INSTITUT FÜR INFORMATIK

A PTAS for Scheduling Unrelated Machines of Few Different Types

Jan Clemens Gehrke, Klaus Jansen,
Stefan E. J. Kraft, Jakob Schikowski

Bericht Nr. 1506
Juni 2015
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

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

A PTAS for Scheduling Unrelated Machines of Few Different Types

Jan Clemens Gehrke, Klaus Jansen,
Stefan E. J. Kraft, Jakob Schikowski

Bericht Nr. 1506
Juni 2015
ISSN 2192-6247

e-mail: {jcg,kj,stkr,schi}@informatik.uni-kiel.de
Corresponding Author: Klaus Jansen (kj)

The research was supported by DFG project JA612/14-2, “Entwicklung und Analyse von effizienten polynomiellen Approximationsschemata für Scheduling- und verwandte Optimierungsprobleme”

Abstract

Scheduling on Unrelated Machines is a classical optimization problem where n jobs have to be distributed to m machines. Each of the jobs $j \in \{1, \dots, n\}$ has on machine $i \in \{1, \dots, m\}$ a processing time $p_{ij} \geq 0$. The goal is to minimize the makespan, i.e. the maximum completion time of the longest-running machine. Unless P = NP, this problem does not allow for a polynomial-time approximation algorithm with a ratio better than $\frac{3}{2}$. A natural scenario is however that many machines are of the same type, like a CPU and GPU cluster: for each of the K machine types, the machines $i \neq i'$ of the same type k satisfy $p_{ij} = p_{i'j}$ for all jobs j. For the case where the number K of machine types is constant, this paper presents an approximation scheme, i.e. an algorithm of approximation ratio $1 + \varepsilon$ for $\varepsilon > 0$, with an improved running time only single exponential in $\frac{1}{\varepsilon}$.

1 Introduction

Scheduling is a classical optimization problem. Jobs—e.g. computing tasks—have to be distributed to machines such that one objective is minimized, normally the maximum completion time of the jobs. One example is a cluster of processors that has to perform a large amount of computing tasks. In general, the machines may be heterogeneous: a processor may have been designed to perform a certain type of calculations very fast, but may not be suited for other ones. However, the number of different machine types may indeed be limited, as can be the case for e.g. a cluster of CPUs and GPUs.

We formally describe the problem. An instance I consists of a set $\mathcal{J} = \mathcal{J}(I)$ of n jobs and a set $\mathcal{M} = \mathcal{M}(I)$ of m machines. Every job j has a processing time on machine i of $p_{ij} \geq 0$ for $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, n\}$. A non-preemptive schedule is a distribution of the jobs to the machines such that every job is processed by exactly one machine. Formally, it is a mapping $\sigma : \mathcal{J} \to \mathcal{M}$ of each job j to a machine i. The objective is to find a schedule σ that minimizes the makespan $\max_{i \in \mathcal{M}} \sum_{j:\sigma(j)=i} p_{ij}$, i.e. the maximum completion time of all jobs. Thus, even the longest-running machine shall finish the processing as soon as possible. This classical problem is called Scheduling on Unrelated Machines and is denoted by $\mathrm{R}\,||\,C_{\max}$ in the 3-field notation [10].

As suggested above, we consider a variant where the machines are only of K different types, where K is considered to be constant: for two different machines i and i' (with $i \neq i'$) of the same type, we have $p_{ij} = p_{i'j}$ for all jobs $j \in \{1, \dots, n\}$. The machines of type k are denoted by $\mathcal{M}_k$ such that the sets $\mathcal{M}_1, \dots, \mathcal{M}_K$ are a disjoint partition of $\mathcal{M}$. The number of machines of one type is $m_k := |\mathcal{M}_k|$ for $k \in \{1, \dots, K\}$. Hence, $m_1 + \cdots + m_K = m$ holds. The problem is denoted by $(\mathrm{P}m_1, \dots, \mathrm{P}m_K)\,||\,C_{\max}$. We can assume without loss of generality that $m_k \leq n$ for all $k \in \{1, \dots, K\}$ and therefore $m = m_1 + \cdots + m_K \leq n \cdot K$. In fact, a solution cannot use more than n machines of a type k because there are only n jobs. For one type, machines whose number exceeds n can therefore be discarded.

1.1 Known Results

Even Scheduling on Identical Machines $\mathrm{P}\,||\,C_{\max}$ (where, as the name suggests, all machines are of the same type) is NP-complete [9]. Thus, finding the optimum objective value $\mathrm{OPT}(I)$ and a corresponding schedule efficiently (i.e. in polynomial time in the input length $|I|$) seems unlikely for the general case. We are therefore looking for efficient approximation algorithms. The approximation ratio of an algorithm A is $\sup_I \frac{A(I)}{\mathrm{OPT}(I)}$, where $A(I)$ is the objective value of the solution found by A. List Scheduling is a well-known heuristic with the approximation ratio $2 - \frac{1}{m}$ for $\mathrm{P}\,||\,C_{\max}$. Hochbaum and Shmoys [11] presented the first polynomial time approximation scheme (PTAS), a family of algorithms $(A_\varepsilon)_{\varepsilon>0}$ where A_ε has an approximation ratio of $1+\varepsilon$. The running time is polynomial in $|I|$, but the degree of the polynomial may depend exponentially (or worse) on $\frac{1}{\varepsilon}$.

Unfortunately, $\mathrm{R}\,||\,C_{\max}$ does not allow for a PTAS unless P = NP: a polynomial algorithm cannot in general have an approximation ratio $c < \frac{3}{2}$ as shown by Lenstra, Shmoys, and Tardos [18]. Approximation algorithms with a ratio of 2 were presented by Lenstra et al. [18], by Shmoys and Tardos [20], and by Gairing, Monien, and Woclaw [8]. A $2 - \frac{1}{m}$ algorithm was found by Shchepin and Vakhania [19]. These algorithms are based on solving a linear program (LP) and rounding the solution to an integer one, with the exception of the purely combinatorial algorithm in [8]. Recently, Arad, Mordechai, and Shachnai [1] have presented a new algorithm that decides that a schedule σ with a makespan of at most T and an average machine load $L = \frac{\sum_{i\in\mathcal{M}}\sum_{j:\sigma(j)=i} p_{ij}}{m}$ does not exist, or it finds one with a makespan of at most $\min\{T + \frac{L}{h}, 2T\}$, where $h = h(T)$ is the so-called feasibility factor.

No algorithm is known for the general problem with a ratio better than 2. For a long time, this was even true for the Restricted Assignment Problem, a special case where $p_{ij} \in \{p_j, \infty\}$. A breakthrough was the estimation algorithm by Svensson [21]. The algorithm does not return an actual solution, but it can estimate the optimal makespan within $\frac{33}{17} + \varepsilon \approx 1.9412 + \varepsilon$, i.e. with a ratio better than 2. Chakrabarty, Khanna, and Li [5] have presented for a constant $\delta^* > 0$ a $(2-\delta^*)$-approximation algorithm (that also returns a solution) for the $(1, \bar{\varepsilon})$-Restricted Assignment Problem. In this case of Restricted Assignment, the finite processing times are additionally either $p_j = 1$ or $p_j = \bar{\varepsilon}$ for constant $\bar{\varepsilon} > 0$.

Bhaskara et al. [2] studied the matrix $P = (p_{ij})_{m\times n}$ of the processing times, more precisely the influence of its rank on the non-approximability of $\mathrm{R}\,||\,C_{\max}$. Rank 1 is the case of identical or uniform machines (where the processing times are of the form $p_{ij} = \frac{p_j}{s_i}$), which allows for PTAS (see above for identical and e.g. [12, 17] for uniform machines). Unless P = NP, rank 4 is already APX-hard (i.e. a PTAS cannot exist), and rank 7 cannot be better approximated than $\frac{3}{2}$, as in the general case (see above). This was improved by Chen, Ye, and Zhang [6] who showed that already rank 4 does not allow for a polynomial-time approximation algorithm better than $\frac{3}{2}$ unless P = NP.

If the number m of machines is constant (i.e. $\mathrm{R}m\,||\,C_{\max}$ is considered), the problem has a PTAS [18] and a fully polynomial time approximation scheme (FPTAS) [13]. An FPTAS is a PTAS where the running time is polynomial in $|I|$ and also $\frac{1}{\varepsilon}$. Faster FPTAS were successively found [7, 16], and the fastest known FPTAS has a running time in

$O(n) + (\frac{m}{\varepsilon})^{O(m)} \leq O(n) + (\frac{\log m}{\varepsilon})^{O(m \log m)}$ [15]. It should be noted that the algorithm by Lenstra, Shmoys, and Tardos [18], while "only" being a PTAS, has a space complexity only polynomial in m, $\log \frac{1}{\varepsilon}$, and the input length.

Interestingly, the special case of Scheduling on a constant number of m identical machines ($\mathrm{P}m \,||\, C_{\max}$) has a lower bound of $n^{O(1)} + (\frac{1}{\varepsilon})^{O(m)}$ on the running time unless the Exponential Time Hypothesis fails [6]. For ε small enough, e.g. $\varepsilon \leq \frac{1}{m}$, the running time of the algorithm in [15] can be bounded by $O(n) + (\frac{1}{\varepsilon})^{O(m)}$ and therefore attains this lower bound.

Finally, Imreh [14] considered the Scheduling problem on $K = 2$ types. He presented heuristic algorithms with ratios $2 + \frac{m-1}{k}$ and $4 - \frac{2}{m}$, where m is the number of processors of the first and k the number of processors of the second type. Bleuse et al. [3] described an algorithm with the approximation ratio $\frac{4}{3} + \frac{1}{3k} + \varepsilon$ for scheduling on m cores (CPUs) and k GPUs. If all jobs are accelerated when executed on a GPU, the algorithm has the ratio $\frac{3}{2} + \varepsilon$. Bonifaci and Wiese [4] presented a PTAS for $(\mathrm{P}m_1, \ldots, \mathrm{P}m_K) \,||\, C_{\max}$ (where $K = O(1)$). It can schedule Δ-dimensional jobs. If $\Delta = 1$ (i.e. in the one-dimensional case), the algorithm has to solve $m^{O(K \cdot ((1/\varepsilon)^{1/\varepsilon \log 1/\varepsilon))}}$ linear programs, which is therefore a lower bound on the overall running time. It is double exponential in $\frac{1}{\varepsilon}$.

1.2 Our Result

This paper presents a PTAS for the one-dimensional case that is only single exponential in $\frac{1}{\varepsilon}$.

Theorem 1. *There is a PTAS for* $(\mathrm{P}m_1, \ldots, \mathrm{P}m_K) \,||\, C_{\max}$ *with a running time in*

$$O(K \cdot n) + m^{O(K/\varepsilon^2)} \cdot \left(\frac{\log m}{\varepsilon}\right)^{O(K^2)} .$$

1.3 Techniques

Our algorithm first preprocesses the instance I with a method presented in [7, 15] to get a new instance I^{merge} whose set of jobs $\mathcal{J}(I^{\text{merge}})$ has a bounded cardinality. Then, the well-known dual approximation approach by Hochbaum and Shmoys [11, 18] is used to find a solution close the optimal makespan of I^{merge} (and therefore an approximate solution to I). The dual approach is a binary search with an oracle: in each iteration, a value T is tested. If there is a schedule with a makespan of at most T, the oracle returns a solution of value at most $(1 + \Theta(\varepsilon))T$ (and T is decreased in the next iteration). If there is not a schedule, the oracle does not return any solution (and T is increased because it was too small). This can be iterated until a solution close enough to the optimum is found.

For a given makespan T, our oracle first partitions the jobs into large and small jobs for every machine type k. The processing times of the jobs are then rounded such that they have discrete values. Every feasible schedule has a profile: it states for every machine type k the total processing time of the small jobs assigned to k. Moreover, it states for machine type k how many machines have which total processing time of large jobs. As the job processing times are discrete, so are the profiles. The dynamic program of the oracle then constructs all possible profiles. When the dynamic program has finished, a

simple condition is checked for every constructed profile to see whether the small jobs can be greedily assigned to the machines. If yes, a real schedule for the large jobs is found by backtracking, and the small jobs are greedily scheduled, which yields a solution with a makespan of at most $(1 + \Theta(\varepsilon))T$. If no profile has been generated by the dynamic program or no profile allows for a distribution of the small jobs, the value T is too small. The binary search adapts T according to the output of the oracle.

