INSTITUT FÜR INFORMATIK

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

Florian Diedrich
Klaus Jansen

Bericht Nr. 0902
February 2009

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

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

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

Florian Diedrich
Klaus Jansen

Bericht Nr. 0902
February 2009

e-mail: fdi@informatik.uni-kiel.de, kj@informatik.uni-kiel.de

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

Florian Diedrich† Klaus Jansen‡

February 3, 2009

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 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 $3/2 + \epsilon$, respectively, which we refine to approximation algorithms with ratio $3/2$. 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. In total, our approximation ratio for both problems is 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

*A preliminary version of this work with the title "Improved Approximation Algorithms for Scheduling with Fixed Jobs" waspresented at the 20th ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York Marriott Downtown, USA, January 4–6, 2009.

†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. 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 project AEOLUS, "Algorithmic Principles for Building Efficient Overlay Computers", EU contract 015964, and in part by DFG priority program 1126, "Algorithmik großer und komplexer Netzwerke", DFG contract SR7/9.

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 [31, 32] or intervals of non-availability of some machines must be taken into account [5, 13, 23, 25, 26]. 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 [28] 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 $p_1, \ldots, p_n \in \mathbb{N}$. 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 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* [5, 23, 25, 26]. 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 [30, 31, 32]. They mainly studied the problem for m constant; 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 [30] and $2 + \epsilon$ [32]; 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 $P||C_{\max}$ [11] to schedule the remaining $n-k$ jobs after the fixed job which finishes last. Finally, Scharbrodt, Steger & Weisser [32] 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* $3/2+\epsilon$ *for any* $\epsilon \in (0, 1/2]$. *Furthermore, 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 [6]. 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. [5]. 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 [22] and Lee et al. [24] 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 [17] found an algorithm with a tight ratio of $5/4$. Furthermore, Hwang et al. [13] briefly pointed out that this problem admits a PTAS. A more general case is the setting where the at most one interval per machine may have an arbitrary position. For this problem Lee [23] 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 $\lambda \leq m/2$ [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 [25], Chapt. 22, or [29] for surveys. For the sake of completeness, some results about single-machine problems can be found in the articles [5, 16, 22]. 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* $3/2+\epsilon$ *for any* $\epsilon \in (0, 1/2]$. *Furthermore, this problem admits an approximation algorithm with ratio* $3/2$. *Finally, 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, whether this is possible is an interesting open problem, as discussed in [9], 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 [10], partition of the instance, linear grouping and rounding known from Bin Packing [7] or Strip Packing [19, 20], and definition of configurations. Our modelization also involves the multiple subset sum problem also denoted by MSSP. As an algorithmic building block we use a PTAS for MSSP from [1] where the knapsack capacities are permitted to be different. Alternatively, a PTAS for the multiple knapsack problem (MKP) can be used [2, 3, 15]. In particular, if the number of target areas is large, the recent PTAS by Jansen [15] yields a runtime bound which is polynomial in both $1/\epsilon$ and the encoding size of the instance. Knapsack type problems belong to the oldest and most fundamental problems in combinatorial optimization and theoretical computer science; we refer the reader to [18, 27] for in-depth surveys or the papers [1, 3, 14, 15, 21] 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, which is then refined to yield a ratio of 3/2. 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. 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.

1. Set $\epsilon' := \epsilon/3$. Set $LB := C_{\max}^{\text{fix}}$ and $UB := C_{\max}^{\text{fix}} + np_{\max}$. Let σ_{saved} be the empty schedule.

2. While $UB - LB > 1$ repeat Steps 2.1–2.3.

   2.1 Set $T := \lceil (UB + LB)/2 \rceil$. Generate gap sets $G_L(T)$ and $G_S(T)$. Generate the sets $J_L(T)$, $J_M(T)$ and $J_S(T)$, as described in Subsect. 2.1. Apply linear grouping and rounding to the jobs in $J_M(T)$ and generate all possible configurations $\kappa^{(1)}, \ldots, \kappa^{(c_2)}$. Set *found* := *false*.

   2.2 For each possible choice of the values $q'_i(T,k)$ for each interval index $k \in \{1, \ldots, c_3\}$ and group index $i \in \{1, \ldots, c_4 + 1\}$ execute Step 2.1.

      2.2.1 For each possible choice of values $c(k,i,\ell)$ for $(k,i,\ell) \in \mathcal{I}$ execute Steps 2.2.1.1–2.2.1.2.

         2.2.1.1 Generate the network flow model $N(T)$ for the specific choice of configurations, values $q'_i(T,k)$ and values $c(k,i,\ell)$ as described in Subsect. 2.4 and solve it. If the value of the network flow is smaller than $|J_L(T)|$, proceed with the next iteration of the loop in Step 2.2.1. Otherwise assign the jobs in $J_L(T)$ to the gaps in $G_L(T)$ as indicated by the network flow, resulting in a schedule σ. Use a PTAS for MSSP as described in Subsect. 2.5 to add a suitable subset of $J_M(T) \cup J_S(T)$ to the schedule σ. Let $P'(\sigma)$ be the total processing time of jobs *not* scheduled in σ.

         2.2.1.2 If $P'(\sigma) > 3\epsilon' Tm$, proceed with the next iteration of the loop in Step 2.2.1. If $P'(\sigma) \leq 3\epsilon' Tm$, set $\sigma_{\text{saved}} := \sigma$, set *found* := *true*, go to Step 2.3.

   2.3 If *found* := *true* set $UB := T$ else set $LB := T$.

3. Use the list scheduling algorithm from Subsect. 2.5 to add the jobs which are not yet scheduled in σ after the makespan of σ.

Figure 1: The approximation algorithm for scheduling with fixed jobs. We assume that each loop is taken to the next iteration if T, the values $q'_i(T,k)$ of the values $c(i,k,\ell)$ are determined to be infeasible.

Our modelization 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.

Our algorithm is described in Fig. 1. It is based on the dual approximation paradigm [10] by using binary seach on the makespan. First we set $\epsilon' := \epsilon/3$.

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 j. Let

$$C_{\max}^{\text{fix}} := \max\{C_j | j \in \{1, \ldots, k\}\}.$$

Note that

$$C_{\max}^{\text{fix}} \leq C_{\max}^{*} \leq C_{\max}^{\text{fix}} + np_{\max}$$

holds, where $p_{\max} := \max\{p_j | j \in \{k+1, \ldots, n\}\}$ denotes the maximum processing time of the non-fixed jobs. In total, the remaining $n-k$ jobs indexed by $\{k+1, \ldots, n\}$ can be scheduled on one machine in the interval $[C_{\max}^{\text{fix}}, C_{\max}^{\text{fix}} + np_{\max})$. 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. If the algorithm in Fig. 1 reaches Step 3, the upper bound UB is the smallest target makespan for which in Step 2.2.1.2 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 a PTAS for MSSP [1, 15] 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 algorithmically find a schedule which executes almost all load in the interval $[0, C_{\max}^{*})$; the remainig 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 $T \in [C_{\max}^{\text{fix}}, C_{\max}^{\text{fix}} + np_{\max})$ 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 $(m_1, s_1), \ldots, (m_k, s_k)$.

