

A deterministic truthful PTAS for scheduling related machines.

George Christodoulou*

Annamária Kovács†

Abstract

Scheduling on related machines ($Q||C_{\max}$) is one of the most important problems in the field of Algorithmic Mechanism Design. Each machine is controlled by a selfish agent and her valuation can be expressed via a single parameter, her *speed*. Archer and Tardos [4] showed that, in contrast to other similar problems, a (non-polynomial) allocation that minimizes the makespan can be truthfully implemented. On the other hand, if we leave out the game-theoretic issues, the complexity of the problem has been completely settled — the problem is strongly NP-hard, while there exists a PTAS [9, 8].

This problem is the most well-studied in single-parameter Algorithmic Mechanism Design. It gives an excellent ground to explore the boundary between truthfulness and efficient computation. Since the work of Archer and Tardos, quite a lot of deterministic and randomized mechanisms have been suggested. Recently, a breakthrough result [7] showed that a randomized, truthful-in-expectation PTAS exists. On the other hand, for the deterministic case, the best known approximation factor is 2.8 [10, 11].

It has been a major open question whether there exists a deterministic truthful PTAS, or whether truthfulness has an essential, negative impact on the computational complexity of the problem. In this paper we give a definitive answer to this important question by providing a truthful *deterministic* PTAS.

1 Introduction

Algorithmic Mechanism Design (AMD) is an area originated in the seminal paper by Nisan and Ronen [14, 15] and it has flourished during the last decade. It studies combinatorial optimization problems, where part of the input is controlled by selfish agents that are either unmotivated to report them correctly, or strongly motivated to report them erroneously, if a false report is profitable. In classical mechanism design more emphasis has been put on incentives issues, and less to computational aspects of the optimization problem at hand. On the other hand, traditional algorithm design disregards the fact that in some settings the agents might have incentive to lie. Therefore, we end up with algorithms that are fragile against selfish behavior. AMD carries challenges from both disciplines, aiming at the design of qualitative algorithms that, at the same time, make

selfish users interested in reporting truthfully, and so are also immune to strategic behavior.

A fundamental optimization problem that has been suggested in [15] as a ground to explore the design of truthful mechanisms, is the *scheduling problem*, where a set of n tasks need to be processed by a set of m machines. There are two important variants with respect to the processing capabilities of the machines, that have been studied within the AMD framework. The machines can be *unrelated*, i.e., each machine i needs t_{ij} units of time to process task j ; or *related*, where machine i comes with a speed s_i , while task j has processing requirement p_j , that is, $t_{ij} = p_j/s_i$ (we will use the settled notation $Q||C_{\max}$ to refer to the latter problem). The objective is to allocate the jobs to the machines so that the maximum finish time of the machines, i.e. the *makespan* is minimized.

In the game-theoretic setting, it is assumed that each machine i is a rational agent who controls the *private* values of row t_i . It is further assumed that each machine wants to minimize its completion time, and without any incentive it will lie, if this can trick the algorithm to assign less work to him. In order to motivate the machines to cooperate, we pay them to execute the tasks. A *mechanism* consists of two parts: an *allocation algorithm* that assigns the tasks to the machines, and a *payment scheme* that compensates the machines in monetary terms. We are interested in devising *truthful* mechanisms in *dominant* strategies, where each player maximizes his utility by telling the truth, regardless of the reports of the other players.

The scheduling problem provides an excellent framework to study the computational aspects of truthfulness. It is a well-studied problem from the algorithmic perspective with a lot of algorithmic techniques that have been developed. Moreover, it is conceptually close to combinatorial auctions, so that solutions and insights can be transferred from the one problem to the other. Indeed, the scheduling problem comes with a variety of objectives to be optimized, that are different than the objectives used in classical mechanism design.

The mechanism design version of scheduling on related machines was first studied by Archer and Tardos [4]. It is the most central and well-studied among single-parameter problems, where each player controls a

*Max-Planck-Institut für Informatik, Saarbrücken, Germany. This author has been partially supported by DFG grant Kr 2332/1-3 within Emmy Noether Program and by EPSRC grant EP/F069502/1. Email: gchristo@mpi-inf.mpg.de.

†Department of Informatics, Goethe University, Frankfurt M., Germany. Email: panni@cs.uni-frankfurt.de.

single real value and his objective is proportional to this value (see also Chapters 9 and 12 of [13]). In particular, in the scheduling setting the cost of player i is $t_i \cdot w_i$, where $t_i = 1/s_i$ is his private value, and w_i is the total processing requirement (sum of the p_j) allocated to machine i . The *profit* that the agent tries to maximize is the payment he receives minus his cost. Myerson [12] gave a characterization of truthful algorithms for one-parameter problems, in terms of a monotonicity condition. Archer and Tardos [4] found a similar monotonicity characterization, and using it they showed that a certain type of optimal allocation is monotone and consequently truthful (albeit exponential-time). A monotone allocation rule for related scheduling is defined as follows:

DEFINITION 1.1. *An allocation rule for $Q||C_{\max}$ is monotone, if increasing the input speed s_i of any single machine i , while leaving the other speeds unchanged, monotonically increases the workload w_i allocated to this machine, i.e., $s_i < s'_i \Rightarrow w_i \leq w'_i$.*

The fact that truthfulness does not exclude optimality, in contrast to the multi-parameter variant of scheduling (the unrelated case)¹, makes the problem an appropriate example to explore the interplay between truthfulness and computational complexity. It has been a major open problem whether or not a *deterministic* monotone PTAS exists for $Q||C_{\max}$ ². In this work, we give a definitive positive answer to that central question and conclude the problem.

1.1 Related Work Auletta et al. [5] gave the first deterministic polynomial-time monotone algorithm for any fixed number of machines, with approximation ratio 4. This result was improved to an FPTAS by Andelman et al. [2]. For an arbitrary number of machines, Andelman, Azar, and Sorani [1] gave a 5-approximation deterministic truthful mechanism, and Kovács improved the approximation ratio to 3 [10] and to 2.8 [11], which was the previous record for the problem.

Randomization has been successfully applied. There are two major concepts of randomization

¹With the scheduling on unrelated machines, we are more in the dark (see [6] for a recent overview of results). There are impossibility results that show that there does not exist any truthful mechanism with approximation ratio better than a constant *even in exponential time*. Therefore, more primitive questions need to be answered before we settle the complexity of the problem. The only known algorithm for the problem is the VCG that has approximation ratio equal to the number of machines.

²We say that a mechanism runs in polynomial time when both the allocation algorithm and the payment algorithm run in polynomial time.

for truthful mechanisms, *universal truthfulness*, and *truthfulness-in-expectation*. The first notion is strongest, and consists of randomized mechanisms that are probability distributions over deterministic truthful mechanisms. In the latter notion, by telling the truth a player maximizes his expected utility. Only the second notion of randomized truthfulness has been applied to the problem. Archer and Tardos [4] gave a truthful-in-expectation mechanism with approximation ratio 3, that was later improved to 2 [3]. Recently, Dhangwatnotai et al. [7], settled the status for the randomized version of the problem by giving a randomized PTAS that is truthful-in-expectation. Both mechanisms apply (among other methods) a randomized rounding procedure. Interestingly, randomization is useful only to guarantee truthfulness and has no implication on the approximation ratio. Indeed, both algorithms can be easily derandomized to provide deterministic mechanisms that preserve the approximation ratio, but violate the monotonicity condition.

1.2 Our results and techniques We provide a deterministic monotone PTAS for $Q||C_{\max}$. The corresponding payment scheme [4] is polynomially computable³, and with these payments our algorithm induces a $(1 + 3\epsilon)$ -approximate deterministic truthful mechanism, settling the status of the problem. As opposed to [10], where a variant of the LPT heuristic was shown to be monotone, our algorithm is the first deterministic monotone adaptation of the known PTAS's [9, 8]. Next we describe the main ideas of the paper.

We start by fixing a common basis for our subsequent considerations. We always assume that input speeds are indexed so that $s_1 \leq s_2 \leq \dots \leq s_m$ holds. For any set of jobs $P = \{p_1, p_2, \dots, p_j\}$, the *weight* or *workload* of the set is $|P| = \sum_{r=1}^j p_r$. We will view an allocation of the jobs to the machines as an (*ordered*) *partition* (P_1, P_2, \dots, P_m) of the jobs into m sets. We search for an output where the workloads $|P_i|$ are in non-decreasing order.

