A truthful constant approximation for maximizing the minimum load on related machines*

George Christodoulou† Annamária Kovács‡ Rob van Stee§

September 8, 2011

Abstract

Designing truthful mechanisms for scheduling on related machines is a very important problem in single-parameter mechanism design. We consider the covering objective, that is we are interested in maximizing the minimum completion time of a machine. This problem falls into the class of problems where the optimal allocation can be truthfully implemented. A major open issue for this class is whether truthfulness affects the polynomial-time implementation.

We provide the first constant factor approximation for deterministic truthful mechanisms. In particular we come up with a approximation guarantee of $2 + \varepsilon$, significantly improving on the previous upper bound of $\min(m, (2 + \varepsilon)s_m/s_1)$.

Keywords: algorithmic mechanism design, scheduling, machine covering, approximation algorithms

1 Introduction

Algorithmic Mechanism Design studies scenarios where there is an optimization problem at hand, but selfish agents control some input parameters. These parameters are unknown to the optimizer and are *private* values of the agents. Moreover, the agents might be only interested in satisfying their own interests and therefore they might have incentive to misreport their values, if this can lead to an output or solution that they prefer. In order to elicit the missing information, the mechanism design approach uses side payments to motivate the agents to reveal their true values. Roughly speaking, a *mechanism* consists of two components: an algorithm that takes as input the reported values, and returns a solution of the optimization problem, and a payment algorithm that hands out payments to the agents. Each agent's goal is to maximize her utility, that is the payment she gets minus her actual value on the solution. A mechanism is *truthful* if it is in the best interest of each agent to report truthfully.

Given a class of problems, the challenge is to characterize the objective functions that one can truthfully optimize/approximate. Under this framework, scheduling is a very natural and well-studied setting to explore the boundaries of truthful implementation. On the one hand, the algorithmic techniques that have been

*An extended abstract of this paper appeared in the 6th Workshop on Internet and Network Economics (WINE 2010), LNCS 6484, p.182–193. Springer, 2010.

†University of Liverpool, U.K. gchristo@liv.ac.uk

‡Department of Informatics, Goethe University, Frankfurt am Main, Germany. panni@cs.uni-frankfurt.de. Research supported by the German Research Foundation (DFG) grant KO 4083/1-1.

§Max Planck Institute for Informatics, Saarbrücken, Germany. vanstee@mpi-inf.mpg.de. Research supported by the German Research Foundation (DFG) grant STE 1727/3-2.

developed are very broad, and the question is to what extent those techniques can be applied to the design of truthful mechanisms. On the other hand scheduling is conceptually similar to a combinatorial auction, a setting that is very important in economics, and therefore insights can be transfered from one problem to the other. In a scheduling setting, there are m machines and n tasks, and each machine is controlled by an agent that has as private values the processing times it needs to execute the tasks. The algorithmic goal is to allocate the jobs to the machines so that some objective (most commonly the makespan) is optimized. In the unrelated machines setup the processing times for each machine are expressed via a vector of size m, while for the related machines setup they are expressed via a single parameter, the speed of the machine.

A natural question that arises in many single-parameter settings is: what is the approximability of *polynomial-time* truthful mechanisms? Taking a problem that one can solve exactly with a truthful mechanism, can one also achieve the best possible approximation guarantee, or does truthfulness have a negative computational impact? Is the class of polynomial-time truthful mechanisms less powerful with respect to approximation, compared to the class of polynomial-time non-truthful algorithms? For makespan minimization such a separation does not exist. Dhangwatnotai et al. [8] showed a randomized truthful-in-expectation PTAS and later Christodoulou and Kovács [7] showed a deterministic truthful PTAS that is the best one can achieve even with non-truthful approximation algorithms [13, 10].

In order to explore further the performance of truthful mechanisms in single-parameter problems, we focus on the covering objective for scheduling on related machines, that is we are interested in maximizing the minimum completion time over all machines. This objective is important in settings where a system is only alive if all of its components are alive. One can think of the jobs as batteries with varying capacities, or hard-drives of various sizes that we want to use as a backup medium [16]. The covering problem is also closely related to the max-min fairness problem, where we want to distribute indivisible goods to players so as to maximize the minimum valuation.

Mu'alem and Schapira [14] showed that maximizing the minimum load for unrelated machines cannot be approximated within a constant factor by a deterministic truthful mechanism.[1] On the other hand, using the arguments in [1], one can show that for related machines the optimal allocation is truthful, although not efficient. For the non-strategic version, Epstein and Sgall showed a PTAS [10].

The question we address in this paper is: 'What is the best deterministic, polynomial-time, truthful approximation mechanism that one can design for the covering problem?' We provide the first deterministic truthful mechanism with constant approximation for the covering objective. In particular, we obtain an approximation guarantee of $2 + \varepsilon$.

Related work The non-strategic version of the problem has been extensively studied in the past in various contexts for online and approximations algorithms. For identical machines, Woeginger [17] designed a polynomial time approximation scheme (PTAS) and gave tight results for deterministic online algorithms. Azar and Epstein [3] studied the randomized online setting. Furthermore, for the case where jobs arrive in non-increasing order and also the optimal value is known in advance, they gave a deterministic 2-competitive online algorithm NEXT COVER.