1.4 General Remarks and Notation

Since we are in the case of K machine types, it is sufficient to state for a job j its processing time on every machine type and not on every individual machine. The processing time of job j on the machine type $k \in \{1, \ldots, K\}$ is therefore denoted by p_{kj}. The value $k(i)$ is the type of a machine $i \in \mathcal{M}$.

We suppose that K is constant, that $0 < \varepsilon \leq \frac{1}{2}$ and that computing the logarithm needs time in $O(1)$.

2 Preprocessing of the Instance

The first step of the algorithm is a preprocessing to reduce the number of items. The technique in this section is taken from [15]. Let $0 < \varepsilon' \leq \frac{1}{3}$ with $\varepsilon' = \Theta(\varepsilon)$. The actual value of ε' will be determined later.

First, let

$$d_j := \min_{k \in \{1,\ldots,K\}} p_{kj}$$

be the smallest processing time of a job j over all machine types, and let $D := \sum_{j \in \mathcal{J}} d_j$. We have $\frac{D}{m} \leq \mathrm{OPT}(I)$ because the jobs could ideally be scheduled uniformly on all machines, where each job is executed on one machine of its fastest type. On the other hand, we have $\mathrm{OPT}(I) \leq D$: a feasible solution is obtained by scheduling each job to one of its fastest machine. In the worst case, all jobs have the same fastest machine type, and there is only one machine of this type.

Hence, we can divide all processing times p_{kj} by $\frac{D}{m}$ such that we get the following:

Assumption 1. Without loss of generality, the jobs are scaled such that $1 \leq \mathrm{OPT}(I) \leq m$ and $D = \sum_{j \in \mathcal{J}} d_j = m$.

The jobs are now partitioned into fast and slow ones for each type k. A job is slow on type k if $p_{kj} \geq \frac{m}{\varepsilon'} d_j$, otherwise it is fast on type k. Should j be slow on type k, we set $p_{kj}^{\text{round}} := \infty$ (or to a sufficiently large value like $2m$) such that a reasonable algorithm will not schedule j on such a machine. If j is fast on type k, we round it down to

$$\text{the nearest lower value } p_{kj}^{\text{round}} := d_j \, (1 + \varepsilon')^h \text{ for } h \in \mathbb{N} \ .$$

We therefore have $d_j(1 + \varepsilon')^h \leq p_{kj} < d_j(1 + \varepsilon')^{h+1}$. The new instance of scaled and rounded jobs $\mathcal{J}^{\text{round}}$ together with the (unchanged) machines is called I^{round}.

Remark 2. In this paper, we will sometimes directly refer to "fast jobs" and "slow jobs" although we mean e.g. "jobs scheduled on machines where they are fast". We may also call jobs "fast" or "slow" when we refer to their processing times, which should therefore be "the processing times of one job on machines where the job is fast" and "the processing times of one job on machines where it is slow." Similarly, we may also use expressions like "jobs on slow/fast machines" when we mean "jobs scheduled on machines where they are slow/fast."

Finally, jobs will later on also be called "large" and "small" such that similar expressions will be used.

Lemma 3 ([15, Lemma 2.1]). *We have* $\mathrm{OPT}(I^{\mathrm{round}}) \leq (1+\varepsilon')\,\mathrm{OPT}(I)$.

Proof. Let I' be the instance where the fast jobs are rounded, but the processing times of slow jobs have not been set to ∞. On the one hand, we obviously have $\mathrm{OPT}(I') \leq \mathrm{OPT}(I)$ because the processing times may only have decreased. On the other hand, we can take an optimum solution to I' and replace every rounded processing time p_{kj}^{round} by its original processing time p_{kj}. Then the schedule increases only by a factor of $1+\varepsilon'$. We get $\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq (1+\varepsilon')\mathrm{OPT}(I')$.

Let $\sigma' : \mathcal{J} \to \mathcal{M}$ be an optimum schedule for I', i.e. with a makespan of $\mathrm{OPT}(I')$. We transform it into a schedule σ'' for I^{round}. If a job j is scheduled by σ' to a machine on which it is slow, it is moved to an arbitrary machine where it is processed in time d_j (i.e. to one of the fastest machines for the job). Let S be the set of jobs that have been scheduled by σ' on slow machines. In the worst case, the processing time for one machine increases with the modified schedule σ'' by

$$\begin{aligned}\sum_{j\in S} d_j &\leq \sum_{j\in S} \frac{\varepsilon'}{m} \cdot p_{k(\sigma'(j))j} = \sum_{i\in\mathcal{M}} \sum_{j:\sigma'(j)=i} \frac{\varepsilon'}{m} \cdot p_{k(i)j} \leq \sum_{i\in\mathcal{M}} \frac{\varepsilon'}{m} \cdot \mathrm{OPT}(I') \\ &= \frac{\varepsilon'}{m} \cdot m \cdot \mathrm{OPT}(I') = \varepsilon'\mathrm{OPT}(I') \leq \varepsilon'\mathrm{OPT}(I)\ .\end{aligned}$$

The modified schedule is obviously a schedule for I^{round}. To sum up, we get

$$\begin{aligned}\mathrm{OPT}(I^{\mathrm{round}}) &\leq \mathrm{OPT}(I') + \varepsilon'\mathrm{OPT}(I') \leq \mathrm{OPT}(I) + \varepsilon'\mathrm{OPT}(I) \\ &= (1+\varepsilon')\,\mathrm{OPT}(I)\ .\end{aligned}$$

□

The following lemma allows us to derive an approximation algorithm for I from an algorithm for I^{round}.

Lemma 4 ([15, Lemma 2.3]). *If there is an approximation algorithm A_r for I^{round} such that $A_r(I^{\mathrm{round}}) \leq \alpha\mathrm{OPT}(I) + \beta$, then there is also an approximation algorithm A for I with*

$$A(I) \leq \alpha\,(1+\varepsilon')^2\,\mathrm{OPT}(I) + \beta(1+\varepsilon') \leq \Big(\alpha(1+\varepsilon')^2 + \beta(1+\varepsilon')\Big)\,\mathrm{OPT}(I)\ .$$

Proof (Sketch). Construct a schedule for I^{round} and replace the modified processing times by the unmodified ones. □

Now, jobs are grouped together in a special way to reduce their overall number. A rounded job j has the profile $(\Pi_{1,j}, \ldots, \Pi_{k,j})$ where $\Pi_{k,j} \in \mathbb{N}$ is the exponent such that $p_{kj}^{\text{round}} = d_j \, (1+\varepsilon')^{\Pi_{k,j}}$. We set $\Pi_{k,j} = \infty$ if $p_{kj}^{\text{round}} = \infty$.

Lemma 5. *Let l be the number of profiles. The bound $l \leq (2 + \log_{1+\varepsilon'}(\frac{m}{\varepsilon'}))^K$ holds.*

Proof. If a job j is fast on the machine type k, its maximum processing time is $\frac{m}{\varepsilon'} d_j$. Hence, we have for the largest exponent $h_{\max}$ that $d_j(1+\varepsilon')^{h_{\max}} \leq \frac{m}{\varepsilon'} d_j$ such that $h_{\max} = \lfloor \log_{1+\varepsilon'}(\frac{m}{\varepsilon'}) \rfloor$ follows. The smallest exponent is $h_{\min} = 0$. If j is slow on type k, then its exponent is ∞. Thus, there are $2 + \lfloor \log_{1+\varepsilon'}(\frac{m}{\varepsilon'}) \rfloor$ possible exponents on K positions, which yields $(2 + \log_{1+\varepsilon'}(\frac{m}{\varepsilon'}))^K$ possible profiles. (This proof is an extension of the proof in [15].) □

Let $\nu := \frac{1}{\lceil m/\varepsilon' \rceil}$. The jobs are partitioned into large jobs $L := \{j \mid d_j > \nu\}$ and small jobs $S := \{j \mid d_j \leq \nu\}$. Take an enumeration of the l profiles such that we can denote a profile directly by its number $\varsigma \in \{1, \ldots, l\}$. The set S is then further partitioned into the sub-sets $S_\varsigma = \{j \in \mathcal{J} \mid j \text{ has the profile } \varsigma\}$ for $\varsigma \in \{1, \ldots, l\}$.

Two jobs j_a and j_b with the same profile ς and for which $d_{j_a}, d_{j_b} \leq \frac{\nu}{2}$ holds are now grouped together to a new composed job j_c with $p_{kj_c} := p_{kj_a} + p_{kj_b}$ for every k. The composing is repeated until there is at most one job $j \in S_\varsigma$ with $d_j \leq \frac{\nu}{2}$ for every profile ς. The other jobs (including the jobs in L) now have all a processing time of at least $\frac{\nu}{2}$. The set of all jobs is called $\mathcal{J}^{\text{merge}}$, which yields together with the (unchanged) machines the instance I^{merge}.

Lemma 6. *If two jobs j_a and j_b are grouped together to j_c, then j_c has the same profile as j_a and j_b.*

Proof. We have $\Pi_{k,j_a} = \Pi_{k,j_b}$ for all $k \in \{1, \ldots, K\}$ because j_a and j_b have the same profile. Let k' be a machine type where $\Pi_{k',j_a} = \Pi_{k',j_b} = 0$, i.e. both jobs have their fastest processing time d_{j_a} and d_{j_b} on the machine type k'. We have

$$\begin{aligned} p_{kj_c}^{\text{round}} &= p_{kj_a}^{\text{round}} + p_{kj_b}^{\text{round}} = d_{j_a} \, (1+\varepsilon')^{\Pi_{k,j_a}} + d_{j_b} \, (1+\varepsilon')^{\Pi_{k,j_b}} \\ &= (d_{j_a} + d_{j_b}) \, (1+\varepsilon')^{\Pi_{k,j_a}} \quad \text{for all } k \in \{1, \ldots, K\} \quad . \end{aligned}$$

Thus, we have $p_{k'j_c}^{\text{round}} = d_{j_a} + d_{j_b}$, and $d_{j_c} = d_{j_a} + d_{j_b}$. The job j_c has therefore the same profile as j_a and j_b. □

Lemma 7. *After composing the items, we still have $1 \leq \text{OPT}(I^{\text{merge}}) \leq m$.*

Proof. Composed jobs have values d_{j_c} that are the sum of several d_j for $j \in \mathcal{J}(I^{\text{round}})$. Hence, we get $D = m = \sum_{j \in \mathcal{J}} d_j = \sum_{j \in \mathcal{J}^{\text{merge}}} d_j$ (see Assumption 1) because one $j \in \mathcal{J}(I^{\text{round}})$ can only be used in the composition of one job $j_c \in \mathcal{J}(I^{\text{merge}})$. The upper and lower bound now follow as before. □

Theorem 8. *The number of jobs in $\mathcal{J}^{merge}$ (i.e. in I^{merge}) is bounded by* $\min\{n, O(\frac{m^2}{\varepsilon'}) + (\frac{\log m}{\varepsilon'})^{O(K)}\}$.

Proof. The number of jobs does obviously not increase so that n is an upper bound. The number of jobs in $\mathcal{J}^{\text{merge}}$ that have a shortest processing time of $d_j > \frac{\nu}{2}$ is bounded by $\frac{\sum_{j\in\mathcal{J}} d_j}{\nu/2} = \frac{2D}{\nu}$. Moreover, there is at most one job left with $d_j \leq \frac{\nu}{2}$ for every profile ς. Therefore, we have at most $l + \frac{2D}{\nu}$ many jobs. Note that $0 < \varepsilon' \leq \frac{1}{3}$ such that $\ln(1+\varepsilon') \geq \varepsilon' - (\varepsilon')^2 = \varepsilon'(1-\varepsilon') \geq \frac{1}{2}\varepsilon'$ holds. We get

$$
\begin{array}{ccccc}
 & \frac{2D}{\nu} & + & l & \\
= & \underbrace{2m \cdot \left\lceil \frac{m}{\varepsilon'} \right\rceil} & + & \underbrace{\left(2 + \log_{1+\varepsilon'}\left(\frac{m}{\varepsilon'}\right)\right)^K} & \leq O\left(\frac{m^2}{\varepsilon'}\right) + \left(\frac{\log m}{\varepsilon'}\right)^{O(K)} \quad . \\
\leq & 2m \cdot \left(\frac{m}{\varepsilon'} + 1\right) & = & \left(2 + \frac{\ln(\frac{m}{\varepsilon'})}{\ln(1+\varepsilon')}\right)^K & \\
\leq & \frac{2m^2}{\varepsilon'} + \frac{m^2}{\varepsilon'} & \leq & \left(2 + \frac{2}{\varepsilon'} \cdot \ln\left(\frac{m}{\varepsilon'}\right)\right)^K & \\
= & O\left(\frac{m^2}{\varepsilon'}\right) & = & \left(\frac{\log m}{\varepsilon'}\right)^{O(K)} &
\end{array}
$$

The proof is an extension of the proof in [15]. □

Theorem 9 ([15, Lemma 2.4]). *We have*

$$\text{OPT}(I^{\text{round}}) \leq \text{OPT}(I^{\text{merge}}) \leq \text{OPT}(I^{\text{round}}) + \varepsilon' \quad .$$

We state the running time to construct I^{merge}.

