + K) = 2K + L - a_i.$$

This means that the small jobs corresponding to $b_{\pi(2i-1)}$ and $b_{\pi(2i)}$ can be executed in the interval $[0, 2K + L - a_i)$ on machine $m - n + i$ for each $i \in \{1, \ldots, n\}$. Consequently, I' has an optimal makespan of $C^*_{\max} = 2K+L$. Conversely, in a schedule with makespan $2K + L$ the dummy jobs must be executed on machines $1, \ldots, m-n$, hence the small jobs must run on machines $m - n + 1, \ldots, m$. Note that the processing time of each small job is larger than K; consequently, we have $3K > 2K + L$, hence it is impossible that more than 2 small jobs run on the same machine in the interval $[0, 2K + L)$. This means that on each machine $i \in \{m - n, m\}$, exactly 2 small jobs are executed, which indicates the desired permutation π. In total, I' has an optimal makespan of $C^*_{\max} = 2K + L$ if and only if I is a yes-instance of the above problem.

Now let I be a no-instance of the above problem. Then in any schedule for I' two cases can occur.

Case 1: The dummy jobs run on the machines in $\{1, \ldots, m - n\}$. Then there is a small job which is either scheduled together with a dummy job or

on one machines $\{m-n+1,\ldots,m\}$ after the interval of non-availability. In total, we obtain a job with completion time at least $3K+L$. *Case 2:* There is a dummy job which runs on one of the machines in $\{m-n+1,\ldots,m\}$. Since its processing time is $2K+L$, it must run after the interval of non-availability; here we also obtain a completion time at least $4K+2L \geq 3K+L$.

In total, the makespan of any schedule of I' must be at least

$$3K+L.$$

Next we show that we can use the algorithm A as an exact algorithm for the above problem as follows. For each instance I of the above problem we generate an instance of our scheduling problem as described above and apply the algorithm A to the instance I'. If the makespan of the generated schedule for I' is smaller than $3K+L$, we decide that I is a yes-instance of the above problem.

Let I be a yes-instance of the above problem. Note that the inequality $K > (1/2-\epsilon)L/(2\epsilon)$ can be rearranged to $K2\epsilon > (1/2-\epsilon)L$. Using this inequality we obtain

$$\begin{aligned}\Big(\frac{3}{2}-\epsilon\Big)(2K+L) &= 3K+\Big(\frac{3}{2}-\epsilon\Big)L-K2\epsilon \\ &< 3K+\Big(\frac{3}{2}-\epsilon\Big)L-\Big(\frac{1}{2}-\epsilon\Big)L = 3K+L.\end{aligned}$$

Now we use this inequality to argue that the algorithm A generates for I' a solution with value

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

where in the last step we used the estimation from above. For a no-instance I of the above problem, the algorithm A generates for I' a schedule with makespan at least $3K+L$, which is a lower bound for the optimal makespan of I'.

In total, we can algorithmically decide whether any instance I of the above problem is a yes-instance or a no-instance within a polynomial runtime bound, which is impossible unless $\mathsf{P}=\mathsf{NP}$ holds. □

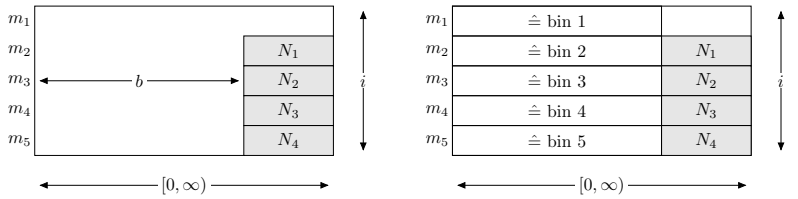
Comment. Note that in the construction from the proof, we can also use $\epsilon := 1/n$, which means that there is also no approximation algorithm for the problem under discussion with approximation ratio $3/2-1/n$.

Furthermore the construction from the proof uses at most one interval of non-availability per machine; hence, the result is also valid if the number of

non-availability intervals per machine is restricted to one. Furthermore, 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 [9, Chapt. 2, p.67]. Theorem 8 can be seen as an informal reason why scheduling with non-availability is hard to approximate.

Theorem 8. *Suppose there is a polynomial time algorithm for scheduling with non-availability and at least one permanently available machine that has 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 as claimed with approximation ratio $c \in \mathbb{N} \setminus \{1\}$; the following construction is illustrated in Fig. 5.



(a) The structure of intervals of non-availability of the generated instance I'_i roughly resembles i bins.