The PTAS in [8] – which is a simplified and polished version of the very first PTAS [9] – defines a directed network on $m + 1$ layers depending on the input job set, where each arc leading between the layers $i - 1$ and i represents a possible realization of the set P_i , and directed paths leading over the $m + 1$ layers correspond to the possible job partitions. An optimal solution is then found using a shortest path computation in this network.

The difficulty in applying any known PTAS to con-

³This is intuitively clear, since our *work curve* is a step function with a polynomial number of steps.

struct a deterministic monotone algorithm for $Q||C_{\max}$ is twofold. First, in all of the known PTAS's, sets of input jobs of approximately the same size form groups, s.t. in the optimization process a common (rounded or smoothed) size is assumed for all members of the same group. Second, jobs that are tiny compared to the total workload of a machine do not turn up individually in the calculations, but just as part of an arbitrarily divisible (e.g., in form of small blocks) total volume.

Note that it must be relatively easy to find an allocation procedure that is in a way 'approximately monotone'. However, (exact) monotonicity intuitively requires exact determination and knowledge of the allocated workloads. To justify this, we just point out that in every monotone (in expectation) algorithm for $Q||C_{\max}$ provided so far, the (expected) workloads either occur in increasing order wrt. increasing machine speeds, or constitute a lexicographically minimal optimal solution wrt. a fixed solution set and a fixed machine indexing.

Thus, both of the mentioned simplifications of the input set – which, to some extent, seem necessary to admit polynomial time optimization – appear to be condemned to destroy any attempt to make a deterministic adaptation monotone. (The authors of [7] used randomization at both points to obtain the monotone in expectation PTAS.) Our ideas to eliminate the above two sources of inaccuracy of the output are the following, respectively:

1. As for rounding the job sizes, note that grouping is necessary only to reduce the (exponential) number of different outputs. We can achieve the same goal if for any group of jobs of similar size we fix the order of jobs in which they appear in the allocation (e.g., in increasing order), and calculate with the *exact* job sizes along the optimization process. Notice that not even the fact is obvious that such a solution with near-optimal makespan, *and* increasing workloads exists. Now, if reducing a machine speed increases the makespan of the (previously optimal) solution, that means that this machine became a bottleneck, so a new upper bound on the optimum makespan over the considered set of outputs is induced *exactly* by the (previous) workload of the changed machine (the same argument as used in [4, 2, 7]). With this idea we derandomize the first type of randomization (*job smoothing*) of [7].

2. Concerning tiny jobs, we observe that with these we can fill up some of the fastest machines nearly to the makespan level. On the other hand, it is easy to show [3] that pre-rounding machine speeds to powers of some predefined $(1 + \epsilon)$ does not spoil monotonicity and increases the approximation bound by only a factor of $(1 + \epsilon)$. Assuming now that the coarsity of tiny blocks is

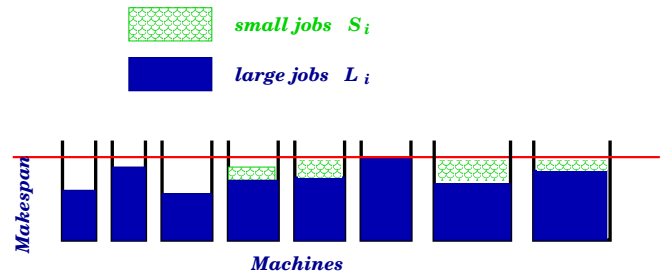


Figure 1: The type of schedule we look for

much finer than the coarsity of machine speeds, we can be sure that (full) machines of higher speed receive more work than slower machines. Moreover, having reduced the speed of such a machine, tiny jobs in its workload 'flow' to other machines to provide a makespan 'much' smaller than implied by the previous workload of this machine.

It is quite a technical challenge to combine these two ideas so smoothly that in the end yields a correct monotonicity proof. We accomplish this task as follows (see Figure 1). We fix (for the proof argument) a set L_i of non-tiny jobs on each machine, so that the L_1, L_2, \dots, L_m have increasing and exactly known weights, and they fulfil the constraints suggested in 1. On top of the sets L_i , each machine has a set S_i of small jobs (due to necessary conditions for rounding the total volume of tiny jobs, some of these are uniform blocks, while some are known exactly). The total set of small jobs is flexible (along the proof), in particular we can always move a small job to a higher index machine, and obtain a valid schedule. Moreover, we set the objectives so that in an optimum solution the small jobs are moved to the higher index machines as much as possible (and so, make them full). We note that the same job size may be large for a slower machine, while it is small on a faster machine.

Our monotonicity proof becomes subtle in case of the first (and so, not necessarily full) machine containing small jobs. It is especially so when this first machine is m , not leaving space for manipulating the small jobs in the output as needed. In order to circumvent this problem we restrict the search to allocations where at least two machines do have some tiny blocks (unless too few tiny jobs exist). Moreover, it seems crucial in our monotonicity argument that every machine has the possibility to get rid of all the tiny blocks (i.e., those inducing uncertain workload) if this is provoked by a reduction of its speed. Combining these two requirements we treat the last *three* machines as a single entity. A carefully optimized assignment of an 'obligatory' set of tiny blocks, and later of the actual tiny jobs to these

machines then implies monotonicity.

$$(D2) \quad \bar{p} > \frac{\delta \cdot |L|}{(1+\delta)^2} \text{ for every } p \in L, \text{ and}$$

$$(D3) \quad \bar{q} \leq \delta |L| \text{ for every } q \in S.$$

1.3 Preliminaries The input is given by a set P_I of n input jobs, and a vector s (or σ) of input speeds $s_1 \leq \dots \leq s_m$. For a job $p \in P_I$ we use p both to denote the individual job, and the *size* of this job in a given formula.

For a desired approximation bound $1+3\epsilon$, we choose a $\delta \ll \epsilon$, that will be the rounding precision of the job sizes. For ease of exposition, we will assume that $(1+\delta)^t = 2$ for some $t \in \mathbb{N}$.⁴ Furthermore, we define ρ as the unique integer power of 2 in $[\delta/6, \delta/3]$. We use the interval notation (e.g., $[1, m]$) for a set of consecutive machine indices.

DEFINITION 1.2. (JOB CLASSES) If p denotes (the size of) a job, then \bar{p} denotes this job rounded up to the nearest integral power of $(1+\delta)$. A job p is in the job class C_l , iff $\bar{p} = (1+\delta)^l$.

Let $C_l = \{p_{l1}, p_{l2}, \dots, p_{ln_i^{\max}}\}$ be the jobs of C_l in some fixed non-decreasing order of size. We use the notation $C_l(a) = \{p_{l1}, \dots, p_{la}\}$ for $0 \leq a \leq n_i^{\max}$, and $C_l(a, b) = C_l(b) \setminus C_l(a)$ for $0 \leq a \leq b \leq n_i^{\max}$.

If $P = \{p_1, p_2, \dots, p_j\}$ is a job set, then $\bar{P} = \{\bar{p}_1, \bar{p}_2, \dots, \bar{p}_j\}$ denotes the corresponding set of rounded jobs. The *weight* or *workload* of P is $|P| = \sum_{r=1}^j p_r$; the *rounded weight* is $|\bar{P}| = \sum_{r=1}^j \bar{p}_r$. Assuming that the jobs are in *non-increasing* order of size, we denote the subset of the r largest jobs by $P^r = \{p_1, p_2, p_3, \dots, p_r\}$.

2 Canonical allocations

This section characterizes the type of allocations – we call them *canonical* – that we will consider. Definitions 2.1 and 2.2 describe the necessary restrictions on the output job partition P_1, \dots, P_m .⁵ Subsequently, our first main result, Theorem 2.1 states that for any input, and any $\delta > 0$, a canonical allocation exists that provides a $1 + \mathcal{O}(\delta)$ approximation to the optimum makespan.