Theorem 10. *I^{merge} can be constructed from I in time $O(n \cdot K)$.*

Proof. We have the following parts:

- Finding the values d_j for all jobs j, the calculation of D as well as the scaling of the instance can be done in $O(n \cdot K)$.
- We suppose that determining the logarithm can be done in $O(1)$ such that the exponent h for which $d_j(1+\varepsilon)^h \leq p_{kj} < d_j(1+\varepsilon')^{h+1}$ holds can be found in $O(1)$. Hence, the jobs can be rounded and their profiles determined in $O(n \cdot K)$. A rounded job is then directly added to the stack that corresponds to its profile. Thus, there are at most $O(n)$ profiles to be considered.
- The partition of the jobs in L and S is done in $O(n)$.
- The composition of the jobs is described in Algorithm 1. First, list(j) is created for every $j \in S$, a list that is used at the end of the algorithm to replace a composed item by the the ones it consists of. The jobs in S are already grouped into their respective profile sets S_ς (see above). Then, the jobs are further partitioned into $S_\varsigma^1 := \{j \in S_\varsigma \mid d_j > \frac{\nu}{2}\}$ and $S_\varsigma^2 := \{j \in S_\varsigma \mid d_j \leq \frac{\nu}{2}\}$. The jobs are iteratively combined into larger ones in each S_ς^2 until at most one job is left in each S_ς^2, i.e. there is at most one job in S_ς with $d_j \leq \frac{\nu}{2}$. The correctness of the algorithm is obvious because each S_ς^2 will always contain all small jobs of profile ς with $d_j \leq \frac{\nu}{2}$.

As for the running time, the division of the small jobs into the sets S_ς^1 and S_ς^2 needs time in $O(n)$ over all profiles ς. Fix one set S_ς^2. The combination of two jobs in S_ς^2 can be done in $O(K)$. For combining $\text{list}(j_a)$ and $\text{list}(j_b)$, suppose that linked lists are used for $\text{list}(\cdot)$. Since j_a and j_b are not used afterwards, we just concatenate $\text{list}(j_a)$ and $\text{list}(j_b)$ and save the result as $\text{list}(j_c)$. With the right implementation of linked lists, this needs only $O(1)$. Finally, checking whether j_c satisfies $d_{j_c} > \frac{\nu}{2}$ (and adapting S_ς^1 and S_ς^2) can be done in $O(1)$. Hence, one iteration of the while-loop needs time in $O(K)$. The while-loop can only be executed at most n times in total over all S_ς because all jobs will then be merged into one. Hence, we get a running time in $O(n \cdot K)$ for the entire algorithm. □

for $j \in S$ **do**
  $\text{list}(j) := \{j\}$; `// Information for undoing the composition of every item`
for *every* S_ς **do**
  Partition S_ς into $S_\varsigma^1 = \{j \in S_\varsigma \mid d_j > \frac{\nu}{2}\}$ and $S_\varsigma^2 = \{j \in S_\varsigma \mid d_j \leq \frac{\nu}{2}\}$;
  while *There are (at least) two jobs* $j_a, j_b \in S_\varsigma^2$ **do**
    Group j_a and j_b together to j_c;
    $\text{list}(j_c) := \text{list}(j_a) \cup \text{list}(j_b)$;
    if $d_{j_c} > \frac{\nu}{2}$ **then**
      $S_\varsigma^2 := S_\varsigma^2 \setminus \{j_c\}$ and $S_\varsigma^1 := S_\varsigma^1 \cup \{j_c\}$;

Algorithm 1: This procedure combines the items in S into larger ones until at most one item with $d_j \leq \frac{\nu}{2}$ is contained in every S_ς.

3 The Main Algorithm

Let $0 < \delta \leq \varepsilon' \leq \frac{1}{3}$ with $\delta = \Theta(\varepsilon')$. We present our algorithm for an instance I with n' items and $1 \leq \text{OPT}(I) \leq m$: it finds a solution of value at most $(1+\delta)\text{OPT}(I)$. Later on, I will in fact be I^{merge} such that we make the following assumption:

Assumption 2. I has $n' \leq O(\frac{m^2}{\varepsilon'}) + (\frac{\log m}{\varepsilon'})^{O(K)} = O(\frac{m^2}{\delta}) + (\frac{\log m}{\delta})^{O(K)}$ items.

3.1 Approximating the Optimum by Binary Search

We introduce another value $\delta' = \Theta(\delta)$ with $0 < \delta' \leq \delta \leq \varepsilon' \leq \frac{1}{3}$. Suppose that we have an oracle `Oracle(`I, T`)` that returns for a given makespan T and a constant $C > 0$ either a solution of value at most $(1 + C\delta')T$ or $\perp$ (false). The answer $\perp$ implies that there is not a solution of value (at most) T, i.e. $T < \text{OPT}(I)$. We employ the well-known dual approximation approach [11, 18], a binary search with such an oracle, to approximate the optimum $\text{OPT}(I)$ up to a given approximation ratio (see Algorithm 2). We start with the lower bound $LB = 1$ and the upper bound $UB = m$ and first check whether $LB = 1$

yields a solution. If not, we iteratively adapt LB and UB until the difference $UB - LB$ is small enough. At the same time, we ensure that $LB < OPT(I)$ always holds and that there is a solution for the value $T = UB$: the solution for UB is returned at the end. Note that the oracle can construct a solution for $T = m$ in $O(n' + m)$ as described at the beginning of Section 2.

$LB := 1$;
$UB := m$;
if `Oracle(`I`,`LB`)` $\neq \perp$ **then** `// Solution for` $T = LB$ `exists`
  return `Oracle(`I`,`LB`)`;
while $UB - LB > \delta'$ **do**
  $T := \frac{UB+LB}{2}$;
  $result :=$ `Oracle(`I`,` T`)`;
  if $result = \perp$ **then** `//` T `is too small`
    $LB := T$;
  else `//` T `is large enough`
    $UB := T$;
$T := UB$;
return `Oracle(`I`,`T`)`;

Algorithm 2: Binary search to approximate $\mathrm{OPT}(I)$ with the oracle

Lemma 11. *Suppose that the* `Oracle` *function has the properties above, i.e. it returns for given T either a solution with a makespan of at most $(1 + C\delta')T$ or $\perp$. In the second case, we have $T < \mathrm{OPT}(I)$. Then, the dual approximation approach, i.e. the binary search, finds a schedule with a makespan of at most $(1 + (C + 1)\delta' + C(\delta')^2)\,\mathrm{OPT}(I)$ and needs $O(\log(\frac{m}{\delta'}))$ iterations.*

Proof. It is clear that the algorithm finishes after $O(\log(\frac{m}{\delta'}))$ iterations of the binary search.

The correctness of the binary search has already been shown in [11, 18] and is easy to see: first, we have at the beginning $1 = LB \leq \mathrm{OPT}(I)$ and that there is a solution for $UB = m$. If the oracle does not return a solution for $T = LB$, we have $LB < \mathrm{OPT}(I)$. The algorithm then makes sure by the adaptation of T that we always have $LB < \mathrm{OPT}(I)$ and that the oracle returns for $T = UB$ a solution (of value at most $(1 + C\delta')T$). Suppose that the binary search terminates, i.e. $UB - LB \leq \delta'$ holds. The oracle then returns a solution with a makespan of at most $(1 + C\delta')UB$. Since we have $UB \leq LB + \delta' \leq \mathrm{OPT}(I) + \delta'$, we get

$$\begin{aligned}
(1 + C\delta')\,UB &\leq (1 + C\delta')\,(\mathrm{OPT}(I) + \delta') \\
&= (1 + C\delta')\,\mathrm{OPT}(I) + (1 + C\delta')\,\delta' \\
&\leq (1 + C\delta')\,\mathrm{OPT}(I) + (1 + C\delta')\,\delta'\mathrm{OPT}(I) \\
&= (1 + C\delta')\,(1 + \delta')\,\mathrm{OPT}(I) \\
&= \left(1 + (C + 1)\delta' + C(\delta')^2\right)\mathrm{OPT}(I)\ .
\end{aligned}$$

□

3.2 The Oracle

We now describe the principle of the oracle. As a first step, the processing times of the jobs in I are divided by T. We get a new instance I^{scale} with $\text{OPT}(I^{\text{scale}}) \leq 1$ (if $\text{OPT}(I) \leq T$). Then, the jobs are rounded to get the instance I^{r} with $\text{OPT}(I^{\text{r}}) \leq (1+\delta')$. A dynamic program `DynProg` is used to iteratively construct the sets $TS_0, \ldots, TS_{n'}$ of *profiles*, where each profile represents several (real) schedules. (A formal definition is given below.) The profiles in TS_j consider the first j jobs $\{1, \ldots, j\}$. At the end, a function `CreateSchedule` tries to construct a discrete schedule σ for the instance I^{p} (which is similar to I^{r}) from each profile $t \in TS_{n'}$, where the profiles in $TS_{n'}$ consider all n' jobs. If there is a solution to I with a makespan of at most T, one discrete schedule σ for I with a makespan of at most $(1+C\delta')T$ will be found by `CreateSchedule`. An overview is shown in Algorithm 3. We first have the following obvious lemma:

Lemma 12. *The set I^{scale} can be constructed in $O(n' \cdot K)$.*

```
Construct I^scale from I;
Round I^scale to I^r;
TS = (TS_0, ..., TS_n') := DynProg(I^r);
for all t ∈ TS_n' do
    σ := CreateSchedule(t);
    if σ ≠ ⊥ then
        // A schedule of value at most (1 + Cδ') has been found for I^p
        return σ;
return ⊥ ; // No schedule for I with a makespan ≤ T
```

Algorithm 3: An overview of the `Oracle(`I, T`)`

In a slight abuse of notation, we still denote the scaled processing times by p_{kj}.

Definition 13. *A (scaled) job j is large on a machine type k if $p_{kj} \geq \delta'$. Otherwise, it is small.*

Take one job j. If its processing time is large on a machine type k, it is rounded up to the next $\gamma \cdot (\delta')^2$ for $\gamma \in \mathbb{N}$. If the processing time is small, i.e. $p_{kj} < \delta'$, the processing time is rounded *down* to the next multiple of $\frac{m_k \cdot \delta'}{n'}$. This new instance with processing times p^{r}_{kj} is denoted by I^{r}.

Note that jobs are large (or small) on a machine type in I^{r} if they are large (or small) in I^{scale}, and vice versa.

Lemma 14. *If I^{scale} has a schedule with a makespan of at most 1, then I^{r} has a schedule with a makespan of at most $1+\delta'$. I^{r} can be constructed in $O(n' \cdot K)$.*

Proof. Let σ be a schedule for I^{scale} with a makespan $T \leq 1$. By definition, there can only be $\frac{1}{\delta'}$ large jobs on one machine. Replace all jobs by their rounded counterpart in I^{r}. The processing times of small jobs may only decrease while each large job increases by at

most $(\delta')^2$. Hence, the increase of the total processing time on a machine is bounded by $(\delta')^2 \cdot \frac{1}{\delta'} = \delta'$. The new total processing time of machine i is therefore at most $T + \delta' \leq 1 + \delta'$. Since this holds for all machines, the bound on the makespan follows for I^{r}.