In [4], a PTAS was designed for related machines, and later this was generalized to capture a large class of objective functions in [10]. Epstein and van Stee [11] provide a PTAS and also an FPTAS for constant number of related machines which they then use as a subroutine for a truthful FPTAS, while Efraimidis and Spirakis [9] show an FPTAS for the more general case of unrelated machines. Dhangwatnotai et al. [8] provide a randomized truthful-in-expectation PTAS. Epstein and van Stee [11] give a monotone approximation

[1] In fact the authors showed this for the combinatorials auctions setup where the agents are utility maximizers, while in the scheduling setting the agents are cost minimizers. However, a simple modification of their argument works for scheduling as well.

algorithm with approximation ratio $\min(m, (2+\varepsilon)s_m/s_1)$ where $\varepsilon > 0$ can be chosen arbitrarily small and s_i is the (real) speed of machine i.

The max-min fairness problem has been studied intensively in recent years, see for instance [2, 5, 6, 12] and references therein.

Our results and techniques. For any positive $\varepsilon < 1/5$, we show a $(2 + 5\,\varepsilon)$-approximation, *monotone* algorithm for the covering problem. Monotonicity of a scheduling algorithm means that whenever a single machine (agent) increases his reported speed (asuming that the other speeds are unchanged), the machine receives not less total jobsize, than with the original speed. As known from the classic work of Myerson [15], and completed with the payment scheme given there, this yields a truthful mechanism, that is computable in polynomial running time for constant ε. With this result we significantly improve upon the previous best approximation ratio of $\min(m, (2+\varepsilon)s_m/s_1)$ given in [11].

As a standard technique applied in all approximation schemes for related scheduling [13, 10, 8, 7], we define a directed acyclic graph with vertices representing possible job-sets allocated to single machines. Relative to the total size of any given set, we distinguish *normal* and *tiny* jobs in the set. We consider a special form of schedules, where the whole sequence of machines is partitioned into *segments*, each segment having either sets of (nearly) only normal jobs, or sets of only tiny jobs. The allocation of jobs *within* segments must adhere to strict regulations, which allow for both good approximation and polynomial-time optimization.

We could exploit some of the ideas used for the monotone PTAS for related machine scheduling (with the makespan objective) [7], while defining an essentially different type of allocation. We point out that the current result is not a straightforward adaptation of [7]. Somewhat surprisingly, we were unable to find such an adaptation for maximizing the cover: although in many aspects the setting is symmetric to that of makespan minimization, this symmetry breaks when handling the tiny jobs.[2]

On the positive side, striving for the weaker approximation ratio admits a simpler and technically less demanding construction than in [7].

2 Preliminaries

The input is given by a set P_I of n input jobs, and a vector σ of input speeds $\sigma_1 \leq \ldots \leq \sigma_m$. We round up every input speed to the nearest integral power of $1 + \varepsilon$. Denoting the respective rounded speeds by s_i, we have $s_1 \leq \ldots \leq s_m$. We use the interval notation (e.g., $[1, m]$) for a set of consecutive machine indices. The letters p or q are used to denote jobs, as well as the respective job sizes in a given formula. We fix a nondecreasing order $p_1 \leq p_2 \leq \ldots \leq p_n$ of all input jobs. If $Q = \{q_1, q_2, \ldots, q_j\}$ is an arbitrary job set, then the *weight* or *workload* of Q is $|Q| = \sum_{r=1}^{j} q_r$. An allocation of the jobs to the machines is an *(ordered) partition* $(P_1, P_2, \ldots, P_m)$ of the jobs into m *sets*. We search for an output where the workloads $|P_i|$ are in non-decreasing order. We assume w.l.o.g. that $n \geq m$, since otherwise the cover is trivial. We are only interested in allocations where $P_i \neq \emptyset$ for $i \in [1, m]$ (otherwise the approximation ratio is ∞).

Our graph-algorithm outputs a schedule of optimum cover over a restricted type of job partitions. We name these partitions *segmented partitions*, because the output partition can be subdivided into *segments*, each consisting of consecutive job sets of the partition. Every job allocated in some earlier segment precedes all jobs allocated in any later segment, with respect to the fixed job order. The allocation of jobs within a partition segment will have to adhere to one of two forms: *smooth allocation*, or *canonical allocation*.

[2] The difficulty lies roughly in the fact that in case of makespan minimization a machine that becomes bottleneck loses (many or all of) its tiny jobs, while in case of maximizing the cover, a bottleneck machine might collect all tiny jobs from faster machines. This makes exact optimization with our methods impossible, since the *exact* workload of a set of tiny jobs is not known.

Input: $(Q_1, \ldots, Q_r)$ be a partition of a subset Q of the input jobs
Output: a partition $(Q'_1, \ldots, Q'_r)$ of Q with canonical allocation

CANON($Q_1, \ldots, Q_r$)
If $r = 1$ then return (Q_1). Else

1. $Q := \bigcup_{k=1}^{r} Q_k$
2. Let Q_i^Q be a set of maximum workload among $\{Q_1^Q, Q_2^Q, \ldots, Q_r^Q\}$.
3. $Q'_r := Q_i^Q$.
4. in $Q_1, \ldots, Q_{i-1}, Q_{i+1}, \ldots, Q_r$ exchange the jobs of Q_i^Q for jobs (of the same class) of Q_i
5. $(Q'_1, \ldots, Q'_{r-1}) :=$ CANON($Q_1, \ldots, Q_{i-1}, Q_{i+1}, \ldots, Q_r$)
6. return $(Q'_1, \ldots, Q'_r)$

Figure 1: Canonization procedure. We use the following definition: Let $Q_i \subset R$ be sets of input jobs. We say that we *maximize Q_i with respect to R*, if for every class l we replace the jobs in $Q_i \cap C_l$ by the largest jobs in $R \cap C_l$ (i.e., if there are t such jobs in $Q_i \cap C_l$, then we replace them with the t jobs of highest index in $R \cap C_l$). We denote the maximized set by Q_i^R.