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 $G(T) := \{G_1, \ldots, G_{q(T)}\}$ 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. Note that $|q(T)| \leq k + m \leq 2n$ since at most 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

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

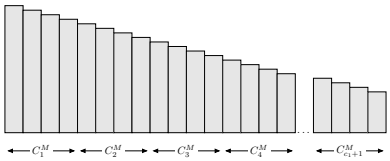
Let $q_L(T) := |G_L(T)|$, $q_S(T) := |G_S(T)|$ be the number of large and small gaps for target makespan T. Since there is at most one large gap per machine, we have $q_L(T) \leq m$. We define

$$\begin{aligned} J_L(T) &:= \{i \in \{k+1, \ldots, n\} | p_i \in (T/2, T]\}, \\ J_M(T) &:= \{i \in \{k+1, \ldots, n\} | p_i \in (\epsilon' T, T/2]\}, \\ J_S(T) &:= \{i \in \{k+1, \ldots, n\} | p_i \in (0, \epsilon' T]\} \end{aligned}$$

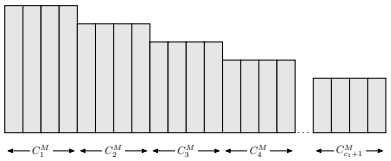
to partition the set of non-fixed jobs into *large*, *medium* and *small* jobs. Note $J_L(T)$ can be only packed into the at most m gaps of $G_L(T)$. Hence, if $|J_L(T)| > q_L(T)$, T is infeasible and can be discarded. In the sequel we assume that there are no unnecessary idle times in the gaps, i.e. a set of jobs placed in a gap is scheduled as a continuous block which starts as early as possible and the idle times are positioned at the end of the gap.

2.2 Definition of Configurations for Medium Jobs

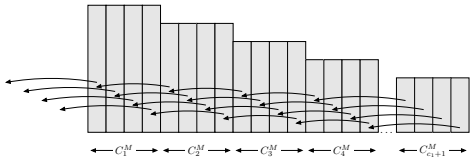
We obtain few distinct job sizes in $J_M(T)$ via rounding. This construction removes some jobs from the schedule; these will not be executed in $[0, T)$ anymore. However, the total processing time of the removed jobs can be



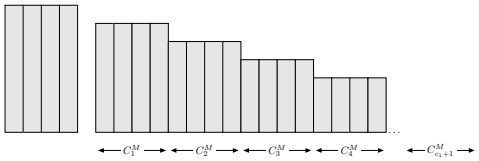
(a) Arrangement of the jobs in $J_M(T)$ in non-increasing order of size. Creation of c_1+1 groups of equal cardinality where the last group is possibly of smaller cardinality.



(b) In each group C_i^M the job sizes are rounded up to the largest size occuring in the respective group.



(c) Each group is shifted to the next larger group; the first group is removed.



(d) The first group is removed from the schedule, the remaining rounded jobs from $J_M(T)$ are accomodated by the rearrangement.

Figure 2: This sketch illustrates the linear grouping and rounding technique used in Subsect. 2.2.

suitably bounded. Now fix a makespan T and a schedule σ with makespan T; note that the construction will be valid in particular for $T = C^*_{\max}$, an argument which will be used later. 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 and rounding as in [7] to the medium jobs in $J_M(T)$ by setting $c_1 := \lceil 1/\epsilon'^2 \rceil$ and creating $c_1 + 1$ groups. The following construction is sketched in Fig. 2.

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 $C_1^M, \ldots, C_{c_1+1}^M$. Next for each $i \in \{1, \ldots, c_1 + 1\}$ the processing times of medium jobs in C_i^M are rounded up to the largest processing time occuring 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 $\tilde{C}_1^M, \ldots, \tilde{C}_{c_1+1}^M$. 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 the items of $\tilde{C}_i^M$ into the space for $\tilde{C}_{i-1}^M$ for each $i \in \{2, \ldots, c_1\}$, 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, the total processing time of C_1^M can be bounded by

$$P(C_1^M) \leq \frac{Tn'}{2c_1} \leq \frac{Tm}{2\epsilon' c_1} = \frac{Tm}{2\epsilon' \lceil 1/\epsilon'^2 \rceil} \leq \frac{\epsilon' Tm}{2}.$$

Note that $G \leq T$ for each gap G and $p_i > \epsilon' T$ for each medium job $i \in J_M(T)$. Hence 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 $(a_1, \ldots, a_{c_1})$ with $\sum_{i=1}^{c_1} a_i \leq 1/\epsilon'$ and $a_i \in \mathbb{N}_0$ for each $i \in \{1, \ldots, c_1\}$. For a configuration 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. Let c_2 denote the number of configurations.

For any $n \in \mathbb{N}$, $k \in \mathbb{N}$ let

$$D_n^k := |\{(a_1, \ldots, a_n) | a_i \in \mathbb{N}_0, \sum_{i=1}^{n} a_i \leq k\}| = \binom{n+k-1}{k};$$

the last equality follows from modeling the combinatorial situation as combinations with repetition. In total, we obtain

$$c_2 \leq \sum_{j=0}^{\lfloor 1/\epsilon' \rfloor} D_n^j = \sum_{j=0}^{\lfloor 1/\epsilon' \rfloor} \binom{\lfloor 1/\epsilon' \rfloor + j - 1}{j} = \frac{1}{\lfloor 1/\epsilon' \rfloor} \sum_{j=0}^{\lfloor 1/\epsilon' \rfloor} \frac{(\lfloor 1/\epsilon' \rfloor + j - 1)!}{j!}.$$

However, in the sequel we will use the simpler bound

$$\begin{aligned} c_2 \leq |\{\kappa \in \{0, \ldots, \lfloor 1/\epsilon' \rfloor\}^{c_1} | \sum_{i=1}^{c_1} \kappa_i \leq \lfloor 1/\epsilon' \rfloor\}| \\ \leq |\{\kappa \in \{0, \ldots, \lfloor 1/\epsilon' \rfloor\}^{c_1}| = (\lfloor 1/\epsilon' \rfloor + 1)^{c_1} \end{aligned}$$

which states that c_2 is independent from the encoding size of the input. We denote all configurations by $\kappa^{(1)}, \ldots, \kappa^{(c_2)}$. For each such a configuration $\kappa^{(\ell)} = (\kappa_1^{(\ell)}, \ldots, \kappa_{c_1}^{(\ell)})$ with upper index $\ell \in \{1, \ldots, c_2\}$ 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 a simpler structure in which only a small amount of total processing time is not scheduled. Furthermore, each job which is not included in the scheduled is a medium job. We have established Lemma 3 and later use enumeration to find a suitable choice of configurations for a target makespan T.