The running time to obtain I^{r} is obvious. □

Take one schedule with a makespan of at most $1 + \delta'$. If only large jobs of I^{r} are scheduled on a machine i, its total processing time is a multiple of $(\delta')^2$. In fact, it must be one of the values

$$\{0\} \cup \left\{\gamma \cdot (\delta')^2 \mid \gamma \in \mathbb{N} \text{ and } \delta' \leq \gamma \cdot (\delta')^2 \leq 1 + \delta'\right\} \ .$$

These processing times can be numbered with $\gamma = 0$ (for the total processing time 0) and $\gamma \in \{\gamma_0 := \lceil \frac{1}{\delta'} \rceil, \gamma_0 + 1, \ldots, \gamma_1 - 1, \gamma_1 := \lfloor \frac{1+\delta'}{(\delta')^2} \rfloor\}$. If j is large, the value $\gamma(k, j)$ is the factor such that $p^{\mathrm{r}}_{kj} = \gamma(k, j) \cdot (\delta')^2$.

Lemma 15. *For I^{r}, there are $O(\frac{1}{(\delta')^2})$ processing times of the form $\gamma \cdot (\delta')^2$ of large jobs on a machine.*

Similarly, take all small jobs assigned to a machine type k. Their total processing time is at most $m_k \cdot (1 + \delta')$ because we consider a schedule with a makespan of at most $1 + \delta'$. Moreover, the total processing time is also a multiple of $\frac{m_k \cdot \delta'}{n'}$ because of the rounding, i.e. it is one of the values in

$$\Sigma_k := \left\{ \tau \cdot \frac{m_k \cdot \delta'}{n'} \;\middle|\; \tau \in \mathbb{N} \text{ and } 0 \leq \tau \cdot \frac{m_k \cdot \delta'}{n'} \leq m_k \cdot (1 + \delta') \right\} \ .$$

Lemma 16. *For one machine type k of I^{r}, there are $O(\frac{n'}{\delta'}(1 + \delta')) = O(\frac{n'}{\delta'})$ possible total processing times of small jobs in Σ_k.*

Proof. The small jobs have a total processing time in the interval $[0, (1 + \delta') \cdot m_k]$. Since the processing times are a multiple of $\frac{m_k \cdot \delta'}{n'}$, the lemma follows. □

Based on the observations above, we introduce several useful definitions.

Definition 17. *Let I' be a sub-instance of I^{r}, i.e. an instance whose jobs $\mathcal{J}'$ are a subset of the jobs in I^{r}, and which has the same machines as I^{r}. Let $\sigma : \mathcal{J}' \to \mathcal{M}$ be a feasible schedule with a makespan of at most $1 + \delta'$. Let b_i be the total processing time of the large jobs assigned to machine i, i.e.*

$$b_i = b_i(\sigma) := \sum_{j : \sigma(j) = i, p^{\mathrm{r}}_{k(i)j} \geq \delta'} p^{\mathrm{r}}_{k(i)j} \ .$$

We also introduce the remaining processing time (or remaining machine capacity) of every machine type k for the makespan $1 + \delta'$:

$$r_k = r_k(\sigma) := \sum_{i \in \mathcal{M}_k} (1 + \delta' - b_i) = m_k \cdot (1 + \delta') - \sum_{i \in \mathcal{M}_k} b_i \ .$$

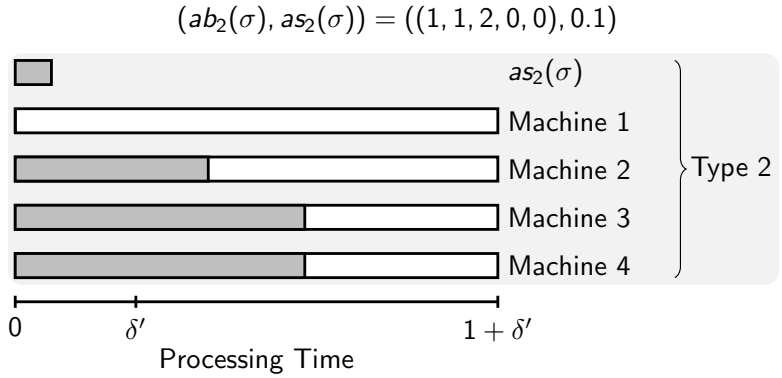


Figure 1: The total processing time of the small jobs assigned to type $k = 2$ is $as_2(\sigma) = 0.1$. There is $ab_2(\sigma, 0) = 1$ machine to which no large job has been assigned, and $ab_2(\sigma, \gamma_0) = 1$ machine with a total processing time of $\gamma_0 \cdot (\delta')^2$ for the large jobs. Moreover, there are $ab_2(\sigma, \gamma_0 + 1) = 2$ machines with a total processing time of $(\gamma_0 + 1) \cdot (\delta')^2$ for the large jobs. There are no machines with processing times $\gamma \cdot (\delta')^2$ for $\gamma > \gamma_0 + 1$.

Moreover, the value $ab_k(\sigma, \gamma)$ denotes the number of machines of type k where $b_i = \gamma \cdot (\delta')^2$:

$$ab_k(\sigma, \gamma) := \left|\left\{i \in \mathcal{M}_k \mid b_i = \gamma \cdot (\delta')^2\right\}\right| \text{ for } \gamma \in \{0, \gamma_0, \ldots, \gamma_1\} \quad .$$

The vector $ab_k(\sigma) = (ab_k(\sigma, \gamma))_{\gamma=0,\gamma_0,\ldots,\gamma_1}$ contains all entries $ab_k(\sigma, \gamma)$.

Furthermore, $as_k(\sigma)$ is the total processing time of all small jobs assigned to machine type k, i.e.

$$as_k(\sigma) := \sum_{j:\sigma(j)\in\mathcal{M}_k, p^{\mathrm{r}}_{kj}<\delta'} p^{\mathrm{r}}_{kj} \quad .$$

As seen above, we have $as_k(\sigma) \in \Sigma_k$. Since the small jobs have to fit into the remaining processing time, $as_k(\sigma) \leq r_k(\sigma)$ holds for all $k \in \{1, \ldots, K\}$.

The values $ab_k(\sigma, \gamma)$ and $as_k(\sigma)$ form the profile *of σ:*

$$\Bigg(\Big[\Big(ab_1(\sigma, 0), ab_1(\sigma, \gamma_0), \ldots, ab_1(\sigma, \gamma_1)\Big), as_1(\sigma)\Big], \ldots, \Big[\Big(ab_K(\sigma, 0), ab_K(\sigma, \gamma_0), \ldots, ab_K(\sigma, \gamma_1)\Big), as_K(\sigma)\Big]\Bigg) \quad .$$

Profiles are illustrated in Figures 1 and 2.

3.3 Dynamic Programming

The dynamic program we introduce determines all possible profiles for I^{r}. Therefore, one profile t for a sub-instance I' is represented like above: it is a tuple of K tuples, one for each machine type:

$$t = ((AB_1, AS_1), \ldots, (AB_k, AS_k), \ldots, (AB_K, AS_K)) \quad .$$

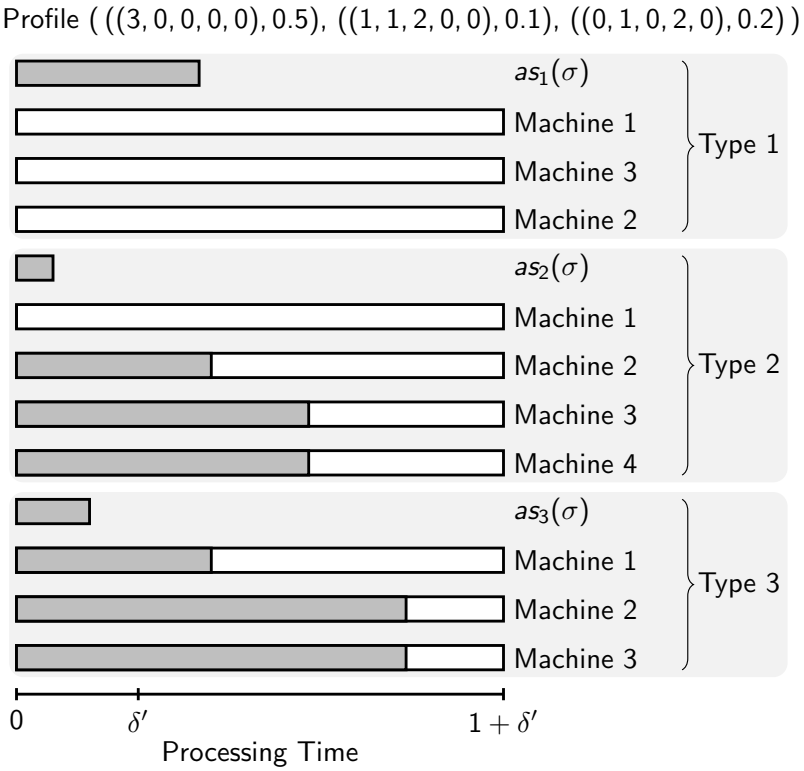


Figure 2: An illustration of the complete profile for a schedule σ.

For each $k \in \{1, \ldots, K\}$, the entry AS_k denotes the total processing time of all small jobs that are assigned to the machines of type k.

One AB_k is again a tuple

$$AB_k = (q_0, q_{\gamma_0}, \ldots, q_\gamma, \ldots, q_{\gamma_1}) \quad .$$

Each entry q_γ denotes the number of machines of type k where the large jobs have the total processing time $\gamma \cdot (\delta')^2$. Obviously, $q_0 + \sum_{\gamma=\gamma_0}^{\gamma_1} q_\gamma = m_k$ holds.

For convenience, $AS_k(t)$ denotes the entry AS_k of a profile t. Similarly, $AB_k(t)$ stands for the tuple AB_k of profile t. Additionally, $(AB_k(t))_\gamma$ is the entry q_γ in the tuple $AB_k(t)$.

Lemma 18. *One profile t has $O(\frac{K}{(\delta')^2})$ entries.*

Proof. There are K entries $AS_k(t)$ and K tuples $AB_k(t)$. Each $AB_k(t)$ has again $O(\gamma_1) = O(\frac{1}{(\delta')^2})$ many entries (see Lemma 15). The overall bound follows. □

The idea of the dynamic program `DynProg` (as shown in Algorithm 5) is quite simple. We start with the tuple set

$$TS_0 = \left\{ \left(\left(\underbrace{(\overbrace{m_1}^{q_0}, 0, \ldots, 0)}_{AB_1}, \underbrace{0}_{AS_1} \right), \ldots, \left(\underbrace{(\overbrace{m_K}^{q_0}, 0, \ldots, 0)}_{AB_K}, \underbrace{0}_{AS_K} \right) \right) \right\} \tag{1}$$

that represents the empty schedule: for every machine type k, small jobs have not been assigned ($AS_k = 0$), and m_k machines (i.e. all machines of type k) have a total processing time of large jobs equal to 0.

Suppose that the set TS_{j-1} has been determined: it contains all profiles that can be obtained for the first $j-1$ jobs $\{1, \ldots, j-1\}$. The profiles for $\{1, \ldots, j\}$ are constructed by considering for each $t \in TS_{j-1}$ all possibilities to add j to t. Fix one $t \in TS_{j-1}$. We go over all k. If j is small on type k, then $AS_k(t)$ is simply increased by p^{r}_{kj} (lines 6–12): we have a new profile t' where additionally j is assigned to the machine type k. If j is large on type k, all $q_\gamma = (AB_k(t))_\gamma > 0$ are taken into account (lines 13–20): there are q_γ machines of type k in t where each has the total processing time $\gamma \cdot (\delta')^2$. If we add j to one of these machines, there is obviously one machine less with the processing time $\gamma \cdot (\delta')^2$ and one machine more with the processing time $\gamma \cdot (\delta')^2 + p^{\mathrm{r}}_{kj} = \gamma \cdot (\delta')^2 + \gamma(k,j) \cdot (\delta')^2$. Hence, q_γ decreases and $q_{\gamma+\gamma(k,j)}$ increases by one. The `add` operation is shown in Algorithm 4. Thus, each $q_\gamma > 0$ in $AB_k(t)$ generates a new profile $t' \in TS_j$.