A *smooth allocation* of a set of consecutive jobs $P = \{p_j, p_{j+1}, \ldots, p_k\}$ into r sets is the partition segment output by the following *smoothing procedure*: We construct a fractional allocation into r sets of equal workloads of size $|P|/r$ (we assume $p_k \ll |P|/r$). We start with the smallest job p_j, add jobs in the fixed order, and cut a job into two whenever the total workload reached $|P|/r$. We continue with the next set, and the remaining part of the divided job, and so on. Next, we allocate each job that was cut into two, to the first one of its two sets. Finally, we order the job partition in increasing order of workloads.

Before we turn to *canonical allocations*, we need to fix the constants δ and ρ, and classify the input job sizes accordingly. For a desired approximation bound of $2 + 5\varepsilon$, we choose a $\delta \ll \varepsilon$. [3] For ease of exposition, we will assume that $(1+\delta)^t = 2$ for some $t \in \mathbb{N}$. Furthermore, we define ρ as the unique integer power of 2 in $(\delta/8, \delta/4]$.

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

A *canonical allocation* within a segment means that the sets have non-decreasing workloads, moreover that jobs that belong to the same job class appear in increasing order over the sets of the segment. Given an arbitrary partition $(Q_1, Q_2, \ldots, Q_r)$ of some *subset* $Q \subseteq P_I$ of the jobs, the *canonization procedure* shown in Figure 1 constructs a partition $(Q'_1, Q'_2, \ldots, Q'_r)$ of Q with canonical allocation. The procedure appeared previously in the full version of [7]; we include it here for completeness. In effect, this procedure
(A) permutes jobs *within job classes*, and thus *perturbs* each set Q_i, so that the *perturbed set* $\widetilde{Q}_i$ has a workload in $[|Q_i|/(1+\delta), |Q_i|(1+\delta)]$; and then
(B) sorts the perturbed $\widetilde{Q}_i$ sets by increasing workloads to obtain $(Q'_1, \ldots, Q'_r)$.

[3] $12\delta < \varepsilon$ suffices

3 Segmented partitions

In the following we introduce *magnitudes*, and make the definition of segmented partitions exact. As the main result of the section, we show in Theorem 1 that for arbitrary input with rounded speeds, a segmented partition of cover within a factor of $\frac{1-\varepsilon}{2}$ of the optimum exists. As done previously in [10, 8, 7], we associate a *magnitude* w_i, an integer power of 2, with each set P_i in the partition. The set with the associated magnitude will also be denoted by (P_i, w_i). Magnitudes are used to focus on the relevant job sizes when representing job sets with the help of integer arrays. We will require $w_i/5 < |P_i| \leq w_i$ $(i \in [1, m])$, and $w_1 \leq w_2 \leq \ldots \leq w_m$.

Definition 2 *A job p is* tiny wrt. magnitude w_i, *if $p \leq \rho \cdot w_i$. A jobset (P_i, w_i) is sandy, if all jobs in P_i are tiny wrt. w_i. A jobset (P_i, w_i) is* normal*, if it has at least one non-tiny job, and a (possibly empty) consecutive sequence of the largest tiny jobs wrt. w_i.*

Note that the property of being a tiny job for some magnitude either holds for a whole class of jobs or for none of them, as $\rho \cdot w_i$ is an integer power of 2. Next we define the two allowed sorts of partition segments: one for sandy sets, and another one for normal sets.

Definition 3 *A* sandy segment *consists of sandy sets of equal magnitude $(P_i, w), (P_{i+1}, w), \ldots, (P_h, w)$ with a smooth allocation of consecutive jobs (of size at most $\rho \cdot w$).*

Definition 4 *A* normal segment *of a partition consists of normal sets of nondecreasing magnitudes (P_i, w_i), $(P_{i+1}, w_{i+1}), \ldots, (P_h, w_h)$, with a canonical job allocation. The union of the sets contains consecutive jobs of the ordered input.*

Definition 5 *A* segmented partition *is a partition of the input jobs into m jobsets P_i of non-decreasing workloads $|P_i|$. It is subdivided into partition segments each of which is either sandy or normal; if job p precedes job q in the fixed ordering, then p belongs to the same segment as q, or to an earlier segment.*

Theorem 1 *Let $0 < \varepsilon < 1/5$ and $\delta \ll \varepsilon$ be fixed. Given an arbitrary set of n input jobs in a fixed non-decreasing order, and m non-decreasing input speeds that are integral powers of $(1 + \varepsilon)$, there exists a segmented partition having a cover of at least $(1 - \varepsilon)\frac{\text{OPT}}{2}$, where* OPT *means the optimum cover.*

Proof We begin by showing the following claim.

Claim 1 *It is possible to allocate the jobs in nondecreasing order, to machines of nondecreasing speeds, so that the machines are filled up to a finish time of at least* $\text{OPT}/2$.