Lemma 3. *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. Almost all medium jobs from $J_M(T)$ are scheduled, i.e. there are only medium jobs from $J_M(T)$ which are not scheduled, and the total processing time of these is at most $\epsilon' Tm/2$. 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 $J_S(T)$.*

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 3. 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 $k \in \{1, \ldots, c_3\}$ let

$$I_k(T) := (k\epsilon' T, (k+1)\epsilon' T].$$

Finally for each $k \in \{1, \ldots, c_3\}$, $\ell \in \{1, \ldots, c_2\}$ 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 $I_k(T)$.

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. We define

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

Now we present a construction to round the large jobs from $J_L(T)$ contained in $G_L(T,k)$ under a small loss of total size of the medium jobs in $J_M(T)$; the approach is illustrated in Fig. 3. There, light grey areas indicate the large jobs, dark grey areas indicate the configurations, and white areas indicate small jobs or idle time.

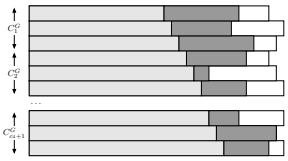
Conceptually arrange the gaps in $G_L(T,k)$ along with their contents in σ_1 in a vertical stack, starting at the top with a gap containing a job from $J_L(T,k)$ of smallest size to the bottom finishing with a gap containing a job from $J_L(T,k)$ of largest size. Except for the small jobs from $J_S(T)$, each of these gaps contains exactly two objects, namely a job from $J_L(T,k)$ and a non-empty configuration. If two such objects occur together in a large gap we will call them *associated*. Similar as in [7], we apply linear grouping to the stack. More precisely, we set $c_4 := \lceil 1/\epsilon' \rceil$ and aim at creating c_4+1 groups; beginning from the top, we create c_4+1 groups $C_1^G, \ldots, C_{c_4+1}^G$ of size $\lfloor |G_L(T,k)|/c_4 \rfloor$ where the last group is possibly of smaller cardinality. However, if in the stack there are $n' < c_4+1$ gaps, we also create c_4+1 groups. In this case, the first n' groups are of cardinality 1; the remaining c_4+1-n' groups are empty and are not used in the construction. Now in each group C_i^G, each large job is rounded up to the size of the largest large job occuring in C_i^G, namely to

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

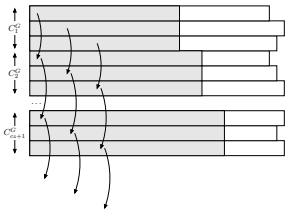
if there are $n' < c_4+1$ gaps in the stack, the values for the empty groups are set to $-\infty$. Now it remains to show that the rounded large jobs in the stack can be packed together with nearly all of their associated configurations.

To this end, we use an elegant cyclic shifting argument which is sketched in Fig. 3. Each rounded large job from $J_L(T,k)$ is shifted downwards exactly $\lfloor |G_L(T,k)|/c_4 \rfloor$ gaps in the stack into at least the next larger group, where it can be safely packed together with the configuration packed there. The large jobs in the $\lfloor |G_L(T,k)|/c_4 \rfloor$ gaps at the bottom are pushed out of the stack. Hence, a total number of $\lfloor |G_L(T,k)|/c_4 \rfloor$ large jobs from $J_L(T)$ not packed. We remove the configurations in the group C_1^G and denote the set of non-packed associated configurations by $I(T,k)$. Then, the set $I(T,k)$ is removed from the schedule. Now the uppermost $\lfloor |G_L(T,k)|/c_4 \rfloor$ gaps in the stack are empty. Every non-packed configuration has a total size of at least $\epsilon' T$ and at most $T/2$ since it was packed together with a large job in σ_1. Consequently, we can use

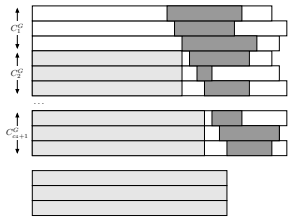
$$P(I(T,k) \leq \frac{T}{2} \lfloor |G_L(T,k)|/c_4 \rfloor \leq \frac{T|G_L(T,k)|}{2c_4} \tag{1}$$



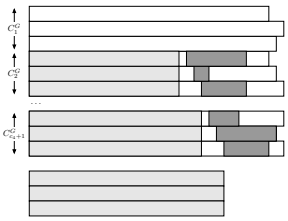
(a) Create a stack for $G_L(T,k)$ by starting at the top with a gap containing a large job of smallest size and finishing at the bottom with a gap containing a large job of largest size. Then we create at most c_4+1 groups of equal size, except for the last group.



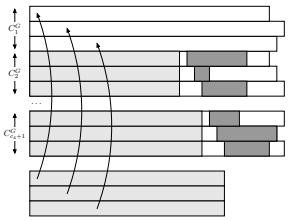
(b) The size of the large jobs is rounded up to the largest job size occurring in the respective group; the configurations are not shown due to overlap. The large jobs are rearranged by shifting them in at least the next lower group; the bottommost large jobs are pushed out of the stack.



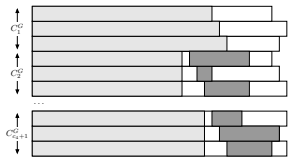
(c) In the resulting arrangement there is no overlap between rounded large jobs in the stack and configurations. Here the configurations are shown again.



(d) To accomodate the rounded large jobs from below the stack, the configurations in the topmost group C_1^G are removed. The idle time 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. For ease of exposition, the job sizes there are restored to the non-rounded value.



(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 3: This sketch illustrates the construction from Subsect. 2.3; light grey areas indicate the large jobs from $G_L(T,k)$, dark grey areas indicate the associated configurations and white areas indicate small jobs or idle time.

to bound the total processing time of the non-packed configurations. Concerning the $\lfloor |G_L(T,k)|/c_4 \rfloor$ 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. Just for ease of exposition, the sizes of the large jobs which are now in the gaps in C_1^G are restored to their non-rounded sizes. By dropping the rounding of the large jobs now in C_1^G, a large job is rounded if and only if it is packed together with a non-empty configuration.