Input: AB_k, γ, j
Set $\gamma' := \gamma + \gamma(k,j)$;
$(AB_k)_{\gamma'} := (AB_k)_{\gamma'} + 1$;
$(AB_k)_\gamma := (AB_k)_\gamma - 1$;
return AB_k;

Algorithm 4: `add`(AB_k, γ, j)

When t' has been constructed, it is checked whether $t' \in TS_j$ already holds (lines 10 and 18), i.e. whether we already have obtained the tuple t' in another way (e.g. from another $t \in TS_{j-1}$). If no, we save t' together with the corresponding backtracking information to later construct a schedule (lines 11–12 and 19–20). If yes, we only keep the old backtracking information. Note that the `add` operation is only executed if $\gamma \cdot (\delta')^2 + p^{\mathrm{r}}_{kj} \leq (1+\delta')$, i.e. $\gamma + \gamma(k,j) \leq \gamma_1$ (see line 15): we only want to find the profiles representing schedules with makespans of at most $1+\delta'$. Similarly, the bound $AS_k(t) + p^{\mathrm{r}}_{kj} \leq m_k \cdot (1+\delta')$ for AS_k is checked in line 7. It is therefore possible that there is not any profile $t \in TS_{j-1}$ such that j can be assigned: the value T is too small. Then, `DynProg` returns the empty set (line 22), and `Oracle` will therefore return $\perp$. We show that a schedule for I^{r} with a makespan of at most $(1+\delta')$ corresponds to at least one profile.

Lemma 19. *Let σ be a schedule for I^{r} with a makespan of at most $1+\delta'$. Then the dynamic program* `DynProg` *generates a profile t for I^{r} where*

- $(AB_k(t))_\gamma = ab_k(\sigma, \gamma)$ *for all* $\gamma \in \{0, \gamma_0, \ldots, \gamma_1\}$ *and* $k \in \{1, \ldots, K\}$, *and*
- $AS_k(t) = as_k(\sigma)$ *for all* $k \in \{1, \ldots, K\}$.

Proof. The statement is easy to see because the dynamic program does exactly what a natural algorithm to construct σ would do: it takes all currently constructed schedules for the first $j-1$ jobs and tries for each of these schedules to assign job j to every machine i. However, only the corresponding profiles are saved.

Since σ has a makespan of at most $1+\delta'$, all conditions checked during the construction of t_σ will be satisfied. □

Input: Instance I^{r}, $\delta' > 0$

Set TS_0 as seen in Equation (1);
for $j = 1, \ldots, n'$ **do**
    $TS_j := \emptyset$;
    for $t \in TS_{j-1}$ **do**
        for $k = 1, \ldots, K$ **do**
            if j *is small on type* k **then**
                if $AS_k(t) + p^{\mathrm{r}}_{kj} \leq m_k \cdot (1 + \delta')$ **then**
                    $t' := t$;
                    $AS_k(t') := AS_k(t') + p^{\mathrm{r}}_{kj}$;
                    if $t' \notin TS_j$ **then**
                        Backtrack$(t') := (t, k)$;
                        $TS_j := TS_j \cup \{t'\}$;
            else // j `is large on type` k
                for $\gamma = 0, \gamma_0, \ldots, \gamma_1 - 1$ **do**
                    if $q_\gamma = (AB_k(t))_\gamma > 0$ *and* $\gamma + \gamma(k, j) \leq \gamma_1$ **then**
                        $t' := t$;
                        `add`$(AB_k(t'), \gamma, j)$;
                        if $t' \notin TS_j$ **then**
                            Backtrack$(t') := (t, k, \gamma)$;
                            $TS_j := TS_j \cup \{t'\}$;
    if $TS_j = \emptyset$ **then**
        return $\emptyset$;
return $TS = (TS_0, TS_1, \ldots, TS_{n'})$;

Algorithm 5: The dynamic program `DynProg`

Lemma 20. *The number κ of profiles in TS_j is bounded by*

$$\kappa \leq m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)} .$$

Proof. Fix one machine type k. The number of possible vectors AB_k is bounded by $(m+1)^{O(1/(\delta')^2)} = m^{O(1/(\delta')^2)}$ because AB_k has $O(\frac{1}{(\delta')^2})$ entries (see Lemma 15), and each entry $(AB_k)_\gamma$ is in $\{0, 1, \ldots, m_k\} \subseteq \{0, \ldots, m\}$.

For every value of AB_k, we will only save $O(\frac{n'}{\delta'})$ possible values for AS_k (see Lemma 16). Note that we have $n' \leq O(\frac{m^2}{\delta}) + (\frac{\log m}{\delta})^{O(K)} = O(\frac{m^2}{\delta'}) + (\frac{\log m}{\delta'})^{O(K)}$ because of Assumption 2 and $\delta' = \Theta(\delta)$. We get

$$O\left(\frac{n'}{\delta'}\right) = O\left(\frac{m^2}{(\delta')^2}\right) + \left(\frac{\log m}{\delta'}\right)^{O(K)} .$$

If we consider all machine types k, we have the upper bound on the number of profiles of

$$\begin{aligned}
\left(m^{O(1/(\delta')^2)} \cdot O\left(\frac{n'}{\delta'}\right)\right)^K &= \left(m^{O(1/(\delta')^2)} \cdot \left(O\left(\frac{m^2}{(\delta')^2}\right) + \left(\frac{\log m}{\delta'}\right)^{O(K)}\right)\right)^K \\
&= \left(m^{O(1/(\delta')^2)} \cdot \frac{m^2}{(\delta')^2} + m^{O(1/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K)}\right)^K \\
&= \left(m^{O(1/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K)}\right)^K \\
&= m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)} .
\end{aligned}$$

We have used that $\frac{1}{(\delta')^2} = 2^{2\log(1/(\delta'))} \leq 2^{O(1/\delta')} \leq m^{O(1/\delta')}$ so that $\frac{m^2}{(\delta')^2} \cdot m^{O(1/(\delta')^2)} = m^{O(1/(\delta')^2)}$. □

Lemma 21. *The dynamic program (Algorithm 5) needs time in $O(\frac{K}{(\delta')^2} \cdot m \cdot \kappa \cdot n') = m^{O(K/(\delta')^2)}(\frac{\log m}{\delta'})^{O(K^2)}$ to construct all profiles $TS_0, \ldots, TS_{n'}$.*

Proof. The for-loop for the items needs n' iterations. When an item is processed in the for-loop, it is tried to assign it to each of the $O(\kappa)$ profiles in TS_{j-1}. Fix one profile $t \in TS_{j-1}$ and one machine type k. If the item is small on machine type k, we need $O(1)$ to check whether j can be added to type k. If the item is large, we need $O(\gamma_1) \overset{\text{Lem. } 15}{=} O(\frac{1}{(\delta')^2})$ to check the entries q_γ and the conditions whether j can be added. Note that there are $O(m_k)$ entries $q_\gamma > 0$ and therefore only $O(m_k)$ cases where an item is added.

When an item is added, we need time in $O(\frac{K}{(\delta')^2})$ to create the new profile t' and add the item to it (see Lemma 18; note that the `add` operation of Algorithm 4 needs time in $O(1)$). Additionally, it has to be checked whether t' is new in TS_j. We can suppose that this can be done in the size of the profile $O(\frac{K}{(\delta')^2})$: all profiles $t'' \in TS_j$ can be saved in

one array where the position of t'' is given by its values $(AB_k(t''))_\gamma$ and $AS_k(t'')$. Finally, the backtracking information only has to save a pointer to the old profile from which the new one was obtained as well as the values k and/or γ. In total, the time needed to add an item is bounded by $O(\frac{K}{(\delta')^2})$.

Obviously, the case where j is large on type k dominates the running time for one machine type. The dynamic program therefore needs time in

$$O\left(n' \cdot \kappa \cdot \sum_{k=1}^{K} \left(\underbrace{\frac{1}{(\delta')^2}}_{\text{checks of } q_\gamma} + \underbrace{m_k}_{\text{number of } q_\gamma > 0} \cdot \underbrace{\frac{K}{(\delta')^2}}_{\text{time to add an item}} \right)\right) = O\left(n' \cdot \kappa \cdot m \cdot \frac{K}{(\delta')^2}\right) \ .$$

We get

$$\begin{aligned}
&O\left(\frac{K}{(\delta')^2} \cdot m \cdot \kappa \cdot n'\right) \\
&= O\left(\frac{K}{(\delta')^2} \cdot m \cdot \left[m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)}\right] \cdot \left[\left(\frac{m^2}{\delta'}\right) + \left(\frac{\log m}{\delta'}\right)^{O(K)}\right]\right) \\
&= \frac{K}{(\delta')^2} \cdot m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)} = K \cdot m^{O(K/(\delta')^2)} \cdot \frac{1}{(\delta')^2} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)} \\
&= m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)}
\end{aligned}$$

with Assumption 2 and Lemma 20. □

Remark 22. Profiles can of course be stored in a more compact form: we only have to save the strictly positive $ab_k(\sigma, \gamma)$ and $(AB_k(t))_\gamma$. The number of entries in one profile is then bounded by $O(K + m)$. However, the number of profiles κ does not decrease: it is independent of the fact whether we save the $q_\gamma = 0$ or not. Lemma 20 still holds so that the asymptotic running time also remains unchanged (see the proof of Lemma 21).

3.4 Construction of a Schedule

The goal of this section is the construction of a schedule with a makespan of at most $1 + 3\delta'$ from a suitable profile. We first state two definitions.

Definition 23. *For a given profile t', the remaining total processing time for every machine type k is defined by*

$$R_k = R_k(t') := m_k \cdot (1 + \delta') - \sum_\gamma (AB_k(t))_\gamma \cdot \gamma \cdot (\delta')^2 \ .$$

The definition corresponds to the one of r_k in Definition 17.

Definition 24. *Take one job j in I. If its processing time is large on machine type k, it is rounded up to the next $\gamma \cdot (\delta')^2$ for $\gamma \in \mathbb{N}$. The small processing times of a job are not changed. This is the new instance I^{p} whose processing times are denoted by p^{p}_{kj}.*

I^{p} is therefore the instance I^{r} without the rounding of the small jobs. We have $p^{\mathrm{p}}_{kj} = p_{kj}$ if j is small on type k, and we have $p^{\mathrm{p}}_{kj} = p^{\mathrm{r}}_{kj}$ if j is large on type k.

After `DynProg`, `Oracle` (Algorithm 3) calls for every $t' \in TS_{n'}$ `CreateSchedule`, a function that is shown in Algorithm 6. First, `CreateSchedule(`t'`)` checks whether enough processing time is left for the small jobs (lines 1–4), i.e. whether $AS_k(t') \leq R_k(t')$ holds for all $k \in \{1, \ldots, K\}$. If yes, this is sufficient to construct a schedule for I^{P} of value at most $1 + 3\delta'$ (lines 5–9) as will be shown below in Lemma 25. In line 6, the function `Backtracking` uses the backtracking information to find the set of small jobs $\mathcal{J}_k$ assigned to machine type k. Moreover, it constructs a schedule σ_k for the large jobs assigned to type k. This is shown in detail below in Algorithm 7. When `Backtracking` has finished, σ is updated according to each σ_k (line 8). The small jobs assigned to k are greedily added to the machines $\mathcal{M}_k$ (line 9): a machine in $\mathcal{M}_k$ gets assigned jobs in $\mathcal{J}_k$ until the total processing time of the machine exceeds $1 + 2\delta'$. Then, the next machine in $\mathcal{M}_k$ is processed in the same way. As mentioned above, it will be shown in Lemma 25 that this procedure will be successful.

Note that `Oracle` returns $\perp$ if it cannot construct a schedule (because $TS_{n'} = \emptyset$ or no t' satisfies $R_k(t') \leq AS_k(t')$ for all k). It will be shown in Theorem 27 that `Oracle` indeed satisfies the properties of Lemma 11.

Input: Profile t'
1 **for** $k = 1, \ldots, K$ **do**
2 Determine $R_k = R_k(t')$;
3 **if** $AS_k(t') > R_k$ **then**
4 **return** $\perp$;
5 $\sigma := \emptyset$; `// The empty schedule`
6 $(\sigma_1, \mathcal{J}_1), \ldots, (\sigma_K, \mathcal{J}_K) :=$ `Backtracking(`t'`)`;
7 **for** $k = 1, \ldots, K$ **do**
8 Set $\sigma(j) := \sigma_k(j)$ for the jobs j scheduled by σ_k;
9 Assign the jobs in $\mathcal{J}_k$ greedily to the machines in $\mathcal{M}_k$ and adapt σ accordingly;
10 **return** σ;

Algorithm 6: `CreateSchedule(`t'`)` from the oracle (Algorithm 3)

Let us present the `Backtracking` function as shown in Algorithm 7. The backtracking information of `DynProg` allows us to go directly from $t' = t'_{n'} \in TS_{n'}$ to $t'_{n'-1} \in TS_{n'-1}$ from which t'_n has been constructed. We continue and get $t'_{n'-2} \in TS_{n'-2}, \ldots, t'_j \in TS_j, \ldots, t'_1 \in TS_1, t'_0 \in TS_0$, where t'_j has been obtained from t'_{j-1}. Next, we define the σ_k and $\mathcal{J}_k$ (line 3).