Proof Consider first the following simple greedy allocation procedure (a variant of the 2-approximation algorithm of Azar and Epstein [3]). It is well known (see e.g., [10]), that a schedule of maximum cover, and increasing workloads over the machines of increasing speeds exists. Suppose that W_i is the workload of machine i in a fixed optimal schedule of increasing workloads. Fill the machines with consecutive jobs, in the given increasing order, each up to workload W_i. Cut a job into two when W_i is reached and continue on the next machine. Observe that in the resulting fractional allocation every (fractional) job p assigned to machine i has full size of at most W_i. Otherwise only jobs smaller than p would be allocated to machines $1, 2, \ldots, i$ in the optimal schedule, contradicting that *all* jobs smaller than p did not fill the total workload $\sum_{h=1}^{i} W_h$. This implies that each machine with a fractional job has at least two (fractional) jobs. Thus,

Input: job set P_I, machine speeds $s_1 \leq \cdots \leq s_m$, optimal cover OPT.

1. Let $C = \text{OPT}/2$. For machine i of speed s_i let the *pre-magnitude* w'_i (a power of 2) be uniquely defined so that $2s_iC \leq w'_i < 4s_iC$.
2. Allocate jobs in the fixed increasing order as follows. For $i = 1$ to $m - 1$ do
   (a) Assign jobs to the current machine until the finish time is at least $(1 - \varepsilon/2)C$.
   (b) If the last (largest) job on the machine has size more than $\rho w'_i$, continue assigning jobs until the finish time is at least C.
3. Assign the remaining jobs to machine m.

Output: job assignment $Q_1, \ldots, Q_m$ with cover of at least $(1 - \varepsilon/2)\text{OPT}/2$.

Figure 2: Greedy integral allocation procedure.

if we allocate every job to the lower-indexed machine where it appears, then each machine i loses, in the worst case, the smallest one among its (at least two) jobs, so that it still keeps a workload of at least $\frac{W_i}{2}$. In summary, we could allocate the *integral* jobs consecutively, in increasing order, so that the machines are filled up to a finish time of at least $\text{OPT}/2$. □

Obviously, this claim also holds if each machine i is filled up to some given finish time $f_i \leq \text{OPT}/2$.

Now we construct the segmented partition for the given input. We start by a greedy integral allocation which is shown in Figure 2. Note that the pre-magnitudes defined in Step 1 are increasing in i. Observe that those machines that were filled up to $C = \text{OPT}/2$ have normal jobsets (call them normal machines), and those filled only to $(1 - \varepsilon/2) \cdot C$ have sandy sets (call them sandy machines), with respect to the pre-magnitudes w'_i. The only possible exception to this is machine m, which is always filled up to at least C, but might contain a sandy set (in this case it is a sandy machine).

Within machines of equal speed, zero or more sandy machines are followed by zero or more normal machines, since such machines have the same pre-magnitude. Each such sequence of sandy machines (i.e., of the same machine speed) will be a sandy segment. The remaining maximal sequences of normal machines, possibly spanning over different machine speeds will be the normal segments. Next, we redistribute the jobs *within* each segment in order to fulfill the conditions of Definitions 3 and 4 and prove the approximation bound of $(1 - \varepsilon) \cdot C = (1 - \varepsilon) \cdot \frac{\text{OPT}}{2}$. The sets of this final allocation will be denoted by $P_1, P_2, \ldots, P_m$.

Sandy segments. Consider first a segment consisting of all the sandy machines of the same speed s and having the same pre-magnitude $w' < 4Cs$. Using $\rho w' < 4\rho Cs \leq \delta Cs$, such machines have a workload of at least $(1 - \varepsilon/2) \cdot C \cdot s$ and at most $(1 - \varepsilon/2 + \delta) \cdot C \cdot s$. We now apply the smoothing procedure (see Section 2) to these machines and the jobs assigned to them. Then for each machine i of the segment we have

$$(1 - \varepsilon/2 - \delta) \cdot C \cdot s \leq |P_i| \leq (1 - \varepsilon/2 + 2\delta) \cdot C \cdot s. \tag{1}$$

The obtained partition on the segment adheres to Definition 3, and the cover is higher than $(1 - \varepsilon) \cdot C$; the pre-magnitudes can remain the valid magnitudes w_i of the jobsets.

An exceptional case occurs when machine m is (sandy and) is part of the segment. In this case the upper bound in (1) might fail, and the common magnitude of the segment needs to be increased acccordingly. However, then the segment is the very last one, and Claim 2 and the theorem still holds.

Normal segments. Consider now an arbitrary normal segment $(Q_s, \ldots, Q_t)$. We create a canonical allocation by running the canonization procedure (see the last lines of 4). Since sorting cannot decrease the cover [10], the cover remains above $C/(1+\delta) > (1-\varepsilon)C$. We also need to find proper magnitudes for machines in the normal segment. Before doing this, we conclude the main line of the proof by showing that the workloads $|P_i|$ are increasing in i. All other conditions of Definition 5 hold by construction.

Claim 2 *The workloads $|P_i|$ are increasing in i.*

Proof Clearly, the workloads within each segment are increasing, since the segments have either a canonical or a smooth allocation. Next we show that they are increasing over the whole schedule. First, we compare a normal set P_i with a sandy set P_j of a *preceding* sandy segment. Assume that $P_i = \widetilde{Q}_{i'}$, for some $s_j \leq s_{i'}$, where i' and i are in the same normal segment. We saw in (1) that for the sandy set

$$|P_j| \leq (1 - \varepsilon/2 + 2\delta) \cdot C \cdot s_j,$$

whereas for the normal set

$$\frac{1}{(1+\delta)} \cdot C \cdot s_{i'} \leq \frac{1}{(1+\delta)} \cdot |Q_{i'}| \leq |\widetilde{Q}_{i'}| = |P_i|.$$

Using $1 - \varepsilon/2 + 2\delta < 1 - \delta < \frac{1}{1+\delta}$, this proves $|P_j| < |P_i|$.