DEFINITION 2.1. (δ -DIVISION) We say that a given set of jobs P is δ -divided into the pair of sets (L, S) (or $P = (L, S)$) if

$$(D1) \quad P = L \cup S \text{ and } L \cap S = \emptyset,$$

⁴This simplifying assumption is unrealistic for computations, but it is not necessary for the result to hold. We could equally well use the rounding function of [8] or [7]. Also, since our result is of purely theoretical interest, we do not try to optimize the ratio δ/ϵ ; it will be clear that, e.g., $30\delta < \epsilon$ suffices in the proofs.

⁵More precisely, the partition Q_1, Q_2, \dots, Q_m obtained in step 4. of PTAS will be canonical (see Figure 2).

The subsets L and S will be the *large* resp. the *small* jobs on a single machine. Note that we cannot set a sharp border between their sizes. For instance, having uniform jobs, $1/\delta$ of them will form L , while the rest (arbitrarily many) belong to S .

DEFINITION 2.2. (CANONICAL ALLOCATION) For a given input, an allocation P_1, P_2, \dots, P_m is called *canonical*, if for every $i \in [1, m]$, the set P_i can be δ -divided into (L_i, S_i) , so that the following properties hold:

$$(A1) \quad \text{if } i < i', \text{ then } |L_i| \leq |L_{i'}|;$$

$$(A2) \quad \text{for jobs } p \text{ and } q \text{ of the same job class } p \leq q \text{ holds if and only if}$$

$$(a) \quad p \in L_i \text{ and } q \in S_{i'} \text{ for some } i, i' \in [1, m], \text{ or}$$

$$(b) \quad p \in L_i \text{ and } q \in L_{i'} \text{ and } i \leq i', \text{ or}$$

$$(c) \quad p \in S_i \text{ and } q \in S_{i'} \text{ and } i \leq i'.$$

THEOREM 2.1. For arbitrary increasing input speeds and input jobs, a canonical allocation inducing a schedule with makespan at most $(1+3\delta)OPT$ exists, where OPT is the optimum makespan of the input.

In the proof of Theorem 2.1, we modify an optimal partition of the rounded input jobs \bar{P}_I to get the canonical allocation. First we take the *core* set of each set in the partition, which yields a ‘preliminary δ -division’ of the sets:

DEFINITION 2.3. (CORE) Given a set P of jobs, we define the *core* $cr(P)$ of P as follows. Consider the jobs $P = \{p_1, p_2, \dots\}$ in a fixed non-increasing order of size. Let j be minimum with the property that $\bar{p}_j \leq \frac{\delta}{1+\delta} |P^{j-1}|$, then $cr(P) \stackrel{\text{def}}{=} P^{j-1} = \{p_1, \dots, p_{j-1}\}$. If no such j exists, then $cr(P) \stackrel{\text{def}}{=} P$.

Then we order the partition sets by increasing order of *core size*, and apply the following result to make the cores (now with original job sizes) fulfil (A2) (b).

PROPOSITION 2.1. Let (Q_1, \dots, Q_m) be a partition of a subset Q of the input jobs such that $|\bar{Q}_1| \leq \dots \leq |\bar{Q}_m|$. There exists a partition (L_1, \dots, L_m) of Q that satisfies

$$1. \quad |L_1| \leq \dots \leq |L_m|;$$

$$2. \quad \text{for any job class } C_l, \text{ if job } p_{lj} \text{ belongs to } L_i \text{ and job } p_{lk} \text{ to } L_{i'} \text{ where } i < i', \text{ then } j < k;$$

3. for all i

$$\frac{1}{1+\delta} |\overline{Q}_i| < |L_i| \leq |\overline{Q}_i|.$$

Finally, it is easy to show that *small* jobs (those outside the cores) can be shifted towards fast machines, in order to 'fit below' the original makespan again. After shifting, the cores and the small jobs induce a δ -division on each machine.

3 Configurations

Like in [8, 7], we introduce so called *configurations* $\alpha(w, \mu, \vec{n}^o, \vec{n}^1)$ in order to represent any possible job set P_i of the partition, up to δ accuracy. We use the configurations to define the vertices of a directed graph \mathcal{H} . A well-defined optimal path in this graph will then specify our output schedule.⁶

The first component of any configuration is a *magnitude* w which is an integer power of 2. As we proceed from slow machines to fast machines in a schedule, the monotonically increasing magnitude keeps track of the largest job size allocated so far, which must be some size in the interval $(w/2, w]$. Thus, the current magnitude also shows, which (larger) job sizes are not yet relevant, and which (tiny) jobs need not be taken into account *individually* anymore in the configuration (note that here we exploit that the $|L_i|$ must be increasing, cf. [8]). This motivates the next definition.

DEFINITION 3.1. (VALID MAGNITUDE) *The value $w = 2^z$ ($z \in \mathbb{Z}$) is a valid magnitude if an input job $p \in P_I$ exists so that $w/2 < p \leq w$. Let w_{\min} and w_{\max} denote the smallest and the largest valid magnitudes, respectively. We call a job tiny for w if it has size at most ρw .*

Recall that ρ is the integer power of 2 between $\delta/6$, and $\delta/3$. Having a magnitude w fixed, let $\lambda = \log_{(1+\delta)} \rho w = t \cdot \log(\rho w)$, and $\Lambda = \log_{(1+\delta)} w = t \cdot \log w$, where $(1+\delta)^t = 2$. Notice that both λ and Λ are integers, and by Definition 1.2, the jobs of size in $(\rho w, w]$ belong to the classes $C_{\lambda+1}, \dots, C_{\Lambda}$. These will constitute the relevant job classes, if the largest jobs size on the current or slower machines is between $w/2$ and w .

If the configuration α represents the set P_i in a job partition, then the so-called *size vector* $\vec{n}^o = (n_{\lambda}^o, n_{\lambda+1}^o, \dots, n_{\Lambda}^o)$ describes the jobs in the cumulative job set $A_{i-1} \stackrel{\text{def}}{=} \bigcup_{h=1}^{i-1} P_h$ as follows. For $\lambda < l \leq \Lambda$,

⁶Roughly speaking, our graph can be thought of as the *line graph* of the graph G defined in [8] (with simple modifications). That is, the vertices of \mathcal{H} correspond to edges of G . This is the reason why our configurations include two vectors \vec{n}_1 and \vec{n}_2 instead of only one.

($l \neq \mu, \mu + 1$), exactly the first (smallest) n_l^o jobs of the class C_l are in the set A_{i-1} . The meaning of n_{λ}^o is that in A_{i-1} the total weight of jobs from $\bigcup_{l \leq \lambda} C_l$ is in the interval $((n_{\lambda}^o - 1) \cdot \rho w, (n_{\lambda}^o + 1) \cdot \rho w)$. However, the particular subset of these small jobs inside A_{i-1} , is not determined by α . The vector \vec{n}^1 represents the set $A_i \stackrel{\text{def}}{=} \bigcup_{h=1}^i P_h$, analogously.

A major difference to the configurations of [8], is that our configurations should not only represent a job set P_i , but also its δ -division (L_i, S_i) . In particular, we will distinguish four types of job sizes in a configuration. *Tiny* jobs have size at most ρw , and, as already seen, are represented by the first coordinates n_{λ} of the two size vectors with their total size rounded to an integer multiple of ρw . Correspondingly, we will sometimes talk about *blocks* of size ρw which are simply re-tailored tiny jobs so as to make our procedure efficient.

DEFINITION 3.2. (BLOCK) *Blocks are imaginary tiny jobs, each having size ρw for some valid magnitude w . We use $S(n_{\lambda}, \rho w)$ to denote a set of n_{λ} blocks of size ρw .*

Small jobs are those that (together with the tiny jobs), can only appear in the set S_i of the δ -division, whereas *large* jobs can only be in the set L_i . However, there must exist job classes – we will call them *middle* size jobs –, which might occur in both L_i and S_i , since by Definition 2.1 there is a flexible border between the job sizes in L_i and in S_i . Therefore, exactly two job classes, μ and $\mu + 1$ will be represented by a *triple* of (increasing) non-negative integers instead of scalar values in both of the vectors \vec{n}^o, \vec{n}^1 . For instance, in the case of $\underline{n}_{\mu}^o = (n_{\mu\ell}^o, n_{\mu m}^o, n_{\mu s}^o)$, the meaning of the three numbers will be that in the set A_{i-1} , from the job class C_{μ} exactly the jobs in $C_{\mu}(n_{\mu m}^o, n_{\mu s}^o)$ are allocated as small jobs, that is, to one of the sets S_1, S_2, \dots, S_{i-1} , and exactly the jobs in $C_{\mu}(n_{\mu\ell}^o)$ as large jobs, i.e., in one of L_1, \dots, L_{i-1} , and similarly in case of \vec{n}_{μ}^1 for the set A_i . The meaning of the numbers for $\mu + 1$ is analogous. We summarize the above description in the next, somewhat technical definitions.