Note that, algorithmically, it is not possible to obtain the rounded job sizes $q'_1(T,k), \ldots, q'_{c_4+1}(T,k)$ for a fixed interval index $k \in \{1, \ldots, c_3\}$ directly since the schedule σ_1 is not available. However, each value $q'_i(T,k)$ is a size of a large job from $J_L(T)$ or $-\infty$, i.e. one of $m+1$ possible values. Hence, the number of possible choices for the values $q'_i(T,k)$ is bounded by $(|J_L(T)|+1)^{c_4+1} \leq (m+1)^{c_4+1}$. This means that the values $q'_i(T,k)$ resulting from the application of the cyclic shifting technique can be found by enumeration within a polynomial runtime bound, since also the number of interval indices $k \in \{1, \ldots, c_3\}$ is bounded by a constant. In total, all values $q'_i(T,k)$ can be enumerated in $(m+1)^{c_3(c_4+1)}$ steps.

Lemma 4. *By applying the rounding and cyclic shifting for all interval indices $k \in \{1, \ldots, c_3\}$, only medium jobs from $J_M(T)$ are lost. The total processing time of these is bounded by $\epsilon' Tm/2$. Furthermore, the number of different sizes for the rounded large jobs is bounded by $c_3(c_4+1)$.*

Proof. By construction only medium jobs from $J_M(T)$ are not packed. Now we use the bound (1), the estimation $\sum_{k=1}^{c_3} |G_L(T,k)| \leq |G_L(T)| \leq m$, and $c_4 = \lceil 1/\epsilon' \rceil$ to obtain the chain of inequalities

$$\begin{aligned} P(\cup_{k=1}^{c_3} I(T,k) = \sum_{k=1}^{c_3} P(I(T,k)) \leq T/(2c_4) \sum_{k=1}^{c_3} |G_L(T,k)| \\ \leq \frac{T}{2c_4} |G_L(T)| \leq \frac{Tm}{2c_4} \leq \frac{\epsilon' Tm}{2} \end{aligned}$$

which yields the desired bound. Since we have c_3 interval indices and for each of these we generate at most c_4+1 rounded sizes for large jobs, we obtain the claim. □

We use σ_2 to denote the resulting partial schedule in which all jobs which are now removed, more precisely the medium jobs in $\cup_{k=1}^{c_3} I(T,k)$, do not

occur. Let

$$\mathcal{I} := \{1, \ldots, c_3\} \times \{1, \ldots, c_4 + 1\} \times \{1, \ldots, c_2\}$$

denote the set of triples for indices for intervals, rounded sizes of large jobs, and configurations. For each $(k, i, \ell) \in \mathcal{I}$ let

$$c(k, i, \ell) := |\{G \in G_L(T) | \text{in } \sigma_2 \text{ gap } G \text{ contains a job from } J_L(T, k) \\ \text{of rounded size } q'_i(T, k) \text{ and a non-empty configuration } \kappa^{(\ell)}\}| \leq m$$

denote the number of large jobs from $J_L(T)$ with rounded size $q'_i(T, k)$ which are packed together with the non-empty configuration $\kappa^{(\ell)}$ in σ_2. In total, we obtain the following result.

Lemma 5. *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. Almost all medium jobs from $J_M(T)$ are scheduled, i.e. there are only medium jobs from $J_M(T)$ which are not scheduled, and the total processing time of these is at most $\epsilon' Tm/2 + \epsilon' Tm/2 = \epsilon' Tm$. In each large gap from $G_L(T)$ there are exactly three objects, namely a possibly rounded large job from $J_L(T)$, possibly of size 0, a possibly empty configuration and a possibly empty subset of small jobs from $J_S(T)$. The number of sizes for the rounded large jobs is bounded by $c_3(c_4 + 1)$. For each $(k, i, \ell) \in (I)$ there is a nonnegative integer $c(k, i, \ell) \leq m$ which indicates how often a rounded large job of size $q'_i(T, k)$ is packed together with a non-empty configuration $\kappa^{(\ell)}$.*

Clearly, there are at most $m^{c_2 c_3 (c_4+1)}$ choices for the values $c(k, i, \ell)$, hence these can be found by enumeration. However, algorithmically we have to deal with the problem that, even if the values $c(k, i, \ell)$ 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 6, 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 confgurations to the gaps in $G_L(T)$ can be extended to a partial schedule under a small loss of processing time. This means that obtaining any such assignment is sufficient for our construction. To this end we define a multiset of large jobs and configurations; more precisely let

$$J_{LC}(T) := \{j \in J_L(T) | \text{job } j \text{ is rounded (or not rounded) as in } \sigma_2\} \\ \cup \{\kappa^{(\ell)} | \kappa^{(\ell)} \text{ occurs in a gap in } G_L(T) \text{ in } \sigma_2\}$$

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

Lemma 6. *Let T be a feasible makespan. Let σ_3 be a partial schedule of makespan at most T which assigns exactly the (possibly rounded) 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 $J_L(T)$ and a subset of almost all medium and small jobs are scheduled in a large gap from $G_L(T)$, i.e. there are only medium and small jobs from $J_M(T) \cup J_S(T)$ which are not scheduled and the total processing time of these is at most $\epsilon' Tm/2 + \epsilon' Tm/2 + \epsilon' Tm = 2\epsilon' Tm$.*

Proof. Let σ_2 denote the schedule from Lemma 5. Let

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

Remove the small jobs from I_S in σ_2 and do not change the schedule in the gaps which are not large. 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 $J_{LC}(T)$ (i.e. the jobs in the large gaps) 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 idle 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 accomodated completely. In this way, at most $|G_L(T)| - 1 \leq m - 1 \leq m$ 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 for each $j \in S$ we have $p_j \leq \epsilon' T$ and $|S| \leq m$, we have $P(S) \leq \epsilon' Tm$. 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. □

2.4 Assignment of Jobs to Large Gaps via Network Flow

In Lemma 6 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, even if we do not know the contents of the remaining gaps. 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. Solutions of the network flow model will also decide whether or not a large job is rounded.

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. Suppose that suitable rounded job sizes $q'_i(T,k)$ for $k \in \{1,\ldots,c_3\}$ and $i \in \{1,\ldots,c_4+1\}$ are given. Furthermore, the associated configurations are given implicitly by the values $c(k,i,\ell) \leq m$ for each $(k,i,\ell) \in \mathcal{I}$.

We define a directed acyclic graph $N(T) = (V(T), E(T))$ where $V(T)$ consists of five layers; the construction is sketched in Fig. 4. More precisely, the *node set* $V(T)$ of $N(T)$ is defined as follows.

1. In the first layer there is only s, the *source* node.
2. In the second layer there are the at most m large jobs from $J_L(T)$ which we call *job* nodes.
3. In the third layer there is a set of nodes to govern the assignment of large jobs and configurations, namely exactly the set $\mathcal{I}$ which will be termed as *interval-size-configuration* nodes.
4. In the fourth layer there are the at most m large gaps from $G_L(T)$ which we call *gap* nodes.
5. In the fifth layer there is only t, the *terminal* node.