Assume now that P_i is a sandy set, and P_j is either a jobset in a preceding sandy segment, or the perturbed $\widetilde{Q}_{j'}$ set of a preceding normal segment. Let $s = s_j$ in the first, and $s = s_{j'}$ in the second case, respectively. In both cases, by construction $s \leq s_i/(1+\varepsilon)$. Furthermore, all the jobs in P_i, and in sets of previous segments have size of at most $\rho w_i \leq \delta C s_i$, which implies the bound

$$|P_j| \leq (C \cdot s + \delta C s_i)(1+\delta) \leq C \cdot s_i \cdot \left(\frac{1}{1+\varepsilon} + \delta\right)(1+\delta)$$

for both cases. Using the lower bound for $|P_i|$ from (1) and the fact that $(\frac{1}{1+\varepsilon} + \delta)(1+\delta) < 1 - \varepsilon/2 - \delta$ for $\delta < \varepsilon/12$, we obtain $|P_j| < |P_i|$. □

Finally, we define magnitudes w_i. Fix a normal segment, and let w'_0 be the smallest pre-magnitude in this segment. For each set P_i of the segment, we define the magnitude as $w_i = \max\{w'_0, 2^{\lceil \log |P_i| \rceil}\}$. For these magnitudes $w_i/5 < |P_i| \leq w_i$ holds.

Claim 3 *The magnitudes are increasing over the whole schedule, and the (P_j, w_j) are normal sets.*

Proof The magnitudes are increasing within the segment because the workloads are increasing. Furthermore, if the magnitude w_j of some set $P_j = \widetilde{Q}_i$ is larger than the pre-magnitude w'_i of Q_i, then

$$2 s_i C \leq w'_i \leq w_j/2 < |\widetilde{Q}_i| < (1+\delta)|Q_i|,$$

whereas $|Q_i|$ had to reach (only) a workload of $s_i C$. Thus, the last job in Q_i is at least as big as (roughly) the sum of all other jobs in Q_i, and in particular for $\delta < 1/5$ we obtain:

$(*)$ If $w_j > w'_i$ then Q_i contains a job of size at least $\frac{w_i}{5(1+\delta)}$.

Since in the subsequent *sandy* segment (of magnitude w) this jobsize is tiny by definition, we have $w_i/[5(1+\delta)] < \rho w$. Therefore, the magnitudes are increasing over the whole partition.

We show that the (P_j, w_j) are normal sets. Let $P_j = \widetilde{Q}_i$. Recall that (Q_i, w'_i) is normal by the definition of normal machines. With respect to the new magnitude w_j, there is at least one normal job in the set. This

is clear if $w_j \leq w'_i$, and follows from $(*)$ if $w_j > w'_i$. Assume now that the machine also contains tiny jobs with respect to w_j. Note that since the jobs are consecutive (disregarding perturbation) in each set, every other set has either only tiny jobs or only normal jobs with respect to w_j. However, any set in the same normal segment that has only tiny jobs with respect to w_j, must have a magnitude less than w_j (since each set does have a normal job for its own magnitude) and so (since magnitudes are increasing) it is a set P_k for some $k < j$. By the definition of canonical allocations and normal sets, this proves the normality of P_j. □
This proves the existence of a segmented partition with cover at least $(1-\varepsilon)\frac{\mathrm{OPT}}{2}$. □

4 Graph construction

In this section we construct a directed acyclic graph, depending on the set of input jobs P_I. The vertices represent either *normal jobsets*, or *sandy partition segments*. An arc between two vertices should indicate that the corresponding sets or segments can be neighbors in the partition (e.g., that some normal set P_i can be followed by a certain sandy segment $(P_{i+1}, \ldots, P_k)$). A given input speed vector $(s_1, \ldots, s_m)$, determines a weight on each graph vertex, meaning the (minimum) finish time induced by the workload(s) $|P_i|$. A path, leading over some $P_1, P_2, \ldots, P_m$, that maximizes the minimum weight over its vertices, represents an optimal solution among all segmented partitions.

The above outlined technique was introduced by Hochbaum and Shmoys [13] for a PTAS for related scheduling, and has been used for (monotone) approximation schemes for related scheduling [10, 4, 8, 7]. Based on this previous work, our graph construction (adapted for segmented partitions) is straightforward. As a difference to all of the known PTAS algorithms, the notion of segmented partitions allows for a pure and *exact* representation of the jobsets, and a very plain graph structure. Of course, we pay for this simplification with a loss of a factor 2 in the approximation ratio.