DEFINITION 3.3. (SIZE VECTOR) *A size vector $\vec{n} = (n_{\lambda}, \dots, n_{\Lambda})$ with middle size $\mu \in [\lambda + 1, \Lambda]$, is a vector of integers, with the exception of the entries $\underline{n}_{\mu} = (n_{\mu\ell}, n_{\mu m}, n_{\mu s})$ and $\underline{n}_{\mu+1} = (n_{(\mu+1)\ell}, n_{(\mu+1)m}, n_{(\mu+1)s})$ both of which consist of three integers, so that $n_{\mu\ell} \leq n_{\mu m} \leq n_{\mu s}$, and $n_{(\mu+1)\ell} \leq n_{(\mu+1)m} \leq n_{(\mu+1)s}$ holds. All integer entries belong to $[0, n]$.*

DEFINITION 3.4. (CONFIGURATION) *A configuration $\alpha(w, \mu, \vec{n}^o, \vec{n}^1)$ consists of four components: a valid*

magnitude w , and two size vectors $\vec{n}^o = (n_\lambda^o, \dots, n_\Lambda^o)$, and $\vec{n}^1 = (n_\lambda^1, \dots, n_\Lambda^1)$ with middle size μ , such that

- (C1) $n_l^o \leq n_l^1 \leq n_l^{\max}$ for $\lambda < l \leq \Lambda$, $l \notin \{\mu, \mu + 1\}$;
(C2) if $w \neq w_{\min}$ then $n_l^1 > 0$ for at least one $l \in (\Lambda - t, \Lambda]$;
(C3) $n_\lambda^o \leq n_\lambda^1 \leq \left\lceil \frac{\sum_{l \leq \lambda} |C_l|}{\rho w} \right\rceil + 3$;
(C4) $n_{\mu\ell}^o \leq n_{\mu\ell}^1 \leq n_{\mu m}^o = n_{\mu m}^1 \leq n_{\mu s}^o \leq n_{\mu s}^1 \leq n_{\mu}^{\max}$, and analogously for $\mu + 1$;

$$T_\alpha \stackrel{\text{def}}{=} S(n_\lambda^1 - n_\lambda^o, \rho w).$$

$$L_\alpha \stackrel{\text{def}}{=} C_\mu(n_{\mu\ell}^o, n_{\mu\ell}^1) \cup C_{(\mu+1)}(n_{(\mu+1)\ell}^o, n_{(\mu+1)\ell}^1) \cup$$

$$\cup \bigcup_{l=\mu+2}^{\Lambda} C_l(n_l^o, n_l^1), \text{ and}$$

$$S_\alpha \stackrel{\text{def}}{=} C_\mu(n_{\mu s}^o, n_{\mu s}^1) \cup C_{(\mu+1)}(n_{(\mu+1)s}^o, n_{(\mu+1)s}^1) \cup$$

$$\cup \bigcup_{l=\lambda+1}^{\mu-1} C_l(n_l^o, n_l^1) \cup T_\alpha.$$

- (C5) either $\vec{n}^o \neq \vec{n}^1$, and $(1 + \delta)^{(\mu+1)} \leq \delta \cdot |L_\alpha| < (1 + \delta)^{(\mu+2)}$;

or α is the empty configuration $(w_{\min}, \lambda_{\min} + 1, \vec{0}, \vec{0})$ where $\lambda_{\min} = t \cdot \log(\rho w_{\min})$.

NOTATION 1. We refer to the whole represented job set $L_\alpha \cup S_\alpha$ (including virtual blocks) simply by α (abusing notation), and $|\alpha|$ stands for the total work of the set α . We denote the set without tiny blocks by $\tilde{\alpha} = \alpha \setminus T_\alpha$.

Clearly, (C1), (C3), and (C4) reflect how \vec{n}^o and \vec{n}^1 represent the cumulative job-sets; (C2) implies that w is always the smallest possible magnitude for representing these job-sets.

(C5) is different in flavor from the previous properties: it establishes the relation between the weight of L_α and of the job sizes at the boundary of L_α , and S_α , according to Definition 2.1. It is easy to verify that for every set $P_i = A_i \setminus A_{i-1}$ (and corresponding w) in a canonical schedule a unique $\mu > \lambda$ exists that fulfils (C5), and vice versa, that due to (C5) any configuration $\alpha = (L_\alpha, S_\alpha)$ represents a δ -division. Finally, we stress that the sets A_i do *not* possess a δ -division, and a single size vector \vec{n} does not represent an (L, S) division at all.

4 The directed graph \mathcal{H}_I

In this section, for arbitrary input instance I , we define a directed, layered graph \mathcal{H}_I . All vertices of this graph are configurations, selected, numbered, and 'chained' to form the graph in an appropriate way.

First, for an arbitrary configuration α , we define a set $Scale(\alpha)$ of configurations. These are the possible configurations of an end-vertex of any arc with a starting vertex having α as configuration. In particular, if $\alpha = (w, \mu, \vec{n}^o, \vec{n}^1)$, $\beta = (w', \mu', \vec{n}'^o, \vec{n}'^1)$, and $\beta \in Scale(\alpha)$, then \vec{n}'^o must represent the same job set A_i , as \vec{n}^1 , from the point of view of a (possibly) increased magnitude w' and a (possibly) increased middle size μ' . Whenever $w' > w$, this involves collecting the remaining jobs of classes that become tiny, and remaining tiny blocks, and forming the proper number of new bigger tiny blocks out of this job set. Importantly, the size ρw of tiny blocks at least doubles (unless unchanged) as we proceed to faster machines. This fact prevents that rounding errors in the total (tiny) job size cumulate to an unpredictable error (see also [8]). Furthermore, the middle size μ is allowed to increase if all large jobs in the class μ have been allocated, that is, $n_{\mu\ell}^1 = n_{\mu m}^1$. We omit the exact definition of $Scale(\alpha)$.

The vertices of \mathcal{H}_I (i.e., the configurations) are arranged in m layers, and in levels I and II , which are orthogonal to the layers. The configurations on level I must have an empty set of small jobs, i.e., $S_\alpha = \emptyset$, and here the layers $\{m-2, m-1, m\}$ are empty. Level II has $m-1$ 'real' layers, and we add a single dummy vertex v_m adjacent to every vertex on layer $m-1$, that alone forms the last layer m .⁷ In general, the i th layer stands for the i th set P_i . Any directed path of m nodes leads to v_m over the m layers, and from level I (or II) to level II . Such a path we call an m -path. The m -paths represent partitions of the input P_I . For a given m -path, the very first vertex on level II is in some layer $k \leq m-2$; we will call it the *switch vertex*, and k the *switch machine*. (Note that k is thus the first machine possibly receiving small jobs.) We shall denote the vertices on the two levels by V_I , and V_{II} , respectively.

NOTATION 2. For any directed path (v_1, v_2, \dots, v_r) , the corresponding configurations of the nodes will be denoted by $(\alpha_1, \alpha_2, \dots, \alpha_r)$.

In what follows, we discuss about the last three sets P_{m-2}, P_{m-1}, P_m of the partition. First of all, observe that for an m -path the last configuration α_{m-1} alone represents $\bigcup_{h=1}^{m-1} P_h$. Thus, we can use α_{m-1} to

⁷More precisely, we will unite layers $m-2$ and $m-1$, and use *double vertices* in the united layer, but it simplifies the discussion to think of these as pairs of individual vertices.

uniquely determine the 'hidden configuration' α_m (not appearing explicitly in the path). We define α_m to have $w_m = w_{m-1}$, $\mu_m = \mu_{m-1}$, and all jobs not allocated in $\bigcup_{h=1}^{m-1} P_h$. In particular, note that α_m includes *all* jobs of class higher than Λ_{m-1} , and that this job set can be handled as a single huge chunk without violating the running time bounds. For technical reasons we overestimate the amount of tiny blocks on m and set $n_\lambda^1 = \left\lceil \frac{\sum_{l \leq \lambda} |C_{il}|}{\rho w} \right\rceil + 3$. We omit the details.