Likewise, the *arc set* $E(T)$ of $N(T)$ is constructed in order to encode the possibilities of arranging large jobs together with configurations in large gaps. More precisely, $E(T)$ is defined as follows.

1. The source s is connected to each large job node, i.e. $(s,j) \in E(T)$ for each $j \in J_L(T)$; these arcs connect the first and the second layer.
2. Each job node is connected to an interval-size-configuration node if it is contained in the interval and can be possibly rounded to the size, i.e.

$$(j,(k,1,\ell)) \in E(T) :\Leftrightarrow p_j \in (0, q'_1(T,k)]$$

and

$$(j,(k,i,\ell)) \in E(T) :\Leftrightarrow p_j \in (q'_{i-1}(T,k), q'_i(T,k)]$$

for $i \in \{2,\ldots,c_4+1\}$ for each $j \in J_L(T)$ and $(k,i,\ell) \in \mathcal{I}$; these arcs connect the second to the third layer.

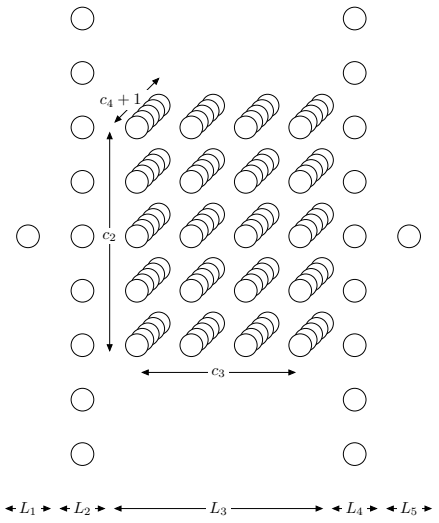


Figure 4: Sketch of the network used for assignment of large jobs and configurations; edges are not shown. Layers 1 (L_1) to 5 (L_5) are arranged from left to right. Arrows labelling layer 3 indicate the number of nodes in the corresponding dimension. In total, there are $|\mathcal{I}|$ nodes in layer 3.

3. Each interval-size-configuration node is connected to a gap node if a job of the rounded size can be packed together with the configuration into the gap, i.e.

$$((k, i, \ell), G) \in E(T) :\Leftrightarrow q_i'(T, k) + s_\ell \leq G$$

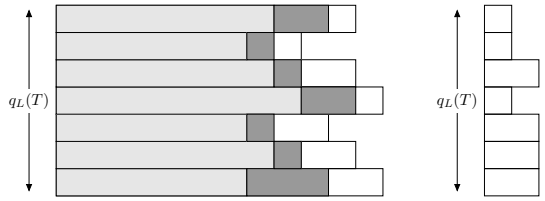
and $q_i'(T, k) \neq -\infty$, for each $(k, i, \ell) \in \mathcal{I}$ and $G \in G_L(T)$; these arcs connect the third to the fourth layer. Routing of flow along such an edge indicates that the corresponding large job is *rounded* and packed together with a non-empty configuration.

4. Each gap node is connected to the terminal node, i.e. $(G, k) \in E(T)$ for each $G \in G_L(T)$; these arcs connect the fourth to the fifth layer.

5. Each job node is connected to each gap node it fits into, more precisely

$$(j, G) \in E(T) :\Leftrightarrow p_j \leq G$$

for each $j \in J_L(T)$, $G \in G_L(T)$; these arcs connect the second to the fourth layer. Routing of flow along such an edge indicates that the corresponding large job is *not rounded* and packed together with the empty configuration.



(a) In the partial schedule σ_3, the large gaps in $G_L(T)$ are filled with large jobs from $J_L(T)$ and configurations as indicated by the choice of values $c(k, \ell)$.

(b) For the assignment of jobs from I_2, we consider the remaining space. The set of smaller "large" gaps is denoted by $\tilde{G}_L(T)$.

Figure 5: This sketch illustrates the large gaps in the schedule σ_3. Light grey areas indicate large jobs, dark grey ares indicate configurations, and white areas indicate idle time.

The arcs in $E(T)$ are endowed with a lower capacity 0 and an upper capacity of 1. Finally for each $(k, i, \ell) \in \mathcal{I}$ we require the flow in the interval-size-configuration node (k, i, ℓ) to be exactly $c(k, i, \ell)$ 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 the jobs of $J_L(T)$ (possibly rounded to the values $q'_i(T, k)$) together with the selected configurations, implicitly given by the values $c(k, i, \ell)$, 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 $c(k, i, \ell)$ for each $(k, i, \ell) \in \mathcal{I}$ have to be enumerated. We have $|G_L(T)| \leq m$ and hence at most m rounded large jobs from $J_L(T)$ of a certain size can be packed together with a certain configuration; consequently, as mentioned in Lemma 5, $c(k, i, \ell) \leq m$ holds for each $(k, i, \ell) \in \mathcal{I}$. Furthermore there is only a constant number of at most $|\mathcal{I}| = c_2 c_3 (c_4 + 1)$ nodes in the third layer. Since each of these nodes gets assigned a capacity value $c(k, i, \ell) \in \{0, \ldots, m\}$, the quantity $(m+1)^{c_2 c_3 (c_4+1)}$ 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 $I := \{k+1, \ldots, n\}$. We enumerate over all possible choices for the rounded job sizes $q'_i(T, k)$ for each $i \in \{1, \ldots, c_4 + 1\}$ and $k \in \{1, \ldots, c_3\}$; we

also enumerate over all possible choices of values $c(k,i,\ell)$ for each $(k,i,\ell) \in \mathcal{I}$. Next we describe how to find a suitable schedule, if one exists, given one such choice of values. First 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 $G_L(T)$.

Next we discuss Step 2.2.1.2 of the algorithm in Fig. 1. 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. 5. 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 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 6 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 a PTAS for MSSP 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 $c(k,i,\ell)$ 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

$$\begin{aligned} P(I) - (1-\epsilon')(P(I) - 2\epsilon' Tm) &\leq 2\epsilon' Tm + \epsilon' P(I) \\ &\leq 2\epsilon' Tm + \epsilon' Tm = 3\epsilon' Tm \end{aligned}$$

to schedule and the size of all of these jobs is bounded by $T/2$. We can use any list scheduling algorithm to execute this small load in the interval $[T,\infty)$; the following analysis is sketched in Fig. 6. 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

$$|[T, T')| \leq \frac{3\epsilon' T m}{m} \leq 3\epsilon' C^*_{\max}$$

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

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

By carrying out the entire construction from Subsect. 2.1–2.5, we have established the first part of Theorem 1. Next we show how we can obtain a ratio of 3/2 via a modification of the list scheduling approach discussed above.