Set configurations *Set configurations* are used to represent normal jobsets. Each set configuration α is a triple $\alpha = (w, \vec{n}^o, \vec{n}^1)$, where w is the magnitude of the set, and the vectors $\vec{n}$ are *size vectors*. If the configuration is supposed to define the set P_i, this is done by the two size vectors defining the cumulative jobsets $\bigcup_{k=1}^{i-1} P_k$, and $\bigcup_{k=1}^{i} P_k$, respectively. Thus, size vectors represent sets of jobs of size between ρw and w (in fact, a prefix set of each job class), and a prefix subset of the tiny jobs of size at most ρw. They are indexed by the integers from $\lambda = \log_{1+\rho} \rho w$ to $\Lambda = \log_{1+\rho} w$, and have nonnegative integer coordinates. The entry n_l for some $l \in (\lambda, \Lambda]$ means that the cumulative jobset contains exactly the first n_l jobs of the class C_l. Observe that this is an adequate representation of canonical allocations, where jobs within each class appear in the fixed order. Finally, the coordinate n_λ stands for some *prefix subset* $\{p_1, p_2, \ldots, p_{n_\lambda}\}$ of all the jobs of size at most ρw. We can speak of a *valid set configuration* only if a handful of conditions are fulfilled. For instance, by the definition of normal sets it is required that either $\vec{n}^o_\lambda = \vec{n}^1_\lambda$ (no tiny jobs), or that $\vec{n}^1_\lambda$ is the number of jobs of size at most ρw (the largest tiny jobs are all in the set). Also, $w/5 < |P| \leq w$ must hold, and can be easily checked. The rest, like $\vec{n}^o \leq \vec{n}^1$, and other bounds on the coordinates, are straightforward, and will not be detailed here. For an illustration see Figure 3.

We bound the number of different valid set configurations. Every size vector has

$$\log_{1+\delta} w - \log_{1+\delta} \rho w = \log_{1+\delta} 1/\rho = \mathcal{O}(1/\delta \cdot \log(1/\rho))$$

integer coordinates between 0 and n. Each possible pair of size vectors determines a set, which has at most 3 possible valid magnitudes. Therefore, for constant δ there is a polynomial number of different set configurations.

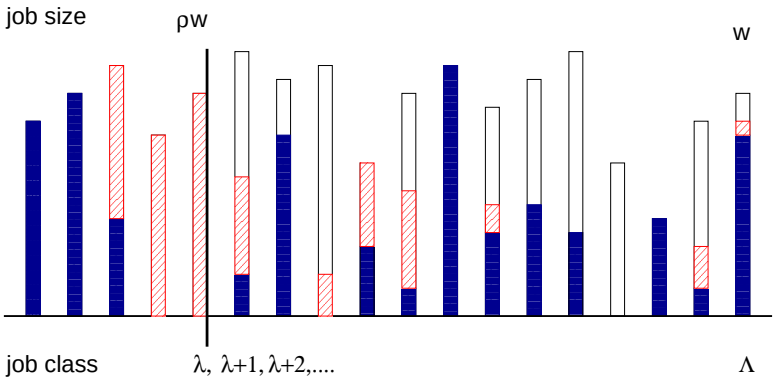

Figure 3: A set configuration: the thin rectangles represent job classes; the solid part belongs to the set $\bigcup_{k=1}^{i-1} P_k$, and the striped part to set P_i. Note that P_i has a contiguous set of the largest tiny jobs. The sets P_i which we construct in Theorem 1 (see Figure 2) always contain jobs of *consecutive* classes, where each class except the first and last class is completely contained in P_i.

Segment configurations A *segment configuration* $\beta = (w, r, n^o, n^1)$ stands for a sandy segment, and has altogether four positive integer entries. This tuple defines a smooth allocation of the jobs $\{p_{n^o+1}, p_{n^o+2}, \ldots, p_{n^1}\}$ into r sets of magnitude w. Notice that the jobs are distributed evenly over the segment, *regardless* of the machine speeds. Moreover, at any point of the calculation, the sets of the segment $P_i, P_{i+1}, \ldots, P_{i+r-1}$ (of increasing workloads) can easily be computed. In order to have a valid configuration, the conditions $p_{n^1} \leq \rho w$, and $w/5 < |P| \leq w$ (for each set P) must hold. The number of different valid segment configurations is bounded by $3mn^2$.

The directed graph $\mathcal{G}(V, A)$ The vertex set V of $\mathcal{G}$ has $m + 2$ layers. Each layer $i \in [1, m]$ contains a vertex (i, α) for every valid set configuration α, and a vertex (i, β) for every valid segment configuration $\beta = (w, r, n^o, n^1)$ for which $i + r \leq m + 1$. Recall that for any configuration on level i, the entry $\vec{n}^o$ (or n^o) uniquely determines the cumulative jobset $\bigcup_{k=1}^{i-1} P_k$. Similarly, the entry $\vec{n}^1$ (resp. n^1), encodes the cumulative set $\bigcup_{k=1}^{i} P_k$ (resp. $\bigcup_{k=1}^{i+r-1} P_k$)). We add a source vertex s on layer 0, that stands for the empty jobset (say, with $n^1 = 0$), and a sink vertex t on layer $m + 1$ for the complete jobset (say, with $n^o = n$).

Next we define the arc set A. There is an arc between two configurations if and only if they satisfy all of the following conditions. From a set configuration (i, α) all arcs lead into (set or segment) configurations of layer $i + 1$. From a segment configuration (i, β), all arcs lead into (set or segment) configurations of layer $i + r$. Obviously, a necessary condition for an arc between two configurations is that the $\vec{n}^1$ or n^1 entry of the first one should represent the same jobset as the $\vec{n}^o$ or n^o entry of the second one. Finally, it is required that the magnitudes w are nondecreasing along every arc, and similarly, the workloads of the represented sets must be nondecreasing (here for set configurations $|P_i|$ is meant, and for segment configurations we consider $|P_i|$ for incoming arcs, and $|P_{i+r-1}|$ for outgoing arcs).