Furthermore, our monotonicity argument requires that even the last *three* workloads $\alpha_{m-2}, \alpha_{m-1}$, and α_m be dealt with as a single entity. Among other restrictions, we will demand that either all of them have the same magnitude, and therefore use $w'_{m-1} = w_{m-2}$ instead of w_{m-1} , or that w_{m-2} is *much* smaller than w_{m-1} , so that all jobs on $m-2$ (if exist), are tiny for machines $m-1$ and m . Correspondingly, α_{m-2} and α_{m-1} of any path must adhere to either type (A) or (B) as specified next. Observe that in case (A) on the last three machines, resp. in case (B) on the last two machines the size of tiny jobs mentioned is the same (i.e. well-defined).

- (A) $w_{m-2} > \rho^2 \cdot w_{m-1}$; in this case we modify the last magnitude to be $w'_{m-1} := w_{m-2}$, and require $|\tilde{\alpha}_{m-2}| \leq |\tilde{\alpha}_{m-1}| \leq |\tilde{\alpha}_m|$, and $|\alpha_{m-2}| \leq |\alpha_{m-1}| \leq |\alpha_m|$; moreover,
- (i) either *all* tiny jobs (measured by blocks) are on machines $m-1$ and m , or
 - (ii) machines $\{m-2, m-1, m\}$ have at least 18 tiny blocks, and at least two of them have each at least 6 tiny blocks.
- (B) $w_{m-2} \leq \rho^2 \cdot w_{m-1}$; then $|\tilde{\alpha}_{m-1}| \leq |\tilde{\alpha}_m|$, and $|\alpha_{m-1}| \leq |\alpha_m|$, and
- (i) all machines but $\{m-1, m\}$ are empty, or
 - (ii) $m-1$ and m together have at least 6 tiny blocks.

The requirements (A) and (B) can be included in the graph definition, e.g., by using (polynomially many) special *double vertices* v'_{m-2} with *double configurations* $(\alpha_{m-2}, \alpha_{m-1})$ in the layers $(m-2, m-1)$ on level II . Applying $w'_{m-1} := w_{m-2} > \rho^2 \cdot w_{m-1}$ can be done by using size vectors of triple length for the double vertices of type (A). Clearly, all restrictions can be represented by the configurations $(\alpha_{m-2}, \alpha_{m-1})$.

The following definition (sketch) of graph \mathcal{H}_I is independent of the speed vector s , and depends only on the job set P_I . We assume, w.l.o.g. that $m \geq 3$, otherwise we include a machine of speed (close to) 0.

DEFINITION 4.1. (GRAPH \mathcal{H}_I) $\mathcal{H}_I(V, E)$ is a directed graph, where every vertex $v \neq v_m$ is a triple $v = (d, i, \alpha)$, so that $d \in \{I, II\}$, and $i \in [1, m-1]$, and $\alpha = (w, \mu, \vec{n}^o, \vec{n}^1)$ is a configuration.

- (V1) if $i = 1$, then for $l \neq \mu, \mu+1$ $n_l^o = 0$, while for $l \in \{\mu, \mu+1\}$ $n_l^o = 0$ and $n_{l_m}^o = n_{l_s}^o$;
- (V2) if $d = I$, then $i \in [1, m-3]$, and $S_\alpha = \emptyset$;

There is an arc from $v = (d, i, \alpha)$ to $v' = (d', i+1, \beta)$ if an only if

- (E1) $\beta \in \text{Scale}(\alpha)$, and
- (E2) $|L_\alpha| \leq |L_\beta|$, and
- (E3) $d \leq d'$;

Finally, for $d = II$, and combined layers $(m-2, m-1)$, include double vertices v' with double configurations $(\alpha_{m-2}, \alpha_{m-1})$ so that for $(\alpha_{m-2}, \alpha_{m-1}, \alpha_m)$ the requirements (E1) (E2) and either (A) or (B) hold.

Next we assign a weight to each vertex, called *finish time*, and define the *makespan* of a path accordingly. Obviously, these values do depend on the machine speeds s .

DEFINITION 4.2. (FINISH TIME OF A VERTEX) Let $v = (d, i, \alpha)$ be a vertex of \mathcal{H}_I , where $\alpha = (w, \mu, \vec{n}^o, \vec{n}^1)$. The finish time of v is then $f(v) = \frac{|\alpha| + \rho w}{s_i}$ if α does have tiny blocks ($n_\lambda^o < n_\lambda^1$), and $f(v) = \frac{|\alpha|}{s_i}$ otherwise.

DEFINITION 4.3. (MAKESPAN OF A PATH) Let $\mathcal{Q} = (v_1, v_2, \dots, v_r)$ be a directed path in \mathcal{H}_I . If $\mathcal{Q} \subset V_I$, or $\mathcal{Q} \subset V_{II}$, then the makespan of \mathcal{Q} is $M(\mathcal{Q}) = \max_{h=1}^r f(v_h)$. If \mathcal{Q} is an m -path with switch vertex $v_k = (II, k, \alpha_k)$, then $M(\mathcal{Q}) = \max\{\frac{|\alpha_k|}{s_k}, \max_{h \neq k} f(v_h)\}$.

The following theorem, saying that an m -path having makespan close to the optimum makespan of the scheduling problem always exists, is a consequence of Theorem 2.1. The obligatory amount of tiny blocks (or *all* tiny jobs for w_{\max}) are collected at the beginning and later allocated to the 3 fastest machines to fulfil (A) or (B). The rest of the proof is rather straightforward, and requires a technical translation of real schedules to m -paths of \mathcal{H}_I , which involves creating blocks of size ρw_i from the actual tiny jobs.

THEOREM 4.1. For every input $I = (P_I, s)$ of the scheduling problem, the optimal makespan over all m -paths in \mathcal{H}_I is at most $OPT \cdot (1 + \mathcal{O}(\delta))$, where OPT denotes the optimum makespan of the scheduling problem.

5 The deterministic algorithm

This section describes the deterministic monotone algorithm, in form of two procedures and the main algorithm PTAS. We will make use of an arbitrary fixed total order \prec over the set of all configurations α , such that configurations of smaller total workload $|\alpha|$ are smaller according to \prec .

One can easily check that every m -path in the graph represents a *canonical* allocation (extending this concept to tiny blocks). Among the m -paths of \mathcal{H}_I , the algorithm selects an m -path having *minimum* makespan, as the primary objective. Among paths of minimum makespan, we maximize the index of the switch machine k . A further order of preference, is to be of type (A), then (B). This selection of the optimal path is done by Procedure OPTPATH (see Appendix A). On the high level, this procedure is a common dynamic programming algorithm that finds an m -path of minimum makespan in \mathcal{H}_I . However, we do not simply proceed from left to right over the m graph layers, but select an optimal path from the first layer to every node in V_I , and similarly, an optimal path from layer m to each node in V_{II} . Finally, we test each vertex in V_{II} to provide a potential switch vertex (i.e., we find optimal paths leading to the switch vertex from both end-layers). When the makespan of two prefix (or suffix) paths is the same, we break ties according to \prec . We choose a switch vertex v_k providing optimum makespan, and of maximum possible k . The case $k = m - 2$ needs careful optimization. Roughly, we choose *deterministically* by some fixed order of the double configurations $(\alpha_{m-2}, \alpha_{m-1})$, but minimize the makespan on the last three machines by redistributing the tiny blocks. The flexibility provided by three machines with 'many' tiny blocks, facilitates monotone allocation in this degenerate case as well.