To this end, we use the algorithm in Fig. 1 with $\epsilon := 3/24$, which results in $\epsilon' = 1/24$. Furthermore we modify Step 3 of the Algorithm in Fig. 1 as follows. When reaching Step 3, let I' denote the set of jobs which are not scheduled; as discussed above, we have $T \leq C^*_{\max}$ and $P(I') \leq 3\epsilon' T m$. Furthermore, we have $p_j \leq T/2$ for each $j \in I'$. Next we partition I' in two sets of larger and smaller jobs by setting

$$\begin{aligned} I'_L(T) &:= \{j \in I' | p_j > T/4\}, \\ I'_S(T) &:= \{j \in I' | p_j \leq T/4\}. \end{aligned}$$

Let $n'' := |I'_L|$; since $P(I') \leq 3\epsilon' T m$, we have $n'' T/4 \leq 3\epsilon' T m$; suitable rearrangement and using $\epsilon' = 1/24$ yields

$$n'' \leq 12\epsilon' m = 12m/24 = m/2.$$

Note that, since n'' is integral, we also have $n'' \leq \lfloor m/2 \rfloor$. Now, since in the time interval $[T, \infty)$ all machines are available, we use the first $n'' \leq \lfloor m/2 \rfloor$ machines to schedule the jobs in I'_L, where each job is scheduled on a machine of its own starting at time T. Similar as before, we use list scheduling to schedule the jobs in I'_S on the last $m - n''$ machines.

Next we distinguish two cases. *Case 1:* Scheduling of the jobs in I'_S does not increase the makespan of the generated schedule. In this case, the makespan of the algorithmically generated schedule is bounded by

$$T + T/2 \leq \frac{3}{2} T \leq \frac{3}{2} C^*_{\max}.$$

Case 2: Scheduling of the jobs in I'_S *does* increase the makespan of the generated schedule. 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. Using the Graham bounds, we obtain $[T', T'') \leq T/4$ for the last part of the schedule and

$$|[T, T')| \leq \frac{3\epsilon' Tm}{m - n''} \leq \frac{3\epsilon' Tm}{m - m/2} = \frac{Tm/8}{m/2} = \frac{T}{4}$$

for the middle part of the schedule. In total, using these bounds, the makespan of the generated schedule can be bounded by

$$|[0, T)| + |[T, T')| + |[T', T'')| \leq T + \frac{T}{4} + \frac{T}{4} = \frac{3}{2}T \leq \frac{3}{2}C^*_{\max};$$

this estimation establishes the second part of Theorem 1. In total, we have proven our first main result.

Comment. In the algorithm in Fig. 1, we can also use a greedy 2-approximation algorithm for MSSP [4] instead of the rather sophisticated PTASes [1, 3, 14, 15, 21]. Consequently, we have modify the algorithm in Fig. 1 as follows. In Step 1, we set $\epsilon' := \epsilon/2$ instead of $\epsilon/3$ and in Step 2.2.1.2, we compare the amount of non-scheduled processing time to $(1/2 + 2\epsilon')Tm$ instead of $3\epsilon' Tm$. If we carry out the same construction as before with this modification, for the list scheduling step we obtain

$$|[T, T')| \leq \frac{(1/2 + 2\epsilon')Tm}{m} \leq (\frac{1}{2} + 2\epsilon')C^*_{\max}.$$

In total, the length of the generated schedule then can be bounded by

$$|[0, T)| + |[T, T')| + |[T', T'')| \leq C^*_{\max} + (\frac{1}{2} + 2\epsilon')C^*_{\max} + \frac{1}{2}C^*_{\max} = (2+\epsilon)C^*_{\max},$$

which means that this approach yields the same approximation ratio as using a PTAS for $P||C_{\max}$ [11] to schedule all non-fixed jobs after the last fixed job.

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 $\mathsf{P} = \mathsf{NP}$ holds. This follows from the

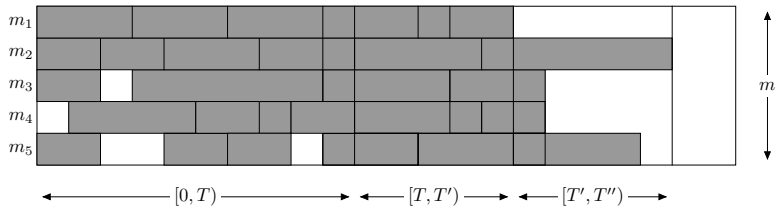


Figure 6: Illustration of the list scheduling from Sect. 2 which finally arranges the hitherto non-scheduled jobs. The jobs are indicated by dark grey areas while light grey areas indicate the periods of non-availability; white areas indicate idle time. The jobs in the interval $[0, T)$ are scheduled via the technique described in Subsect. 2.1–2.4 and a PTAS for MSSP; the jobs in $[T, T'')$ are assigned via list scheduling, hence in $[T, T')$ there is no idle time.

fact that scheduling parallel jobs on parallel machines with non-availability is inapproximable unless $\mathsf{P} = \mathsf{NP}$ [6]. Earlier, Lee [23] 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.

Theorem 7. *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* $\mathsf{P} = \mathsf{NP}$.

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 an available machine. We use a reduction from the following NP-complete problem Equal Cardinality Partition (ECP) [8]. The construction is sketched in Fig. 7.

- *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 an

available machine as follows. First we define two intervals of non-availability by setting $p_1 := p_2 := A(n+1)$ and fixing these via $(1, 0)$ and $(2, A(n+1))$; This means that job 1 is scheduled on machine 1 starting at time 0 and job 2 is scheduled on machine 2 starting at time $A(n+1)$. Furthermore we define additional intervals of non-availability by setting

$$p_{2+\ell} := 2A$$

for each $\ell \in \{1, \ldots, (n+1)\lceil c \rceil\}$ and fix these via list entries

$$(1 + (\ell - 1 \bmod 2), 2A(n+\ell))$$

for each $\ell \in \{1, \ldots, (n+1)\lceil c \rceil\}$. Furthermore we define dummy intervals of non-availability by setting

$$p_{2+(n+1)\lceil c \rceil+\ell} := 2A(n+1)(\lceil c \rceil + 1)$$

for every $\ell \in \{1, \ldots, m-2\}$ and fix these via

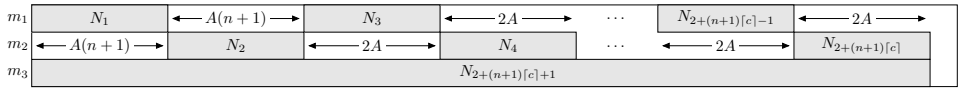
$$(2 + \ell, 0)$$

for each $\ell \in \{1, \ldots, m-2\}$, which means that all machines except machine 1 and machine 2 are not available in the time interval $[0, 2A(n+1)(\lceil c \rceil + 1))$. Finally we copy the items of I by defining