Starting from t'_0 (which is the profile of the empty schedule, see (1)), we have two cases. Either t'_j has been constructed from t'_{j-1} by adding j to a machine type k where it is large, and in $\mathcal{M}_k$ to one machine with the current processing time $\gamma \cdot (\delta')^2$ for one γ. The corresponding value γ and the machine type k are exactly the values stored in Backtrack(t'_j) (see line 19 of Algorithm 5). The current schedule σ_k is then updated accordingly by assigning j to one machine in $\mathcal{M}_k$ with a current processing time of $\gamma \cdot (\delta')^2$ (lines 8–11). Otherwise, j has been assigned to a type k where it is small. The value of k is stored in Backtrack(t_j) (see line 11 of Algorithm 5). The job j is then added to $\mathcal{J}_k$ (lines 12–14). The assignment of large jobs to the machines is rendered more comfortable

by the function extime(·): it keeps track of the current total processing time of large jobs on every machine.

Input: Profile t'

Determine by backtracking the profiles $t' = t'_{n'} \in TS_{n'}, t'_{n'-1} \in TS_{n'-1}, \ldots, t'_j \in TS_j, \ldots, t'_1 \in TS_1, t'_0 \in TS_0$ from which t' has been constructed;
for $k = 1, \ldots, K$ **do**
  $\sigma_k := \emptyset$ and $\mathcal{J}_k := \emptyset$;
for $i = 1, \ldots, m$ **do**
  extime$(i) := 0$; `// Total processing time of large jobs on machine` i
for $j = 1, \ldots, n'$ **do**
  if Backtrack(t'_j) *is of the form* (t'_{j-1}, k, γ) **then**
    `//` j `is added to a machine type where it is large`
    Take k and γ from Backtrack(t'_j);
    Use extime(·) to find one machine i in $\mathcal{M}_k$ with a total processing time of large jobs equal to $\gamma \cdot (\delta')^2$;
    Add j to i: $\sigma_k(j) := i$;
    extime$(i) :=$ extime$(i) + p^{\mathrm{r}}_{kj}$;
  else `//` j `is added to a machine type where it is small`
    Take k from Backtrack(t'_j) ; `//` Backtrack$(t'_j) = (t'_{j-1}, k)$
    $\mathcal{J}_k := \mathcal{J}_k \cup \{j\}$;
return $(\sigma_1, \mathcal{J}_1), \ldots, (\sigma_K, \mathcal{J}_K)$;

Algorithm 7: `Backtracking` (t') from Algorithm 6

Lemma 25. *Let $t \in TS_{n'}$ be a profile for which $AS_k(t) \leq R_k(t)$ holds for all k. Then,* `CreateSchedule` *and* `Backtracking` *(Algorithms 6 and 7) return a schedule σ for I^{p} with a makespan of at most $1 + 3\delta'$. It is also a schedule for I of value at most $(1 + 3\delta')T$.*

Proof. As we have $AS_k(t) \leq R_k(t)$, `CreateSchedule(`t`)` calls `Backtracking(`t`)`, which returns the schedules σ_k and the job sets $\mathcal{J}_k$ for all $k \in \{1, \ldots, K\}$. For each $k \in \{1, \ldots, K\}$, the algorithm takes the schedule σ_k for the machines $\mathcal{M}_k$ and greedily assigns the small jobs in $\mathcal{J}_k$ as explained above. When this is done for all $k \in \{1, \ldots, K\}$, the schedule σ for I^{p} has been constructed. Note that the σ_k are formally schedules for jobs in I^{r}, and the sets $\mathcal{J}_k$ are also taken from $\mathcal{J}(I^{\mathrm{r}})$. However, they can be trivially adapted to I^{p}: the instances I^{p} and I^{r} have a one-to-one correspondence of the jobs and differ only in the processing times p^{p}_{kj} and p^{r}_{kj}.

We have to show that all small jobs in the sets $\mathcal{J}_k$ can indeed be greedily assigned and that the resulting schedule has a makespan of at most $1 + 3\delta'$. We first state several identities. For the total processing time of large jobs on a machine, $b_i(\sigma_k) = b_i(\sigma)$ holds: we have $p^{\mathrm{r}}_{kj} = p^{\mathrm{p}}_{kj}$ for jobs j that are large on a machine type k, and σ_k is σ restricted to the large jobs assigned to machine type k. Hence, we also have $ab_k(\sigma, \gamma) = ab_k(\sigma_k, \gamma)$ for each γ. Moreover, $ab_k(\sigma_k, \gamma) = (AB_k(t))_\gamma$ holds: it is clear that the backtracking

constructs σ_k in such a way that it has a profile for the large jobs on k that corresponds to t. By induction, it can indeed be shown that $(AB_k(t_j))_\gamma = ab_k((\sigma_k)^{(j)}, \gamma)$, where $(\sigma_k)^{(j)}$ is the schedule σ_k restricted to the large jobs in $\{1, \ldots, j\}$, and $t_j \in TS_j$ is the profile of which t is constructed. To sum up, we have $ab_k(\sigma, \gamma) = ab_k(\sigma_k, \gamma) = (AB_k(t))_\gamma$ for all $\gamma \in \{0, \gamma_0, \ldots, \gamma_1\}$.

From this, we can directly derive the following identity for the remaining machine capacity on machines of type k:

$$\begin{aligned} r_k(\sigma) &= \sum_{i \in \mathcal{M}_k} (1 + \delta' - b_i(\sigma)) = (1 + \delta') \cdot m_k - \sum_{\gamma} \sum_{i : b_i(\sigma) = \gamma \cdot (\delta')^2} \gamma \cdot (\delta')^2 \\ &= (1 + \delta') \cdot m_k - \sum_{\gamma} ab_k(\sigma, \gamma) \cdot \gamma \cdot (\delta')^2 \\ &= (1 + \delta') \cdot m_k - \sum_{\gamma} (AB_k(t))_\gamma \cdot \gamma \cdot (\delta')^2 = R_k(t) \ . \end{aligned}$$

Note first that $p^{\mathrm{p}}_{kj} \leq p^{\mathrm{r}}_{kj} + \frac{m_k \cdot \delta'}{n'}$ holds for small jobs. Thus, we have $\sum_{j \in \mathcal{J}_k} p^{\mathrm{p}}_{kj} \leq \sum_{j \in \mathcal{J}_k} (p^{\mathrm{r}}_{kj} + \frac{m_k \cdot \delta'}{n'}) \leq (\sum_{j \in \mathcal{J}_k} p^{\mathrm{r}}_{kj}) + m_k \cdot \delta'$. As we have the identity $AS_k(t) = \sum_{j \in \mathcal{J}_k} p^{\mathrm{r}}_{kj}$ for the total processing time of small jobs assigned to type k, we get $\sum_{j \in \mathcal{J}_k} p^{\mathrm{p}}_{kj} \leq AS_k(t) + m_k \cdot \delta'$. Since we have $AS_k(t) \leq R_k(t)$ by assumption, we finally get $\sum_{j \in \mathcal{J}_k} p^{\mathrm{p}}_{kj} \leq R_k(t) + m_k \cdot \delta'$: the small jobs in $\mathcal{J}_k$ with their processing times p^{p}_{kj} only slightly exceed the remaining capacity of the machines of type k.

Assume for the sake of contradiction that there is one type k for which all small jobs in $\mathcal{J}_k$ cannot be greedily scheduled. Thus, every machine $i \in \mathcal{M}_k$ has a processing time larger than $1 + 2\delta'$, and there are still small jobs left. Let $\delta_i > 0$ be the processing time by which all jobs assigned to machine i exceed $1 + 2\delta'$. We have

$$\begin{aligned} \sum_{j \in \mathcal{J}_k} p^{\mathrm{p}}_{kj} &\geq \sum_{i \in \mathcal{M}_k} (1 + 2\delta' + \delta_i - b_i(\sigma)) \\ &> \sum_{i \in \mathcal{M}_k} (1 + 2\delta' - b_i) \\ &= \sum_{i \in \mathcal{M}_k} (1 + \delta' - b_i) + m_k \cdot \delta' \\ &= r_k(\sigma) + m_k \cdot \delta' = R_k(t) + m_k \cdot \delta' \\ &\geq \sum_{j \in \mathcal{M}_k} p^{\mathrm{p}}_{kj} \ . \end{aligned}$$

This is a contradiction. All small jobs can therefore be greedily scheduled. As we have $p^{\mathrm{p}}_{kj} < \delta'$ for small jobs on type k, the bound $\delta_i \leq \delta'$ holds: at most one small job is assigned to a machine i such that the processing time exceeds $1 + 2\delta'$. Hence, we have for the makespan $\max_k \max_{i \in \mathcal{M}_k} 1 + 2\delta' + \delta_i \leq 1 + 3\delta'$.

Note that the proof by contradiction above is in fact independent of whether some small jobs are still left or all small jobs have been assigned, but all machines $i \in \mathcal{M}_k$ have a processing time that exceeds $1 + 2\delta'$. It is shown that there must be at least one machine with a total processing time of at most $1 + 2\delta'$ because we get the contradiction otherwise.

Finally, the schedule σ for I^{p} is also one for I^{scale} with a makespan of at most $1 + 3\delta'$. In fact, small processing times are identical in I^{p} and I^{scale}, and if we undo the rounding

of the large processing times, the makespan does not increase because the large jobs were rounded up in I^{p}. After undoing the scaling of I^{scale}, the schedule σ is a solution to I of value at most $(1+3\delta')T$. □

The assignment of the large jobs is illustrated in Figure 3 and of the small jobs in Figure 4. We now state the overall running time for the functions `Backtracking` and `CreateSchedule`.

Lemma 26. *One call of* `Backtracking(`t'`)` *(Algorithm 7) needs time in* $O(K + n' \cdot m)$. *Therefore, the total running time for all calls of* `CreateSchedule` *(Algorithm 6), i.e. of the for-loop in* `Oracle` *(Algorithm 3), is in* $O(\kappa \cdot \frac{K}{(\delta')^2} + n' \cdot m) = m^{O(K/(\delta')^2)} \cdot (\frac{\log m}{\delta'})^{O(K^2)}$.

Proof. Let us first consider `Backtracking`. Creating a list consisting of pointers to the t'_j can be done in $O(n')$: we just follow the pointers given by the backtracking information $\mathrm{Backtrack}(t'_j)$. The creation of the σ_k and $\mathcal{J}_k$ is in $O(K)$, and setting the values $\mathrm{extime}(i)$ to 0 in $O(m)$. This is followed by the n' iterations of the for-loop. In one iteration, the case where j is small can be done in $O(1)$. The case where j is large needs $O(m)$: we have to check all $i \in \mathcal{M}_k$ and their corresponding value $\mathrm{extime}(i)$. In total, we need time $O(K + n' \cdot m)$ for one call of `Backtracking`.

Let us now consider `CreateSchedule`. One call of `CreateSchedule(`t'`)` first determines all R_k. Note that $R_k = \sum_\gamma (AB_k(t'))_\gamma \cdot \gamma \cdot (\delta')^2$ so that determining all R_k needs $O(K \cdot \frac{1}{(\delta')^2})$ including the checks of the if-condition (see the bound on γ_1 in Lemma 15). Should one of the if-conditions fail, the call of `CreateSchedule(`t'`)` is terminated such that the oracle checks the next $t'' \in TS_{n'}$. In the worst case, `CreateSchedule` is therefore called κ times without constructing a schedule, which needs in total $O(\kappa \cdot \frac{K}{(\delta')^2})$.