(b) For an optimal solution of I'_i for $i \geq \mathrm{OPT}(I)$ has a makespan of $C^*_{\max} \leq b$.

Figure 5: This sketch illustrates the proof of Theorem 8.

For each instance I of Bin Packing with n items and bin size b we define n instances I'_i for $i \in \{1, \ldots, n\}$ of scheduling with non-availability by setting $m = i$ and defining intervals of non-availability $(j+1, b)$ of size ∞ for each $j \in \{1, \ldots, i-1\}$. 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 $\mathrm{OPT}(I)$, the minimum number of bins in which the items of I can be packed. Our main interest will be the instance with $\mathrm{OPT}(I)$ machines.

Let

$$n' := \min\{i \in \{1, \ldots, n\} \mid A(I'_i) \leq cb\}$$

which can be found in polynomial time by enumeration since n is a lower bound for the encoding length of I. Hence $A(I'_{n'-1}) > cb$, where we set

$A(I'_0) = \infty$ by convention. (An instance I'_0 would have no machines whatsoever.) Since A is a c-approximation, it follows that $C^*_{\max}(I'_{n'-1}) \geq \frac{1}{c}A(I'_{n'-1}) > b$. This means that it is impossible to pack the items of I in less than n' bins of size b, hence $\mathrm{OPT}(I) \geq n'$ holds. Consider the schedule for $I'_{n'}$ generated by A. The schedule for the machines $2, \ldots, n'$ 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

$$n' - 1 + 1 + 2(c-1) \leq \mathrm{OPT}(I) + 2(c-1),$$

hence the approach yields an algorithm for Bin Packing which uses at most $2(c-1)$ additional bins. □

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 finally obtained a polynomial time algorithm with ratio 3/2, which is tight unless $\mathsf{P} = \mathsf{NP}$ holds. The techniques 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 which is tight unless $\mathsf{P} = \mathsf{NP}$ holds.

In total, our approach yields two tight approximation results. However, our algorithm uses a very general MSSP EPTAS; it is interesting whether there is a faster algorithm for the special case of $\epsilon = 1/8$ that we need. So far, the best known non-PTAS result is a 3/4-approximation due to Caprara et al. [2].

Acknowledgements The authors express their gratitude towards the anonymous referees, whose suggestions greatly improved the presentation of this paper.

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

[3] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *SODA*, pages 213–222, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

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

[5] 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 *Lecture Notes in Computer Science*, pages 297–307. Springer, 2007.

[6] F. Diedrich, K. Jansen, F. Pascual, and D. Trystram. Approximation algorithms for scheduling with reservations. *Algorithmica*, 2011. To appear.

[7] L. Eyraud-Dubois, G. Mounie, and D. Trystram. Analysis of scheduling algorithms with reservations. In *IPDPS*, pages 1–8. IEEE, 2007.

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

[9] D. S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1996.

[10] 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.

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

[12] H.-C. Hwang and S. Y. Chang. Parallel machines scheduling with machine shutdowns. *Computers and Mathematics with Applications*, 36(3):21–31, 1998.

[13] 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.

[14] 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.

[15] K. Jansen. A fast approximation scheme for the multiple knapsack problem. Technical Report 0916, Christian-Albrechts-Univresität zu Kiel, 2009.

[16] K. Jansen. Parameterized approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 39(4):1392–1412, 2009.

[17] 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*, 17(2):117–133, 2009.

[18] H. Kellerer. Algorithms for multiprocessor scheduling with machine release times. *IEEE Transactions*, 30(11), 1998.

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

[20] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

[21] C.-Y. Lee. Parallel machines scheduling with non-simultaneous machine available time. *Discrete Applied Mathematics*, 30:53–61, 1991.

[22] C.-Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9:363–384, 1996.

[23] 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.

[24] J. Y.-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[25] 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.

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

[27] N. Megow, R. H. Möhring, and J. Schulz. Turnaround scheduling in chemical manufacturing. In *MAPSP*, 2007.

[28] E. Sanlaville and G. Schmidt. Machine scheduling with availability constraints. *Acta Informatica*, 35(9):795–811, 1998.

[29] M. Scharbrodt. *Produktionsplanung in der Prozessindustrie: Modelle, effiziente Algorithmen und Umsetzung*. PhD thesis, Fakultät für Informatik, Technische Universität München, 2000.

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

[31] G. Scheithauer and J. Terno. The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research*, 84(3):562 – 571, 1995. Cutting and Packing.