Once an optimal m -path is found, we have to allocate the jobs of P_I to the machines. This is obvious for jobs that appear individually in some configuration of the path, but we need an accurate description of how the *tiny* jobs are distributed, given the block representation. Procedure PARTITION is detailed in Appendix A. Importantly, depending on whether the switch machine k is filled high (above $(1 - \epsilon/2)$ times the makespan) or low, it gets filled with tiny jobs below, resp. above $|\alpha_k|$. This, again, will play an important role when showing monotonicity. Distributing the tiny jobs when $k = m - 2$, is a slightly more subtle procedure, operating with the same principle (a *low* machine is filled over $|\alpha_i|$). In general, by having a careful look at PARTITION, one can see that the machines never get filled above the makespan of the input path, that is, $\frac{|Q_i|}{s_i} \leq M(Q)$. This is trivial for machines without tiny jobs, and follows from the definition of finish time with

Algorithm 1 PTAS

Input: machine speeds $\sigma_1 \leq \sigma_2 \leq \dots \leq \sigma_m$, and job set $P_I = \{p_1, p_2, \dots, p_n\}$, desired precision ϵ .

Output: A partition P_1, P_2, \dots, P_m of P_I .

1. for each $i \in [1, m]$, round the speed σ_i up to the nearest power of $(1 + \epsilon)$;
 2. based on the jobs P_I , rounded speeds $s_1 \leq s_2 \leq \dots \leq s_m$, and an appropriate $\delta \ll \epsilon$, construct the graph \mathcal{H}_I ;
 3. run Procedure OPTPATH in order to obtain the optimal m -path $Q = (v_1, \dots, v_m)$;
 4. from Q compute the partition Q_1, Q_2, \dots, Q_m by Procedure PARTITION;
 5. let P_1, P_2, \dots, P_m be the sets of $\{Q_1, Q_2, \dots, Q_m\}$, sorted by increasing order of weight $|Q_i|$; output P_1, P_2, \dots, P_m .
-

Figure 2: The deterministic monotone PTAS.

the extra tiny block, for other machines.

Finally, the monotone PTAS is presented in Figure 2. A substantial property of the output is that on machines $i < k$, the workloads Q_i do not get permuted in step 5. of PTAS (see also Appendixapp:mono). This is due to the fact that the sets L_i of large jobs are increasing by (E2). On the other hand, the machines $i > k$ have finish time close to the makespan $M(Q)$ (resp. finish time of small difference in (A)). As a consequence, we obtain that in step 5. the sets Q_i are permuted only among machines $i \geq k$ of *equal* rounded speed s_i . Therefore, even for the permuted workloads, $|P_i|/s_i \leq M(Q)$ holds for all i . This, in turn, together with Theorem 4.1 implies the approximation bound:

THEOREM 5.1. *For arbitrary input I , and any given $0 < \epsilon \leq 1$, the deterministic algorithm PTAS outputs a $(1 + 3\epsilon)$ -approximate optimal allocation in time $\text{Poly}(n, m)$.*

5.1 Monotonicity A high-level formulation of the monotonicity argument is the following. Since the workloads $|P_i|$ of the output partition are non-decreasing, it is easy to see that it suffices to prove monotonicity in the special case when a single rounded speed $s_i = (1 + \epsilon)$ is reduced to $s'_i = 1$. Let the output path of OPTPATH be Q in the first, and Q' in the second case, moreover let f and M denote finish time and makespan for

input (s_i, s_{-i}) and f' and M' for input (s'_i, s_{-i}) , respectively. Our argument is based on the following observation. The makespan of any (sub)path in \mathcal{H}_I cannot decrease by reducing a machine speed. The objectives concerning the optimal path are defined so that $\mathcal{Q}' \neq \mathcal{Q}$ may occur only if machine i becomes a bottleneck machine, either concerning the whole path (meaning $M(\mathcal{Q}) < M'(\mathcal{Q}) = f'(v_i)$), or some relevant subpath (e.g., prefix path, or the subpath (v_{m-2}, v_{m-1}, v_m)). In any of these cases, \mathcal{Q} is a possible solution of (local) makespan $f'(v_i)$, and \mathcal{Q}' is preferred only in case the respective (sub)path of \mathcal{Q}' has no higher (local) makespan than $f'(v_i)$, implying that i gets not more workload than $f'(v_i) = (1 + \epsilon)f(v_i)$. This proves monotonicity if $(1 + \epsilon)f(v_i)$ is the exact workload, or at least a lower bound on what i received with the original speed (e.g., if $i < k$). On the other hand, if $(1 + \epsilon)f(v_i)$ was just a δ -estimate, then the machine was close to full with input s_i , and, by reallocating 'many' or all tiny blocks, the new makespan (no matter how the output path looks like!) becomes 'much' smaller than $f(v_i)(1 + \epsilon)$.

The other way round, the output path \mathcal{Q} changes to $\mathcal{Q}' \neq \mathcal{Q}$ only if i became a bottleneck machine w.r.t. some relevant subpath. As long as all considered subpaths remain optimal, there is no reason to change by some other optimization criterion: a better path \mathcal{Q}' would have been better than \mathcal{Q} with input s_i as well.

Our main result is thus the following theorem. See Appendix B for a more detailed proof.

THEOREM 5.2. *Algorithm PTAS is monotone.*

References

- [1] Nir Andelman, Yossi Azar, and Motti Sorani. Truthful approximation mechanisms for scheduling selfish related machines. In *22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 69–82, 2005.
- [2] Nir Andelman, Yossi Azar, and Motti Sorani. Truthful approximation mechanisms for scheduling selfish related machines. *Theory of Computing Systems*, 40(4):423–436, 2007.
- [3] Aaron Archer. *Mechanisms for Discrete Optimization with Rational Agents*. PhD thesis, Cornell University, January 2004.
- [4] Aaron Archer and Éva Tardos. Truthful mechanisms for one-parameter agents. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2001.
- [5] Vincenzo Auletta, Roberto De Prisco, Paolo Penna, and Giuseppe Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In Volker Diekert and Michel Habib, editors, *STACS*,

volume 2996 of *Lecture Notes in Computer Science*, pages 608–619. Springer, 2004.

- [6] George Christodoulou and Elias Koutsoupias. Mechanism design for scheduling. *Bulletin of EATCS*, 97:39–59, February 2009.
- [7] Peerapong Dhangwatnotai, Shahar Dobzinski, Shaddin Dughmi, and Tim Roughgarden. Truthful approximation schemes for single-parameter agents. In *FOCS*, pages 15–24, 2008.
- [8] Leah Epstein and Jiri Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):43–57, 2004.
- [9] Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988.
- [10] Annamária Kovács. Fast monotone 3-approximation algorithm for scheduling related machines. In *Algorithms - ESA 2005, 13th Annual European Symposium*, pages 616–627, 2005.
- [11] Annamária Kovács. Tighter approximation bounds for lpt scheduling in two special cases. *J. Discrete Algorithms*, 7(3):327–340, 2009.
- [12] Roger B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.
- [13] N. Nisan, T. Roughgarden, E. Tardos, and V.V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.
- [14] Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC)*, pages 129–140, 1999.
- [15] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

Appendix

A The procedures OPTPATH and PARTITION

The pseudo-codes are presented in Figures 3 and 4. Observe, that by the definition of $M()$ and $opt()$ values, $M(v_k) = M(\mathcal{Q})$. Moreover, since in each case the pointers $pred()$ and $succ()$ determine an incoming, and an outgoing path of minimum makespan, respectively, we can make the following observation:

OBSERVATION 1. *For all $v \in V_{II}$, the value $M(v)$ is the minimum makespan over all m -paths having vertex v as switch vertex. Consequently, $M(v_k) = M(\mathcal{Q})$ is the minimum makespan over all m -paths of \mathcal{H}_I .*

B Monotonicity

The next lemma is a descriptive characterization of the final output allocation P_1, \dots, P_m to be readily used in the monotonicity proof.

Procedure 2 OPTPATH

Input: The directed graph \mathcal{H}_I .Output: The optimal m -path $\mathcal{Q} = (v_1, \dots, v_m)$ of \mathcal{H}_I .