The schedule σ is constructed if we have $AS_k(t') \leq R_k(t')$ for all machine types k. `CreateSchedule` is not called again afterwards because the oracle terminates if σ has been found. The construction of σ first calls `Backtracking(`t'`)`, which needs $O(K + n' \cdot m)$, and then greedily assigns the small jobs to the machines. Let n_k be the number of large jobs assigned to $\mathcal{M}_k$. The greedy procedure then needs $O(n_k + m_k + |\mathcal{J}_k|)$ for one k, i.e. $O(n' + m)$ for all K machine types. In total, we have a time complexity first for all checks of the if-condition and then for the construction of σ in

$$
\begin{aligned}
&O\left(\kappa \cdot \frac{K}{(\delta')^2} + (K + n' \cdot m) + (n' + m)\right) = O\left(\kappa \cdot \frac{K}{(\delta')^2} + n' \cdot m\right) \\
&= \frac{K}{(\delta')^2} \cdot m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)} + m \cdot \left(O\left(\frac{m^2}{\delta'}\right) + \left(\frac{\log m}{\delta'}\right)^{O(K)}\right) \\
&= m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)},
\end{aligned}
$$

where we have used Assumption 2 and Lemma 20. □

Theorem 27. *Let I be an instance with $1 \leq \mathrm{OPT}(I) \leq m$. The binary search (Algorithm 2) with the* `Oracle` *function of Algorithm 3 is a PTAS: it finds for given $\delta > 0$ and $\delta' := \frac{\delta}{5}$ a solution of value at most $(1+\delta)\mathrm{OPT}(I)$. The running time is in*

$$
m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)} = m^{O(K/\delta^2)} \cdot \left(\frac{\log m}{\delta}\right)^{O(K^2)}.
$$

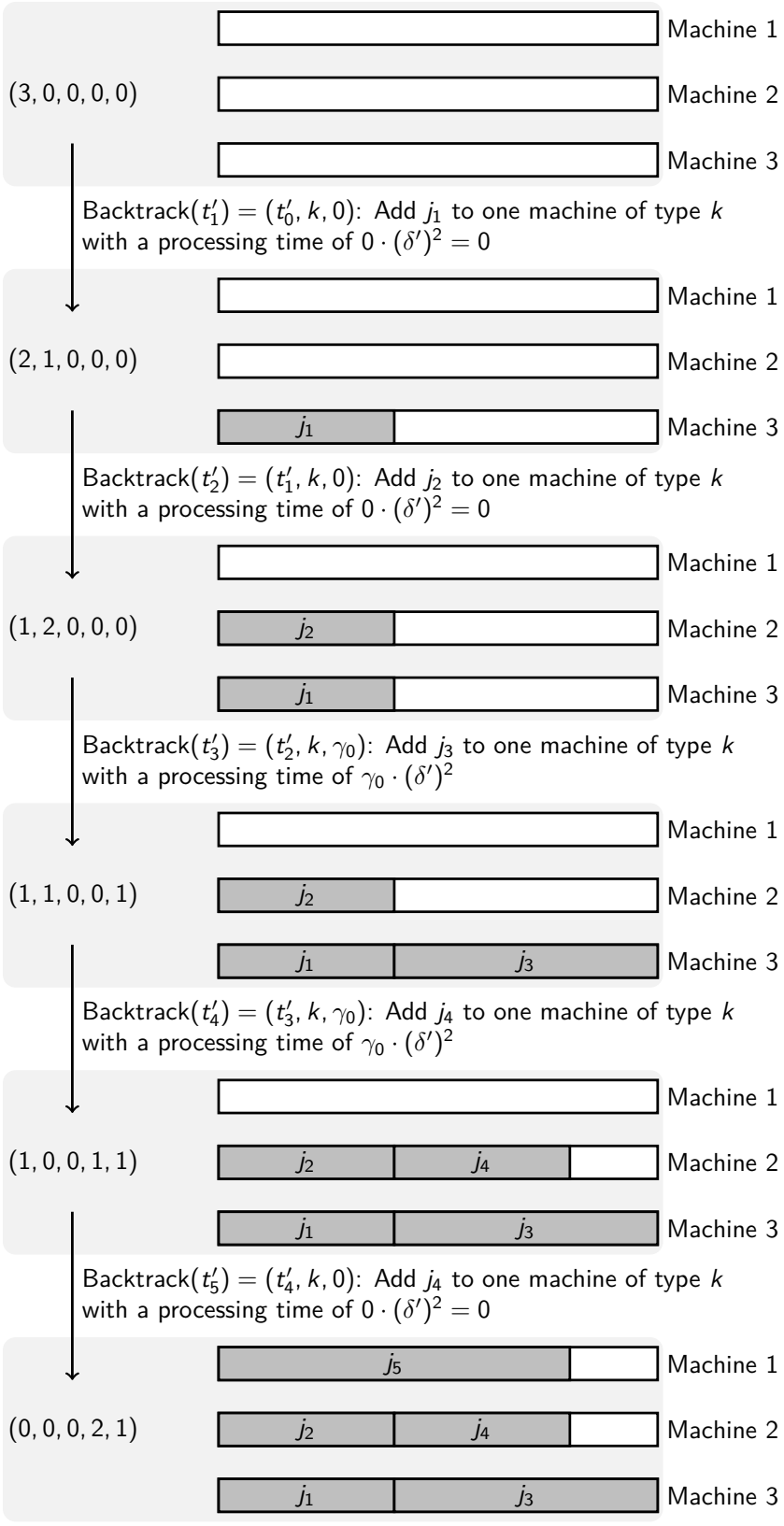
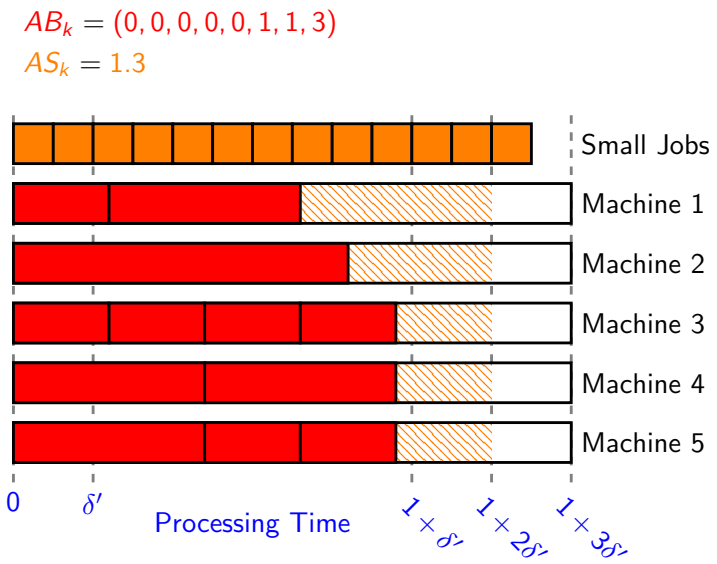
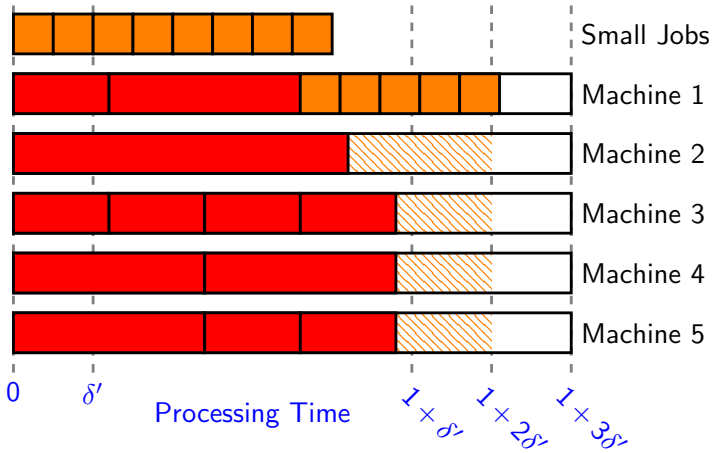


Figure 3: Assignment of the large jobs on machine type k by using the backtracking information

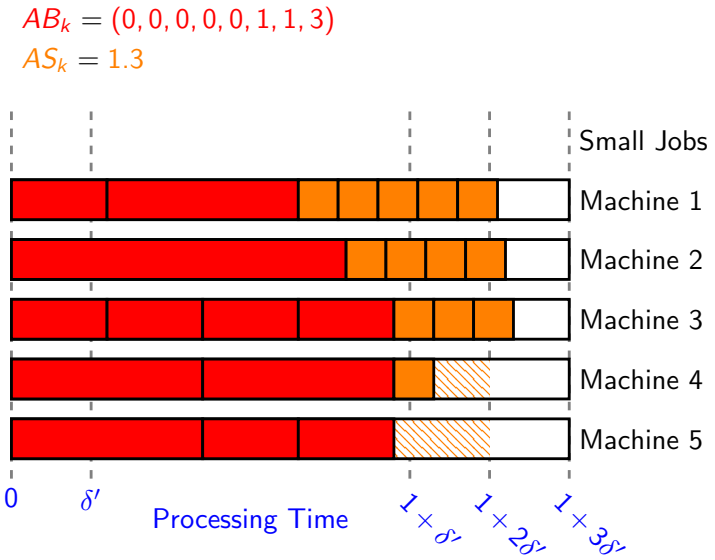


(a) `Backtracking` has assigned the large jobs to the machines. The small jobs still have to be greedily scheduled.



(b) The small jobs are assigned to the first machine until $1 + 2\delta'$ is exceeded. Note that the processing time of the machine is still smaller than $1 + 3\delta'$.

Figure 4: The scheduling of the small jobs after the assignment of the large jobs.



(c) As shown in the proof of Lemma 25, all small jobs can be greedily assigned without exceeding the makespan of $1 + 3\delta'$.

Figure 4: (continued) The scheduling of the small jobs after the assignment of the large jobs.

Proof. Lemma 11 states that the binary search returns a solution of value at most $(1 + (C + 1)\delta' + C(\delta')^2)\mathrm{OPT}(I)$ if `Oracle` has the stated properties. These will now be shown.

Suppose first that `Oracle` (Algorithm 3) returns a solution σ for T and input I. First, the instance is scaled by `Oracle` and becomes I^{scale}, which is rounded to get I^{r}. The solution σ is then determined for I^{p}. It is clear that a schedule is only returned if there is a profile $t \in TS_{n'}$ for which $AS_k(t) \leq R_k(t)$ holds. Hence, the returned schedule σ has a makespan of at most $1 + 3\delta'$ as seen in Lemma 25. It is easy to see that it is also a schedule for I with a makespan of at most $(1 + 3\delta')T$.

It remains to prove that there is not a solution to I of value (at most) T if `Oracle`(I, T) returns $\perp$. We show this by demonstrating that the oracle will always return a solution if there is a solution of value (at most) T:

- The instance I satisfies $\mathrm{OPT}(I) \leq T$, which implies that
- the instance I^{scale} has an optimum $\mathrm{OPT}(I^{\mathrm{scale}}) \leq 1$, which again implies that
- the instance I^{r} satisfies $\mathrm{OPT}(I^{\mathrm{r}}) \leq 1 + \delta'$ (see Lemma 14). Let σ be an optimal schedule for I^{r}. Then,
- `DynProg`, `CreateSchedule` and `Backtracking` will find a schedule: Lemma 19 states that there is a profile t_σ for which we have $ab_k(\sigma, \gamma) = (AB_k(t_\sigma))_\gamma$ and $as_k(\sigma) = AS_k(t_\sigma)$ for all k and γ. As in the proof of Lemma 25, the identity $ab_k(\sigma, \gamma) =$

$(AB_k(t_\sigma))_\gamma$ implies $r_k(\sigma) = R_k(t_\sigma)$. Since the small jobs assigned by σ to the machine type k have to fit into the remaining machine capacity $r_k(\sigma)$, we have $AS_k(t_\sigma) = as_k(\sigma) \leq r_k(\sigma) = R_k(t_\sigma)$. By Lemma 25, the function `CreateSchedule`(t_σ) will therefore construct a schedule σ' for I^{p} with a makespan of at most $1 + 3\delta'$, which is also a solution to I with a makespan of at most $(1 + 3\delta')T$.

Hence, the oracle has the desired properties. We now want that the bound $(1 + (C + 1)\delta' + C(\delta')^2)\mathrm{OPT}(I) \leq (1 + \delta)\mathrm{OPT}(I)$ is satisfied. Note that we have $C = 3$. Set $\delta' := \frac{\delta}{5}$ such that $(C + 1)\delta' + C(\delta')^2 = 4\delta' + 3(\delta')^2 \leq \delta$ holds because $\delta \leq \frac{1}{3}$.