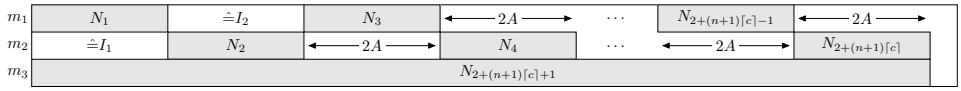
$$p_{j+2+(n+1)\lceil c \rceil+m-2} := 2A + a_j > 2A$$

for each $j \in \{1, \ldots, n\}$. Note that I' can be generated algorithmically from I in a running time which is polynomially bounded in the encoding length of I. Since $p_{2+(n+1)\lceil c \rceil+\ell} > 2A$, no job of I' can be scheduled in the interval

$$[2A(n+1), 2A(n+1)(\lceil c \rceil + 1)).$$



(a) In the structure of intervals of non-availability of the generated instance I', for every time step there is an available machine.



(b) For a yes-instance I of ECP, for I' we have $C^*_{\max} = 2A(n+1)$; for every no-instance I of ECP, for the instance I' we have $C^*_{\max} > 2A(n+1)(\lceil c \rceil + 1)$.

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

Note that for a yes-instance I of ECP, I' has an optimal makespan of value $C^*_{\max} = 2A(n+1)$; if I_1 and I_2 constitute the desired partition, we have

$$\sum_{i \in I_1} 2A + a_i = \sum_{i \in I_2} 2A + a_i = 2A\frac{n}{2} + |A| = An + A = A(n+1),$$

which means that we can execute the job sets indicated by I_1 and I_2 in the intervals $[0, A(n+1))$ and $[A(n+1), 2A(n+1))$ respectively. Conversely, in a schedule with makespan $2A(n+1)$ all jobs must be scheduled during the time interval $[0, 2A(n+1))$ since no more than $n/2$ jobs fit into an availability interval of length $A(n+1)$; such a schedule indicates the partition of I into I_1 and I_2. In total, I' has an optimal makespan of $C^*_{\max} = 2A(n+1)$ if and only if I is a yes-instance of ECP.

Now let I be a no-instance of ECP and consider an optimal schedule of I'. The makespan of the optimal schedule of I' is at least

$$2A(n+1)(\lceil c \rceil + 1)$$

since every job in I' has processing time larger than $2A$ and there must be a job which is not scheduled in $[0, 2A(n+1))$.

Next we show that we can use the algorithm B as an exact algorithm for ECP as follows. For each instance I of ECP we generate an instance I' of our scheduling problem as described above and apply the algorithm B to the instance I'. If the makespan of the generated schedule for I' is smaller than $2A(n+1)(\lceil c \rceil + 1)$, we decide that I is a yes-instance of ECP.

If I is a yes-instance of ECP, the algorithm B generates for I' a schedule with makespan

$$C_{\max} \leq cC^*_{\max} = c2A(n+1) < 2A(n+1)(\lceil c \rceil + 1);$$

for a no-instance I of ECP, the algorithm B generates for I' a schedule with makespan at least $2A(n+1)(\lceil c \rceil + 1)$, which is a lower bound for the optimal makespan of I'.

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

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 restriction is algorithmically hard to approximate. Theorem 8 yields the inapproximability result from Theorem 2.

Theorem 8. *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 o $3/2-\epsilon$, 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 string 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], page 224.

- *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. 8. 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. 8, 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 $3K + L$.

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

$$3K + L.$$

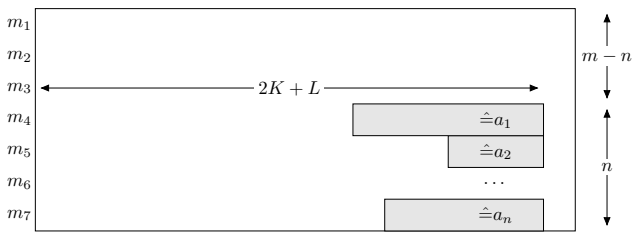


Figure 8: This sketch illustrates the proof of Theorem 8.

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

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

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

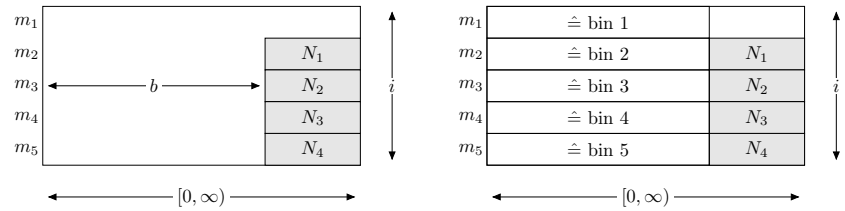
$$C_{\max} \leq (3/2 - \epsilon)C^*_{\max} = (3/2 - \epsilon)(2K + L) < 3K + L,$$

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, page 67. Theorem 9 can be seen as an informal reason for scheduling with non-availability being hard to approximate.



(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 9: This sketch illustrates the proof of Theorem 9.

Theorem 9. *Suppose there is a polynomial time algorithm for scheduling with non-availability where at least one machine is permanently available 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 $c \in \mathbb{N} \setminus \{1\}$; the following construction is illustrated in Fig. 9.

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.

Let I be an instance of Bin Packing. Let

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

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

Now we present the approximation algorithm for scheduling with non-availability where the ratio of permanently available machines is constant. Similar to [13], we use $\lambda \in \{1, \ldots, m-1\}$ 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, we may assume that $m \leq n$ holds. Next we desribe the first algorithm mentioned in Theorem 2.

Let $I := \{1, \ldots, n\}$; for scheduling with non-availability, the total processing time of the instance is bounded by $P(I) \leq np_{\max}$. This 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 $\epsilon' Tm$ of processing time. In the innermost loop of our algorithm, we pack the remaining small and medium jobs using a PTAS for MSSP from [1, 15]. Similar as in the algorithm for scheduling with fixed jobs, we set $\epsilon' := \epsilon(1-\lambda/m)/3 = \epsilon\rho/3$.

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 $M := \{1, \ldots, m-\lambda\}$ denote the set of these. The analysis is illustrated in Fig. 10.