1. for every double $v'_{m-2} = (\alpha_{m-2}, \alpha_{m-1}) \in V_{II}$ do
 $opt(v'_{m-2}) := \max\{f(v_{m-2}), f(v_{m-1}), f(v_m)\}$
(where α_{m-1} determines α_m , see Section 4)
 $M(v'_{m-2}) := \max\{\frac{|\alpha_{m-2}|}{s_{m-2}}, \frac{|\alpha_{m-1}|}{s_{m-1}}, \frac{|\alpha_m|}{s_m}\};$
for $i = m - 3$ downto 1 do
for every $v = (d, i, \alpha) \in V_{II}$ do
 - (i) $succ(v) := w$, if $opt(w)$ is minimum over all vertices y with $(v, y) \in E$, and among such vertices the configuration α of w is \prec -minimal.
 - (ii) $opt(v) := \max\{f(v), opt(succ(v))\};$
 $M(v) := \max\{\frac{|\alpha|}{s_i}, opt(succ(v))\}.$
2. for every $v = (d, 1, \alpha) \in V_I$ do $opt(v) := f(v);$
for $i = 2$ to $m - 2$ do
for every $v = (d, i, \alpha) \in V_I \cup V_{II}$ do
 - (i) $pred(v) := w \in V_I$, if $opt(w)$ is minimum over $y \in V_I$ with $(y, v) \in E$, and among such vertices the configuration α of w is \prec -minimal.
 - (ii) if $v \in V_I$ then
 $opt(v) := \max\{f(v), opt(pred(v))\};$
if $v \in V_{II}$ then
 $M(v) := \max\{M(v), opt(pred(v))\}.$
3. select an optimal *switch vertex* $v_k = (II, k, \alpha_k) \in V_{II}$, by the following objectives:
 - (i) $M(v_k) = \min\{M(v) \mid v \in V_{II}\};$
 - (ii) k is maximum over all v of minimum $M(v);$
 - (iii) if $k \leq m - 3$ then α_k is minimal wrt. \prec over all v of minimum $M(v)$ in layer $k;$
 - (iv) if $k = m - 2$, then among all double vertices $v'_{m-2} = (\alpha_{m-2}, \alpha_{m-1})$ of minimum $M(v'_{m-2})$ select $v_k = v'_{m-2}$ by the following objectives:
 - (a) keep the order (A) (B) (cf. Section 4);
 - (b) in case of (A), select an $(\tilde{\alpha}_{m-2}, \tilde{\alpha}_{m-1}, \tilde{\alpha}_m, (T_{\alpha_{m-2}} \cup T_{\alpha_{m-1}} \cup T_{\alpha_m}))$ (i.e., with a common pool of tiny blocks) by some predefined ordering, and then minimize (also) the *second* highest finish time among $\frac{|\alpha_{m-2}|}{s_{m-2}}, \frac{|\alpha_{m-1}|}{s_{m-1}}$, and $\frac{|\alpha_m|}{s_m}$ (by redistributing the tiny jobs);
 - (c) in case of (B), select an $(\alpha_{m-2}, \alpha_{m-1}, \alpha_m)$ by some predefined ordering, but so that $|\alpha_{m-1}| + |\alpha_m|$ is maximized;
4. for $i = k - 1$ downto 1 do $v_i := pred(v_{i+1});$
for $i = k + 1$ to m do $v_i := succ(v_{i-1});$
 $\mathcal{Q} := (v_1, \dots, v_k, \dots, v_m).$

Figure 3: Procedure OPTPATH finds an m -path \mathcal{Q} of minimum $M(\mathcal{Q})$ in the directed graph.

LEMMA B.1. Let P_I denote the set of input jobs and s be the vector of rounded speeds. Let $\mathcal{Q} = (v_1, \dots, v_m)$ be the path output by OPTPATH with configurations $(\alpha_1, \dots, \alpha_m)$ in the vertices, and $v_k = (II, k, \alpha_k)$ be the switch vertex of the path. If P_1, \dots, P_m is the final partition output by PTAS, then

(a) for $i < k$, $P_i = Q_i = \alpha_i$.

Moreover, if $k \leq m - 3$, then

(b) for $i > k$, $\frac{|P_i|}{s_i} \geq (1 - 6\delta) \cdot M(\mathcal{Q});$

(c) for k either $P_k = Q_k$ or (b) holds.

LEMMA B.2. In case $k = m - 2$, step 3b. of PARTITION can be realized so that $|Q_{m-2}| \leq |Q_{m-1}| \leq |Q_m|$ holds.

COROLLARY B.1. In step 5. of PTAS, the sets Q_i are permuted only among machines $i \geq k$ of equal rounded speed, and only if $k \leq m - 3$. Consequently, $|P_i|/s_i \leq M(\mathcal{Q})$ for all i .

Below we show an incomplete proof of the monotonicity, in that we omit the proof of Lemma B.3, treating the case $k = m - 2$. The omitted proof is neither longer nor more difficult than the presented part; it operates with the same type of arguments, and needs a slightly more complex case analysis.

Theorem 5.2. Algorithm PTAS is monotone.

Proof. Assume that machine i alone decreased its speed σ_i to σ'_i in the input. If the vector of rounded speeds (s_1, s_2, \dots, s_m) remains the same, then the deterministic PTAS outputs the same allocation, and i receives the same, or smaller workload, since the output workloads P_1, P_2, \dots, P_m are in increasing order. Assuming that the rounded speed s_i decreased as well, it is enough to consider the special case when i is the first (smallest index) machine of rounded speed $s_i = (1 + \epsilon)$ in input $I(P, s)$, and after reducing its speed, it becomes the last (highest index) machine of rounded speed 1 in input $I'(P, s')$. Since the workloads in the final allocation P_1, \dots, P_m are ordered, this implies monotonicity for every 'one-step' speed change (like $(1 + \epsilon) \rightarrow 1$). Monotonicity in general can then be obtained by applying such a step repeatedly. Note that for both inputs the algorithm constructs the same graph, independently of the speed vector. We assume that with inputs I , and I' OPTPATH outputs $\mathcal{Q} = (v_1, \dots, v_m)$, and $\mathcal{Q}' = (v'_1, \dots, v'_m)$, where the switch vertices have index k and k' , respectively. Finish time, makespan, etc. wrt. the *new* speed vector s' are denoted by $f'()$, $M'()$ etc. We prove that $|P_i| \geq |P'_i|$.

Procedure 3 PARTITION

Input: The job set P_I , and an m -path $\mathcal{Q} = (v_1, \dots, v_m)$ with switch vertex v_k in the graph \mathcal{H}_I .

Output: A partition Q_1, Q_2, \dots, Q_m of the set P_I .

Case LOW- k : $|\alpha_k|/s_k \leq (1 - \epsilon/2) \cdot M(\mathcal{Q})$

Case HIGH- k : $|\alpha_k|/s_k > (1 - \epsilon/2) \cdot M(\mathcal{Q})$.

1. for $i = 1$ to m do
 $Q_i := \tilde{\alpha}_i$;
2. let $T = \{t_1, t_2, t_3, \dots, t_\mu\} = P_I \setminus \bigcup_{i=1}^m Q_i$, so that the jobs t_j are in increasing order;
- 3a. if $k \leq m - 3$, then let $W = 0$ and $r = 0$;
 for $i = k$ to $m - 1$ do
 given $\alpha_i = (w, \mu, \vec{n}^o, \vec{n}^1)$ and $\lambda = \log \rho w$, let $W_i := (n_\lambda^1 - n_\lambda^o) \cdot \rho w$;
 (i) $W := W + W_i$;
 (ii) if HIGH- k then let u be the maximum index in T so that $\sum_{j=1}^u t_j \leq W$;
 if LOW- k then let u be the minimum index in T so that $\sum_{j=1}^u t_j \geq W$;
 (iii) $Q_i := Q_i \cup \{t_{r+1}, t_{r+2}, \dots, t_u\}$;
 (iv) $r := u$.
 $Q_m := Q_m \cup \{t_{r+1}, t_{r+2}, \dots, t_\mu\}$.
- 3b. if $k = m - 2$, then start with an allocation of tiny jobs to $\{m - 2, m - 1, m\}$ (in the given order) s. t. each machine i gets at most $|T_{\alpha_i}|$ amount of tiny jobs (this is doable, because the total number of tiny blocks is overestimated by 3 blocks in α_m)
 let $M = \max\{\frac{|\alpha_{m-2}|}{s_{m-2}}, \frac{|\alpha_{m-1}|}{s_{m-1}}, \frac{|\alpha_m|}{s_m}\}$;
 call $i \in \{m - 2, m - 1, m\}$ *low*, if $|\alpha_i|/s_i \leq (1 - 2\epsilon/3) \cdot M$, and *high* if $|\alpha_i|/s_i \geq (1 - \epsilon/2) \cdot M$;
 Correct the partition of tiny jobs (with keeping the job order) so that
 (i) if there is one low machine i , and two high machines, then i receives at least $|\alpha_i|$ work;
 (ii) if there are two non-high machines, then both receive at least $6\rho w$ work of tiny jobs.