Let us bound the overall running time of the binary search, i.e. Algorithm 2. The binary search needs $O(\log(\frac{m}{\delta'})) = O(\log(\frac{m}{\delta}))$ iterations as seen in Lemma 11. In each iteration, `Oracle` (Algorithm 3) is called. When called, `Oracle` needs $O(K \cdot n')$ to obtain first I^{scale} and then I^{r} as stated in Lemmas 12 and 14. Lemma 21 bounds the running time for the dynamic program by $m^{O(K/(\delta')^2)}(\frac{\log m}{\delta'})^{O(K^2)}$. The for-loop of `Oracle` also needs time in $m^{O(K/(\delta')^2)} \cdot (\frac{\log m}{\delta'})^{O(K^2)}$ (see Lemma 26). Overall, we have a time complexity in

$$
\begin{aligned}
&O\left(\log\left(\frac{m}{\delta'}\right)\right) \cdot \left(O(K \cdot n') + m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)}\right) \\
&= O\left(\log\left(\frac{m}{\delta'}\right)\right) \cdot m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)} \\
&= m^{O(K/(\delta')^2)} \cdot \left(\frac{\log m}{\delta'}\right)^{O(K^2)} = m^{O(K/\delta^2)} \cdot \left(\frac{\log m}{\delta}\right)^{O(K^2)} .
\end{aligned}
$$

We have used Assumption 2 to see that $O(K \cdot n') = O(K \cdot [\frac{m^2}{\delta'} + (\frac{\log m}{\delta'})^{O(K)}]) \leq m^{O(K/(\delta')^2)} \cdot (\frac{\log m}{\delta'})^{O(K^2)}$. □

4 The General PTAS

The PTAS is presented in Algorithm 8.

Input: Instance I, $\varepsilon > 0$
1 Set $\varepsilon' := \frac{\varepsilon}{5}$ and $\delta := \varepsilon'$;
2 Construct from I the instance I^{merge} as shown in Section 2;
3 Call the binary search for I^{merge} (Algorithm 2) with $\delta' = \frac{\delta}{5} = \frac{\varepsilon}{25}$;
4 Undo the combination of the items in I^{merge} and then the rounding of I^{round} to get a schedule σ for I;
5 **return** σ;

Algorithm 8: An overview of the PTAS

Lemma 28. *For the values of $\varepsilon' = \frac{\varepsilon}{5}, \delta = \varepsilon'$ and $\delta' = \frac{\varepsilon}{25}$, the algorithm returns a solution of value $A_\varepsilon(I) \leq (1 + \varepsilon)\,\mathrm{OPT}(I)$.*

Proof. As seen in Theorem 27, the binary search returns a solution to I^{merge} whose value is bounded by $(1 + \delta)\,\mathrm{OPT}(I^{\mathrm{merge}})$. For convenience, we denote the makespan of its solution

by $A_m(I^{\text{merge}})$. By undoing the combination of the items, we get a solution to I^{round} of the same makespan. Thus,

- transforming I^{round} into I^{merge},
- applying the binary search and then
- undoing the combination of items

is an algorithm A_r for I^{round} with

$$A_r\left(I^{\text{round}}\right) = A_m(I^{\text{merge}}) \leq (1+\delta)\text{OPT}(I^{\text{merge}}) = (1+\varepsilon')\text{OPT}(I^{\text{merge}}) \ .$$

Since $\text{OPT}(I^{\text{round}}) \leq \text{OPT}(I^{\text{merge}}) \leq \text{OPT}(I^{\text{round}}) + \varepsilon'$ (see Theorem 9), we have an algorithm with

$$\begin{aligned} A_r\left(I^{\text{round}}\right) &\leq (1+\varepsilon')\,\text{OPT}(I^{\text{merge}}) \leq (1+\varepsilon')\left(\text{OPT}(I^{\text{round}}) + \varepsilon'\right) \\ &= (1+\varepsilon')\,\text{OPT}(I^{\text{round}}) + \varepsilon'(1+\varepsilon') \ . \end{aligned}$$

By Lemma 4, this implies an algorithm for I with

$$A_\varepsilon(I) \leq \left((1+\varepsilon')^3 + \varepsilon' \cdot (1+\varepsilon')^2\right)\text{OPT}(I) \leq (1+\varepsilon)\,\text{OPT}(I) \ .$$

This upper bound holds because

$$\begin{aligned} (1+\varepsilon')^3 + \varepsilon' \cdot (1+\varepsilon')^2 &= \left(1+\frac{\varepsilon}{5}\right)^3 + \frac{\varepsilon}{5} \cdot \left(1+\frac{\varepsilon}{5}\right)^2 \\ &= \left(1+3\frac{\varepsilon}{5}+3\frac{\varepsilon^2}{25}+\frac{\varepsilon^3}{125}\right) + \frac{\varepsilon}{5} + 2\frac{\varepsilon^2}{25} + \frac{\varepsilon^3}{125} \\ &\overset{\varepsilon\leq 1/2}{\leq} 1 + \frac{3}{5}\varepsilon + \frac{3}{50}\varepsilon + \frac{\varepsilon}{4\cdot 125} + \frac{\varepsilon}{5} + \frac{\varepsilon}{25} + \frac{\varepsilon}{4\cdot 125} \\ &\leq 1+\varepsilon \ . \end{aligned}$$

(Main parts of the proof are taken from [15].) □

Lemma 29. *The PTAS has a running time in*

$$O(K\cdot n) + m^{O(K/\varepsilon^2)} \cdot \left(\frac{\log m}{\varepsilon}\right)^{O(K^2)} \ .$$

Proof. The time to construct I^{merge} is in $O(n\cdot K)$ as stated in Theorem 10. The running time for the binary search of $m^{O(K/(\delta')^2)\cdot(\frac{\log m}{\delta'})^{O(K^2)}} = m^{O(K/\varepsilon^2)\cdot(\frac{\log m}{\varepsilon})^{O(K^2)}}$ is shown in Theorem 27. When I^{merge} is constructed from I^{round}, it is saved for every item $j \in I^{\text{merge}}$ which items in I^{round} have been combined into j. (This is the set list(j) in Algorithm 1.) Reversing the combination of the items is therefore in $O(n)$. Undoing the rounding can finally be done in $O(K\cdot n)$. □

These two lemmas show Theorem 1.

5 Concluding Remarks

We have described a PTAS for Scheduling on Unrelated Machines of Few Different Types that is single exponential in $\frac{1}{\varepsilon}$. A natural question is of course the existence of an efficient PTAS (a PTAS with a running time of the form $f(\frac{1}{\varepsilon}) \cdot |I|^{O(1)}$, i.e. where the degree of the polynomial is independent of $\frac{1}{\varepsilon}$). Another interesting task is the generalization of this algorithm to jobs with Δ dimensions because Bonifaci and Wiese [4] considered this more general case. However, we run into trouble when it comes to the preprocessing of jobs in Section 2. For one dimension $\Delta = 1$, jobs can be partitioned into fast and slow ones on every machine type. The partition is then explicitly used to set $p_{kj}^{\text{round}} = \infty$ for jobs that are slow on a machine type k. This allowed us to bound the number of profiles in Lemma 5 and therefore the number of jobs n' in I^{merge}. The bound on n' helped us to limit the overall running time of the dynamic program. In $\Delta \geq 2$ dimensions, a partition into fast and slow jobs on a machine type may not be possible in a way similar to ours because a job may be fast in one, but slow in another dimension.

Optimizations for the running time of our algorithm are probably possible. For instance, the jobs large on a machine type are rounded linearly and then the rounded processing times are used to derive the profiles. An adapted rounding may lead to a smaller number of profiles. Geometric rounding may be an option, but then the processing times are not necessarily a discrete multiple of a value like $(\delta')^2$. One possibility may therefore be not to round the processing times. Instead, the interval $[0, 1 + \delta']$ is partitioned geometrically. For each interval of the partition, the number of machines is saved whose total processing time lies in this interval. This may lead to an improved running time.

Finally, an open question is whether the FPTAS for a constant number of machines m presented by Jansen and Mastrolilli [15] can also be adapted to $(\mathrm{P}m_1, \ldots, \mathrm{P}m_K) \,|\, | \, C_{\max}$ and yield an FPTAS for a constant number of machine types K.

References

[1] D. Arad, Y. Mordechai, and H. Shachnai. *Tighter Bounds for Makespan Minimization on Unrelated Machines*. 2014. arXiv: 1405.2530.

[2] A. Bhaskara, R. Krishnaswamy, K. Talwar, and U. Wieder. "Minimum Makespan Scheduling with Low Rank Processing Times". In: *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*. SIAM, 2013, pp. 937–947.

[3] R. Bleuse, S. Kedad-Sidhoum, F. Monna, G. Mounié, and D. Trystram. "Scheduling independent tasks on multi-cores with GPU accelerators". In: *Concurrency and Computation: Practice and Experience* 27.6 (2015), pp. 1625–1638. DOI: 10.1002/cpe.3359.

[4] V. Bonifaci and A. Wiese. *Scheduling Unrelated Machines of Few Different Types*. 2012. arXiv: 1205.0974.

[5] D. Chakrabarty, S. Khanna, and S. Li. "On $(1, \varepsilon)$-Restricted Assignment Makespan Minimization". In: *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*. SIAM, 2015, pp. 1087–1101.

[6] L. Chen, D. Ye, and G. Zhang. “An improved lower bound for rank four scheduling”. In: *Operations Research Letters* 42.5 (2014), pp. 348–350.

[7] A. V. Fishkin, K. Jansen, and M. Mastrolilli. “Grouping Techniques for Scheduling Problems: Simpler and Faster”. In: *Algorithmica* 51.2 (2008), pp. 183–199.

[8] M. Gairing, B. Monien, and A. Woclaw. “A faster combinatorial approximation algorithm for scheduling unrelated parallel machines”. In: *Theoretical Computer Science* 380.1–2 (2007), pp. 87–99.

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

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. “Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey”. In: *Discrete Optimization II. Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium*. Vol. 5. Annals of Discrete Mathematics. Elsevier, 1979, pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X.

[11] D. S. Hochbaum and D. B. Shmoys. “Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results”. In: *Journal of the ACM* 34.1 (1987), pp. 144–162.

[12] D. S. Hochbaum and D. B. Shmoys. “A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach”. In: *SIAM Journal on Computing* 17.3 (June 1988), pp. 539–551.

[13] E. Horowitz and S. Sahni. “Exact and Approximate Algorithms for Scheduling Nonidentical Processors”. In: *Journal of the ACM* 23.2 (1976), pp. 317–327. DOI: 10.1145/321941.321951.

[14] C. Imreh. “Scheduling Problems on Two Sets of Identical Machines”. In: *Computing* 70.4 (2003), pp. 277–294.

[15] K. Jansen and M. Mastrolilli. *Scheduling unrelated parallel machines: linear programming strikes back*. Tech. rep. 1004. Christian-Albrechts-Universität zu Kiel, Mar. 2010. ISSN: 2192-6247.

[16] K. Jansen and L. Porkolab. “Improved Approximation Schemes for Scheduling Unrelated Parallel Machines”. In: *Mathematics of Operations Research* 26.2 (2001), pp. 324–338.

[17] K. Jansen and C. Robenek. “Scheduling Jobs on Identical and Uniform Processors Revisited”. In: *Workshop on Approximation and Online Algorithms, WAOA 2011*. LNCS 7164. 2012, pp. 109–122.

[18] J. K. Lenstra, D. B. Shmoys, and É. Tardos. “Approximation Algorithms for Scheduling Unrelated Parallel Machines”. In: *Mathematical Programming* 46 (1990), pp. 259–271.

[19] E. V. Shchepin and N. Vakhania. “An optimal rounding gives a better approximation for scheduling unrelated machines”. In: *Operations Research Letters* 33.2 (2005), pp. 127–133.

[20] D. B. Shmoys and É. Tardos. “An approximation algorithm for the generalized assignment problem”. In: *Mathematical Programming* 62 (1993), pp. 461–474. DOI: 10.1007/BF01585178.

[21] O. Svensson. “Santa Claus Schedules Jobs on Unrelated Machines”. In: *SIAM Journal on Computing* 41.5 (2012), pp. 1318–1341.