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

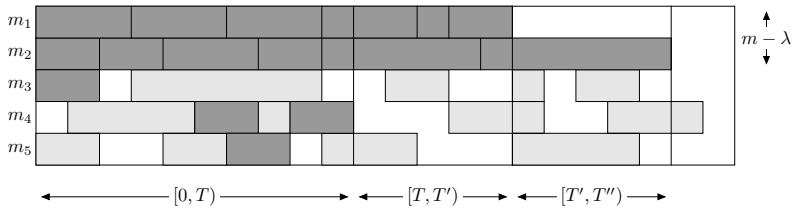


Figure 10: Illustration of the list scheduling from Sect. 3 for the respective list scheduling which finally arranges the hitherto non-scheduled jobs. The jobs are indicated by dark grey areas while light grey areas indicate the periods of non-availability; white areas indicate idle time. The jobs in the interval $[0, T)$ are scheduled via the technique described in Subsect. 2.1–2.4 and a PTAS for MSSP; 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.

obtain $|[T', T'')| \leq T/2 \leq C^*_{\max}/2$ and

$$|[T, T')| \leq \frac{2\epsilon' Tm + \epsilon' P(I)}{m - \lambda} \leq \frac{2\epsilon' Tm + \epsilon' m C^*_{\max}}{m - \lambda} \leq \frac{2\epsilon' C^*_{\max} + \epsilon' C^*_{\max}}{1 - \lambda/m} = \frac{3\epsilon'}{1 - \lambda/m} C^*_{\max},$$

Hence the makespan of the generated schedule can be bounded by

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

In total, we have presented the first algorithm mentioned in Theorem 2. Next we sho how we can obtain a ratio of 3/2 via a the same modification of the list scheduling approach as discussed in Subsect. 2.5. To this end, we use the algorithm discussed above with $\epsilon := 3/24$, which results in $\epsilon' = \rho/24$. When reaching Step 3 of our algorithm, let again denote I' the set of jobs which are not scheduled; as discussed above, we have $T \leq C^*_{\max}$ and $P(I') \leq 3\epsilon' Tm$. Furthermore, we have $p_j \leq T/2$ for each $j \in I'$. As in Subsect. 2.5, we partition I' by defining

$$\begin{aligned} I'_L(T) &:= \{j \in I' | p_j > T/4\}, \\ I'_S(T) &:= \{j \in I' | p_j \leq T/4\}. \end{aligned}$$

Again let $n'' := |I'_L|$; since $P(I') \leq 3\epsilon' Tm$, we have $n''T/4 \leq 3\epsilon' Tm$; suitable rearrangement and using $\epsilon' = \rho/24$ yields

$$n'' \leq 12\epsilon' m = 12\rho m/24 = \rho m/2.$$

Again, since n'' is itegral, we have $n'' \leq \lfloor \rho m/2 \rfloor$. Similar as in Subsect. 2.5, in the time interval $[T, \infty)$ the first $m - \lambda = \rho m$ machines are available. We use the first $n'' \leq \lfloor \rho m/2 \rfloor$ machines to schedule the jobs in I'_L, where each job is scheduled on a machine of its own starting at time T. Again we use list scheduling to schedule the jobs in I'_S on the next $m - \lambda - n''$ machines.

Next we distinguish two cases. *Case 1:* Scheduling of the jobs in I'_S does not increase the makespan of the generated schedule. In this case, the makespan of the algorithmically generated schedule is bounded by

$$T + T/2 \leq \frac{3}{2}T \leq \frac{3}{2}C^*_{\max}.$$

Case 2: Scheduling of the jobs in I'_S *does* increase the nmakespan of the generated schedule. Let T' denote the last step in $[T, \infty)$ where there is no idle machine and let T'' denote the last time step in $[T, \infty)$ were there is a busy machine. Using the Graham bounds as before, we obtain $[T', T'') \leq T/4$ for the last part of the schedule and

$$|[T, T')| \leq \frac{3\epsilon' Tm}{m - \lambda - \rho m/2} = \frac{3\epsilon' Tm}{\rho m - \rho m/2} = \frac{3\epsilon' Tm}{\rho m/2} = \frac{6\epsilon' T}{\rho} = \frac{6\epsilon' \rho T}{24\rho} = \frac{T}{4}$$

for the middle part of the schedule. As before, using these bounds, the makespan of the generated schedule is bounded by

$$|[0, T)| + |[T, T'')| + |[T', T'')| \leq T + \frac{T}{4} + \frac{T}{4} = \frac{3}{2}T \leq \frac{3}{2}C^*_{\max};$$

this estimation establishes the second algorithm mentioned in Theorem 2. In total, we have shown our second main result.

Comment. In our algorithm with ratio $3/2+\epsilon$, we can also use a greedy 2-approximation algorithm for MSSP [4] instead on a PTAS; we have to modify our algorithm as follows. In Step 1, we set $\epsilon' := \epsilon(1 - \lambda/m)/2 = \epsilon\rho/2$ instead of $\epsilon(1 - \lambda)/3$ and in Step 2.2.1.2, we compare the amount of non-scheduled processing time to $(1/2 + 2\epsilon')Tm$ instead of $3\epsilon' Tm$. If we carry out the same construction as before with this modification, for the list scheduling step we obtain

$$\begin{aligned} |[T, T')| &\leq \frac{(1/2 + 2\epsilon')Tm}{m - \lambda} = \frac{(1/2 + 2\epsilon')Tm}{\rho m} \\ &= \frac{(1/2 + 2\epsilon')T}{\rho} = \frac{(1/2 + \epsilon/\rho)T}{\rho} = (\frac{1}{2\rho} + \epsilon)T \leq (\frac{1}{2\rho} + \epsilon)C^*_{\max} \end{aligned}$$

for the middle part of the schedule. In total, the length of the generated schedule then can be bounded by

$$|[0,T)| + |[T,T')| + |[T',T'')| \\ \leq C^*_{\max} + (\frac{1}{2\rho} + \epsilon)C^*_{\max} + \frac{1}{2}C^*_{\max} = (\frac{3}{2} + \frac{1}{2\rho} + \epsilon)C^*_{\max}$$

which means that this approach yields an approximation ratio which depends on ρ, the percentage of permanently available machines.

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. 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 which is tight unless $\mathsf{P} = \mathsf{NP}$ holds.

In total, our approach yields two tight approximation results. However, our algorithm uses large enumeration steps; hence it is interesting whether there is an algorithm with ratio 3/2 but more desireable runtime bound.

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

[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 *LNCS*, pages 297–307. Springer, 2007.

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

[7] 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.

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

[16] 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.

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

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

[19] C. Kenyon and E. Rémila. Approximate strip packing. In *FOCS*, pages 31–36, 1996.

[20] C. Kenyon and E. Rémila. A near-optimal solution to a two dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.

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

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

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

[24] 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.

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

[26] 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.

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

[28] 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.

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

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

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

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