Figure 4: Procedure PARTITION allocates the jobs based on path \mathcal{Q} output by OPTPATH.

We start with a simple observation. Since we decreased a machine speed, it follows from the definition of makespan that for any path $\mathcal{R} = (v_1, v_2, \dots, v_r)$ within level V_I , or within level V_{II} , and for any m -path, $M'(\mathcal{R}) \geq M(\mathcal{R})$. Similarly, for any vertex v , $opt'(v) \geq opt(v)$, and for any $v \in V_{II}$ $M'(v) \geq M(v)$ (cf. Procedure OPTPATH). Obviously, also the optimum makespan over all m -paths could not decrease. We elaborate on the subtle case of $k = m - 2$ in a separate lemma; in what follows, we assume $k \leq m - 3$.

CASE 1: $M'(\mathcal{Q}) > M(\mathcal{Q})$

In this case machine i with the new rounded speed $s'_i = 1$, becomes a bottleneck in path \mathcal{Q} . That is, $M'(\mathcal{Q}) = f'(v_i) = \frac{|\alpha_i|(1+\rho w)}{1}$.

If $i < k$, then $\alpha_i = P_i$, so the machine received exactly $|\alpha_i|$ work with speed s_i , and now \mathcal{Q} is a path with makespan $|\alpha_i|$, so by Corollary B.1, and by the optimality of \mathcal{Q}' we have $|P'_i| = |P_i|/s'_i \leq M'(\mathcal{Q}') \leq M'(\mathcal{Q}) = |\alpha_i| = |P_i|$.

Let us introduce the notation $B \stackrel{\text{def}}{=} (1 - 6\delta) \cdot M(\mathcal{Q})$ for the lower bound in Lemma B.1 on finish times. Recall that $s_i = (1 + \epsilon)$. Assume now that $i \geq k$ and $|P_i|/(1 + \epsilon) \geq B$.

Due to (E2), for the job partition Q_1, \dots, Q_m (before ordering the sets by size), it holds that $|Q_h| \geq |L_{\alpha_i}|$ for every $h \geq i$. Therefore, for the $m - i + 1$ st largest set P_i , we have $|P_i| \geq |L_{\alpha_i}|$, and so $|P_i| \geq \max\{|L_{\alpha_i}|, (1 + \epsilon) \cdot B\}$.

We modify the path \mathcal{Q} and construct a new path \mathcal{Q}'' by 'putting' small jobs from S_{α_i} (of machine i) onto machine $i + 1$, until S_{α_i} becomes empty, or the moved jobs have total weight of at least $(\epsilon/3) \cdot (1 + \epsilon) \cdot M(\mathcal{Q})$. For the new finish time we have $f'(v'_i) \leq \max\{|L_{\alpha_i}|, (1 + \epsilon) \cdot M(\mathcal{Q})(1 - \epsilon/3)\} \leq \max\{|L_{\alpha_i}|, (1 + \epsilon) \cdot B\} \leq |P_i|$. If $i = m$, then we put only *tiny* blocks of the common magnitude w_{m-1} onto $m - 1$, and use $|\tilde{\alpha}_m|$ instead of $|L_{\alpha_i}|$ in the calculation.

It is easy to see that with speed $s'_i = 1$ machine i is still a bottleneck machine in path \mathcal{Q}'' , since it is filled up to about $(1 + 2\epsilon/3) \cdot M(\mathcal{Q})$, while other machines are filled not higher than $(1 + \epsilon/3) \cdot M(\mathcal{Q})$, even with the jobs received from i . Thus, $f'(v'')$ is an upper bound on the new optimal path-makespan, and so on $|P'_i|$, while it is less than $|P_i|$.

By Lemma B.1, it remains to consider the case $i = k$, and $P_i = Q_i$. Given $M'(\mathcal{Q}) > M(\mathcal{Q})$, we have $M'(\mathcal{Q}') \leq M'(\mathcal{Q}) = |\alpha_i|$ as an upper bound on $|P'_i|$. Assuming that for $i = k$ LOW- k holds with speed s_i , $|P_i| = |Q_i| \geq |\alpha_i|$ by PARTITION 3a, and we are done. Assuming HIGH- k , $|P_i| = |Q_i| \geq \max\{|\tilde{\alpha}_i|, |\alpha_i| - \rho w_i\}$. On the other hand, $|\alpha_i| > (1 - \epsilon/2)M(\mathcal{Q}) \cdot (1 + \epsilon) > (1 + \epsilon/3)M(\mathcal{Q})$. By putting one tiny block onto the

next machine (if there are any), we still obtain a path \mathcal{Q}'' , so that $\max\{|\tilde{\alpha}_i|, |\alpha_i| - \rho w_i\} = |\alpha_i''| = M'(\mathcal{Q}'') \geq M'(\mathcal{Q}') \geq |P'_i|$, and we are done. Note that here again we used that with α_i'' machine i is still bottleneck, by $|\alpha_i| > (1 + \epsilon/3)M(\mathcal{Q})$, and so $|\alpha_i''|$ is the makespan.

As follows from Lemma B.1 (b), the only remaining case is

CASE 2. $M'(\mathcal{Q}) = M(\mathcal{Q})$, and $i \leq k$.

If $M'(\mathcal{Q}) = M(\mathcal{Q})$, and $\mathcal{Q} = \mathcal{Q}'$, then the output of PARTITION can only be different if $i = k$. Since the makespan did not change, and s_k decreased, the change is from LOW- k to HIGH- k , and machine $i = k$ receives less work with s'_k , by PARTITION 3a. If, on the other hand, PARTITION outputs the same partition, then every machine gets the same workload.

Suppose $M'(\mathcal{Q}) = M(\mathcal{Q})$, but $\mathcal{Q} \neq \mathcal{Q}'$. Since \mathcal{Q} has minimum makespan, $M'(\mathcal{Q}') = M'(\mathcal{Q})$. We claim that also $k' = k$. Otherwise \mathcal{Q}' would have been better than \mathcal{Q} for input s as well, because $M(\mathcal{Q}') \leq M'(\mathcal{Q}') = M(\mathcal{Q})$ and $k' > k$. Similarly, also $v_k = v'_k$, otherwise $\alpha' \prec \alpha$ would hold, and \mathcal{Q}' would have been better for input s as well.

Now, since $\mathcal{Q} \neq \mathcal{Q}'$, a maximum $h < k$ exists so that $v_h \neq v'_h$. This means that $\text{pred}(v_{h+1}) \neq \text{pred}'(v_{h+1})$. If v'_h was preferred in \mathcal{Q}' because $\alpha'_h \prec \alpha_h$, then $\text{opt}(v_h) < \text{opt}(v'_h) \leq \text{opt}'(v'_h) \leq \text{opt}'(v_h)$. The first inequality holds, otherwise $v'_h = \text{pred}(v_{h+1})$ would have been the choice. The second holds for every vertex. The third holds, otherwise $v_h = \text{pred}'(v_{h+1})$ would have been the choice of OPTPATH. Similarly, if v'_h was preferred in \mathcal{Q}' because $\text{opt}'(v'_h) < \text{opt}'(v_h)$, then $\text{opt}(v_h) \leq \text{opt}(v'_h) \leq \text{opt}'(v'_h) < \text{opt}'(v_h)$. In both cases we obtained $\text{opt}(v_h) < \text{opt}'(v_h)$. Recall that $\text{opt}(v_h)$ is the optimum makespan over all paths leading to v_h from layer 1. This could strictly increase only if $i \leq h$, and i with workload $\alpha_i = P_i$ and speed $s'_i = 1$ became a bottleneck machine in (v_1, v_2, \dots, v_h) . Therefore, $|P'_i| \leq \text{opt}'(v'_h) \leq \text{opt}'(v_h) \leq \frac{|\alpha_i|}{s'_i} = |P_i|$.

LEMMA B.3. *If on input $I(P, s)$, for the output path \mathcal{Q} of OPTPATH the switch machine is $k = m - 2$, then $|P_i| \geq |P'_i|$.*