The first two arc conditions ensure that any (s, t)-path of the graph induces a *partition* of the input jobs into m sets. Due to the fact that here the graph vertices represent jobsets *exactly*, (as opposed to different rounding techniques applied in previous work), the following statement is straightforward:

Proposition 1 *There is a one-to-one correspondence between segmented partitions of the input jobs P_I into m sets, and the directed (s, t)-paths in graph $\mathcal{G}$.*

Input: the directed acyclic graph $\mathcal{G}(V, A)$, and weights $f : V \to \mathbb{R}$

1. let $opt(t) = \infty$, and $opt(v) = -\infty$ $(v \in V - \{t\})$, and $f(s) = \infty$
2. for $i = m$ downto 0 do
   for every vertex x in layer i do
   i. let $succ(x) = y$, if $(x, y) \in A$, and $opt(y) = max_{(x,v)\in A} opt(v)$, and the configuration of y is minimal among all such vertices wrt. the total order $\prec$
   ii. let $opt(x) = \min\{f(x), opt(y)\}$
3. let $r = 0$ and $v_0 = s$
4. repeat $\{r := r + 1; v_r := succ(v_{r-1})\}$ until $v_r = t$

Output: optimal (s, t)-path $s = v_0, v_1, \ldots, v_r = t$.

Figure 4: Procedure MAXCOVER. Recall that the vertices of $\mathcal{G}$ are arranged on $m + 2$ layers, with the singleton vertices s and t in layers 0 and $m + 1$, respectively.

Finish times Note that both segmented partitions and the graph $\mathcal{G}(V, A)$ were defined independently of the speed vector. Now for given (rounded) input speeds $s_1 \leq s_2 \leq \ldots \leq s_m$, we can assign a *finish time* $f(v)$ (a weight) to every vertex $v \in V$ of the graph. For a vertex $v = (i, \alpha)$ with a set configuration α representing set P_i, the finish time is $f(v) = \frac{|P_i|}{s_i}$. If $v = (i, \beta)$ with induced jobsets $P_i, \ldots, P_{i+r-1}$, the (minimum) finish time is defined as

$$f(v) = \min\left\{ \frac{|P_k|}{s_k} \;\middle|\; i \leq k < i + r \right\}.$$

5 Monotone algorithm for covering

Once the graph $\mathcal{G}$ is constructed, the problem boils down to finding an (s, t)-path of maximum cover. This can be done by a standard dynamic programming algorithm which we call MAXCOVER (Figure 4); a very similar algorithm was used in [7]. Because tie-breaking rules are crucial for monotonicity, we fix an arbitrary (e.g., lexicographical) linear order $\prec$ over all valid (set and segment) configurations.

The monotone algorithm MONCOVER is presented in Figure 5. Since for constant ε the number of different configurations is polynomial in n and m, the size of $\mathcal{G}$ is polynomial, and the algorithm runs in polynomial time. Let OPT_σ and OPT_s denote the optimal cover values with the original and the rounded speeds, respectively. Clearly, we have $\text{OPT}_\sigma/(1 + \varepsilon) \leq \text{OPT}_s$, since this ratio holds for the cover of every fixed allocation. Moreover, by Theorem 1 and Proposition 1, the output of MONCOVER has a cover of at least $\text{OPT}_s(1 - \varepsilon)/2$. Altogether we obtain that the cover of the output is at least

$$\frac{\text{OPT}_\sigma}{2} \cdot \frac{(1 - \varepsilon)}{(1 + \varepsilon)} \geq \frac{\text{OPT}_\sigma}{2 + 5\varepsilon}.$$

Theorem 2 *Algorithm* MONCOVER *is monotone.*

Input: job set P_I, machine speeds $\sigma_1 \leq \cdots \leq \sigma_m$, and $\varepsilon \in (0, 1/5)$.

1. Fix an appropriate $\delta < \varepsilon/12$, fix a nondecreasing order of the jobs, determine the job classes C_l, and construct the graph $\mathcal{G}(V, A)$.
2. Round up each speed σ_i to s_i, the nearest integral power of $(1 + \varepsilon)$.
3. Using the rounded speeds, compute the finish time $f(v)$ of every graph vertex $v \in V$.
4. Compute the optimal (s, t)-path (of maximum cover) of $\mathcal{G}$ with procedure MAXCOVER.
5. Output the job partition $P_1, P_2, \ldots, P_m$ determined by the path.

Output: a partition of the input jobs with a cover within a factor $2 + 5\varepsilon$ of the optimum cover.

Figure 5: The algorithm MONCOVER

Proof The proof is analogous to part of the monotonicity proof in [7]. With given input speed vector $\sigma_1, \sigma_2, \ldots, \sigma_m$, let the output of MONCOVER be $P_1, P_2, \ldots, P_m$. We assume that for some machine $i \in [1, m]$, the speed σ_i is increased to $\sigma' > \sigma_i$, the new rounded speed being s', and show that with this new input the algorithm allocates at least as much workload to the machine, as with speed σ_i.

We start with a couple of simple observations. Since the machines are indexed in increasing order of speed (breaking ties by some fixed machine order), the new index i' of the machine is at least i. If $s' = s_i$ (the rounded speed remains the same), then the output of the algorithm is exactly the same, and the allocated workload will be $|P_{i'}| \geq |P_i|$, and the theorem holds. Further, it is enough to prove the theorem for the case when $s' = (1 + \varepsilon)s_i$, and the machine index does not change, i.e., i was the highest index of speed s_i, and becomes the lowest index of speed $s'_i = s' = (1 + \varepsilon)s_i$. For all other cases the proof easily follows by 'continuously' increasing σ_i to σ'.

Observe that the graph $\mathcal{G}(V, A)$ constructed in step 1 does not change, and $f'(v) \leq f(v)$ holds for the new finish time $f'(v)$ of each vertex $v \in V$. Now we turn to procedure MAXCOVER. Because the finish times cannot increase, the minimum of the finish times over any path in $\mathcal{G}$ cannot increase. In particular, for every vertex v the optimum cover $opt(v)$ over all (v, t)-paths cannot increase either, i.e., $opt'(v) \leq opt(v)$ holds, where $opt'()$ denotes the new optimum. Note that the optimal (v, t)-path itself might change.

If the path which is output by MAXCOVER is the same for both input speeds, then the theorem holds. So, let $s, v_1, v_2, \ldots, v_r = t$ be the output path with speed s_i, and $s, v'_1, v'_2, \ldots, v'_{r'} = t$ be the output path with speed s'_i, and k be the minimum index s.t. $v_k \neq v'_k$. That is, $v_k = succ(v_{k-1})$ for the first, and $v'_k = succ'(v_{k-1})$ for the second input. Since no vertex could increase its $opt()$ value, in the second input v'_k could improve its relative position to v_k only due to $opt'(v_k) < opt(v_k)$. In particular, the path $v_k, v_{k+1}, \ldots, v_r$, decreased its cover from $opt(v_k)$ to *at most* $opt'(v_k)$ when s_i increased. That is, i must have become a bottleneck machine, and the minimum finish time over $v_k, v_{k+1}, \ldots, v_r$ became $|P_i|/s'_i = f'(v_q)$, where machine i is represented by the configuration of vertex v_q in the path. So, we have

$$\frac{|P_i|}{s'_i} = f'(v_q) \leq opt'(v_k).$$

On the other hand, $opt'(v_k) \leq opt'(v'_k)$, since v'_k was selected over v_k, and $opt'(v'_k) \leq |P'_i|/s'_i$, because P'_i is determined by the new optimal path. Putting it together, we obtain $|P_i| \leq |P'_i|$. □

6 Conclusions

The question whether there is a monotone PTAS for related scheduling with cover optimization remains open. The same holds for minimizing the L_p-norm of finish times for any $p > 1$. While for the respective (non-strategic) problems the classic PTAS, as well as the randomized monotone PTAS are easy to adapt [10, 8], the same does not seem to hold concerning the deterministic monotone PTAS.

References

[1] Aaron Archer and Éva Tardos. Truthful mechanisms for one-parameter agents. In *Proc. 42nd Annual Symposium on Foundations of Computer Science*, pages 482–491, 2001.

[2] Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *Proc. 39th Annual ACM Symp. Theory of Comp. (STOC)*, pages 114–121, New York, NY, USA, 2007. ACM.

[3] Yossi Azar and Leah Epstein. On-line machine covering. In *Proc. 5th Annual Eur. Symp. Algs. (ESA)*, volume 1284 of *LNCS*, pages 23–36. Springer, 1997.

[4] Yossi Azar and Leah Epstein. Approximation schemes for covering and scheduling on related machines. In *Proc. 1st Intl. Workshop Approx. Algs. for Comb. Opt. (APPROX)*, volume 1444 of *LNCS*, pages 39–47. Springer, 1998.

[5] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *Proc. 41st Annual ACM Symp. Theory of Comp.*, pages 543–552, New York, NY, USA, 2009. ACM.

[6] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *Proc. 50th Annual IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 107–116. IEEE Computer Society, 2009.

[7] George Christodoulou and Annamária Kovács. A deterministic truthful PTAS for scheduling related machines. In *Proc. 21st SIAM Symp. on Disc. Algs. (SODA)*, pages 1005–1016. SIAM, 2010.

[8] Peerapong Dhangwatnotai, Shahar Dobzinski, Shaddin Dughmi, and Tim Roughgarden. Truthful approximation schemes for single-parameter agents. In *Proc. 49th IEEE Symp. on Found. of Comp. Sci. (FOCS)*, 2008.

[9] Pavlos S. Efraimidis and Paul G. Spirakis. Approximation schemes for scheduling and covering on unrelated machines. *Theoretical Computer Science*, 359((1-3)):400–417, 2006.

[10] Leah Epstein and Jiří Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):43–57, 2004.

[11] Leah Epstein and Rob van Stee. Maximizing the minimum load for selfish agents. *Theoretical Computer Science*, 411(1):44–57, 2010.

[12] Uriel Feige. On allocations that maximize fairness. In *Proc. 19th annual ACM-SIAM Symp. Discr. Algs. (SODA)*, pages 287–293, Philadelphia, PA, USA, 2008. SIAM.

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

[14] Ahuva Mu'alem and Michael Schapira. Setting lower bounds on truthfulness. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1143–1152, 2007.

[15] Roger B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6:58–73, 1981.

[16] Martin Skutella and José Verschae. A robust PTAS for machine covering and packing. In *Proc. 18th Annual European Symposium on Algorithms (ESA 2010) (1)*, volume 6346 of *LNCS*, pages 36–47. Springer, 2010.

[17] Gerhard J. Woeginger. A polynomial time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20(4):149–154, 1997.